
TA-nmon-hec Documentation

Release 1.3.0

Guilhem Marchand

Nov 06, 2018

Contents

1	Overview:	3
1.1	About the TA-nmon-hec, technical addon for Nmon Performance app for Splunk	3
1.2	Release notes	4
1.3	Known Issues	5
1.4	Support	6
1.5	Issues and enhancement requests	7
1.6	Operating Systems compatibility	7
1.7	Introduction to Nmon processing	9
1.8	Benchmarks & TA-nmon foot print	11
1.9	Scripts and Binaries	31
1.10	Pre-requisites	33
1.11	Deployment Matrix	34
2	Processing:	35
2.1	Processing workflow in action	35
3	Deployment and configuration:	49
3.1	Deployment	49
3.2	Upgrade	53
3.3	Eventgen testing	53
3.4	Extend Nmon with external data	55
3.5	Configure your options with nmon.conf	59
4	Troubleshoot:	69
4.1	Troubleshoot	69
5	Various:	77
5.1	FAQ	77



Nmon Performance is now associated with Octamix to provide professional solutions for your business, and professional support for the Nmon Performance solution.

For more information: : <http://nmon-for-splunk.readthedocs.io/en/latest/support.html#octamix-support>

Contents:

1.1 About the TA-nmon-hec, technical addon for Nmon Performance app for Splunk

- Author: Guilhem Marchand
- First release published in August 2017
- Purposes:

The TA-nmon-hec for the Nmon Performance application for Splunk implements the excellent and powerful nmon binary known as Nigel's performance monitor. Originally developed for IBM AIX performance monitoring and analysis, it is now an Open source project that made it available to many other systems. It is fully available for any Linux flavor, and thanks to the excellent work of Guy Deffaux, it also available for Solaris 10/11 systems using the sarmon project.

The Nmon Performance monitor application for Splunk will generate performance and inventory data for your servers, and provides a rich number of monitors and tools to manage your AIX / Linux / Solaris systems.



Nmon Performance is now associated with Octamix to provide professional solutions for your business, and professional support for the Nmon Performance solution.

For more information:

1.1.1 Splunk versions

The TA-nmon-hec is compatible with any version of Splunk Enterprise 6.x and Splunk Universal Forwarder 6.x.

1.1.2 Index time operations

The application operates index time operation, the PA-nmon_light add-on must be installed in indexers in order for the application to operate normally.

If there are any Heavy forwarders acting as intermediate forwarders between indexers and Universal Forwarders, the TA-nmon add-on must be deployed on the intermediate forwarders to achieve successfully index time extractions.

1.1.3 About Nmon Performance Monitor

Nmon Performance Monitor for Splunk is provided in Open Source, you are totally free to use it for personal or professional use without any limitation, and you are free to modify sources or participate in the development if you wish.

Feedback and rating the application will be greatly appreciated.

- Join the Google group: <https://groups.google.com/d/forum/nmon-splunk-app>
- App's Github page: <https://github.com/guilhemmarchand/nmon-for-splunk>
- Videos: <https://www.youtube.com/channel/UCGWHd40x0A7wjk8qskyHQcQ>
- Gallery: <https://flic.kr/s/aHskFZcQBn>

1.2 Release notes

1.2.1 Requirements

- Splunk 6.x / Universal Forwarder v6.x and later Only
- Universal Forwarders clients system lacking a Python 2.7.x interpreter requires Perl WITH Time::HiRes module available

1.2.2 What has been fixed by release

V1.3.37:

- fix: nmon_helper.sh is referring to TA-nmon #10

V1.3.36:

- fix: OpenSSL lib issues on full Splunk instances #9

V1.3.35:

- feature: Solaris - SARMON upgrade to v1.12 (Sparc FIFO mode) #6
- fix: Rename eventgen.conf to avoid splunkd WARN messages #7

V1.3.34:

- feature: Solaris - SARMON upgrade to v1.12 (Sparc FIFO mode) #6

V1.3.34:

- fix: reactivating the JFSFILE / JFSINODE collections until new core release is available to prevent missing features

V1.3.33:

- fix: Python parser - header detection correction for nmon external monitoring
- fix: Perl parser issue - UARG parsing issue for AIX #2
- fix: unexpected operator issue during process identification #3
- fix: prevent bundle validation warn messages with spec files in README directory
- feature: Add df information for improved file system monitoring and storage capacity planning
- feature: JFSFILE/JFSINODE are being replaced (and deactivated) by external collection with DF_STORAGE/DF_INODES

V1.3.32:

- fix: Python parser - preserve data ordering when possible during key value transformation
- feature: eventgen implementation (2 AIX and 2 Linux servers with stress load on systems)

V1.3.31:

- fix: Python parser issue - epoch time stamp incorrectly parsed for dynamic data #1

V1.3.30:

- fix: check curl availability in hec_wrapper.sh
- fix: redirect to null any curl stderr/stdout in hec_wrapper.sh
- fix: missing log_date function in hec_wrapper.sh

V1.3.29:

Notes: This is the first version of the TA-nmon-hec, for practical purposes and because it shares most of the components, the TA-nmon-hec follows the TA-nmon version

- First release of the TA-nmon-hec

1.3 Known Issues

Major or minor bug, enhancement requests will always be linked to an opened issue on the github project issue page:

<https://github.com/guilhemmarchand/nmon-for-splunk/issues>

Please note that once issues are considered as solved, by a new release or support exchanges, the issue will be closed. (but closed issues can still be reviewed)

Current referenced issues:

- **fifo implementation not ready for Solaris on Sparc architectures:**

The sarmon binary for Sparc processor has not been released yet and is under compilation.

Once the binary will have been released, the TA-nmon using fifo will be compatible with Solaris Sparc processors.

1.4 Support

1.4.1 Octamis professional support for business



support contract by Octamis limited.

Contact us at: sales@octamis.com

Nmon Performance is now available with professional

1.4.2 Community support

Nmon Performance Monitor for Splunk is provided in Open Source, you are totally free to use it for personal or professional use without any limitation, and you are free to modify sources or participate in the development if you wish.

This application and all of its components are provided under the Apache 2.0 licence, please remember that it comes with no warranty even if i intend to do my best in helping any people interested in using the App.

DISCLAIMER:

Unlike professional services, community support comes in “best effort” with absolutely no warranties.

Companies using this great piece are kindly invited to subscribe for a professional support contract to help us continuing developing the Nmon Performance solution!

Github

The Nmon Performance application is hosted on Github at the following location:

<https://github.com/guilhemmarchand/TA-nmon>

Use Github to open an issue for errors and bugs to be reported, or to ask for enhancements requests.

You can even provide your own improvements by submitting a pull request.

Splunk Answers

Splunk has a strong community of active users and Splunk Answers is an important source of information.

Access previous messages of users or open your own discussion:

<https://splunkbase.splunk.com/app/3668>

<http://answers.splunk.com/answers/app/1753>

Google Group Support

An App dedicated Google Group has been created:

<https://groups.google.com/d/forum/nmon-splunk-app>

This is also a great source of support from people using the Application, and you can also (if you subscribe to mailing news) receive important notifications about the App evolution, such as main release announcements.

1.5 Issues and enhancement requests

For any bug reporting, or enhancement request about the Nmon Performance application, you can:

- Open a question on Splunk Answers related to the app: <https://answers.splunk.com/app/questions/3668.html>
- Open an issue on the Git project home page: <https://github.com/guilhemmarchand/TA-nmon/issues>
- Get in touch by mail: guilhem.marchand@gmail.com

1.6 Operating Systems compatibility

1.6.1 OS compatibility

The TA-nmon is compatible with any version of:

- IBM AIX 6.1 (certified starting OSlevel 6.1.9.101, TL09)
- IBM AIX 7.1 (certified starting OSlevel 7.1.4.1, TL04)
- IBM AIX 7.2 (certified starting OSlevel 7.2.0.1, TL00)
- Linux x86 (32 / 64 bits)
- Linux PowerPC (32 / 64 bits in LE and BE)
- Linux on z Systems (s390 / s390x)
- Linux ARM
- Solaris 10, Solaris 11 on Sparc and x86

1.6.2 OS certification

Here is a non exhaustive list of systems and version that the TA-nmon is / has been intensively qualified:

IBM AIX:

- IBM AIX 6.1 (IBM POWER 8)
- IBM AIX 7.1 (IBM POWER 8)
- IBM AIX 7.2 (IBM POWER 8)

Linux on IBM PowerPC:

- SUSE Linux 12.2 LE (IBM POWER 8)
- SUSE Linux 11.4 BE (IBM POWER 8)
- RedHat Linux 7.3 LE (IBM POWER 8)

- RedHat Linux 7.2 LE (IBM POWER 8)
- Red Hat Linux 6.9 BE (IBM POWER 8)
- Red Hat Linux 6.5 BE (IBM POWER 8)
- Ubuntu 17.04 LTS (IBM POWER 8)
- Ubuntu 16.04 LTS (IBM POWER 8)
- Ubuntu 14.04 LTS (IBM POWER 8)

Linux x86, 32 bits and 64 bits:

Ubuntu:

- ubuntu-1704-64, ubuntu-1704-32
- ubuntu-1610-64, ubuntu-1610-32
- ubuntu-1604-64, ubuntu-1604-32
- ubuntu-1404-64, ubuntu-1404-32
- ubuntu-1204-64, ubuntu-1204-32

Oracle Linux (OL):

- oraclelinux-73-64
- oraclelinux-72-64
- oraclelinux-68-64
- oraclelinux-67-64
- oraclelinux-67-32

CentOS:

- centos-73-64
- centos-72-64
- centos-68-64, centos-68-32
- centos-67-64, centos-67-32

Debian:

- debian-8-64, debian-8-32
- debian-7-64, debian-7-32

SUSE Linux Enterprise Server: (SLES)

- sles11sp3
- sles12
- sles12sp1

OpenSuse:

- opensuse-13

Redhat Enterprise (RHEL):

- rhel5
- rhel65

- rhel73

Fedora:

- fedora24
- fedora25

Amazon AMI:

- AMI 2017.03
- AMI 2016.09

SOLARIS:

- solaris-11.3
- solaris-10

1.7 Introduction to Nmon processing

1.7.1 Nmon processing

The TA-nmon-hec embeds different scripts to perform various tasks from starting nmon binaries to the creation of the final data to be indexed by Splunk.

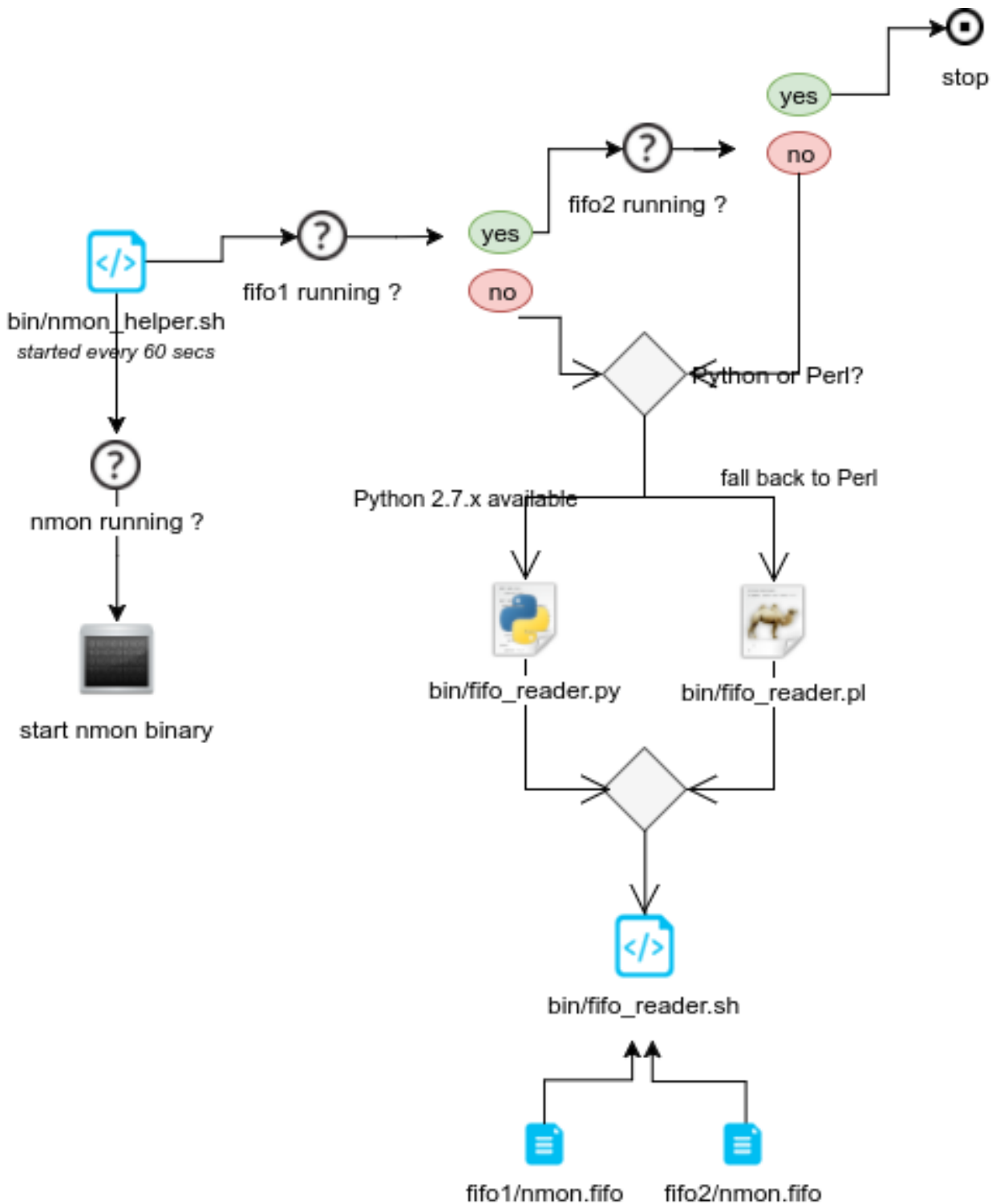
The following items expose the main steps of the processing, the technical details of each scripts and options are globally covered by the documentation.

bin/nmon_helper.sh

The “nmon_helper.sh” script is scheduled to be run every 60 seconds by Splunk, it is responsible for various tasks such as:

- identify the system (OS, processor architecture...)
- load default and custom configurations
- identify the best nmon binary candidate
- identify the running nmon process
- identify fifo_reader process running, and start the reader
- start the nmon binary

Simplified representation of the processing tasks:



bin/fifo_consumer.sh

The “fifo_consumer.sh” script is scheduled to be run every 60 seconds by Splunk, its purpose is consuming the dat files (different part of the nmon file) and stream its content to nmon2csv parsers:

- access the fifo files, wait at least 5 seconds since its last update

- Stream the content of the files to the nmon2kv parser
- empty the nmon_data.dat files for the next cycle

bin/nmon2kv.sh|.py|.pl

These are the nmon2kv parsers:

- the nmon2kv.sh is a simple wrapper which will choose between the Python and Perl parser
- the content is being read in stdin, and piped to the nmon2kv Python or Perl parser
- the Python or Perl parser reads the data, does various processing tasks and stream the data to the Splunk HEC endpoint over http/https
- additionally, the parser can write data to local files on the machine file system, which is disabled by default

1.8 Benchmarks & TA-nmon foot print

1.8.1 What does cost the TA-nmon?

Before deploying the TA-nmon on your systems, you will want to know accurately what will be its costs in term of CPU, memory, and disk I/O.

Since the release 1.3.x of the TA-nmon, we have worked real hard to reduce it at the most, and good news, we got it minimal now!

Splunk Universal Forwarder

The TA-nmon needs off course the Splunk Universal Forwarder (or Splunk Enterprise!) to operate on top of it.

The Splunk Universal Forwarder is highly optimised to have the lowest level of costs in CPU and memory, on its own the Splunk Universal Forwarder has a very negligible foot print.

Please consult the official Splunk Universal Forwarder documentation: <http://docs.splunk.com/Documentation/Forwarder/latest/Forwarder/Abouttheuniversalforwarder>

nmon binaries

Depending on the operating system, the TA-nmon will start the nmon binary when required. (topas-nmon for AIX, nmon for Linux, sarmon for Solaris)

On its own, the resources foot print of the nmon binary is really very low, almost not detectable, its foot print is negligible.

TA-nmon

The TA-nmon does various processing tasks on the data being generated by nmon, this is where there are some risks of a CPU, memory and I/O foot print.

To avoid these risks, and limit at the maximum the amount of resources to be used, we implement fifo files, which allow controlling a constant volume of data to be proceeded.

Thanks to this implementation, the TA-nmon foot print is now very low and constant, the following analysis will factually demonstrate real costs of the TA-nmon processing:

Analysis scenario:

To analyse the costs, we will use a small Linux server (1 CPU, 2 GB RAM) and compare different situation in details:

- run a Splunk Universal Forwarder instance (with connection to a deployment server) + the TA-nmon
- run a Splunk Universal Forwarder instance (with connection to a deployment server) + an independent nmon process to collect performance statistics
- run a Splunk Universal Forwarder instance (without connection to a deployment server) + an independent nmon process to collect performance statistics
- nothing running else but the independent nmon process to collect performance statistics

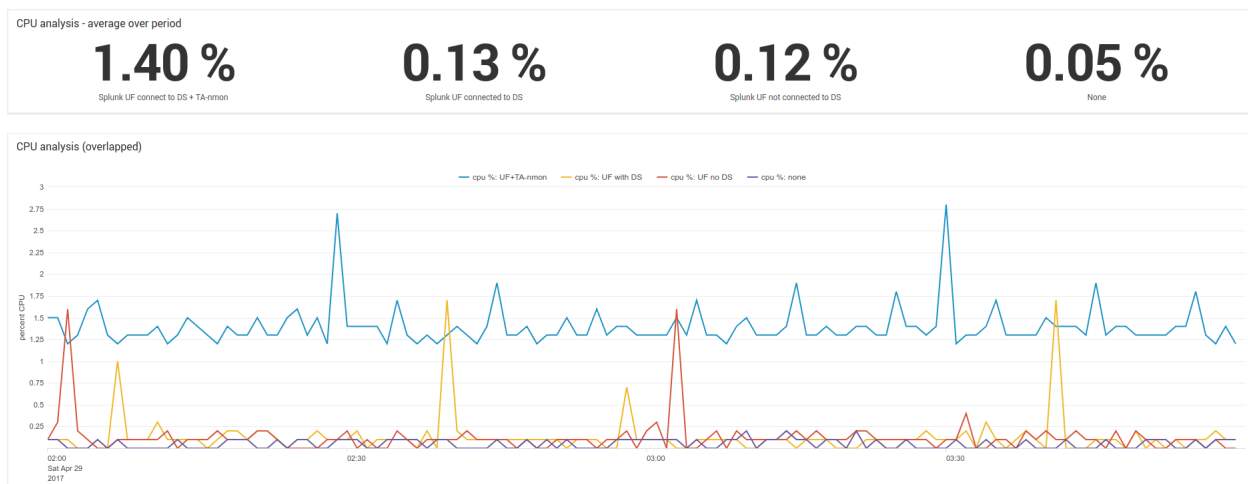
Notes:

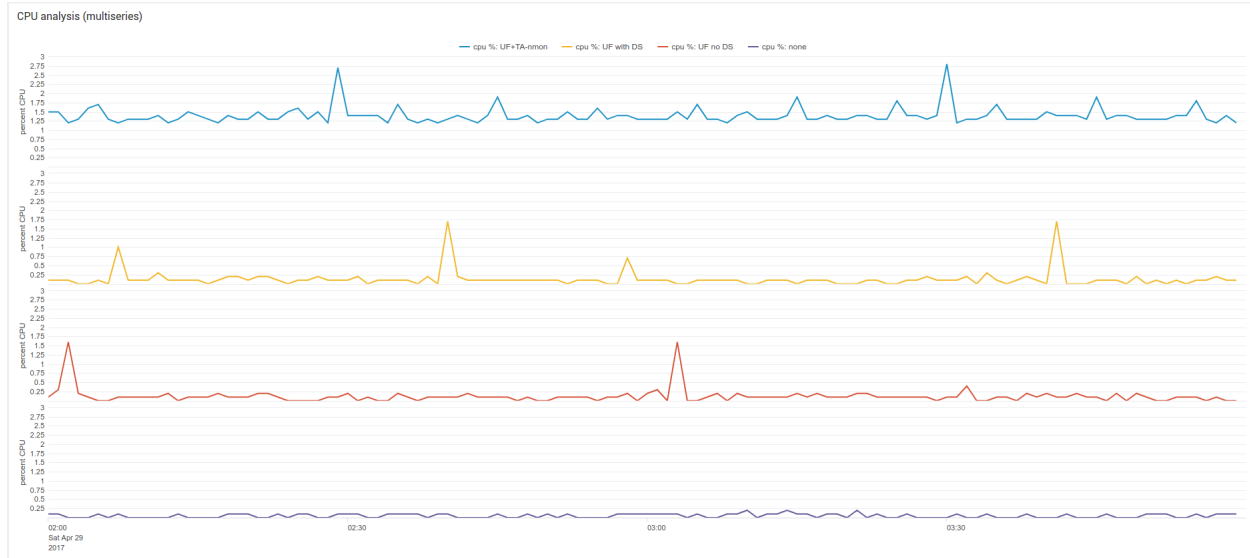
- analysing the system load with and without connection to a Splunk deployment server (DS) will be useful to isolate the cost of “calling home” from the UF to the DS

Finally, after having a run of 2 hours minimum for each scenario, we ingest the external nmon file (for non TA-nmon scenarios) and perform the analysis and comparison.

CPU usage comparison:

Average CPU % usage over periods for each scenario:

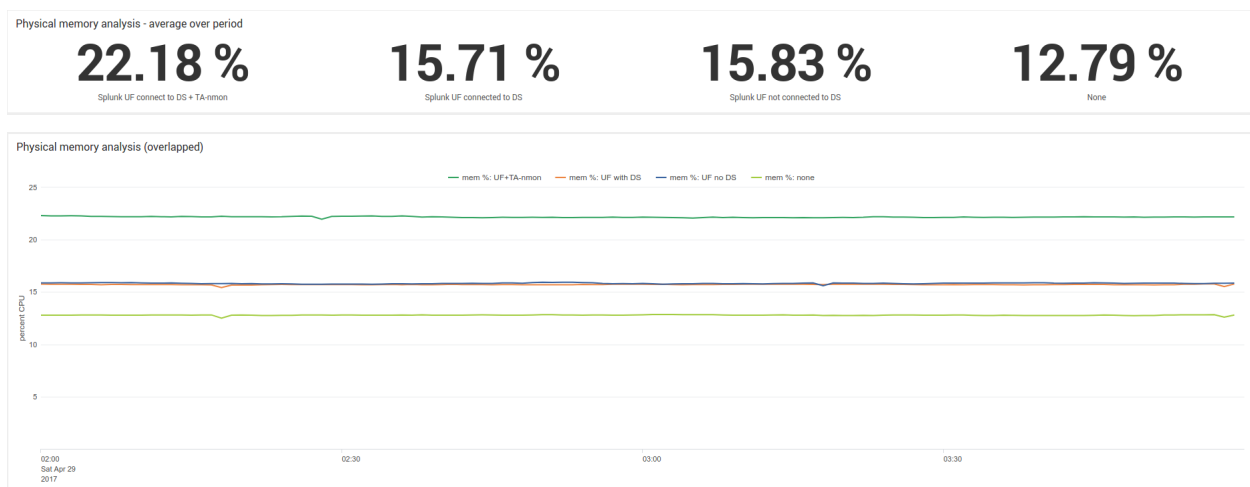




Observations:

- the average cost of running the Splunk Universal Forwarder (doing nothing) is similar with or without a connection to a deployment server
- the average CPU cost is approximately 0.10 % of global CPU usage on this server (usage-usage without UF)
- the imputable average CPU cost of running UF + TA-nmon is approximately 1.35% (TA-nmon processing costs, Splunk UF ingestion costs)
- the average CPU usage of system + UF + TA-nmon is approximately 1.40%
- We can observe an hourly task consuming CPU and imputable to the Splunk Universal Forwarder only (peaks exist without the TA-nmon, but only when running the UF)
- due to this hourly task of the Splunk Universal Forwarder (quick CPU peaks up to 1.7% without DS, up to 1.6% with DS connection), we can observe quick CPU peaks with UF + TA-nmon up to 2.8% CPU

Average physical memory % usage over periods for each scenario:

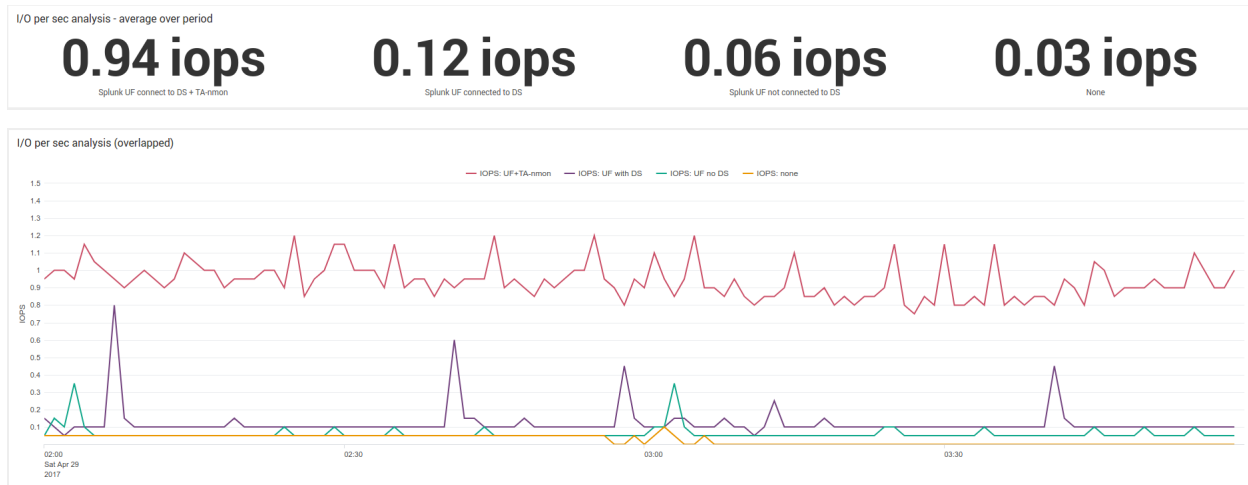


Observations:

- there is a small memory footprint of running the Splunk Universal Forwarder (approximately 3.20% of physical memory)

- this is not necessary what will proportionally cost on any system running a UF, this statistic has to be considered in the context of this configuration
- as well, we can observe a supplementary approximately 6% of memory costs running the UF + TA-nmon (TA-nmon processing costs, Splunk UF ingestion costs)
- caution: these memory utilisation statistics are what the system really uses, not necessary what the Splunk Universal Forwarder or the TA-nmon will use

Average I/O per second (IOPS) over periods for each scenario:



Observations:

- the level of IOPS imputable to the activity of the Universal Forwarder (when doing nothing) is obviously almost null
- when running the UF + TA-nmon, the level of IOPS is approximately 1 I/O per second.

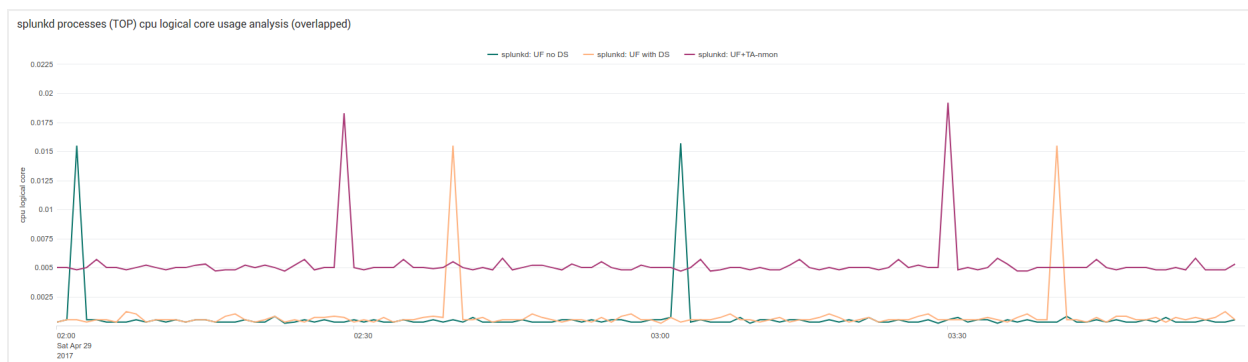
splunkd process (TOP data):

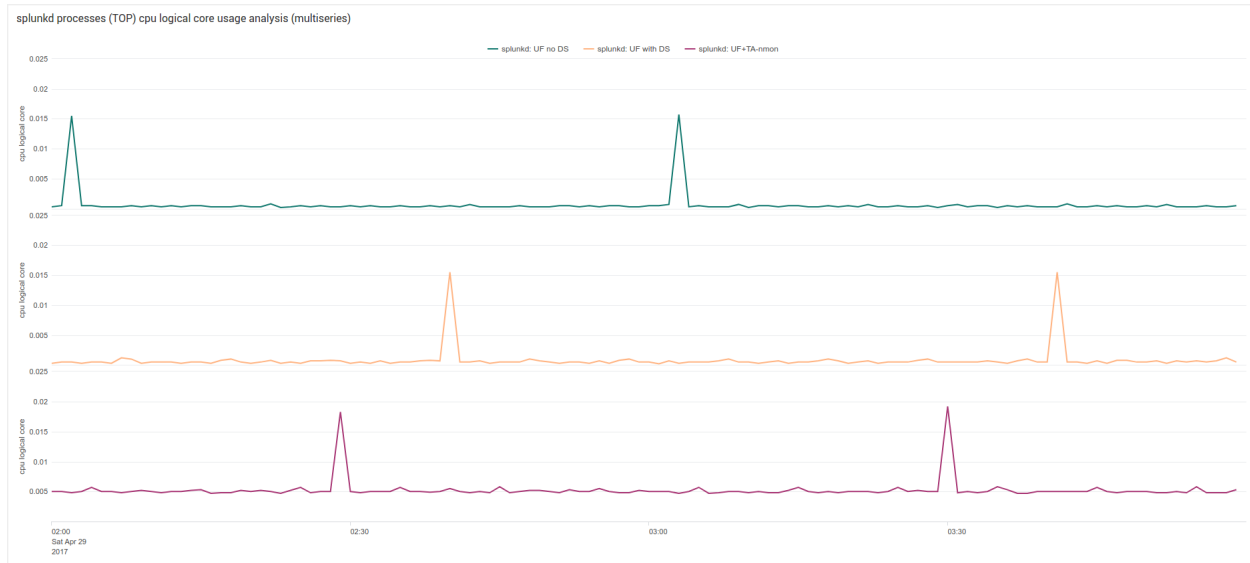
Notes:

For this exercise, we use the nmon binary in unlimited processes capture mode (option -I -1), this mode allows capturing the full processes table even such that capture low consuming processes.

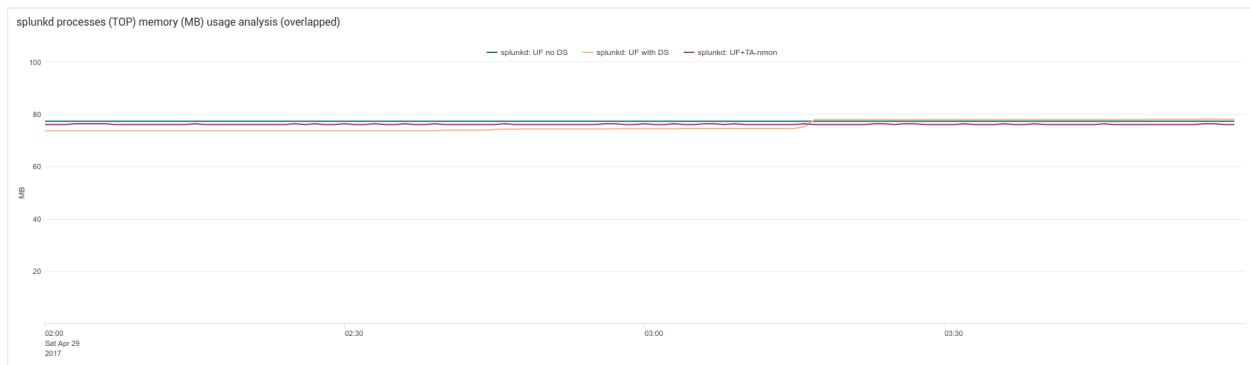
See: [Linux OS specific options](#)

splunkd CPU logical core usage:





splunkd memory usage:



Observations:

- we can clearly observe the hourly peak of CPU due to the Splunk Universal Forwarder
- CPU utilisation with or without deployment server connection is almost identical, the cost of calling home from the UF to the DS is almost null

Conclusions:

- the TA-nmon usage is stable and constant over time
- due to this internal Splunk Universal Forwarder hourly task, we can observe small hourly peaks of CPU usage
- running the Splunk Universal Forwarder + the TA-nmon generates approximately 1.35% of CPU usage on this machine
- the Splunk Universal Forwarder itself but doing nothing has obviously a very limited CPU foot print (but this mysterious hourly task!)
- the fifo implementation introduced in the TA-nmon 1.3.x allows now a very limited and constant system footprint!

The dashboard xml code used for this analysis is available in the Git docs directory, it has hardcoded host and time ranges but can be useful if you want to do your own analysis:

https://github.com/guilhemmarchand/TA-nmon/blob/master/docs/resources/footprint_analysis_and_comparison.xml

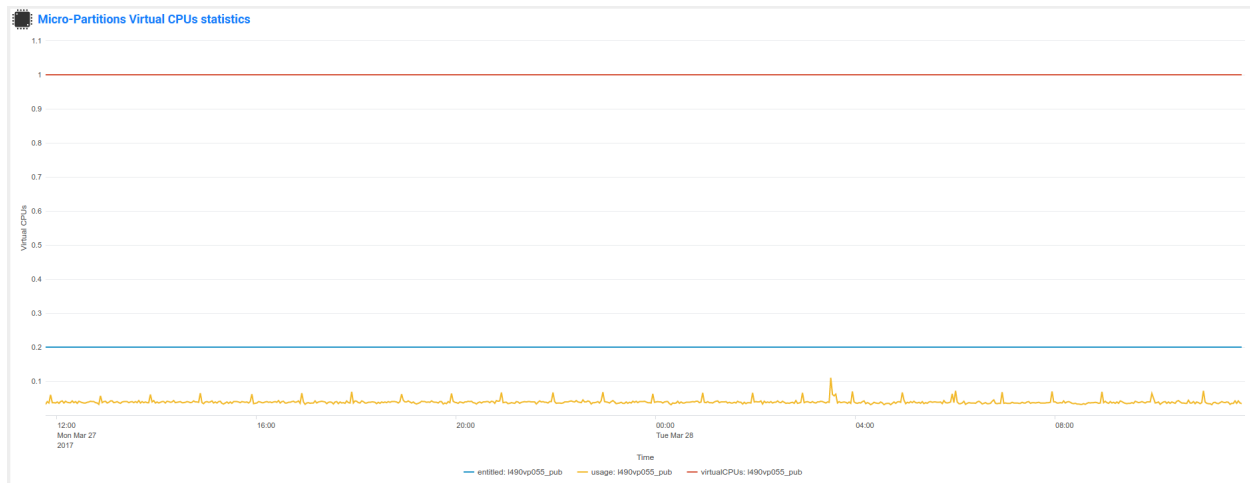
Enjoy!

IBM AIX BENCHMARKS:

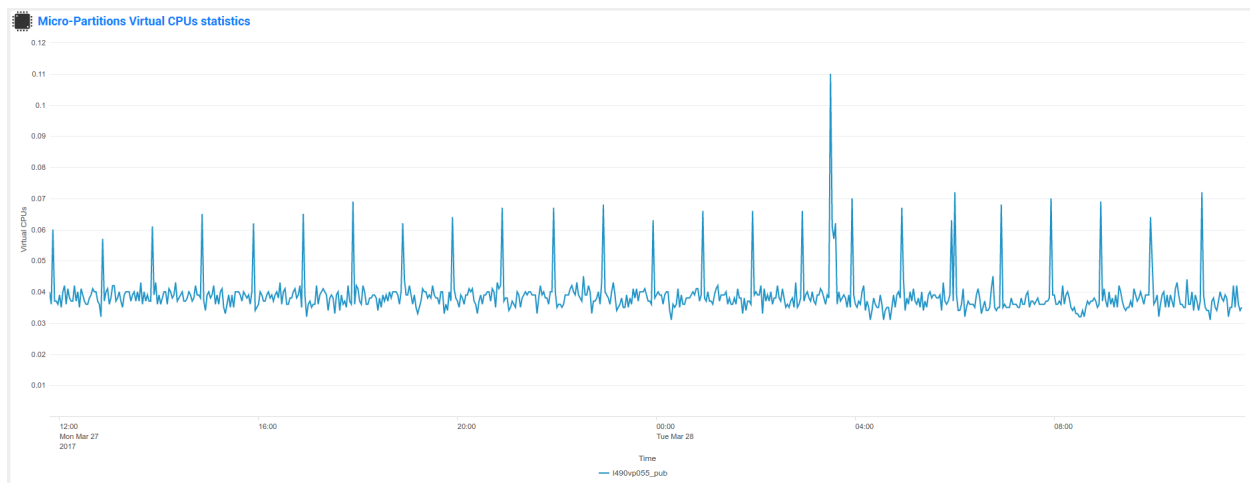
IBM AIX 6.1 ON POWER8 / Entitled 0.2 / VirtualCPUs 1:

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Perl interpreter

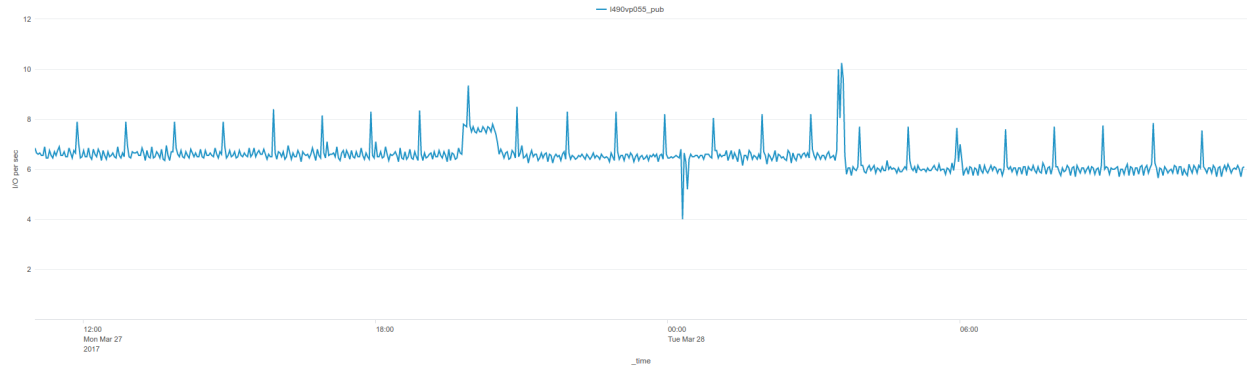
lpar usage over 24 hours:



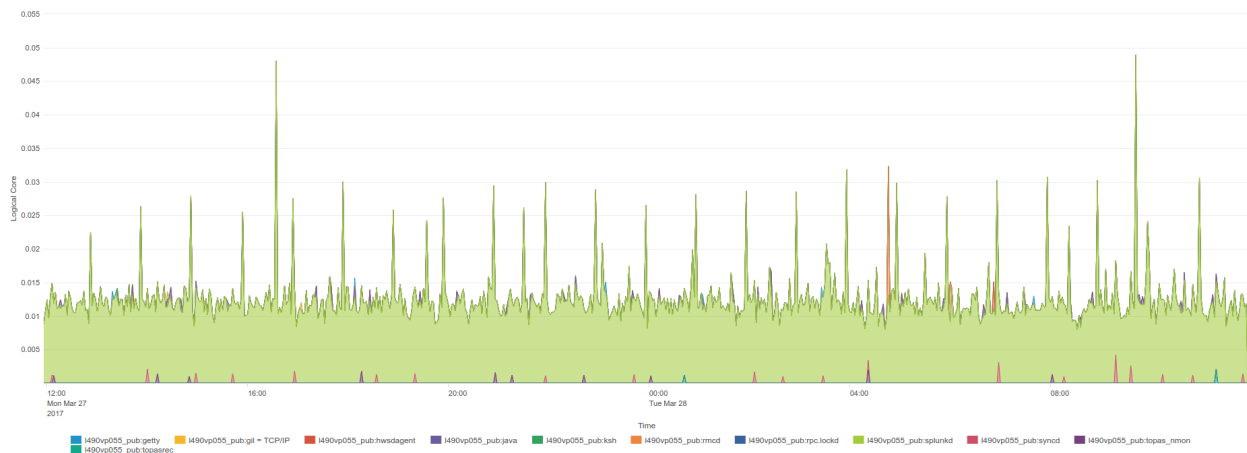
lpar usage_zoom over 24 hours:



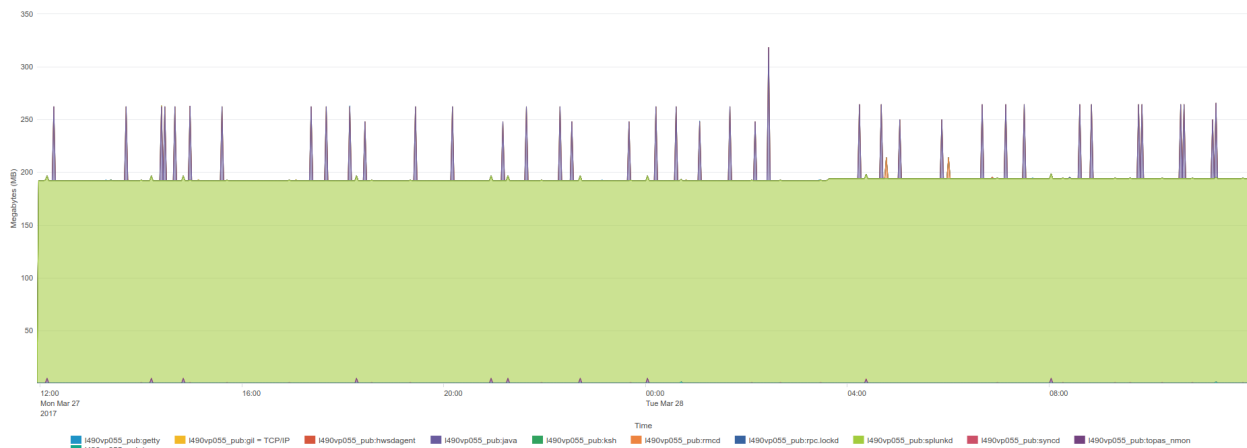
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



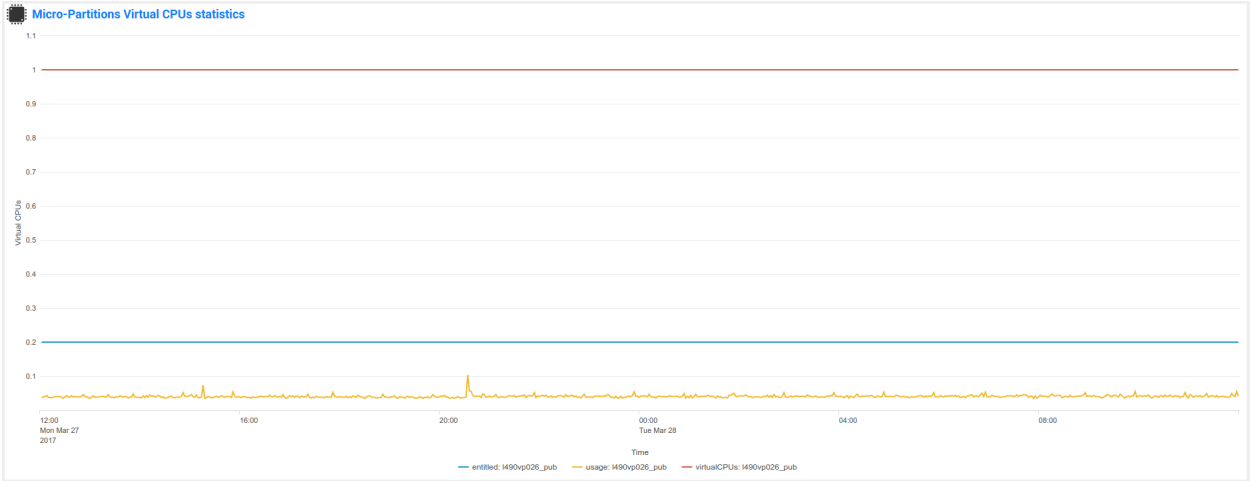
TOP processes memory usage over 24 hours:



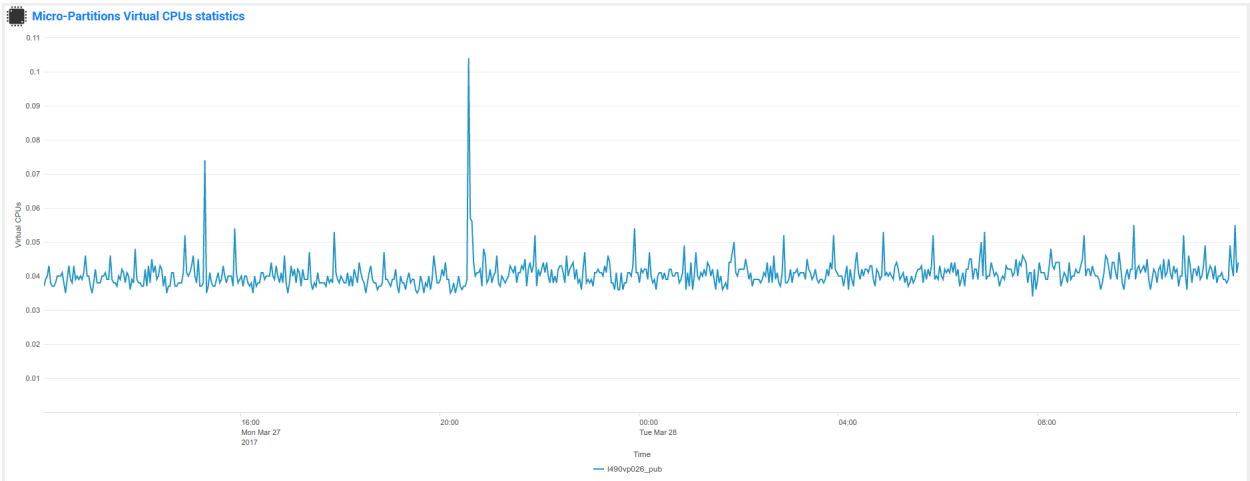
IBM AIX 7.1 ON POWER8 / Entitled 0.2 / VirtualCPU 1:

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Perl interpreter

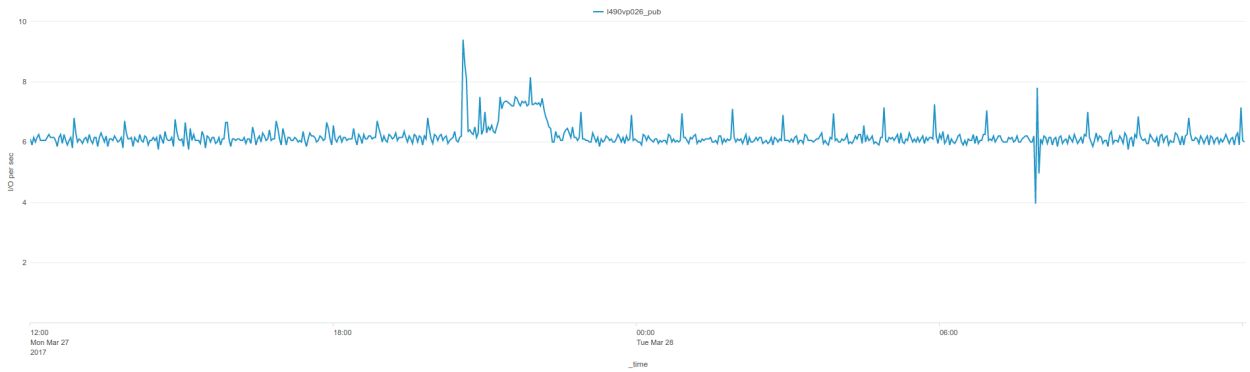
lpar usage over 24 hours:



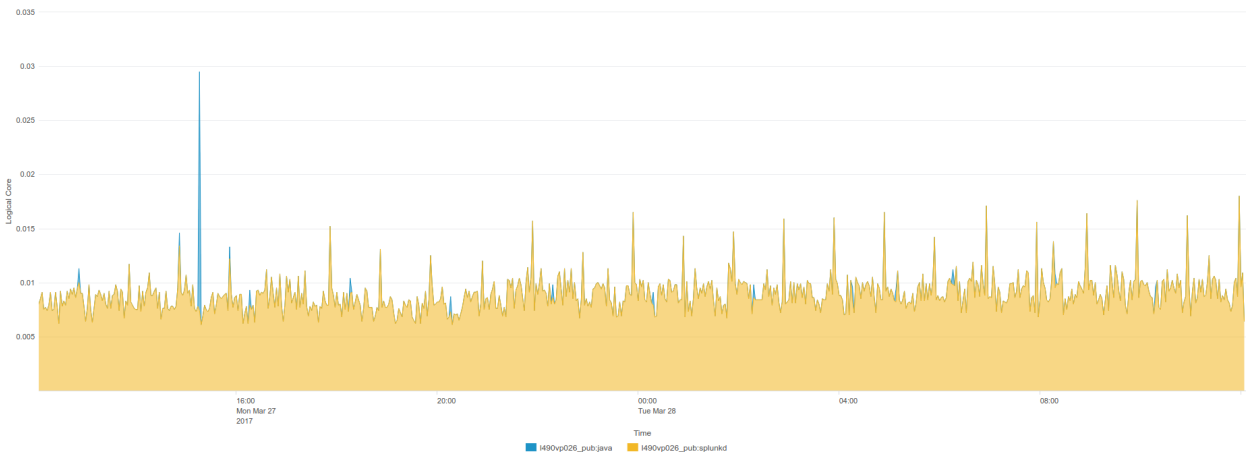
lpar usage_zoom over 24 hours:



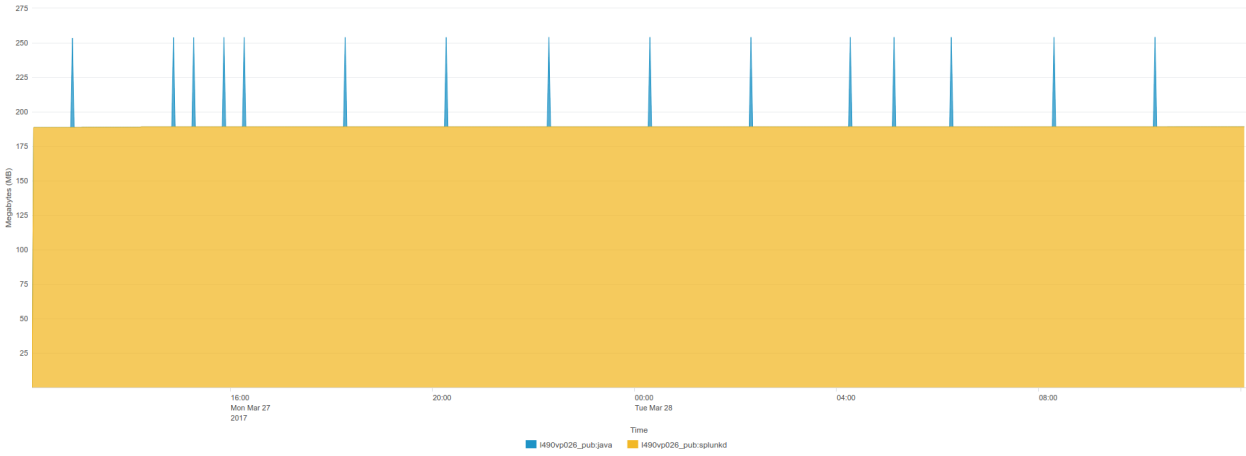
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



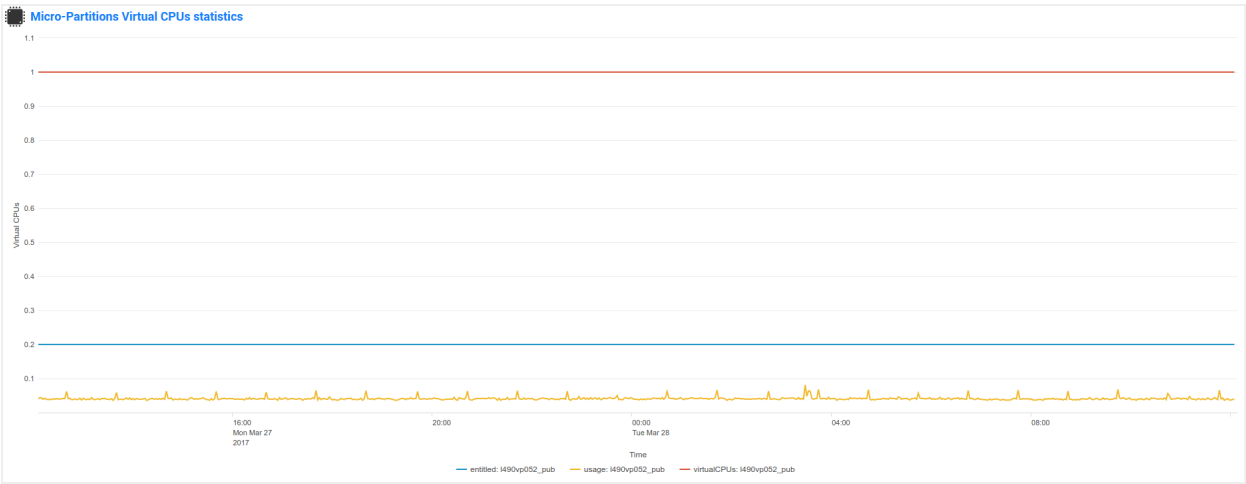
TOP processes memory usage over 24 hours:



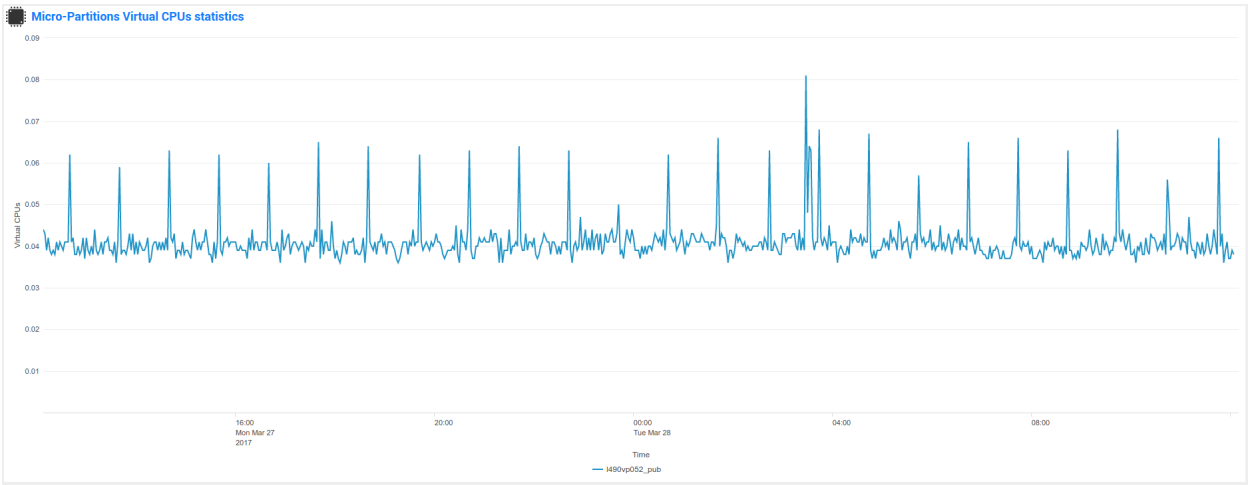
IBM AIX 7.2 ON POWER8 / Entitled 0.2 / VirtualCPUs 1:

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2

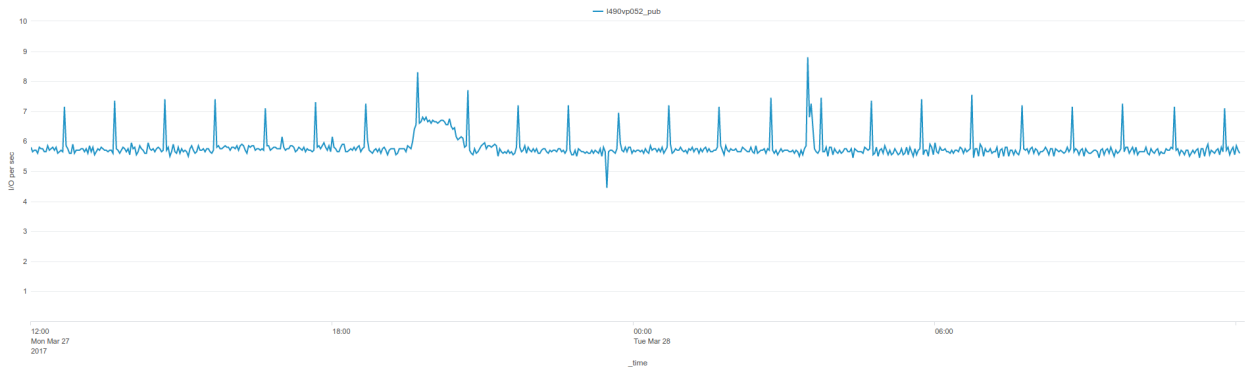
lpar usage over 24 hours:



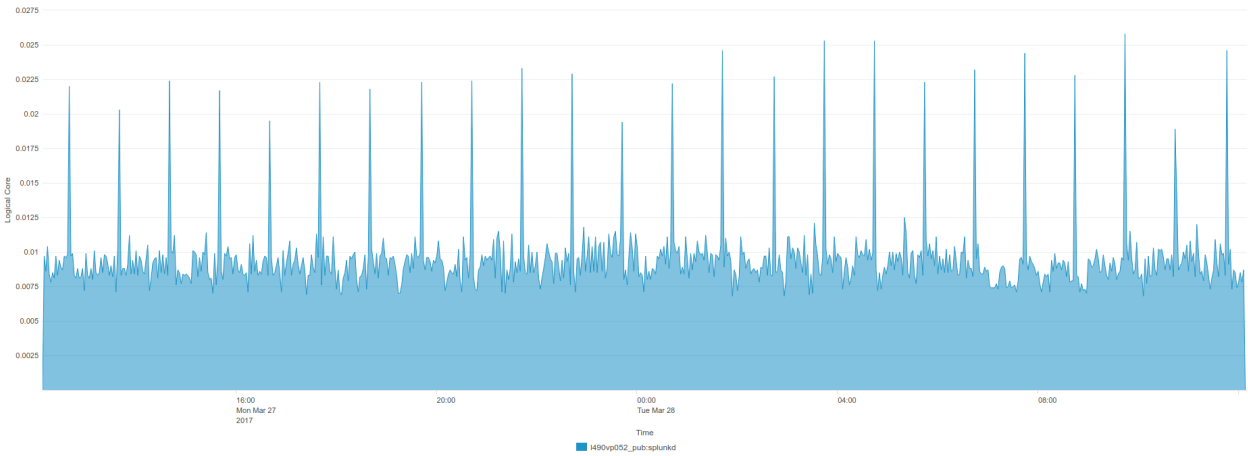
lpar usage_zoom over 24 hours:



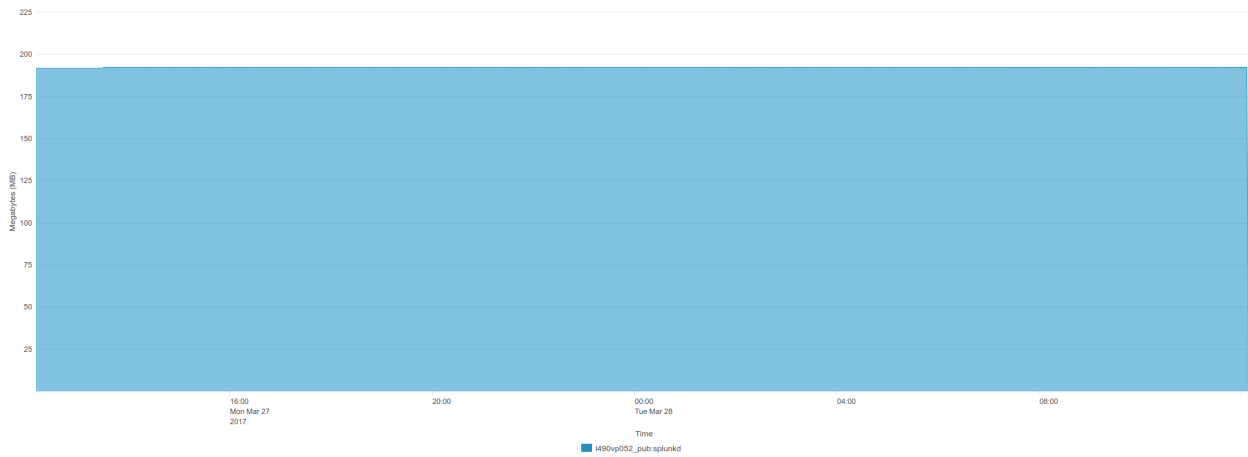
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



TOP processes memory usage over 24 hours:



LINUX BENCHMARKS:

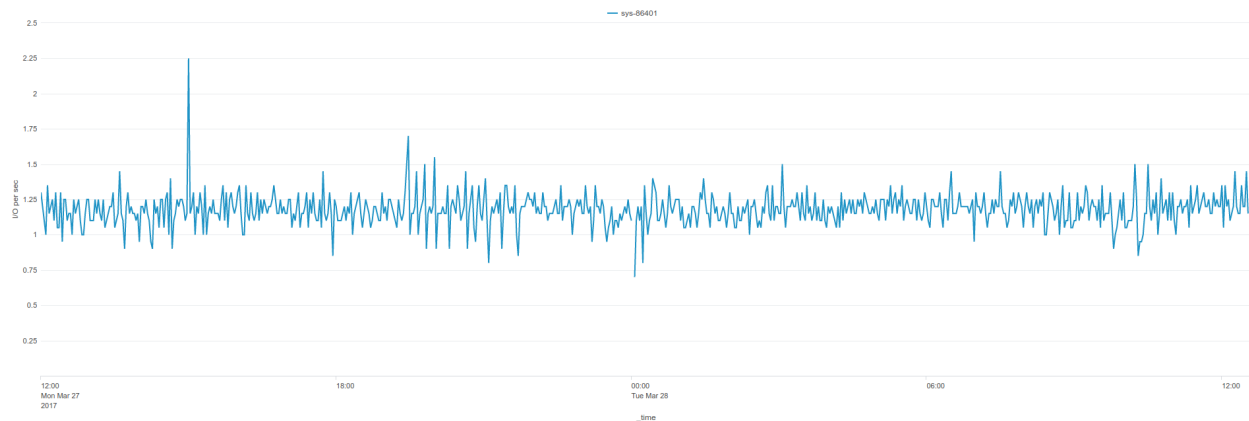
SUSE Linux 11.4 BE (IBM POWER 8)

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Perl interpreter, 1 CPU

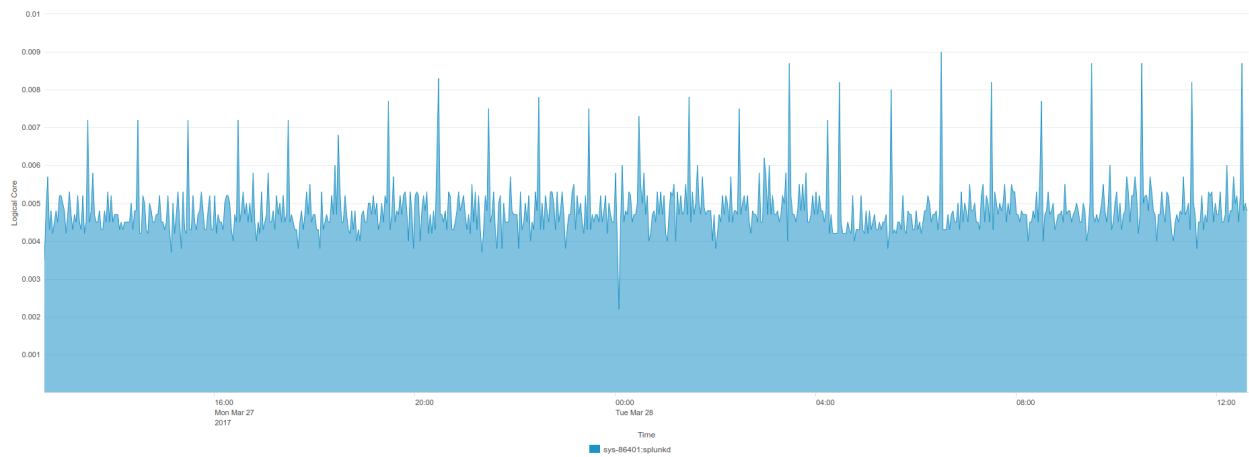
CPU percentage usage over 24 hours:



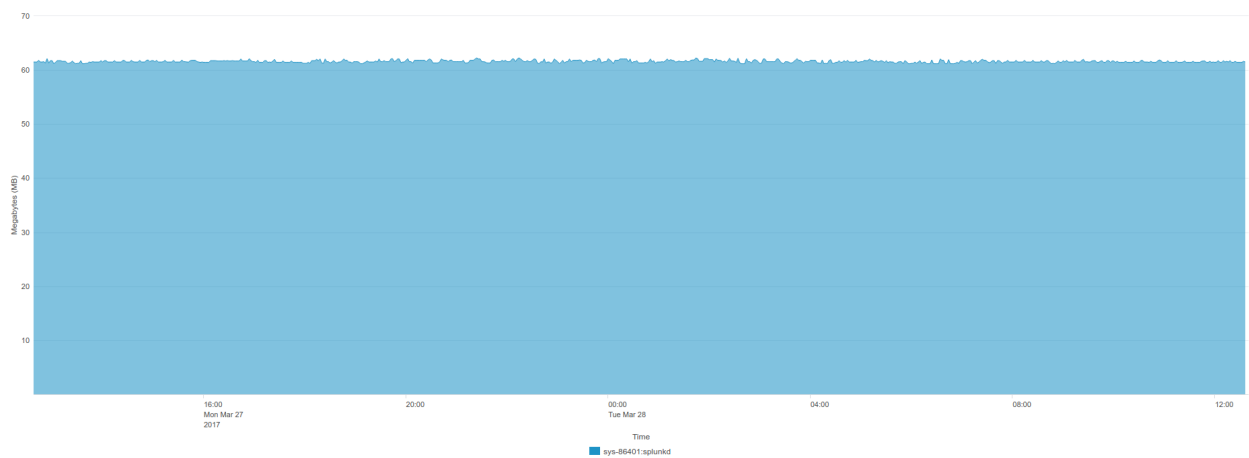
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



TOP processes memory usage over 24 hours:



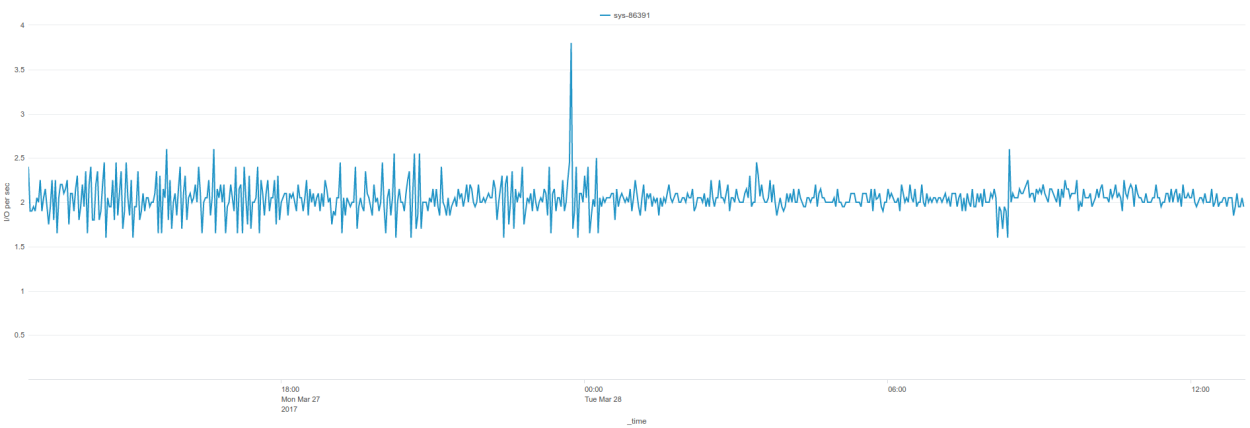
SUSE Linux 12.2 LE (IBM POWER 8)

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Python interpreter, 1 CPU

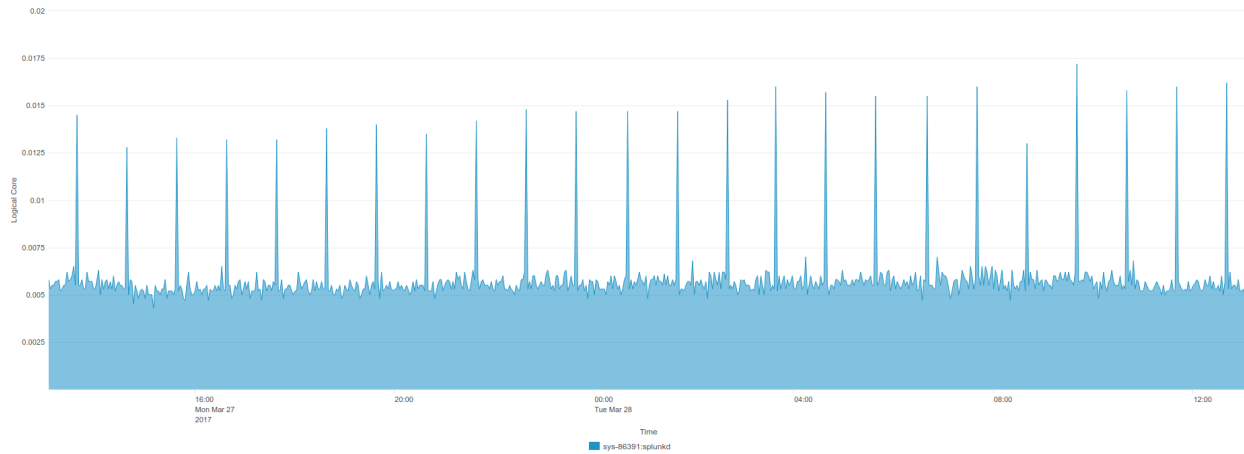
CPU percentage usage over 24 hours:



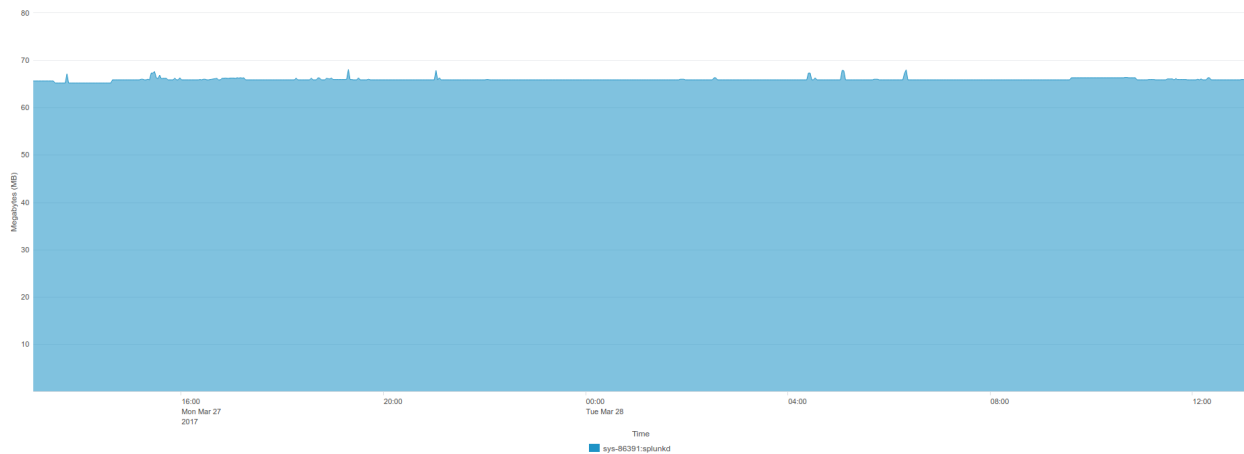
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:

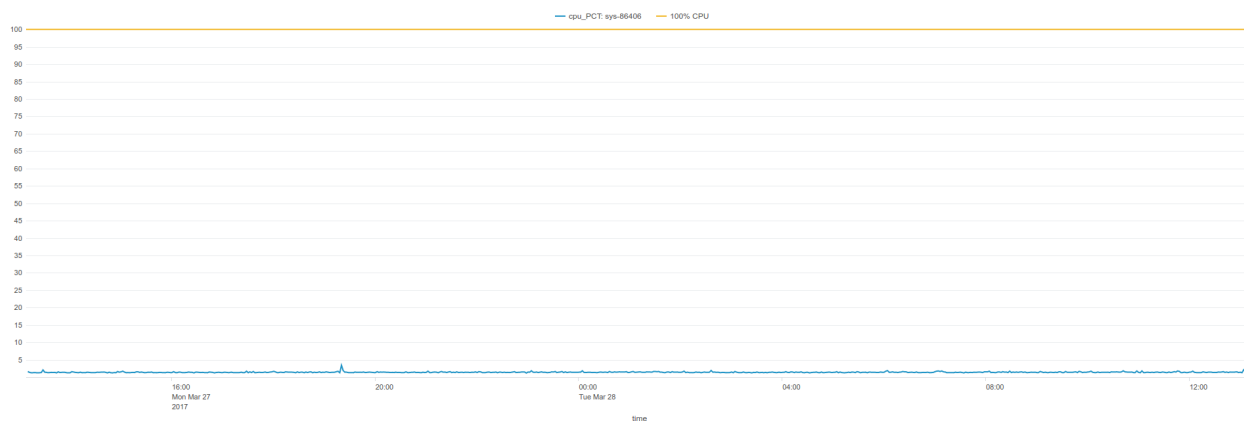


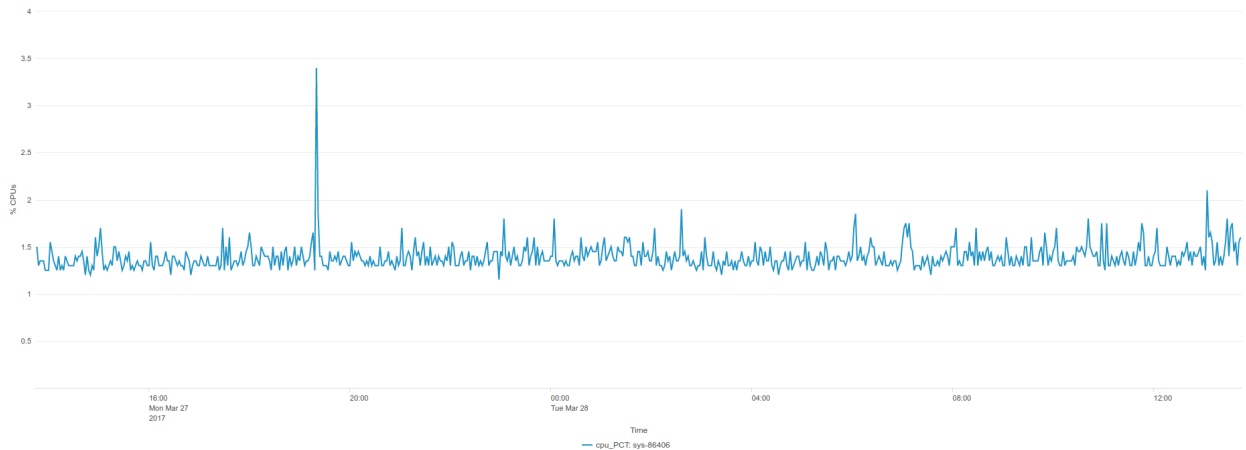
TOP processes memory usage over 24 hours:



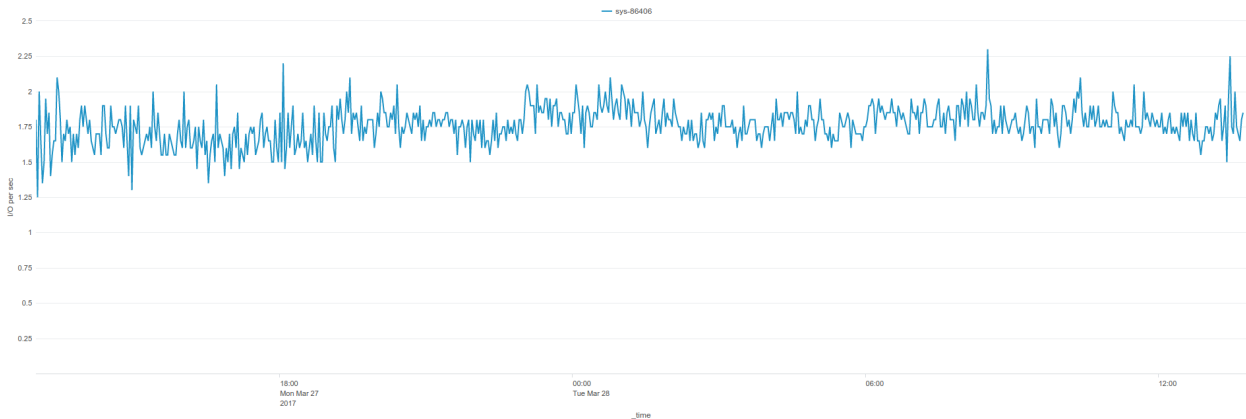
Red Hat Linux 6.9 BE (IBM POWER 8)

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Perl interpreter, 1 CPU

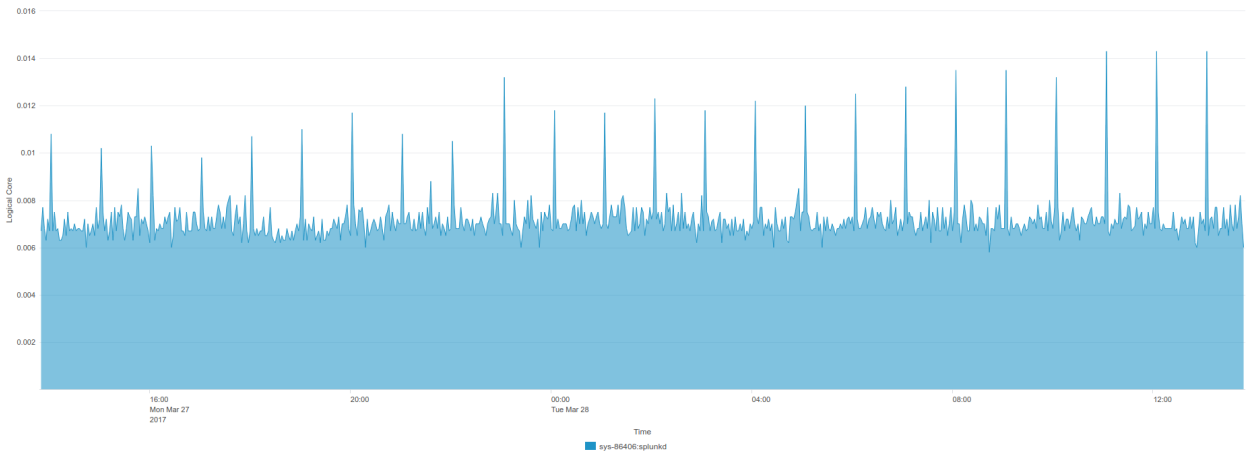




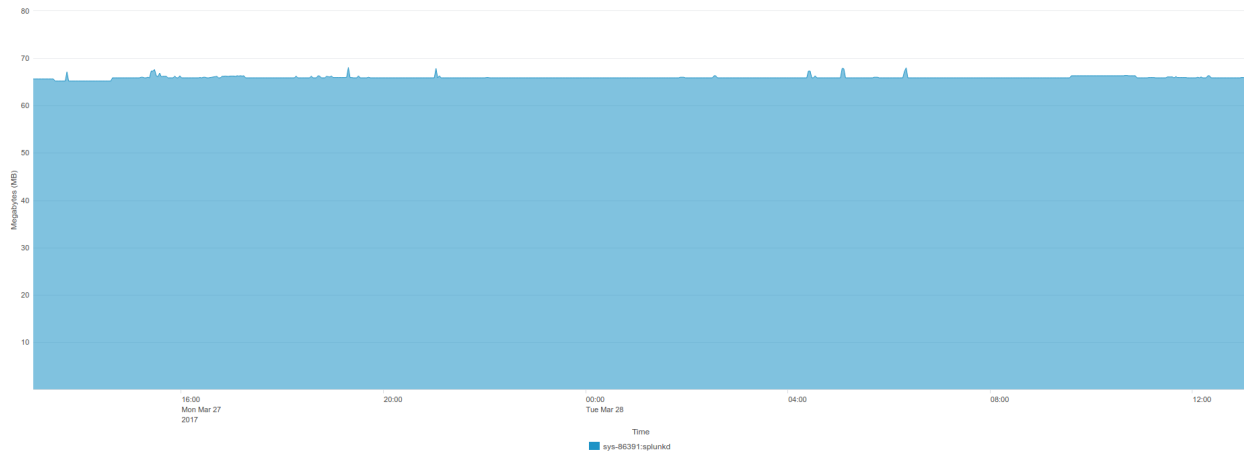
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:

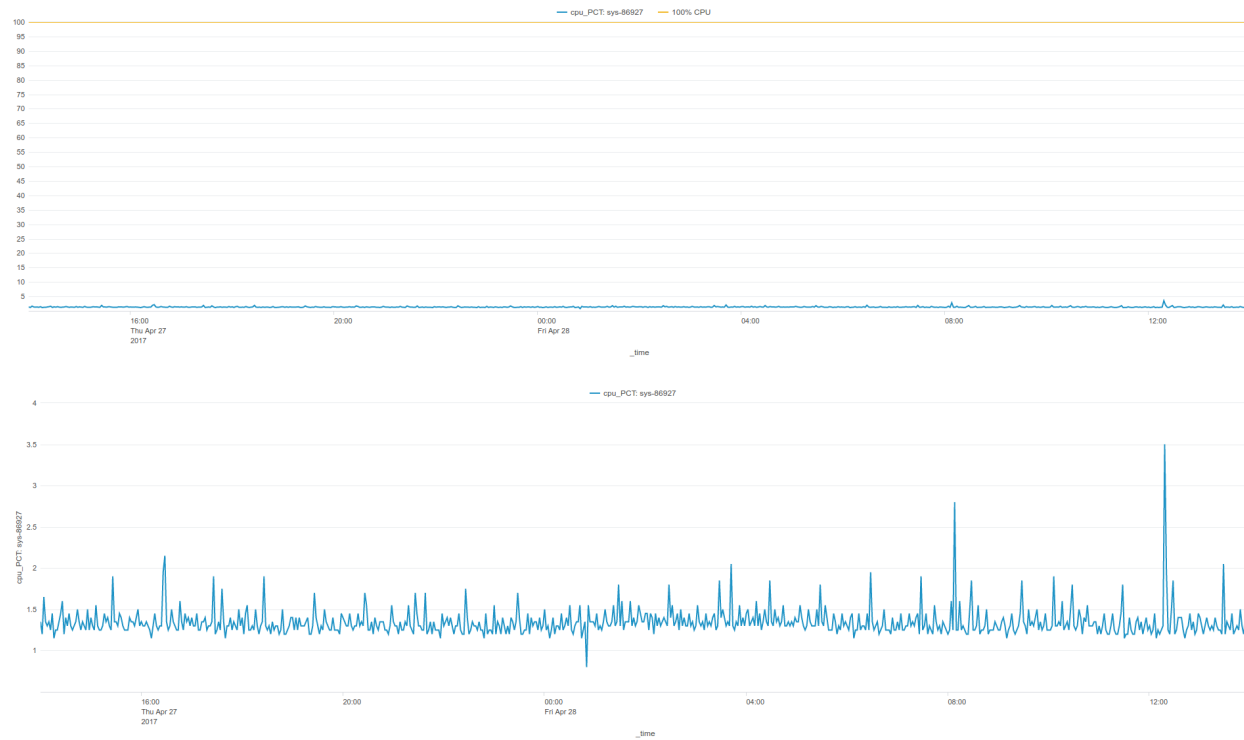


TOP processes memory usage over 24 hours:

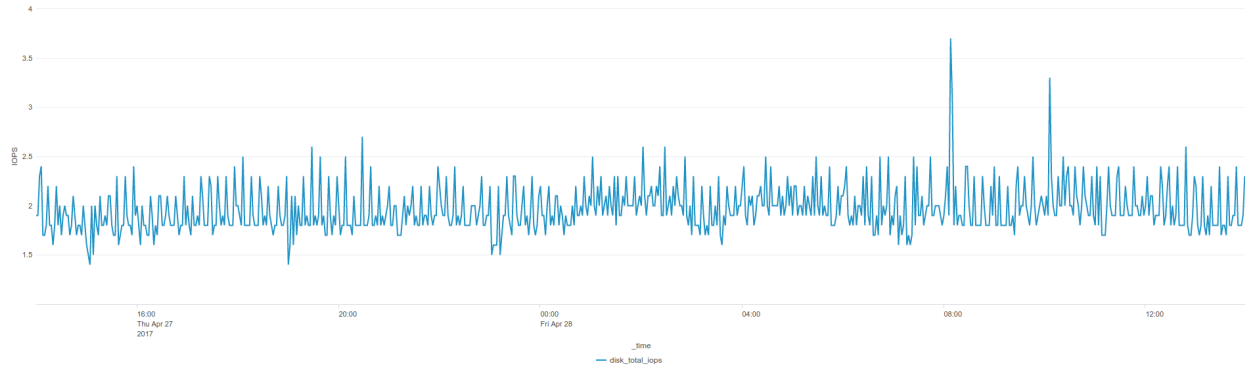


RedHat Linux 7.2 LE (IBM POWER 8)

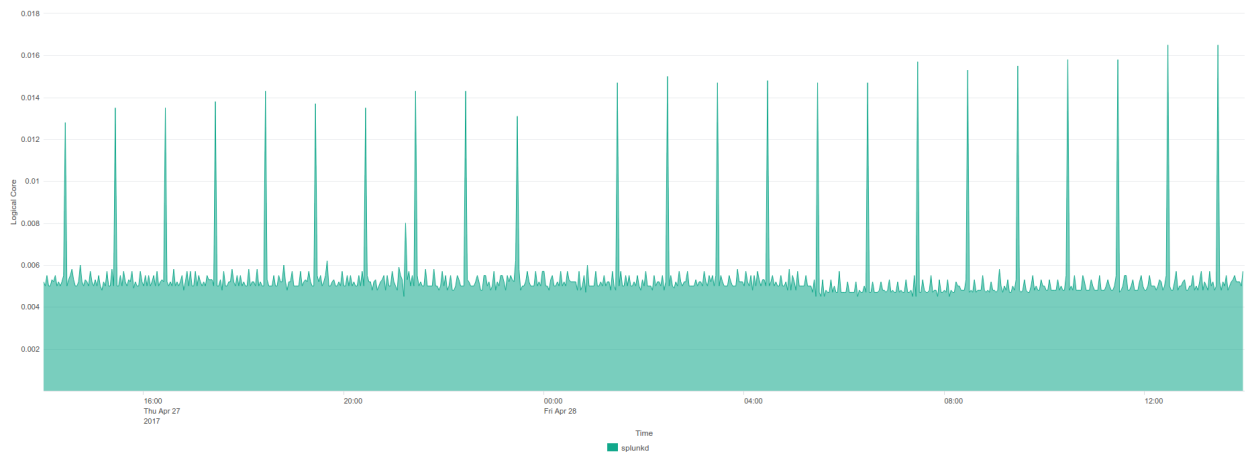
date 29/04/2013, TA-nmon release 1.3.15, Splunk Universal Forwarder 6.5.3, Python interpreter, 1 CPU



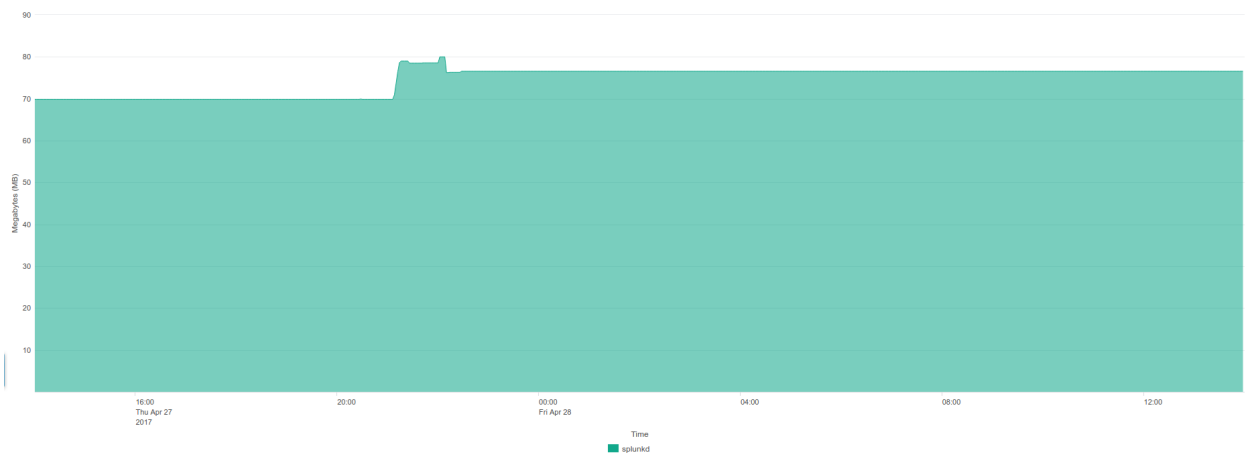
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



TOP processes memory usage over 24 hours:

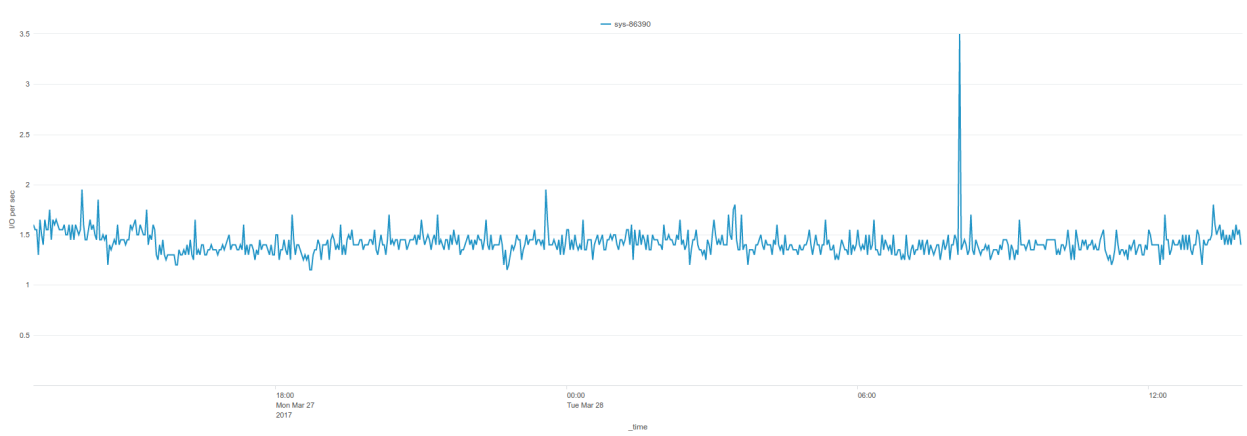


RedHat Linux 7.3 LE (IBM POWER 8)

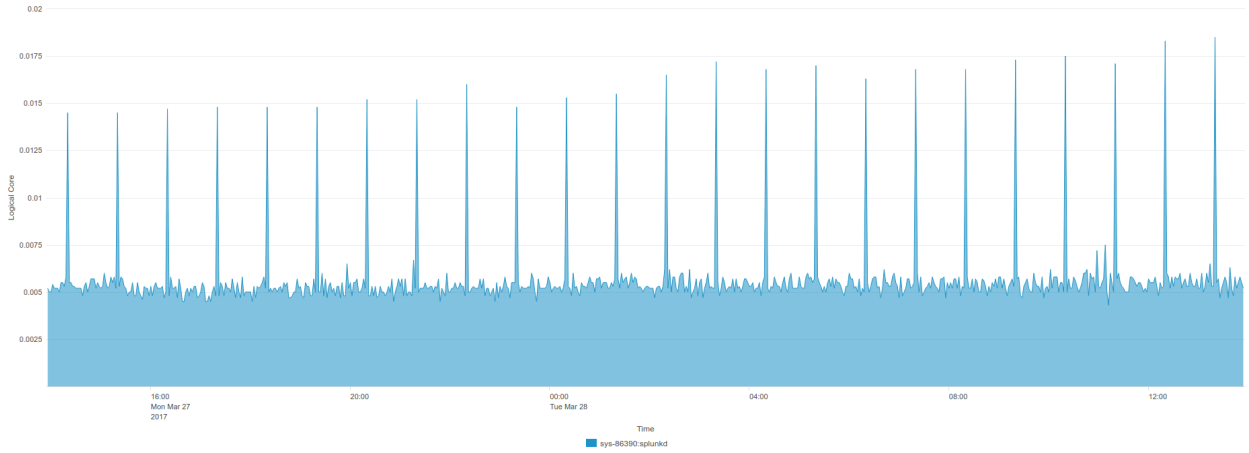
date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Python interpreter, 1 CPU



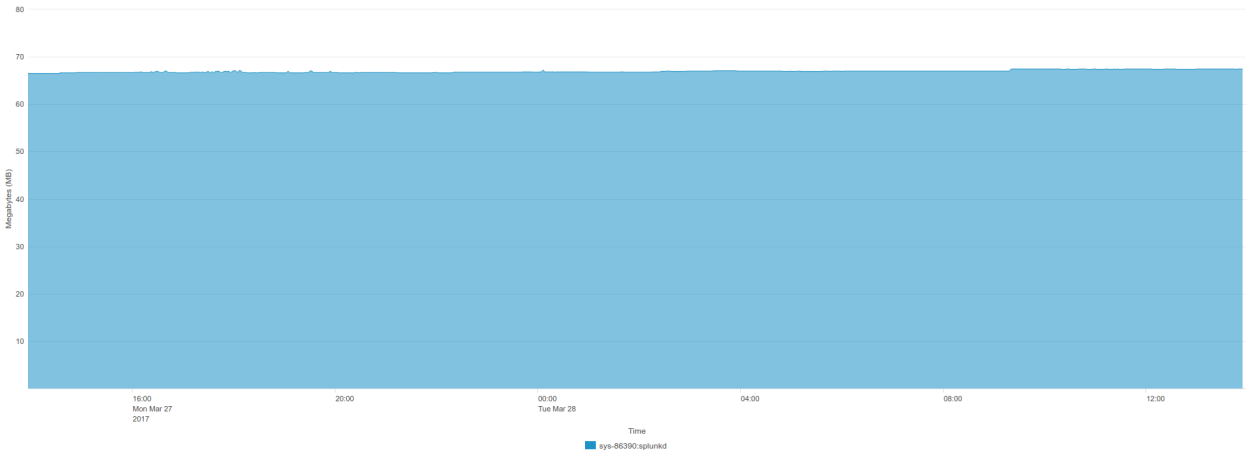
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:

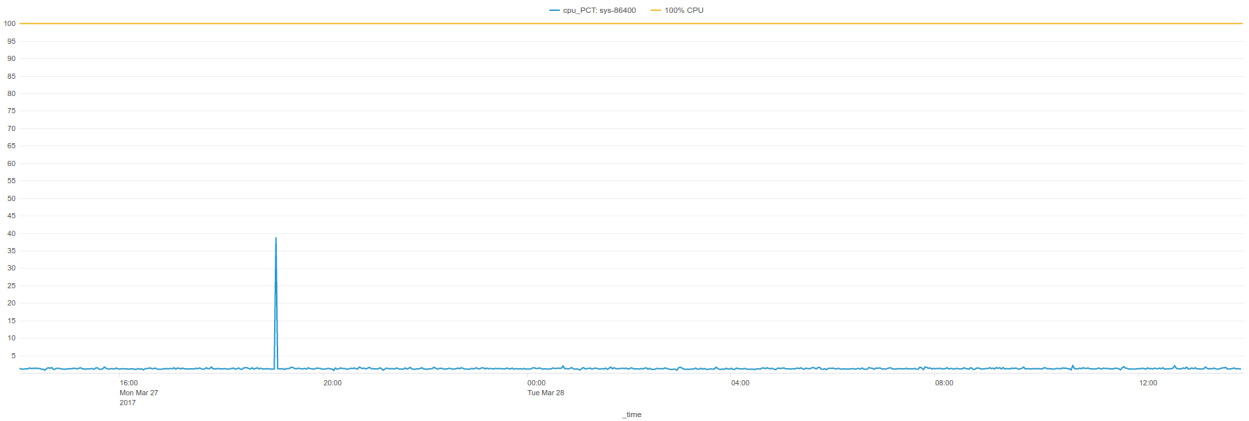


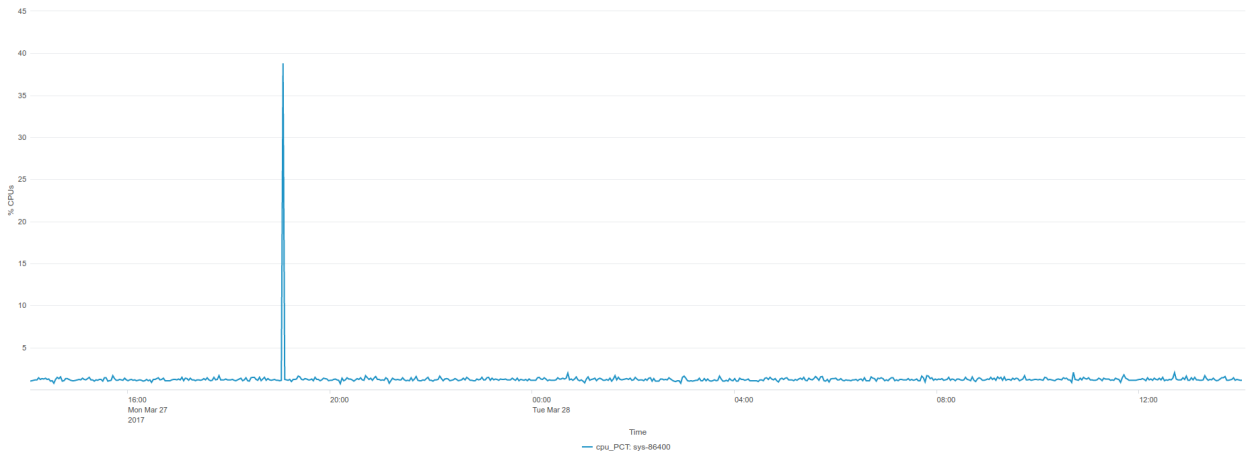
TOP processes memory usage over 24 hours:



Ubuntu 16.04 LTS (IBM POWER 8)

date 27/03/2017, TA-nmon release 1.3.05, Splunk Universal Forwarder 6.5.2, Python interpreter, 1 CPU

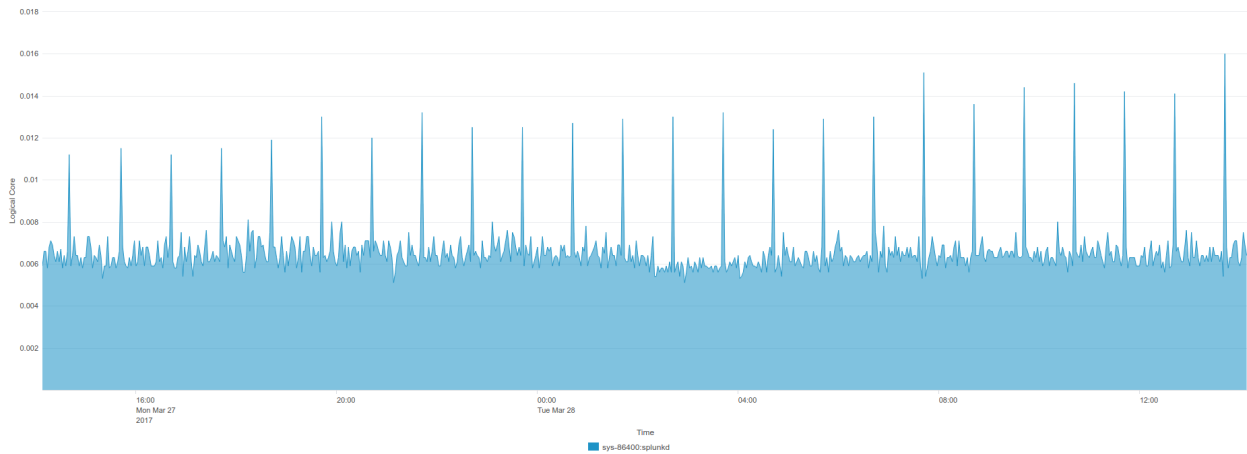




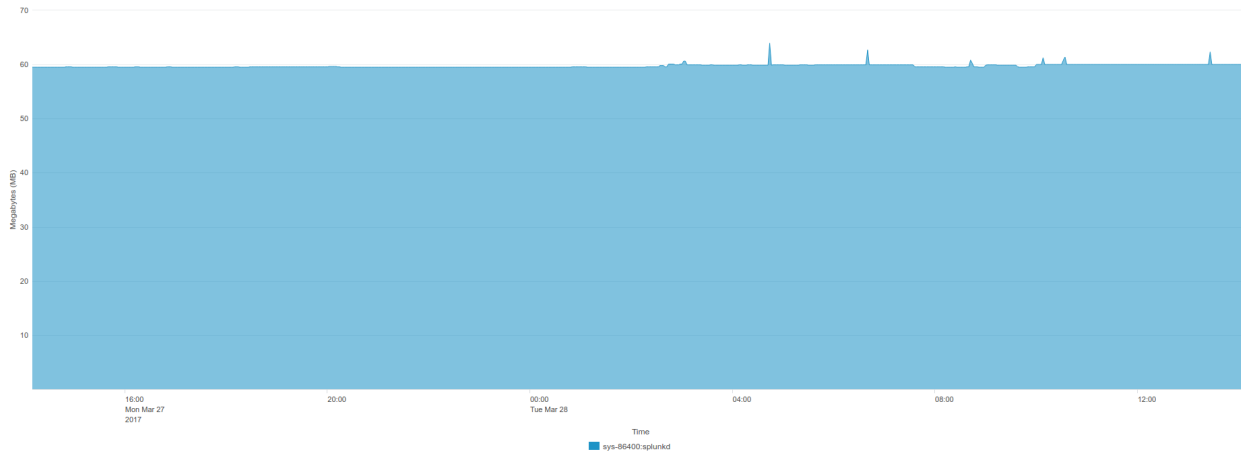
Average I/O over 24 hours:



TOP processes CPU core usage over 24 hours:



TOP processes memory usage over 24 hours:



1.9 Scripts and Binaries

1.9.1 Embedded Scripts in the TA-nmon-hec

nmon_helper:

- bin/nmon_helper.sh:

This shell script is being used by the application to launch Nmon binaries whenever it is detected as required.

It is as well responsible in launching the fifo_reader scripts. (introduced in version 1.3.x)

fifo_reader:

- bin/fifo_reader.pl
- bin/fifo_reader.py
- bin/fifo_reader.sh

These scripts are continuously running as back ground processes on server running the technical addons. Their purpose is to read the fifo files (named pipe) nmon processes are writing to, and extract the different typologies of data from them nmon data

fifo_consumer:

- bin/fifo_consumer.sh

This script is scheduled by default to run every 60 seconds. Its purpose to recompose the nmon flow of data to be parsed by the nmon parser scripts. (see bellow)

nmon_parser:

- bin/nmon2kv.sh | bin/nmon2kv.py | bin/nmon2kv.pl:

Shell / Python / Perl scripts used to manage and process Nmon raw data into key value data streamed to the Splunk HEC endpoint.

The Shell script is a wrapper script to Python / Perl scripts. (decision is made on local interpreter availability with Python as the default choice)

hec_wrapper:

- bin/hec_wrapper.sh

This Shell script is used to read the output of other wrappers scripts and stream it to the Splunk HEC endpoint.

Other wrappers:

- bin/wrapper_nmon_helper.sh
- bin/wrapper_fifo_consumer.sh
- bin/wrapper_nmon_cleaner.sh

These Shell scripts are very simple wrappers that call the main script and output its content to the `hec_wrapper.sh` for data streaming. They are being called by the Splunk internal scheduler.

nmon_cleaner:

- bin/nmon_cleaner.sh / bin/nmon_cleaner.py / nmon_cleaner.pl

Shell / Python / Perl scripts used to manage retention and purge of old nmon data.

Alternatively, it will also ensure that no outdated csv data is being left by Splunk in Performance and Configuration repositories

The Shell script is a wrapper script to Python / Perl scripts. (decision is made on local interpreter availability with Python as the default choice)

1.9.2 Embedded Binaries in the TA-nmon-hec

The TA-nmon-hec embeds Nmon binaries for Linux vendors and Solaris OS. AIX embeds by default its own version of Nmon, known as “topas-nmon”.

For Linux OS:

- bin/linux: Main directory for Linux specific Nmon binaries
- bin/linux/amzn: 64 bits binaries for Amazon Linux (AMI)
- bin/linux/centos: 32/64 bits binaries for Centos
- bin/linux/debian: 32/64 bits binaries for Debian GNU/Linux
- bin/linux/fedora: 32/64 bits binaries for Fedora project
- bin/linux/generic: 32/64/ia64/power/mainframe binaries compiled for generic Linux
- bin/linux/mint: 32/64 bits binaries for Linux Mint
- bin/linux/opensuse: 32/64 bits binaries for Linux Opensuse
- bin/linux/ol: 32/64 bits binaries for Oracle Linux
- bin/linux/rhel: 32/64/ia64/mainframe/power binaries for Redhat Enterprise Server
- bin/linux/sles: 32/64/ia64/mainframe/power binaries for Suse Linux Enterprise Server
- bin/linux/ubuntu: 32/64/power/arm binaries for Ubuntu Linux
- bin/linux/arch: 32/64 bits binaries for Archlinux
- bin/raspbian: arms binaries for Raspbian Linux

Most of these binaries comes from the official Nmon Linux project site. On x86 processor and for Centos / Debian / Ubuntu / Oracle Linux these binaries are being compiled by myself using Vagrant and Ansible automation. (See <https://github.com/guilhemmarchand/nmon-binaries>)

Associated scripts resource (`nmon_helper.sh`) will try to use the better version of Nmon available, it will fall back to generic or system embedded if none of specially compiled versions can fit the system.

For Solaris OS:

sarmon binaries for Oracle Solaris x86 and Sparc:

- bin/sarmon_bin_i386: sarmon binaries for Solaris running on x86 arch
- bin/sarmon_bin_sparc: sarmon binaris for Solaris running on sparc arch

sarmon binaries comes from the official sarmon site project.

For AIX:

Nmon is shipped within AIX by default with topas-nmon binary.

1.10 Pre-requisites

1.10.1 Splunk requirements

Compatibility matrix:

Metricator for Nmon stack	Major version branch
Splunk Universal Forwarder 6.x, 7.x	Version 1.x

1.10.2 Technical Add-on requirements

Operating system

The Technical Add-on is compatible with:

- Linux OS X86 in 32/64 bits, PowerPC (PowerLinux), s390x (ZLinux), ARM
- IBM AIX 7.1 and 7.2
- Oracle Solaris 11

Third party software and libraries

To operate as expected, the Technical Add-ons requires a Python **OR** a Perl environment available on the server:

Python environment: used in priority

Requirement	Version
Python interpreter	2.7.x

Perl environment: used only in fallback

Requirement	Version
Perl interpreter	5.x
Time::HiRes module	any

In addition, the Technical Addon requires:

Requirement	Version
curl	Any

Notes:

- IBM AIX does not generally contain Python. Nevertheless, Perl is available as a standard. More, Time::HiRes is part of Perl core modules.
- Modern Linux distribution generally have Python version 2.7.x available and do not require any further action.
- Linux distributions lacking Python will fallback to Perl and must satisfy the Perl modules requirements.
- If running on a full Splunk instance (any Splunk dedicated machine running Splunk Enterprise), the Technical Add-on uses Splunk built-in Python interpreter.

1.11 Deployment Matrix

1.11.1 What goes where ?

- The TA-nmon-hec is available for download as an independent application in Splunk base: <https://splunkbase.splunk.com/app/>
- The TA-nmon-hec is also available for download in its Git repository: <https://github.com/guilhemmarchand/TA-nmon-hec>

Standalone deployment: A single Splunk instance does all

Splunk Instance (role)	TA-nmon-hec
Standalone	X (optional)

The TA-nmon-hec provides nmon performance and configuration collection for the host than runs the add-on, which is optional

Distributed deployment:

Splunk Instance (role)	TA-nmon-hec
Search head (single instance or clustered)	X (optional)
Indexer (single instance or clustered)	
Master node	X (optional)
Deployment servers	X (optional)
Heavy Forwarder	X
Universal Forwarder	X

The TA-nmon-hec provides nmon performance and configuration collection for the host than runs the add-on, which is optional

2.1 Processing workflow in action

2.1.1 Generating Nmon data

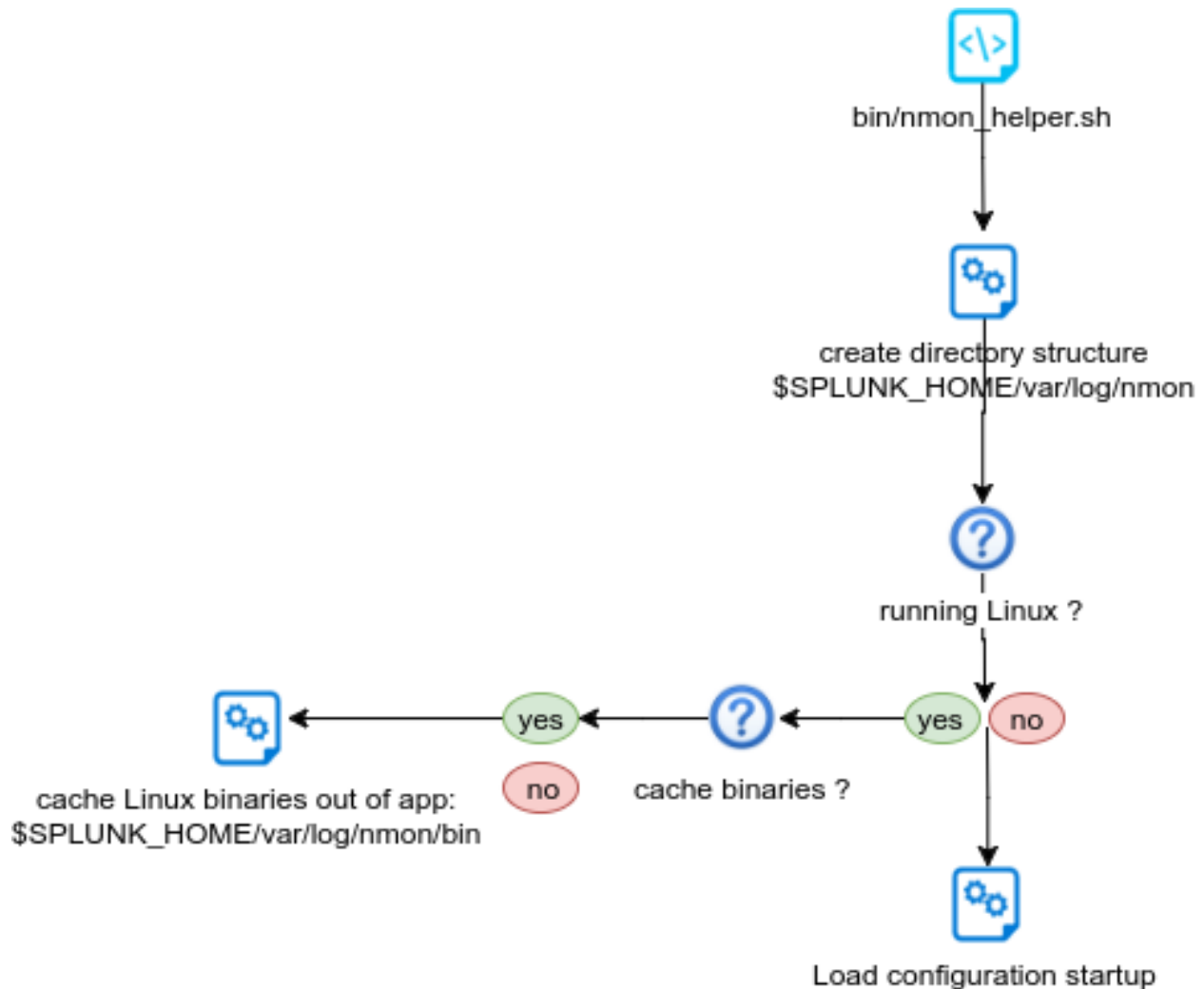
Generating the Nmon data which basically contains the performance measures and the configuration data is called “nmon_collect”.

The activity of the scripts involves in these operations is being logged by Splunk into:

- sourcetype=nmon_collect: for standard output
- index=_internal sourcetype=splunkd: for stderr (unexpected errors)

Most of these operations are done by the main script: bin/nmon_helper.sh:

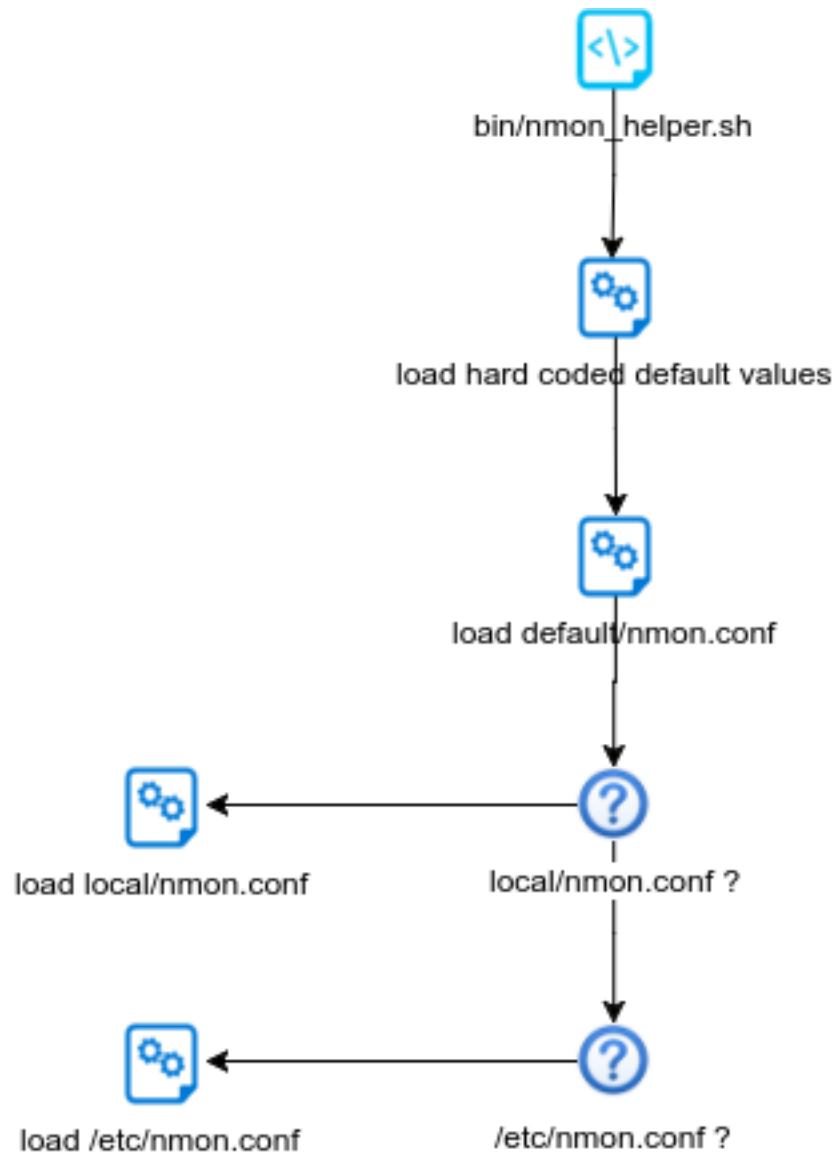
nmon_helper.sh tasks part1: initial startup tasks



Steps are:

- basics startup tasks: load `SPLUNK_HOME` variable, identify the application directories
- directory structure: load and verify directory structure, create if required in `$SPLUNK_HOME/var/log/nmon`
- binaries caching: for Linux OS only, verify if cache is existing and up to date, unpack `linux.tgz` to `$SPLUNK_HOME/var/log/nmon` if required
- start loading default and custom configurations

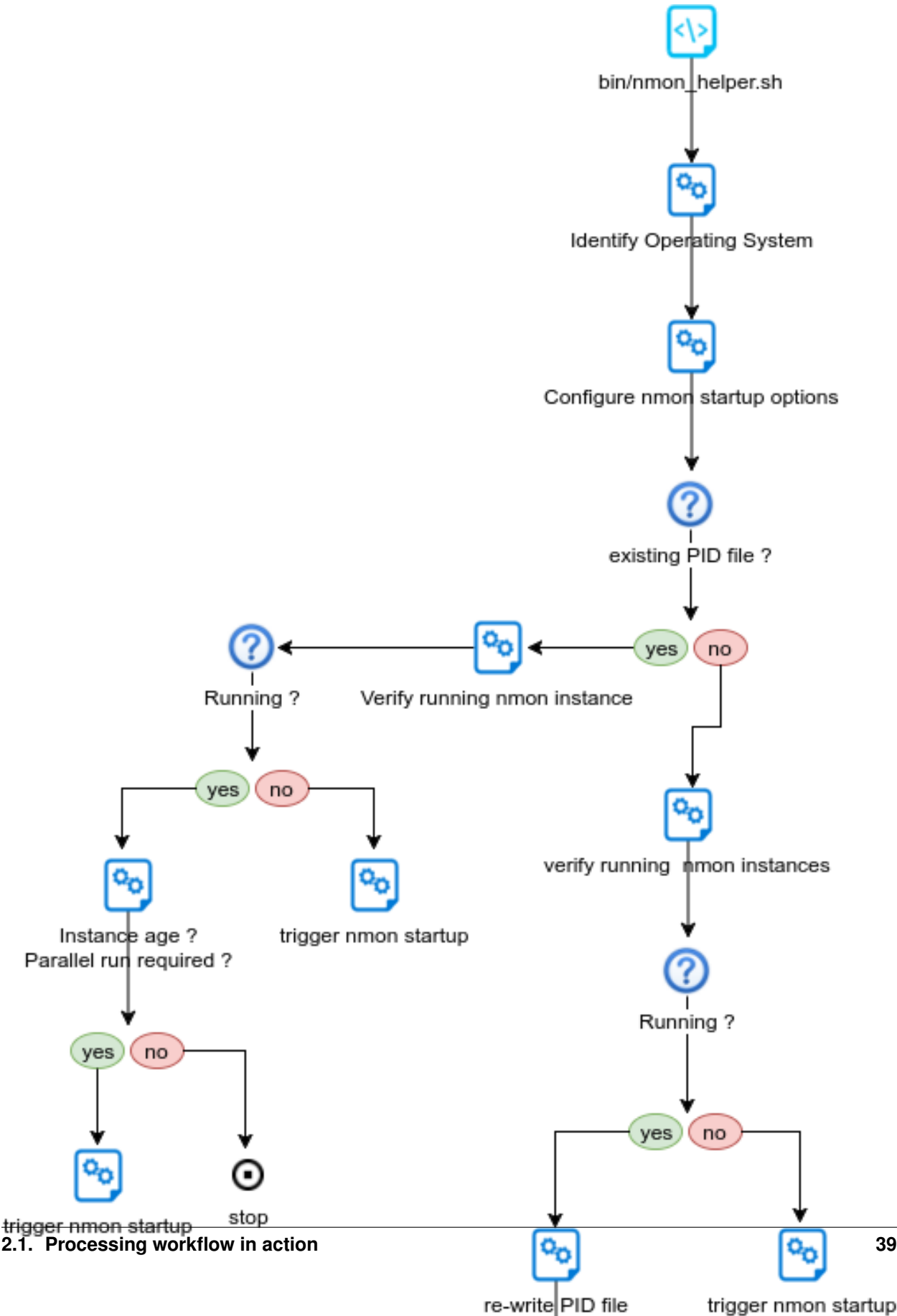
nmon_helper.sh tasks part2: load values and options



Steps are:

- load hard coded values for nmon options, such as interval and snapshot values
- load default/nmon.conf values (source default/nmon.conf)
- check if local/nmon.conf exist, and source file (override any previously defined values)
- check if /etc/nmon.conf exist (provide override mechanism at local server level), and source file (override any previously defined values)

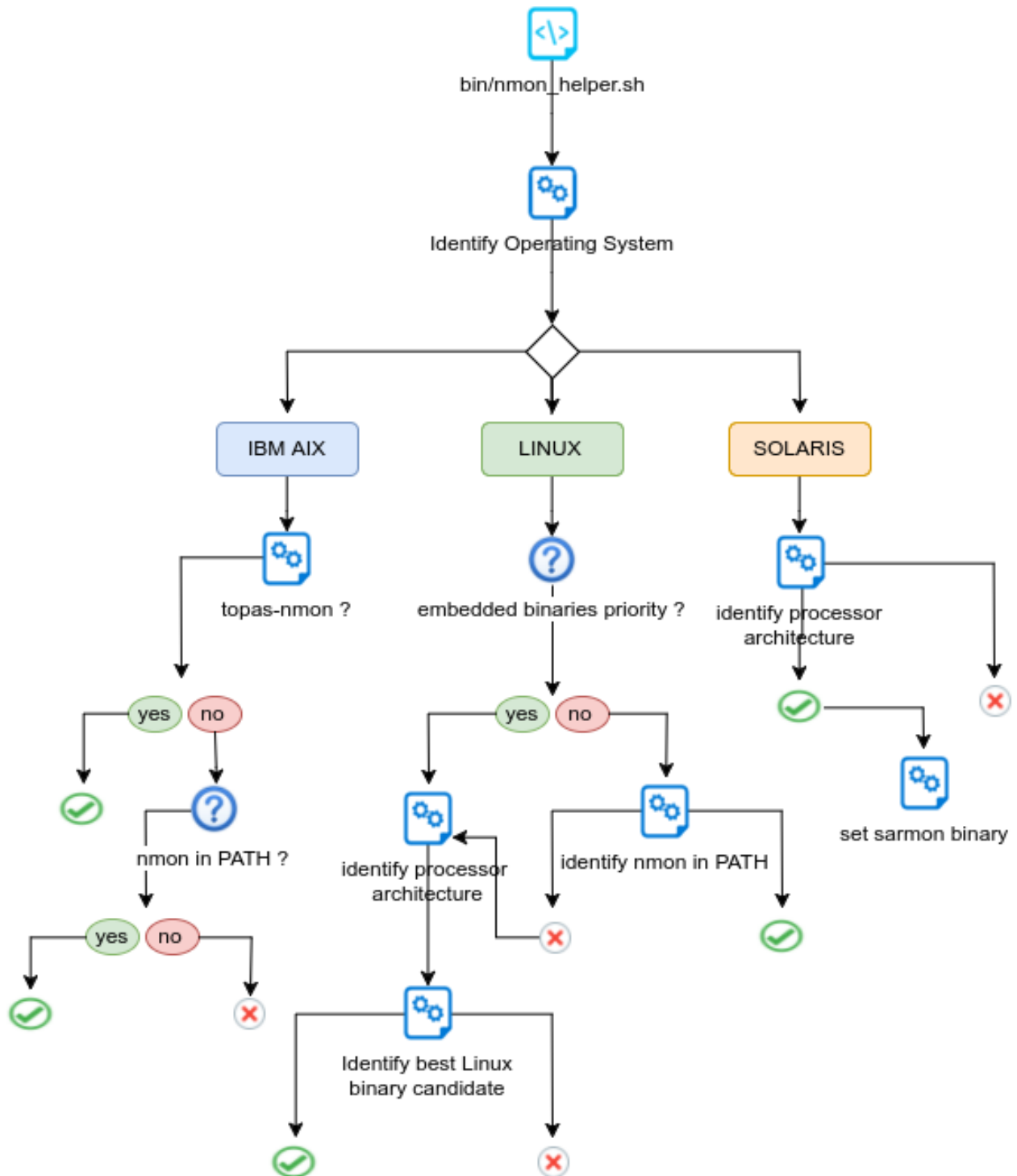
nmon_helper.sh tasks part3: identify instances



Steps are:

- Identify Operating System
- Verify PID file existence
- Is PID file valid ?
- If PID file exists, is the nmon instance running ?
- If PID not found, is there an nmon instance running ?
- rewrite PID file if nmon is running
- trigger nmon startup in other cases

nmon_helper.sh tasks part4: identify binaries



Steps are:

- Identify Operating System
- Identify Processor architecture for Linux / Solaris

- Identify local nmon binary for AIX
- For Linux, verify embedded binaries priority
- For Linux, identify best binary candidate and set nmon binary
- For Solaris, set sarmon binary

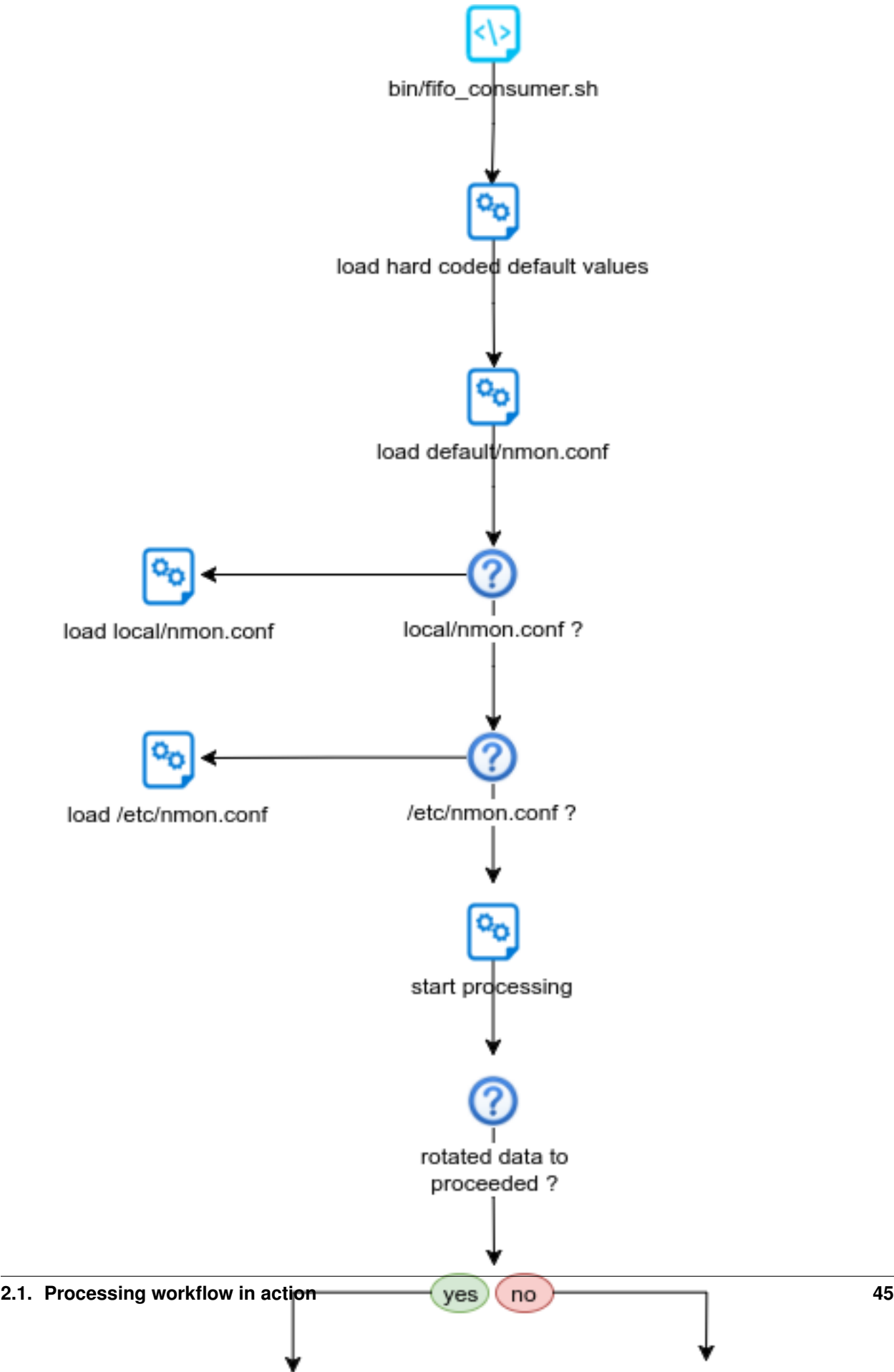
- set the nmon destination (fifo1 / fifo2)
- start nmon binary
- Identify interpreter to be used
- start the fifo_reader

2.1.2 Consuming Nmon data

Consuming the nmon data generated by the `nmon_helper.sh` and associated scripts (`fifo_reader`) is in first step operated by the “`bin/fifo_consumer.sh`” script.

This is a very simple shell script that will recompose the nmon data in the correct order, and stream its content to nmon parsers. (`nmon2kv`)

This script does as well the files rotation, such that next cycle starts such that the `nmon_data.dat` file is empty after consumption, which guarantees a low level of CPU and resources usage over each iteration.



Steps are:

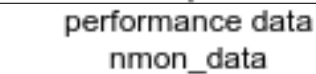
- Initialization and configuration loading
- Verify if fifo1 and fifo2 have rotated data to be proceeded
- Verify if fifo1 and fifo2 have non empty nmon_data.dat
- If so, wait at least 5 seconds age file before processing the data
- Stream nmon data to nmon2kv.sh

2.1.3 Parsing Nmon data

This is the final step in the nmon processing tasks, the nmon2kv parsers will consume the data receive in standard input (stdin), and stream to the Splunk HEC endpoint over http/https.

There 3 scripts involved in these tasks:

- bin/nmon2kv.sh: simple shell wrapper that will decide to use Python or Perl parser
- bin/nmon2kv.py: the Python version parser
- bin/nmon2kv.pl: the Perl version parser



Steps are:

- nmon2kv.sh reads data from standard input
- nmon2kv.sh decides to use Python if version 2.7.x.available, or fall back to Perl
- nmon2kv.pyl.pl parse the data, generates configuration and performance data, as well as internal flag files
- If the HEC endpoint has been configured (URL and token defined in local/nmon.conf), stream the data to the HEC endpoint
- If the option “--no_log_local” is used, the parser does not write to local log files on the machine. Otherwise, write the data to local logs in \$SPLUNK_HOME/var/log/nmon

Deployment and configuration:

3.1 Deployment

However, Splunk is an highly distributable solution and some good practices have to be respected, please consult the Nmon Performance core documentation:

- Standalone deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_standalone.html
- Distributed deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_distributed.html
- Splunk Cloud deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_splunkcloud.html

A Splunk HEC has several advantages:

- **server easy set up:** Splunk http input is easy to configure and implement
- **secure:** Splunk http traffic can easily be encrypted via SSL and integrated into any DMZ or similar restricted networking layer
- **resilient and scalable:** using load balancers and multiple nodes provides resiliency and horizontal scalability
- **network friendly:** as a Web service, it can be easily used across wide networks and over the Internet
- **easy management:** since the http input is managed on a token basis, you can easily configure different tokens to ingest the data into different indexes without any package modification or complexity

3.1.1 HEC performance considerations

For best HEC performance purposes, the nmon-logger works the following way:

- performance and configuration data are streamed in “batch” mode, which means we only generate one HEC connection for each during an occurrence of the nmon_processing (which occurs every minute)
- collection, processing and other data being generated by the nmon-logger work as well in batch mode, one connection per processing streams the full data
- most of Metadata are part of each event sent to the HEC

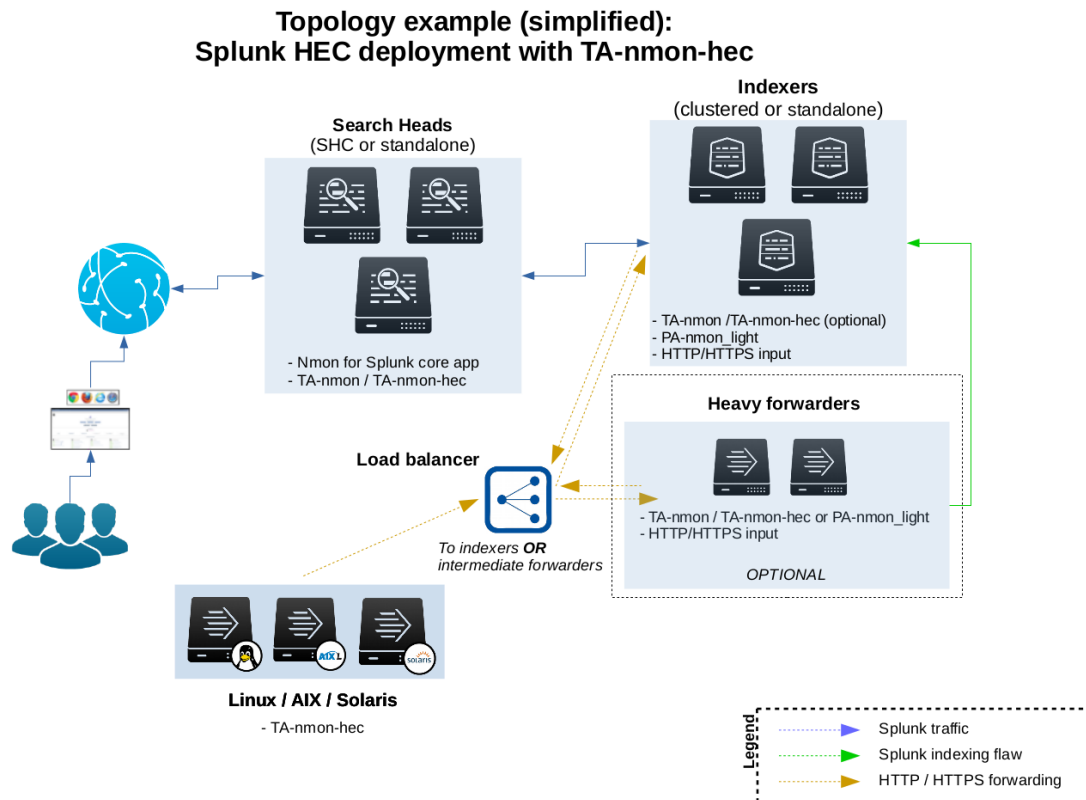
See: <http://dev.splunk.com/view/event-collector/SP-CAAEE73>

3.1.2 HEC resiliency and scalability

The TA-nmon-hec does not implement any kind of resilient mechanism to prevent against data loss during the transport to your HEC endpoint. (if your http endpoint is not available, the data will not be re-sent and will be lost)

Instead of that, we rely on the http/https HEC endpoint high resiliency you should implement when configuring your HEC implementation, which is quite straightforward.

In other words, ensure that you create a load balanced Web service (HA-proxy, nginx, F5 BigIp, etc...) such that you always have multiple http endpoints available, for resiliency and horizontal scalability.



3.1.3 Activate the Splunk http input and create a token

The Splunk configuration is really straightforward, it is all about:

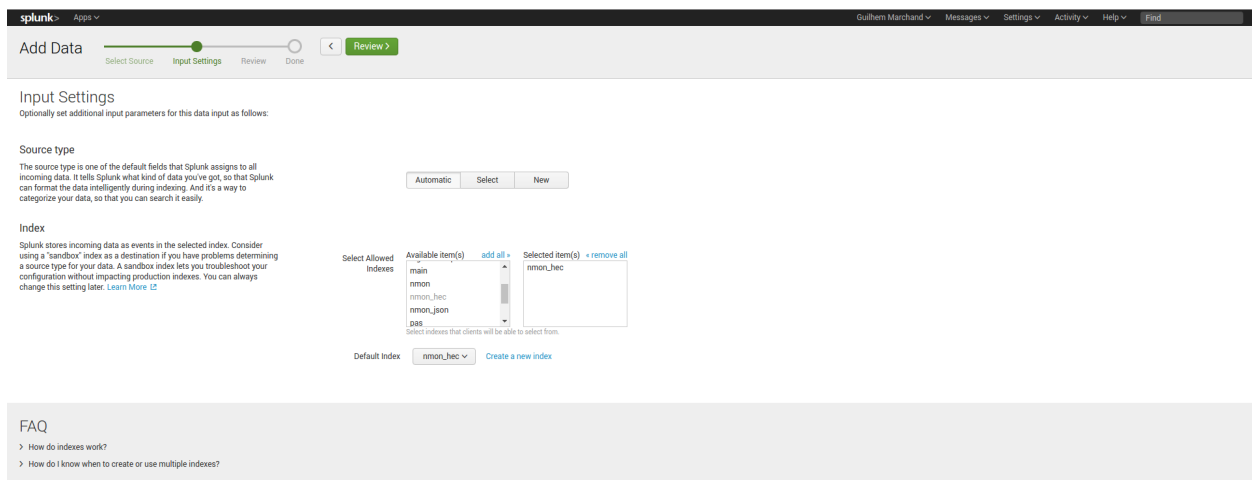
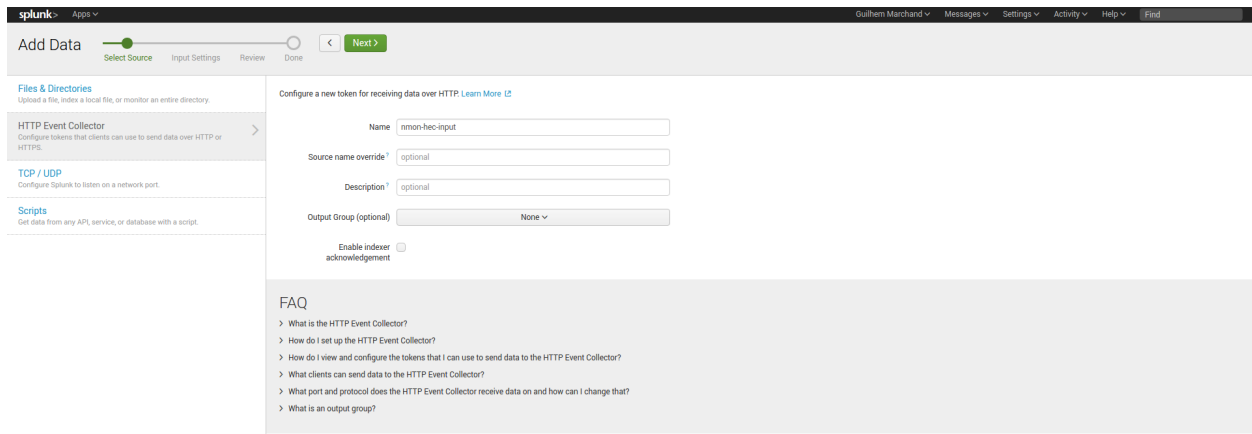
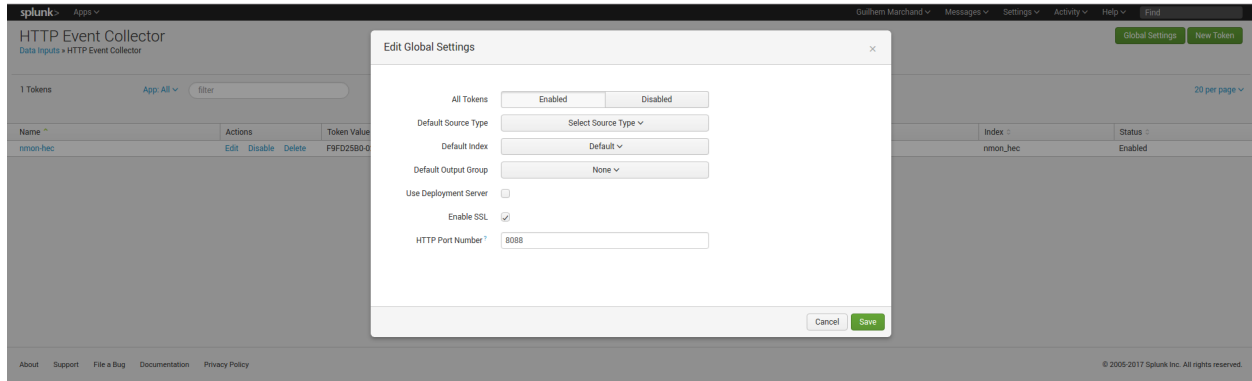
- Activating and the http input: configuring the http port, choosing between http and https
- Creating a token for the nmon data (1 token for all data, but you can create multiple tokens for different servers deployment)

Notes:

- http and https are supported
- indexer acknowledgment is not currently supported (configured per token)
- the nmon-logger will not explicitly specify an index, you choose the index to be used on a per token basis

- Any index name starting by “nmon” is natively taken in charge by the Nmon Performance application
- If you choose a different index name that does not match the rule above, you just need to customize the event-types.conf and macros.conf of the Nmon app
- it is not required to define any sourcetype / source by default

In a nutshell:



Configuration files:

- “\$SPLUNK_HOME/etc/apps/splunk_http_input/local/inputs.conf”:

```
[http]
disabled = 0
```

- “\$SPLUNK_HOME/etc/apps/<appname>/local/inputs.conf”:

Notes: replace <appname> with the application context where you want to store the configuration inputs.conf file

```
# inputs.conf

# Enable the HEC
[http]
disabled = 0
enableSSL = 1

# HEC endpoint for clients
[http://nmon-hec-input]
disabled = 0
index = nmon_hec
indexes = nmon_hec
token = CEE56643-BA2D-48EE-94EF-AD0909718B2A
```

3.1.4 Deploying the TA-nmon-hec

When the TA-nmon-hec has been deployed to Splunk, no data will be available in Splunk as long as you don’t configure the HEC parameters.

The TA-nmon-hec expects your configuration in:

```
$SPLUNK_HOME/etc/apps/TA-nmon-hec/local/nmon.conf
```

Create a local/nmon.conf and insert your URL / Token:

Example:

```
# HEC server configuration

nmon2csv_options="--mode fifo --silent --no_local_log --splunk_http_url https://192.
↪168.33.100:8088/services/collector/event --splunk_http_token CEE56643-BA2D-48EE-
↪94EF-AD0909718B2A"
```

To deploy the TA-nmon-hec using the Splunk deployment server, you will:

- extract the TA-nmon-hec archive in `$SPLUNK_HOME/etc/deployment-apps/` of your deployment server
- create the `$SPLUNK_HOME/etc/deployment-apps/TA-nmon-hec/local/nmon.conf` and insert your HEC parameters
- associate the TA-nmon-hec with the relevant service class and deploy the TA to your servers (Ensure to set “restart splunkd” !)

Once Splunk is restarted, the collection starts and if you have properly configured your HEC endpoint, the data starts to be available in Splunk.

Et voila!

3.2 Upgrade

Upgrading the TA-nmon-hec is nothing more than reproducing the initial installation steps, basically uncompressing the content of the TA-nmon.tgz archive.

Please refer to the installation documentations:

- Standalone deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_standalone.html
- Distributed deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_distributed.html
- Splunk Cloud deployment: http://nmon-for-splunk.readthedocs.io/en/latest/installation_splunkcloud.html

Additional information:

The TA-nmon-hec has an internal procedure that will cache the “/bin” directory from:

```
$SPLUNK_HOME/etc/apps/TA-nmon-hec/bin
```

To:

```
$SPLUNK_HOME/var/log/nmon/bin
```

This procedure is useful because:

- The Splunk deployment server starts by first completely removing the entire TA-nmon removing, this would let running nmon processes orphan (for Linux and Solaris)
- In Search Head Cluster, a constantly running nmon process with the application directory would generate an error during the bundle publication

The cache directory will be updated every time the “app.conf” files in the application directory differs from the version in cache, and is operated by the “bin/nmon_helper.sh” script.

3.3 Eventgen testing

3.3.1 Testing Nmon performance with evengen

Splunk Evengen is a pretty good and straightforward way to test the application.

Starting the TA-nmon version 1.3.28 and TA-nmon-hec version 1.3.32, we provide sample data for 2 AIX and 2 Linux servers. The data has been generated on IBM Power Development Cloud servers.

Finally, we use to run a system stress tool on 1 server of each category, such that you will have quickly active alerts and system statistic anomalies.

Eventgen will generate data for:

- performance metrics (sourcetype=nmon_data / eventtype=nmon:performance)
- configuration data (sourcetype=nmon_config / eventtype=nmon:config)

Additional data normally available within the application is related to the nmon data collection and will not be generated by Eventgen.

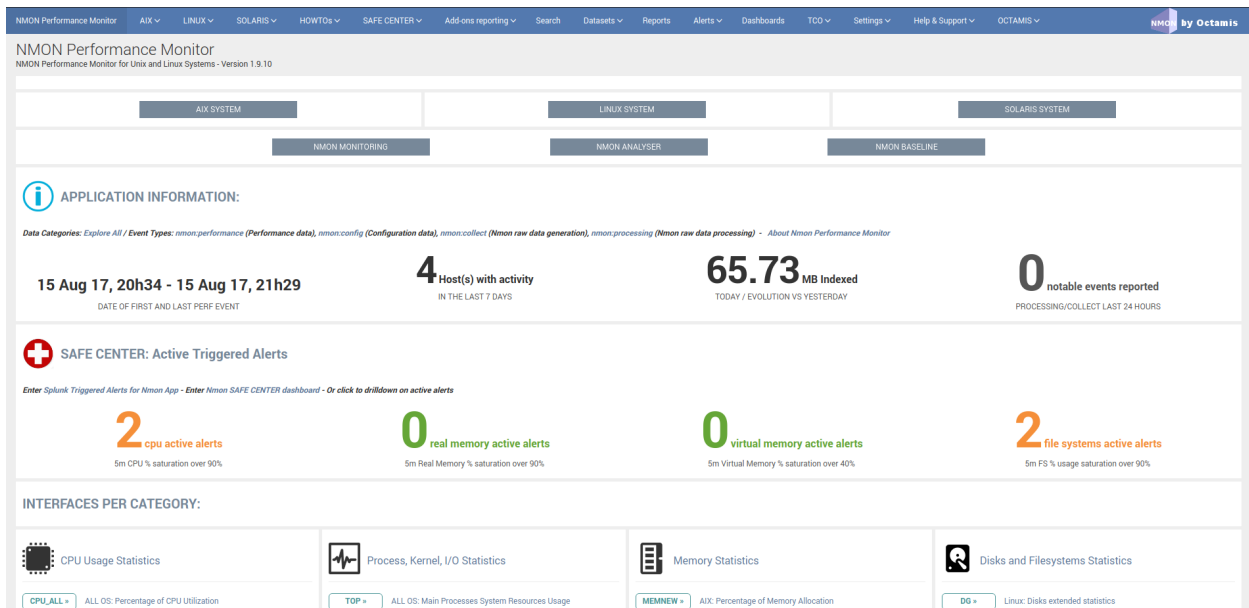
3.3.2 Get it working in 30 seconds

- Have a Splunk instance up and running
- Download the current eventgen version from <https://github.com/splunk/eventgen>
- Install the eventgen application, you should name the application directory as:

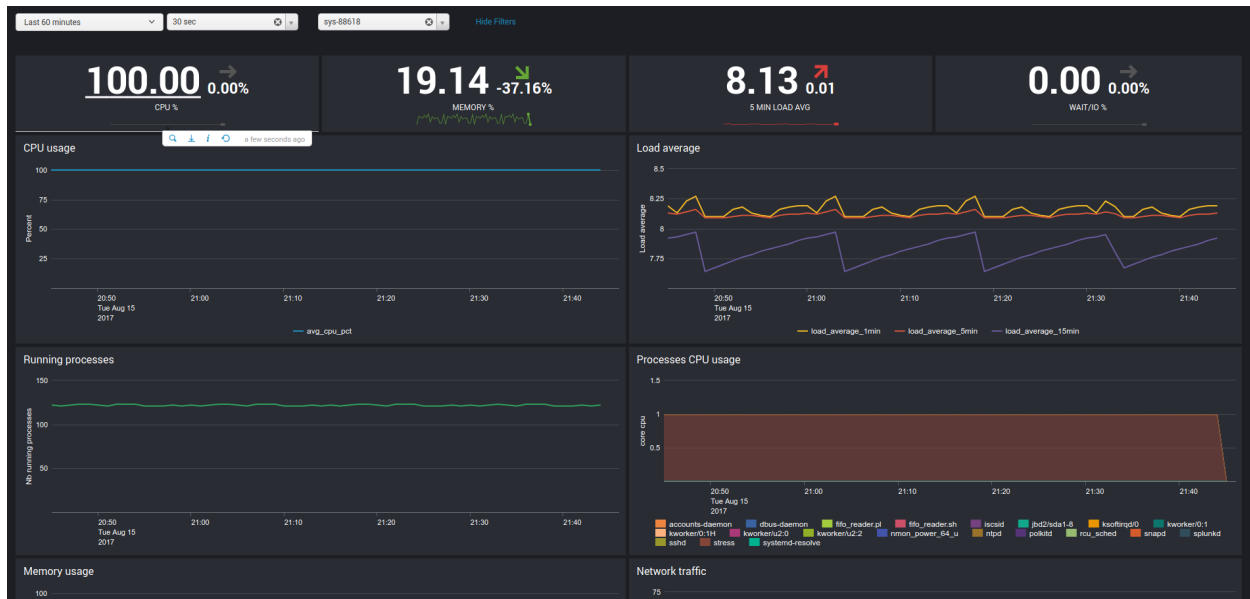
```
$SPLUNK_HOME/etc/apps/SA-Eventgen
```

- If not done already, install the Nmon Performance application (obvious!)
- Install either the TA-nmon or the TA-nmon-hec on this instance
- Create an index called “nmon”
- Restart Splunk

Immediately after Splunk restart, eventgen starts to generate nmon data, as visible from the application home page:



Example of a server running with abnormal load:



3.4 Extend Nmon with external data

You can extend very easily the context of the nmon data using the nmon external scripts.

Integrating external data is an integrated feature in nmon binaries (for AIX, Linux and Solaris), and it has been integrated within the TA-nmon such that it is even much easier to integrate anything that matters for you.

The data can be retrieved from:

- Any command available to the operating system
- Script of any kind that can be called within a shell script: other shell script, Python, Perl...

Basically anything you can retrieve from your systems!

3.4.1 How does it work ?

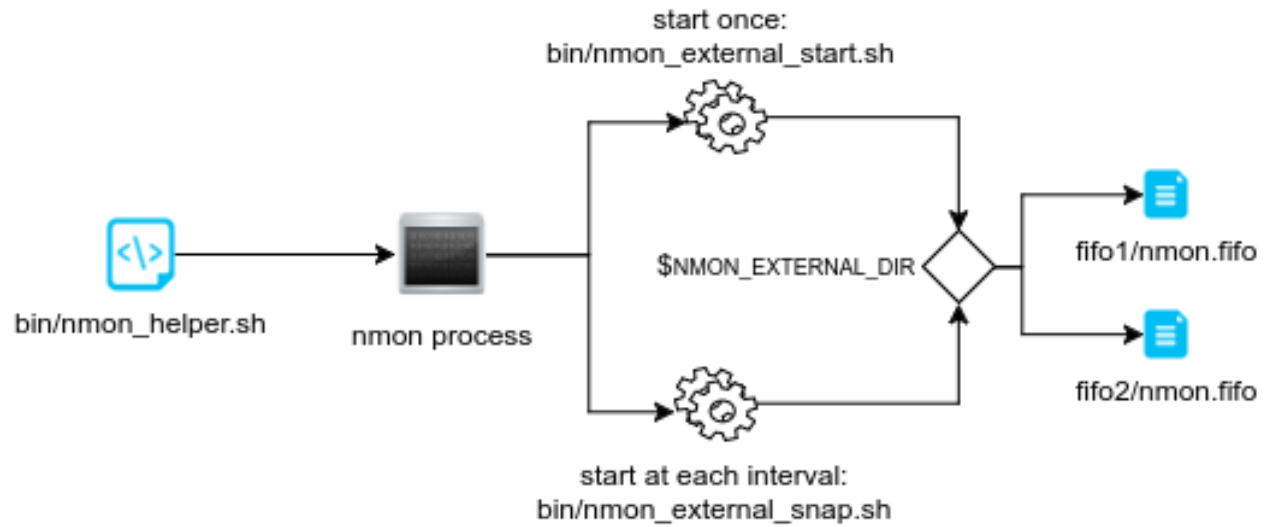
Very simple, we start with 2 simple shell scripts involved:

- bin/nmon_external_cmd/nmon_external_start.sh
- bin/nmon_external_cmd/nmon_external_snap.sh

Both scripts are being called automatically by nmon binaries, using the global environment variables (set by bin/nmon_helper.sh):

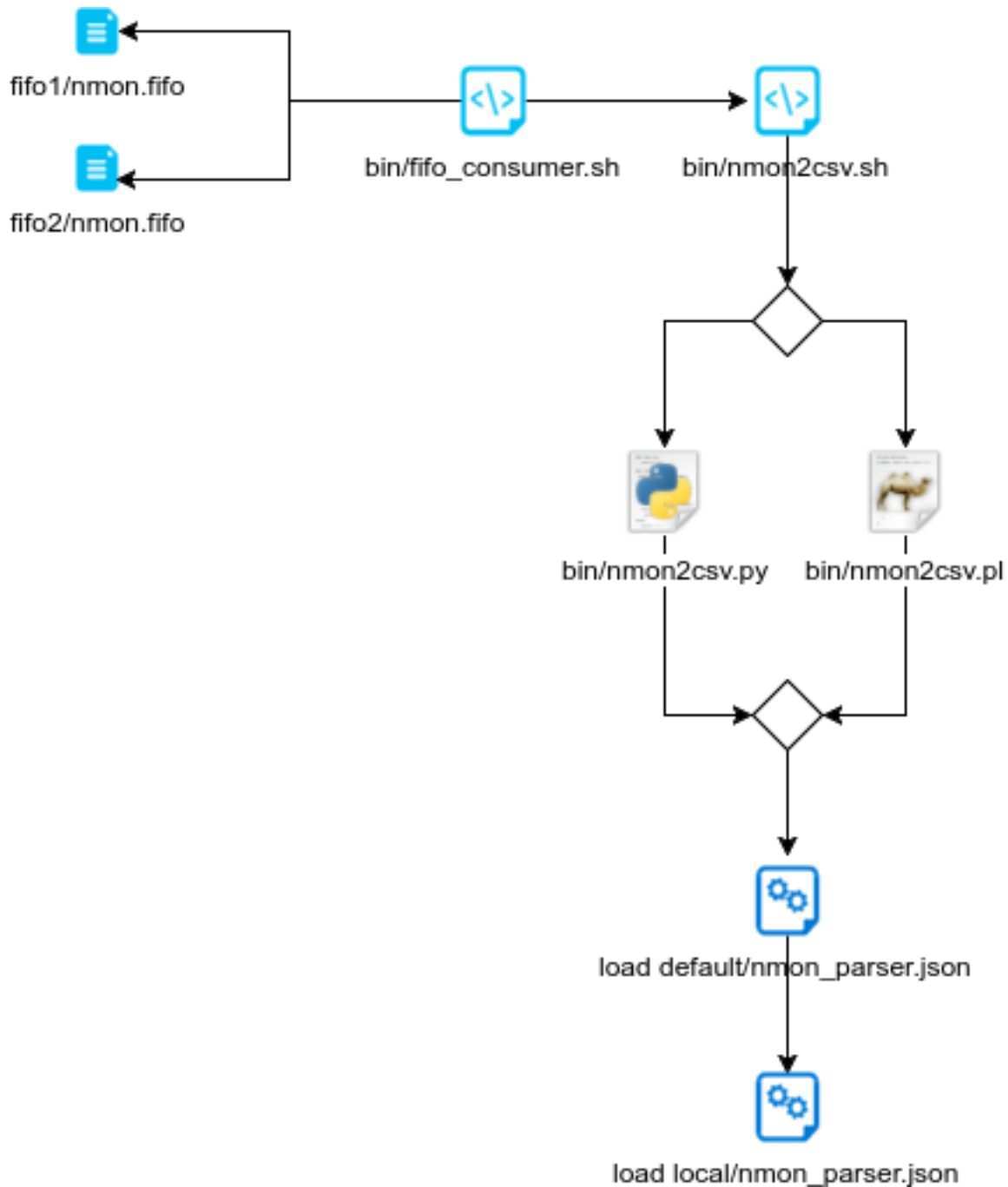
- NMN_START which equals to the full path of the nmon_external_start.sh script
- NMN_SNAP which equals to the full path of the nmon_external_snap.sh script

*An additional variable set by bin/nmon_helper.sh defines the fifo file path were to write the data (used by bin/nmon_external_cmd/.sh script)**



Then, nmon parsers will automatically load the list of nmon sections to be parsed (the “type” field in Splunk) defined in:

- `default/nmonparser_config.json` (for the default configuration)
- `local/nmonparser_config.json` (for upgrade resilient customization)



3.4.2 Ok got it, how do I add mine ?

Right, very simple, let's take the example of the uptime command output:

Running the “uptime” command outputs various interesting information: server uptime, number of Unix users connected, system load for last minute, 5 minutes, 15 minutes:

```
19:08:45 up 11:00,  1 user,  load average: 0.13, 0.22, 0.19
```

STEP 1: header definition

Within the “bin/nmon_external_start.sh” script, we add:

```
# uptime information
echo "UPTIME,Server Uptime and load,uptime_stdout" >>$NMON_EXTERNAL_DIR/nmon.fifo
```

Explanations:

- The first field , in our “UPTIME”, with the nmon section name, indexed in the “type” field in Splunk, and to be added in the nmonparser_config.json
- The second field is a descriptive field
- All the other fields are the data fields
- All fields must be comma separated

STEP 2: data generation

In the step 2, we modify the “bin/nmon_external_snap.sh” script to add:

```
# Uptime information (uptime command output)
echo "UPTIME,$1,\"`uptime | sed 's/^s//g' | sed 's/,/;/g'`\"" >>$NMON_EXTERNAL_DIR/
↪nmon.fifo
```

Explanations:

- The first field refers to the nmon section we previously defined in “bin/nmon_external_snap.sh”
- The second field “\$1” refers to the value of the nmon time stamp (Txxxx), and will be defined automatically by nmon
- The first field defines here our data field (we could have more than one!)
- In the uptime example, our command produces commas, to avoid any trouble we replace any comma by colons, and we protect the field by double quotes

STEP 3: add the monitor in nmonparser_config.json

The uptime example is integrated in the TA-nmon, its definition can be found in “default/nmonparser_config.json”.

However, if you add your own monitors, please create a “local/nmonparser_config.json” and add your monitors declaration.

There is 2 types of declaration to be used:

- “nmon_external:” This is a simple literal parsing of the data, the output will be indexed the same way it has been produced
- “nmon_external_transposed”: This is a different case where data will be transposed, it has to be used when you have a notion of “device/value”

Example:

In our example, we just need to have:

```
"nmon_external": [ "UPTIME" ],
```

More explanations about the “nmon_external_transposed”:

Here is an example of nmon data that the parser automatically transpose:

```
DISKXFER,Disk transfers per second sys-86400,sr0,sda,sda1,sda2,sdb
DISKXFER,T0001,0.5,0.7,0.2,0.5,0.0
DISKXFER,T0002,0.0,3.1,0.0,3.1,0.0
```

(continues on next page)

(continued from previous page)

```
DISKXFER,T0003,0.0,2.1,0.0,2.1,0.0
DISKXFER,T0004,0.0,1.1,0.0,1.1,0.0
```

Using the “nmon_external_transposed” will produce the following type of data in Splunk:

```
DISKXFER,sys-86391,sys-86391,Linux,60,1440,28-03-2017 14:36:14,sda,2.0 DISKXFER,sys-
86391,sys-86391,Linux,60,1440,28-03-2017 14:36:14,sda1,0.0 DISKXFER,sys-86391,sys-
86391,Linux,60,1440,28-03-2017 14:36:14,sda2,2.0 DISKXFER,sys-86391,sys-
86391,Linux,60,1440,28-03-2017 14:36:14,sdb,0.0 DISKXFER,sys-86391,sys-86391,Linux,60,1440,28-
03-2017 14:36:14,sr0,0.0
```

With the following major fields:

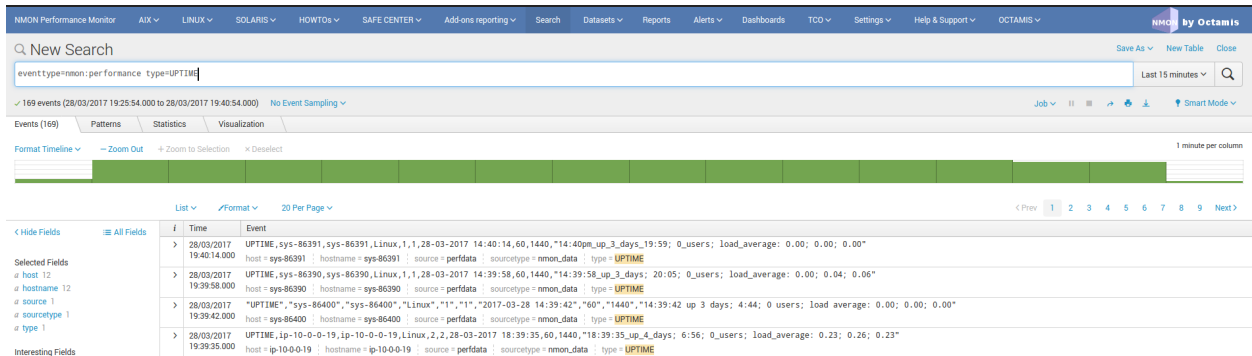
- type=DISKXFER
- host=xxxxxxx
- device=xxxxxxx (sda, sda1...)
- value=xxxxxxx (with the relevant value for that device, at that time stamp)

Which will be much more easy to analyse in Splunk, and allow the management of very large volume of data.

Et voila !

FINAL:

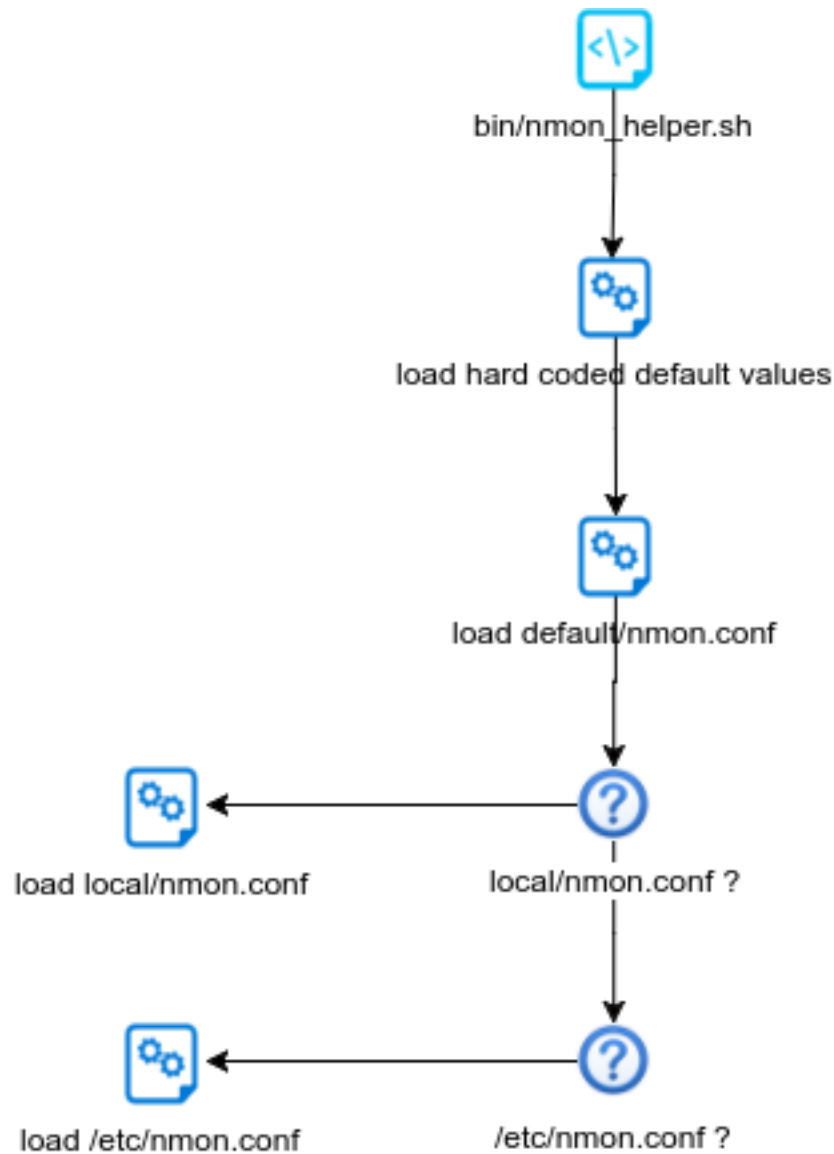
Finally deploy you new configuration to your servers, kill the running nmon processes or wait for their current cycle to end.



3.5 Configure your options with nmon.conf

The TA-nmon implements a main configuration file “nmon.conf” to be used for your customizations.

The configuration file uses the Splunk configuration philosophy (default versus local, etc...) such that you set your own options, and retain them over updates.



3.5.1 Splunk HEC specific options

The TA-nmon-hec implements specific options:

Splunk HEC endpoint URL:

```
--splunk_http_url https://splunk.mydomain.com:8088/services/collector/event
```

Replace:

- “splunk.mydomain.com” by the fully qualified domain name or IP of your Load balanced VIP / Splunk HEC endpoint
- “https” by “http” if you don’t need to use SSL

Splunk HEC token:

```
--splunk_http_token insert_your_splunk_http_token
```

Replace:

- “insert_your_splunk_http_token” by the value of your token

local logging:

This option is active by default:

```
--no_local_log
```

When set, this will **disable** local logging on the machine running the TA-nmon-hec.

If you remove this option, the TA-nmon-hec will log every piece of data on the local machine:

```
$SPLUNK_HOME/var/log/nmon/nmon_perfdata.log
$SPLUNK_HOME/var/log/nmon/nmon_configdata.log
$SPLUNK_HOME/var/log/nmon/nmon_processing.log
$SPLUNK_HOME/var/log/nmon/nmon_collect.log
$SPLUNK_HOME/var/log/nmon/nmon_cleaner.log
```

3.5.2 interval & snapshot

The “interval” and “snapshot” variables are the 2 nmon options that will define respectively the time in seconds between 2 measures of performance, and the number of measures to be achieved.

These 2 factors will define the time to live in seconds of a given nmon process, since the branch 1.3 of the TA-nmon, it is by default a cycle of 24 hours.

Legacy options for the TA-nmon for non fifo

The previous branch of the TA-nmon (not using fifo files) was having short cycle of nmon processes life time, to prevent from generating CPU load due to the processing of the nmon data:

```
###
### Legacy options for nmon writing to regular files (these values are used by the TA-
↳nmon not using fifo files)
###

# The "longperiod_high" mode is a good compromise between accuracy, CPU / licensing,
↳cost and operational intelligence, and can be used in most case
# Reducing CPU foot print can be achieved using one of the following modes,
↳increasing the interval value and limiting the snapshot value are the factors that
↳will impact the TA footprint
# If you observe a too large CPU foot print on your servers, please choose a
↳different mode, or a custom mode

# Available modes for proposal below:

#   shortperiod_low)
#                       interval="60"
#                       snapshot="10"

#   shortperiod_middle)
#                       interval="30"
```

(continues on next page)

(continued from previous page)

```

#             snapshot="20"

#  shortperiod_high)
#             interval="20"
#             snapshot="30"

#  longperiod_low)
#             interval="240"
#             snapshot="120"

#  longperiod_middle)
#             interval="120"
#             snapshot="120"

#  longperiod_high)
#             interval="60"
#             snapshot="120"

# custom --> Set a custom interval and snapshot value, if unset short default values,
↳ will be used (see custom_interval and custom_snapshot)

# Default is longperiod_high
mode="longperiod_high"

# Refresh interval in seconds, Nmon will use this value to refresh data each X seconds
# UNUSED IF NOT SET TO custom MODE
custom_interval="60"

# Number of Data refresh occurrences, Nmon will refresh data X times
# UNUSED IF NOT SET TO custom MODE
custom_snapshot="120"

```

These options are not used anymore by the TA-nmon and will be removed in the future

New options for the fifo implementation

The new branch of the TA-nmon use the following parameters to define the interval & snapshot values:

```

###
### FIFO options: used since release 1.3.0
###

# Using FIFO files (named pipe) are now used to minimize the CPU footprint of the
↳ technical addons
# As such, it is not required anymore to use short cycle of Nmon run to reduce the
↳ CPU usage

# You can still want to manage the volume of data to be generated by managing the
↳ interval and snapshot values
# as a best practice recommendation, the time to live of nmon processes writing to
↳ FIFO should be 24 hours

# value for interval: time in seconds between 2 performance measures
fifo_interval="60"

```

(continues on next page)

(continued from previous page)

```
# value for snapshot: number of measure to perform
fifo_snapshot="1440"
```

The minimal value for the “fifo_interval” should not be less than 10 seconds to let enough time for the “bin/fifo_consume.sh” script to be able to manage the nmon_data.

The recommended cycle for the time to live of an nmon process is 24 hours.

3.5.3 NFS Statistics

NFS options for AIX and Linux: Activate NFS statistics:

Out of the box, NFS statistics generation is disabled. You may enable this feature but note it is only applicable to Linux and AIX - not Solaris.

To activate NFS statistics generation, you must activate this in a local/nmon.conf, as shown below:

```
### NFS OPTIONS ###

# Change to "1" to activate NFS V2 / V3 (option -N) for AIX hosts
AIX_NFS23="0"

# Change to "1" to activate NFS V4 (option -NN) for AIX hosts
AIX_NFS4="0"

# Change to "1" to activate NFS V2 / V3 / V4 (option -N) for Linux hosts
# Note: Some versions of Nmon introduced a bug that makes Nmon to core when
↳activating NFS, ensure your version is not outdated
Linux_NFS="0"
```

3.5.4 End time marging (Nmon parallel run)

Nmon processes generated by technical add-ons have specific time of live which is the computation of INTERVAL * SNAPSHOT.

Between two run of nmon collections, there can be several minutes required by nmon to collect configuration items before starting collecting performance metrics, moreover on very large systems.

For this reason, a parallel run of two nmon concurrent processes will occur a few minutes before the current process ends, which prevents from having gaps in charts and data.

This feature can be controlled by changing the value of the endtime_margin, and can also be totally deactivated if you like:

```
### VARIOUS COMMON OPTIONS ###

# Time in seconds of margin before running a new iteration of Nmon process to prevent
↳data gaps between 2 iterations of Nmon
# the nmon_helper.sh script will spawn a new Nmon process when the age in seconds of
↳the current process gets higher than this value

# The endtime is evaluated the following way:
# endtime=$(( ${interval} * ${snapshot} - ${endtime_margin} ))

# When the endtime gets higher than the endtime_margin, a new Nmon process will be
↳spawned
```

(continues on next page)

(continued from previous page)

```
# default value to 240 seconds which will start a new process 4 minutes before the
↳current process ends

# Setting this value to "0" will totally disable this feature

endtime_margin="240"
```

3.5.5 Linux OS specific options

Embedded nmon binaries versus locally available nmon binaries

In default configuration, the “nmon_helper.sh” script will always give the priority to embedded nmon binary.

The Application has embedded binaries specifically compiled for almost every Linux OS and versions, such that you can manage from a center place nmon versions for all your Linux hosts!

The nmon_helper.sh script will proceed as above:

- Search for an embedded binary that suits processor architecture, Linux OS version (example: RHEL), that suite vendor version (example: RHEL 7) and vendor subversion (RHEL 7.1) Best result will be achieved using /etc/os-release file, if not available specific information file will be searched (example: /etc/issue, /etc/redhat-release, etc...)
- In the worst case (no binary found for vendor OS (example: Linux RHEL), the nmon_helper.sh search for generic binary that fits the local processor architecture
- If none of these options are possible, the script will search for nmon binary in PATH
- If this fails, the script exists in error, this information will stored in Splunk and shown in home page “Notable events reported”.

```
### LINUX OPTIONS ###

# Change the priority applied while looking at nmon binary
# by default, the nmon_helper.sh script will use any nmon binary found in PATH
# Set to "1" to give the priority to embedded nmon binaries
# Note: Since release 1.6.07, priority is given by default to embedded binaries
Linux_embedded_nmon_priority="1"

# Change the limit for processes and disks capture of nmon for Linux
# In default configuration, nmon will capture most of the process table by capturing
↳main consuming processes
# This function is percentage limit of CPU time, with a default limit of 0.01
# Changing this value can influence the volume of data to be generated, and the
↳associated CPU overhead for that data to be parsed

# Possible values are:
# Linux_unlimited_capture="0" --> Default nmon behavior, capture main processes (no -
↳I option)
# Linux_unlimited_capture="-1" --> Set the capture mode to unlimited (-I -1)
# Linux_unlimited_capture="x.xx" --> Set the percentage limit to a custom value, ex:
↳"0.01" will set "-I 0.01"
Linux_unlimited_capture="0"

# Set the maximum number of devices collected by Nmon, default is set to 1500 devices
# This option will be ignored if you set the Linux_unlimited_capturation below.
# Increase this value if you have systems with more devices
```

(continues on next page)

(continued from previous page)

```
# Up to 3000 devices will be taken in charge by the Application (hard limit in
↳nmon2csv.py / nmon2csv.pl)
Linux_devices="1500"

# Enable disks extended statistics (DG*)
# Default is true, which activates and generates DG statistics
Linux_disk_dg_enable="1"

# Name of the User Defined Disk Groups file, "auto" generates this for you
Linux_disk_dg_group="auto"
```

Unlimited capture

Recently introduced, you can set nmon linux to run its mode of capture in unlimited mode, specially for the TOP section (processes) and block devices.

CAUTION: This option is experimental and can cause increasing volume of data to be generated

```
# Change the limit for processes and disks capture of nmon for Linux
# In default configuration, nmon will capture most of the process table by capturing
↳main consuming processes
# You can set nmon to an unlimited number of processes to be captured, and the entire
↳process table will be captured.
# Note this will affect the number of disk devices captured by setting it to an
↳unlimited number.
# This will also increase the volume of data to be generated and may require more cpu
↳overhead to process nmon data
# The default configuration uses the default mode (limited capture), you can set
↳bellow the limit number of capture to unlimited mode
# Change to "1" to set capture of processes and disks to no limit mode
Linux_unlimited_capture="0"
```

Maximum number of disk devices

The maximum number of disk devices to be taken in charge by nmon for Linux has to be set at starting time.

Note that currently, nmon2csv parsers have a hard limit at 3000 devices

```
# Set the maximum number of devices collected by Nmon, default is set to 1500 devices
# Increase this value if you have systems with more devices
# Up to 3000 devices will be taken in charge by the Application (hard limit in
↳nmon2csv.py / nmon2csv.pl)
Linux_devices="1500"
```

disk extended statistics:

```
# Enable disks extended statistics (DG*)
# Default is true, which activates and generates DG statistics
Linux_disk_dg_enable="1"

# Name of the User Defined Disk Groups file, "auto" generates this for you
Linux_disk_dg_group="auto"
```

3.5.6 Solaris OS specific options

Using a local/nmon.conf file, you can activate the generation of statistics for VxVM volumes:

```
### SOLARIS OPTIONS ###

# CHange to "1" to activate VxVM volumes IO statistics
Solaris_VxVM="0"
```

You can manage the activation / deactivation of UARG generation: (full commands arguments)

```
# UARG collection (new in Version 1.11), Change to "0" to deactivate, "1" to activate,
↳ (default is activate)
Solaris_UARG="1"
```

3.5.7 AIX OS specific options

For AIX hosts, you can customize the full command line sent to nmon at launch time, at the exception of NFS options. (see previous section)

```
### AIX COMMON OPTIONS ###

# CAUTION: Since release 1.3.0, we use fifo files, which requires the option "-
↳ yoverwrite=1"

# Change this line if you add or remove common options for AIX, do not change NFS,
↳ options here (see NFS options)
# the -p option is mandatory as it is used at launch time to save instance pid
AIX_options="-T -A -d -K -L -M -P -^ -p -yoverwrite=1"

# enable this line if you want to get only active disks
# AIX_options="-T -A -d -K -L -M -P -^ -p -k `lspsv|grep active|awk '{print $1","}'
↳ '|tr -d '\040\011\012\015'` -yoverwrite=1"
```

3.5.8 Global options

These options are not related to nmon binary options but to the TA-nmon global configuration:

```
# This option can be used to force the technical add-on to use the Splunk configured,
↳ value of the server hostname
# If for some reason, you need to use the Splunk host value instead of the system,
↳ real hostname value, set this value to "1"

# We will search for the value of host=<value> in $SPLUNK_HOME/etc/system/local/
↳ inputs.conf
# If no value can be found, or if the file does not exist, we will fallback to the,
↳ normal behavior

# Default is use system hostname

# FQDN management in nmon2csv.pl/nmon2csv.py: The --fqdn option is not compatible,
↳ with the host name override, if the override_sys_hostname
# is activated, the --fqdn argument will have no effect

override_sys_hostname="0"

# nmon external generation management
```

(continues on next page)

(continued from previous page)

```
# This option will manage the activation or deactivation of the nmon external data_
↳ generation at the lower level, before it comes to parsers
# default is activated (value=1), set to "0" to deactivate

nmon_external_generation="1"

# Fifo options

# This option will deactivate the auto switch to fifo mode, in other words the TA-
↳ nmon will use the file mode and the old mechanism
# unless you encounter unexpected issues, you should not switch to the old mechanism_
↳ as the foot print is much higher

# Default is "1" which means write to fifo

mode_fifo="1"

# Since the release 1.3.0, AIX and Linux OS use the fifo_consumer.sh script to_
↳ consume data produced by the fifo readers
# the following option allows specifying the options sent to the nmon2csv parsers

# consult the documentation to get the full list of available options

# --mode realtime/colddata/fifo --> explicitly manage realtime data
# --use_fqdn --> use the host fully qualified domain name
# --json_output --> generate the performance data in json format instead of regular_
↳ csv data

# In fifo mode, options are sent by the fifo_consumer.sh
# In file mode, options are sent by Splunk via the nmon_processing stanza in props.
↳ conf

nmon2csv_options="--mode fifo"
```


4.1 Troubleshoot

There is a nice and complete troubleshoot guide in the Nmon Core application:

<http://nmon-for-splunk.readthedocs.io/en/latest/Userguide.html#troubleshooting-guide-from-a-to-z>

In a nutshell:

4.1.1 Expected running processes

Since the 1.3.x branch, you should find various processes running:

- 1 x nmon process (or 2 x nmon processes during the parallel interval)
- 1 x main Perl or Python fifo_reader process (or 2 x processes during the parallel interval)
- 1 x subshell fifo_reader process (or 2 x processes during the parallel interval)

On a Linux box:

```

root@centos-73-64:~
[root@centos-73-64 ~]# ps -ef | egrep nmon
root      5710      1  0 20:34 ?        00:00:00 python /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.py --fif
o fifo1
root      5712 5710  0 20:34 ?        00:00:00 /bin/sh /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.sh /opt
/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo
root      5733      1  0 20:34 ?        00:00:00 /opt/splunkforwarder/var/log/nmon/bin/linux/centos/nmon_x86_64_centos
7 -F /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo -T -s 60 -c 1440 -d 1500 -g auto -D -p
root     11634 6150  0 21:42 pts/0    00:00:00 grep -E --color=auto nmon
[root@centos-73-64 ~]#

```

On AIX, the *nmon* process will be called “*topas-nmon*”

On Solaris, the *sarmon* process will be called “*sadc*”

4.1.2 Starting processes

If you run in trouble and want to troubleshoot the situation, the easiest approach is stopping Splunk, kill existing *nmon* process and run the tasks manually:

- Stop Splunk and kill the *nmon* process:

```
./splunk stop
```

```

root@centos-73-64:~
[root@centos-73-64 ~]# ps -ef | egrep nmon
root      5710      1  0 20:34 ?        00:00:00 python /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.py --fif
o fifo1
root      5712 5710  0 20:34 ?        00:00:00 /bin/sh /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.sh /opt
/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo
root      5733      1  0 20:34 ?        00:00:00 /opt/splunkforwarder/var/log/nmon/bin/linux/centos/nmon_x86_64_centos
7 -F /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo -T -s 60 -c 1440 -d 1500 -g auto -D -p
root     11634 6150  0 21:42 pts/0    00:00:00 grep -E --color=auto nmon
[root@centos-73-64 ~]# /opt/splunkforwarder/bin/splunk stop
Stopping splunkd...
Shutting down. Please wait, as this may take a few minutes.
..... [ OK ]
Stopping splunk helpers... [ OK ]
Done.
[root@centos-73-64 ~]# kill 5733
[root@centos-73-64 ~]# ps -ef | egrep nmon
root     12314 6150  0 21:54 pts/0    00:00:00 grep -E --color=auto nmon
[root@centos-73-64 ~]#

```

You will observe that killing the nmon process will automatically terminate the fifo_reader.pll.py and the subshell fifo_reader.sh. This the expected behavior, and mandatory.

If the processes do not stop, then your problem became mine and please open an issue !

- Now we can manually starting the processes, example:

```
/opt/splunkforwarder/bin/splunk cmd /opt/splunkforwarder/etc/apps/TA-nmon/bin/
↪nmon_helper.sh
```

Please adapt the paths to your context

```
root@centos-73-64:~
[root@centos-73-64 ~]# ps -ef | egrep nmon
root      5710      1  0 20:34 ?        00:00:00 python /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.py --fif
o fifo1
root      5712 5710  0 20:34 ?        00:00:00 /bin/sh /opt/splunkforwarder/etc/apps/TA-nmon/bin/fifo_reader.sh /opt
/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo
root      5733      1  0 20:34 ?        00:00:00 /opt/splunkforwarder/var/log/nmon/bin/linux/centos/nmon_x86_64_centos
7 -F /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo -T -s 60 -c 1440 -d 1500 -g auto -D -p
root     11634 6150  0 21:42 pts/0    00:00:00 grep -E --color=auto nmon
[root@centos-73-64 ~]# /opt/splunkforwarder/bin/splunk stop
Stopping splunkd...
Shutting down. Please wait, as this may take a few minutes.
..... [ OK ]
Stopping splunk helpers...
[ OK ]
Done.
[root@centos-73-64 ~]# kill 5733
[root@centos-73-64 ~]# ps -ef | egrep nmon
root     12314 6150  0 21:54 pts/0    00:00:00 grep -E --color=auto nmon
[root@centos-73-64 ~]# /opt/splunkforwarder/bin/splunk cmd /opt/splunkforwarder/etc/apps/TA-nmon/bin/nmon_helper.sh
Fri Mar 31 21:59:50 BST 2017, centos-73-64 INFO: Removing stale pid file
Fri Mar 31 21:59:50 BST 2017, centos-73-64 INFO: starting the fifo_reader fifo1
Fri Mar 31 21:59:51 BST 2017, centos-73-64 INFO: starting nmon : /opt/splunkforwarder/var/log/nmon/bin/linux/centos/n
mon_x86_64_centos7 -F /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/nmon.fifo -T -s 60 -c 1440 -d 1500
-g auto -D -p in /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1
[root@centos-73-64 ~]#
```

Let's summarize what happened here:

- nmon_helper.sh starts the fifo reader, if there is no fifo_reader running, the “fifo1” process will be started
- the fifo_reader.pll.py starts a fifo_reader.sh process in the background
- nmon_helper.sh starts the nmon process which will write its data to the relevant fifo file
- the nmon process cannot start if the fifo_reader has not started

If something unexpected happens and that the fifo_reader and nmon process do not start normally, you may want to trouble shoot the nmon_helper.sh script.

You can do very easily by commenting out “# set -x”, re-run the script and analyse the output. (you might need to add the set-x within the functions as well)

4.1.3 Checking fifo_reader processes

The fifo_reader processes will continuously read the fifo file written by the nmon process, and generate various dat files that represent the different typologies of nmon data:

```

root@centos-73-64:~
[root@centos-73-64 ~]# ls -ltr /opt/splunkforwarder/var/log/nmon/var/
total 8
drwx--x---. 4 root root  44 Mar 31 20:34 nmon_repository
drwx--x---. 2 root root   6 Mar 31 20:35 config_repository
drwx--x---. 2 root root 4096 Mar 31 20:36 centos-73-64_centos-73-64
drwx--x---. 2 root root   6 Mar 31 21:51 csv_workingdir
drwx--x---. 2 root root   6 Mar 31 21:51 csv_repository
-rw-----. 1 root root   6 Mar 31 21:59 fifo_reader_fifo1.pid
[root@centos-73-64 ~]# ls -ltr /opt/splunkforwarder/var/log/nmon/var/nmon_repository/fifo1/
total 80
-rw-----. 1 root root 24488 Mar 31 20:34 nmon_config.dat.rotated
-rw-----. 1 root root  2635 Mar 31 20:35 nmon_header.dat.rotated
-rw-----. 1 root root  2560 Mar 31 21:53 nmon_timestamp.dat.rotated
-rw-----. 1 root root  3660 Mar 31 21:53 nmon_data.dat.rotated
-rw-----. 1 root root  2599 Mar 31 21:59 nmon_header.dat
-rw-----. 1 root root 24502 Mar 31 21:59 nmon_config.dat
-rw-----. 1 root root   256 Mar 31 22:06 nmon_timestamp.dat
prw-----. 1 root root    0 Mar 31 22:06 nmon.fifo
-rw-----. 1 root root  9404 Mar 31 22:06 nmon_data.dat
[root@centos-73-64 ~]#

```

How this it work?

- The `fifo_reader.sh` reads every new line of data written to the fifo file (named pipe) and sends the data to the `fifo_reader.pll.py`
- The `fifo_reader.pll.py` parses the lines and applies various regular expressions to decide where to write the data, depending on its content
- If there were existing `.dat` files at the startup of the `fifo_reader` processes, those `dat` files are rotated and renamed to `“.rotated”`
- The `nmon.fifo` is not regular file but a named pipe (observe the `“prw——”`), its size will always be equal to 0

4.1.4 Testing the Splunk HEC endpoint

Splunk HEC specificity:

By default, the TA-nmon-hec does not generate any data locally on the machine, the data is generated and streamed directly over http/https.

As well, there won't be any output of the streaming operation, if there are any failures like networking failure or invalid parameters (invalid URL or token), this will be silently removed.

Sending a testing data:

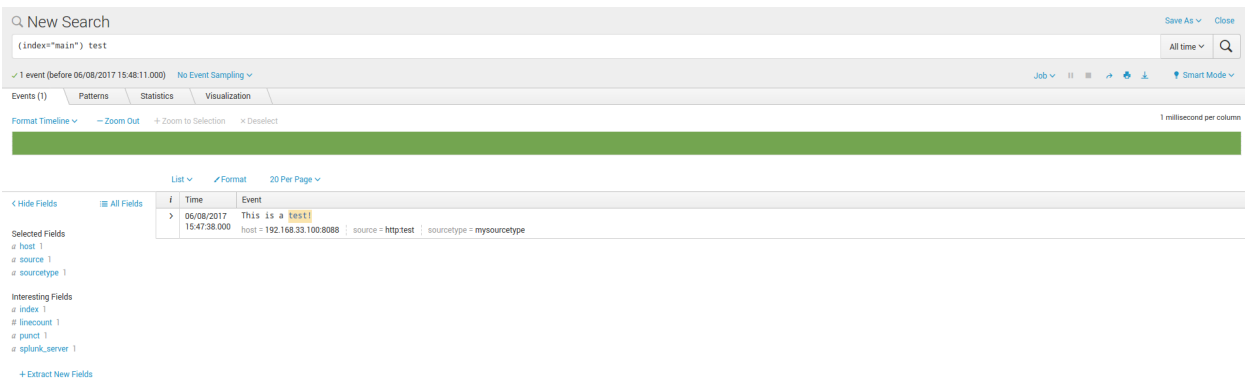
```

curl -k -H "Authorization: Splunk 12345678-1234-1234-1234-1234567890AB" https://
mysplunkserver.example.com:8088/services/collector/event -d '{"sourcetype":
"mysourcetype", "event": "This is a test!"}'

```

```
vagrant@hec-client1: ~
vagrant@hec-client1:~$ curl -k -H "Authorization: Splunk 1A498F57-C224-4C11-AB0B-0998905E3947" https://192.168.33.100:8088/services/collector/event
-d '{"sourcetype": "mysourcetype", "event": "This is a test!"}'
{"text": "Success", "code": 0}vagrant@hec-client1:~$
```

This must result in an event indexed in Splunk: (check the default index you have set in your token)



If this test works from the machine, this validates:

- The network path to your HEC endpoint from the server (routing, firewall, etc. . .)
- your Splunk HEC configuration

4.1.5 Checking the data parsing

The parsing of those dat files is being achieved in 2 main steps:

- The “bin/fifo_consumer.sh” script is started every 60 seconds by Splunk
- This script will check if an `nmon_data.dat` file exists and that its size is greater than 0
- If the size of the `nmon_dat.data` file equals to 0, then the `fifo_consumer.sh` has nothing to do and will exit this fifo file
- If the size is greater than 0 but its modification time (`mtime`) is less than 5 seconds, the script will loop until the condition is true

- The `fifo_consumer.sh` reads the `dat` file, recompose the `nmon` file and stream its content to the “`bin/nmon2csh.sh`” shell wrapper
- After this operation, the `nmon_data.dat` file will be empty for the next cycle
- The shell wrapper reads in `stdin` the data, and send it to the `nmon2kv` parser (`bin/nmon2kv.pll.py`)
- The parser reads the `nmon` data, parses it and produces the final data and stream to the Splunk HEC endpoint

Since the TA-nmon-hec does not write any local data by default, for testing purposes you can remove the “`--no_local_log`” from:

```
$SPLUNK_HOME/etc/apps/TA-nmon-hec/local/nmon.conf
```

Within a minute, the log file will be created and will contain the data to be streamed to the HEC endpoint:

```
root@hec-client1:~
root@hec-client1:~# cat /opt/splunkforwarder/etc/apps/TA-nmon-hec/local/nmon.conf
# HEC server configuration

nmon2csv_options="--mode fifo --silent --splunk_http_url https://192.168.33.100:8088/services/collector/event --splunk_http_token CEE56643-BA2D-48EE-94EF-AD0909718B2A"
root@hec-client1:~# ls -ltr /opt/splunkforwarder/var/log/nmon/*.log
-rw-r--r-- 1 root root 524 Aug 6 15:54 /opt/splunkforwarder/var/log/nmon/nmon_clean.log
-rw-r--r-- 1 root root 58462 Aug 6 15:57 /opt/splunkforwarder/var/log/nmon/nmon_perfdata.log
-rw-r--r-- 1 root root 3484 Aug 6 15:57 /opt/splunkforwarder/var/log/nmon/nmon_processing.log
-rw-r--r-- 1 root root 1056 Aug 6 15:57 /opt/splunkforwarder/var/log/nmon/nmon_collect.log
root@hec-client1:~# tail /opt/splunkforwarder/var/log/nmon/nmon_perfdata.log
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGXFER"
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGREADS"
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGREADMERGE"
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGREADSERV"
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGWRITES"
value="0.7" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGWRITEMERGE"
value="8.0" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGWRITESERV"
value="0.8" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGINFLIGHT"
value="1.0" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGIOTIME"
value="8.0" interval="60" timestamp="1502031551" hostname="hec-client1" ZZZZ="06-08-2017 15:59:11" snapshots="1440" device="sda" serialnum="hec-clie
nt1" Otype="Linux" type="DGBACKLOG"
root@hec-client1:~#
```

This validates:

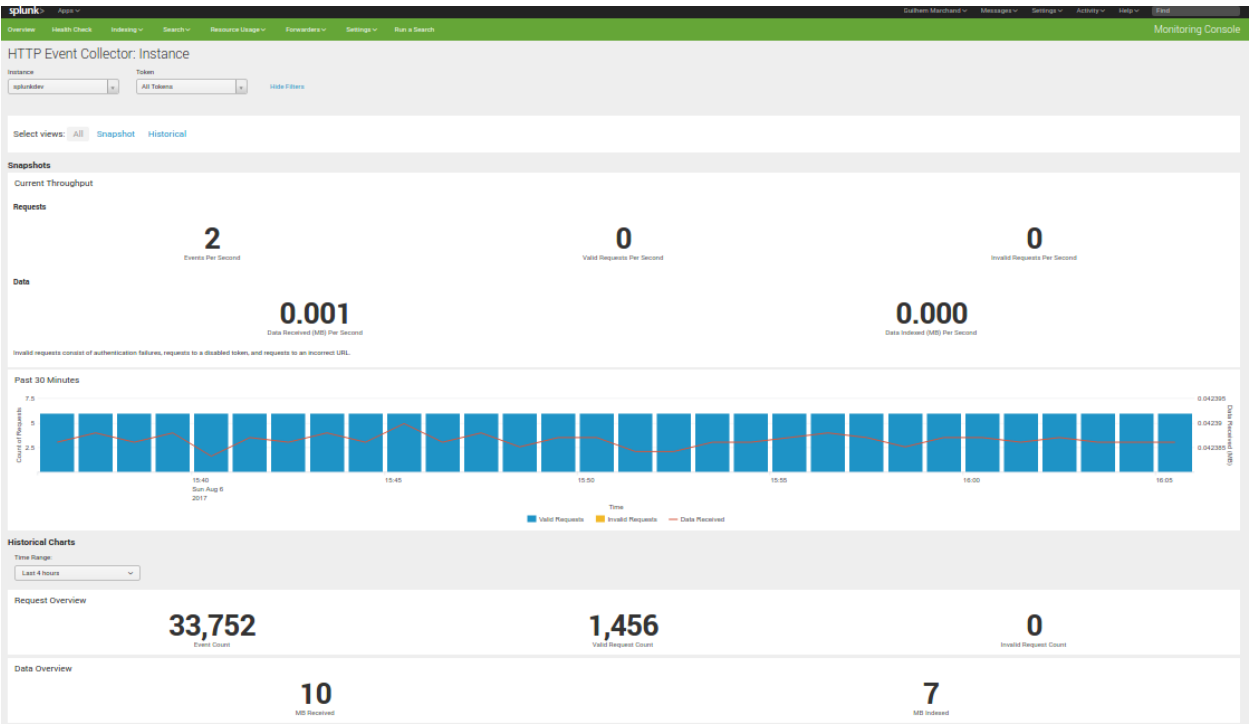
- that the collection works as expected
- that the performance data is properly read and generated

4.1.6 Splunk Monitoring Console

The Splunk Monitoring Console (ex DMC!) provides a view to analyse the Splunk HEC activity:

Indexing / Inputs / HTTP Event Collector: Instance

This view provides information about your HEC instance(s) that will demonstrate if you are receiving bad requests and any error like an invalid or deactivated token:



5.1 FAQ

Can I have my own nmon collection while running the TA-nmon-hec ?

The answer is no, and this won't be supported.

For foot print optimization purposes, the TA-nmon-hec has his own workflow which implements the so called "fifo mode".

Can I have the TA-nmon-hec and the TA-Unix in the same time ?

You might need the TA-unix for specific tasks that are out of the scope of the Nmon application, such as ingesting security related data.

Running both addons in same servers is not a problem at all.

The TA-nmon-hec is CIM compatible, for most performance related metrics, the TA-nmon-hec can be transparently used in replacement of the TA-Unix.

Can I have the TA-nmon-hec and the TA-nmon in the same time ?

The answer is no.

You must run either the TA-nmon, or the TA-nmon-hec. Both TAs share the same parsing configuration.

Is the TA-nmon-hec CIM compatible ?

Yes it is. The TA-nmon-hec is CIM compatible, it will specially deal with the following CIM data models:

- Application State
- Inventory
- Network Traffic
- Performance

If you are an Enterprise Security customer for instance, all you need is having the TA-nmon deployed in search heads as well.