Apache Warble Documentation

Release 0.1

The Apache Warble Community

Jul 01, 2018

Contents:

CHAPTER 1

Setting up Apache Warble

1.1 Understanding the Components

Warble currently consists of two major components:

The Warble Master Server (warble-server) This is the main database and UI Server. It serves as the hub for the nodes/agents to connect to, and provides the overall management of hosts, tests, as well as the visualizations and API end points.

The Warble Node Applications (warble-node) This is a daemon with a collection of test classes used to test external hosts for various services and/or response values. Nodes send results back to the master, which then processes and responds accordingly (for instance, in the case of downtime).

A third major component, the Warble Agent Applications, are being worked on, but is not completed.

1.2 Component Requirements

1.2.1 Server Component

The main Warble Server is a hub for nodes/agents and tests, and as such, is generally speaking only needed on one machine. It is recommended that, for larger instances of warble, you place the application on a machine or VM with sufficient resources to handle the database load and memory requirements.

We will be working towards a multi-master setup option, but that is currently not available.

As a rule of thumb, the Server does not require a lot of disk space (enough to hold the compiled database and time-series), but it does require CPU and RAM. The nodes/agents require virtually no disk space, as all test results are sent to the master server for storage.

1.2.2 Node Component

The node component can either consist of one instance, or be spread out across multiple machines for a distributed test coverage. Nodes will auto-adjust the test speed to match the number of CPU cores available to it; a node with two cores available will run up to 256 simultaneous jobs, whereas a scanner with eight cores would run up to 1024 simultaneous jobs to speed up processing. A node will typically require somewhere between 256 and 512MB of memory, and thus can safely run on a VM with 2GB memory (or less).

1.3 Source Code Location

Apache Warble does not currently have any releases. You are however welcome to try out the development version.

For the time being, we recommend that you use the `master` branch for testing Warble. This applies to both scanners and the server.

The Warble Server can be found via our source repository at https://github.com/apache/incubator-warble-server

The Warble Node Application can be found via: https://github.com/apache/incubator-warble-node

1.4 Installing the Server

1.4.1 Pre-requisites

Before you install the Warble Server, please ensure you have the following components installed and set up:

- A web server of your choice (Apache HTTP Server, NGINX, lighttp etc)
- Python 3.4 or newer with the following libraries installed:
- – yaml
- – certifi
- – sqlite3
- – bcrypt
- – cryptography >= 2.0.0
- Gunicorn for Python 3.x (often called gunicorn3) or mod_wsgi

1.4.2 Configuring and Priming the Warble Server

Once you have the components installed and Warble Server downloaded, you will need to prime the databases and create a configuration file.

Assuming you wish to install warble in /opt/warble, you would set it up by issuing the following:

- `git clone https://github.com/apache/incubator-warble-server.git /opt/warble`
- `cd /opt/warble/setup`
- `python3 setup.py`
- Enter the configuration parameters the setup process asks for

This will set up the database, the configuration file, and create your initial administrator account for the UI. You can later on do additional configuration of the data server by editing the `api/yaml/warble.yaml` file.

1.4.3 Setting up the Web UI

Once you have finished the initial setup, you will need to enable the web UI. Warble is built as a WSGI application, and as such you can use mod_wsgi for apache, or proxy to Gunicorn. In this example, we will be using the Apache HTTP Server and proxy to Gunicorn:

- Make sure you have mod_proxy and mod_proxy_http loaded (on debian/ubuntu, you would run: *a2enmod proxy_http*)
- Set up a virtual host in Apache:

```
<VirtualHost *:80>
   # Set this to your domain, or add warble.localhost to /etc/hosts
   ServerName warble.localhost
   DocumentRoot /opt/warble/ui/
   # Proxy to gunicorn for /api/ below:
   ProxyPass /api/ http://localhost:8000/api/
</VirtualHost>
```

- Launch gunicorn as a daemon on port 8000:

```
cd /opt/warble/api
gunicorn -w 10 -b 127.0.0.1:8000 handler:application -t 120 -D
```

Once httpd is (re)started, you should be able to browse to your new Warble instance.

1.5 Installing Nodes

1.5.1 Pre-requisites

The Warble Nodes rely on the following packages:

- Python >= 3.4 with the following packages:
- – python3-yaml
- – python3-ldap
- – python3-dns

Custom node tests may require additional packages.

1.5.2 Configuring a node

First, check out the node source in a file path of your choosing:

`git clone https://github.com/apache/incubator-warble-node.git`

Then edit the `conf/config.yaml` file to point towards the proper Warble Master server.

Then fire up the node software as a daemon:

`python3 node.py start`

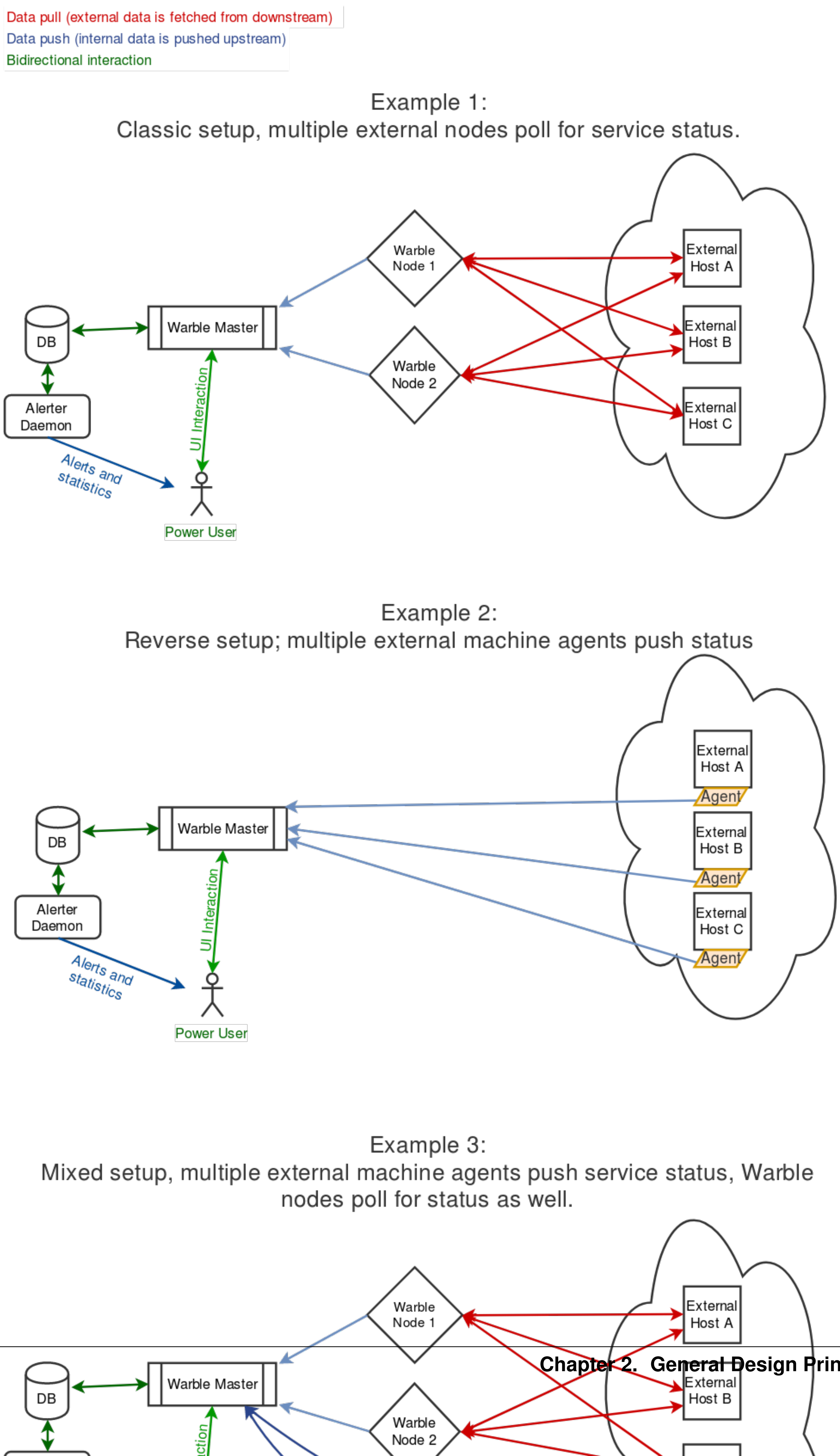
Warble Node apps will, when run the first time, set up an async key pair for encryption and verification, and request a spot in the Warble Master node registry. Spots are verified/approved in the Warble UI, and once completed, the node will receive an API key that corresponds with its ID and key pair, and get to work. It is worth noting, that the Warble node software needs write access to the configuration directory on disk, so it can store the API key and async key pair.

CHAPTER 2

General Design Principles

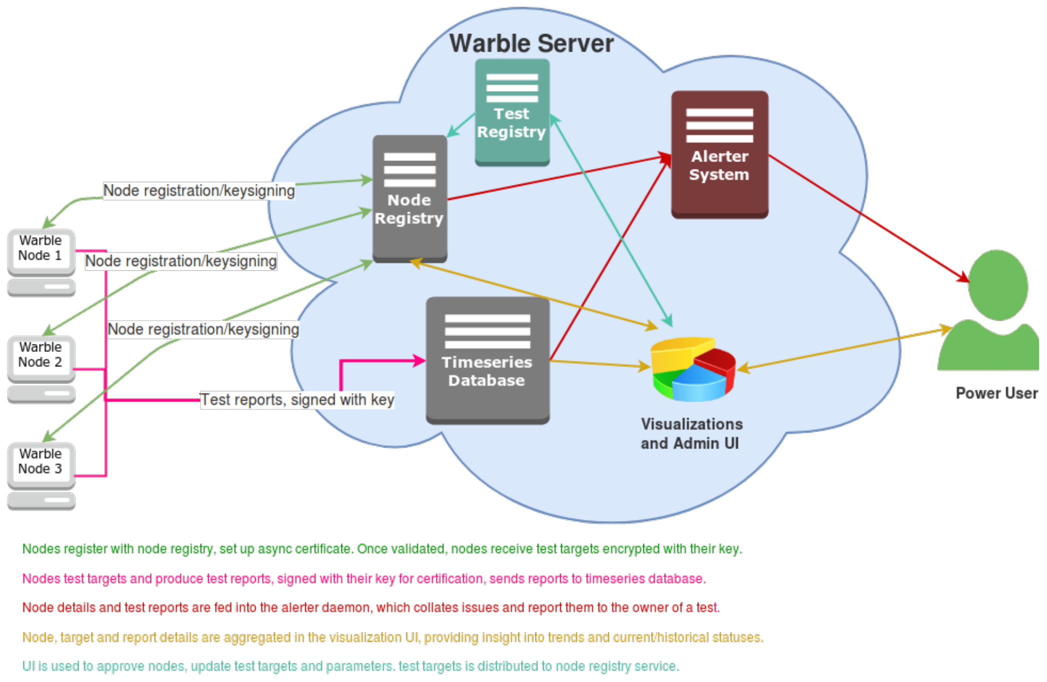
2.1 Client/Server breakdown

(Work in progress!) This section shows the basic three applications of Warble.


Data pull (external data is fetched from downstream)
Data push (internal data is pushed upstream)
Bidirectional interaction
Example 1:
Classic setup, multiple external nodes poll for service status.
Warble Node 1
Warble Node 2
External Host A
External Host B
External Host C
DB
Warble Master
Alerter Daemon
UI Interaction
Alerts and statistics
Power User
Example 2:
Reverse setup; multiple external machine agents push status
External Host A
Agent
External Host B
Agent
External Host C
Agent
DB
Warble Master
Alerter Daemon
UI Interaction
Alerts and statistics
Power User
Example 3:
Mixed setup, multiple external machine agents push service status, Warble nodes poll for status as well.
Warble Node 1
Warble Node 2
External Host A
External Host B
DB
Warble Master

2.2 Agent/Node and Server data flows

(Work in progress!) This section shows the various components inside the Warble Server and how they interact. More to come :)


Warble Server
Test Registry
Alerter System
Node Registry
Node registration/keysigning
Node registration/keysigning
Node registration/keysigning
Warble Node 1
Warble Node 2
Warble Node 3
Test reports, signed with key
Timeseries Database
Visualizations and Admin UI
Power User

Nodes register with node registry, set up async certificate. Once validated, nodes receive test targets encrypted with their key.

Nodes test targets and produce test reports, signed with their key for certification, sends reports to timeseries database.

Node details and test reports are fed into the alerter daemon, which collates issues and report them to the owner of a test.

Node, target and report details are aggregated in the visualization UI, providing insight into trends and current/historical statuses.

UI is used to approve nodes, update test targets and parameters. test targets is distributed to node registry service.

2.3 Client and Server communication

Agents and Nodes (referred to in this segment as *clients*) communicate with the Warble Server using a three-stage protocol:

1. First time a client is started, it generates an async RSA key pair (default key size is 4096 bits) for encryption and subsequent verification/signing. The private key is stored on-disk on the client host, and the public key is sent to the node registry on the master, along with a request to add the client to the node registry as a verified client. The Server registers a unique API key for each client, and binds the public key to this API key.
2. Once verified, a client can request test targets and parameters from the node registry at the Server. This data is sent back to the client in encrypted form, using the previously sent public key. Thus, only a verified client can get test targets, and only the client should be able to decrypt the payload and get clear-text target data.
3. Once a client has completed a test (or a batch of tests), the result is sent to the server and signed using the private key. Thus, the server can use the public key to verify that the test results came from the client.

Once test data has been successfully verified and stored on the server, both the alerting system and the visualization system can retrieve and process it, ensuring that what they (and you) see is genuine.

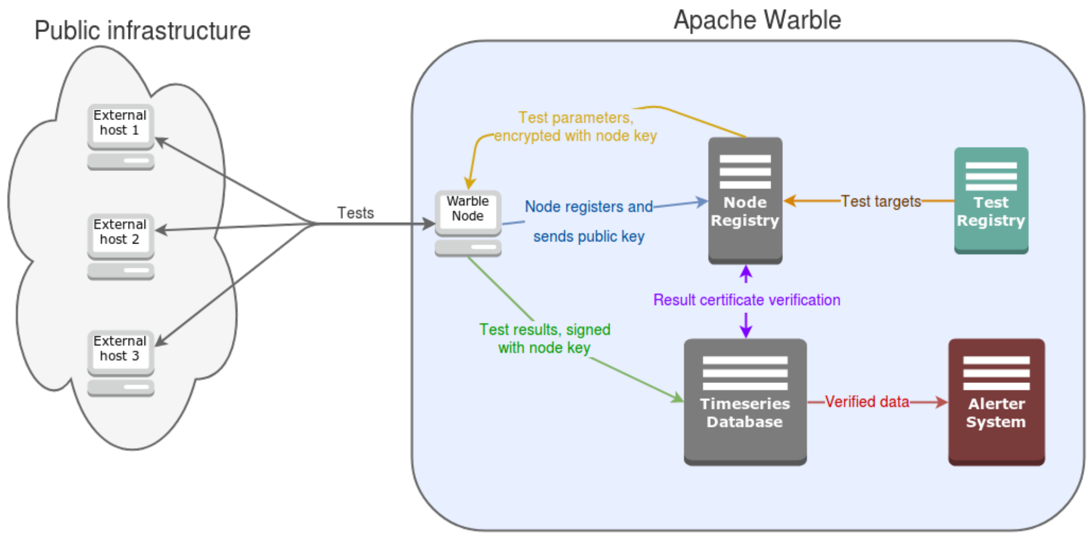


Fig. 1: This figure shows the communication channels as outlined in the above paragraph.

CHAPTER 3

Node Task Registry Design

3.1 Node Tasks

3.1.1 Basic Task Design

Warble Nodes can have one or more (or all) tasks assigned to it. Each task consists of a target to test, as well as what to test and how to go about that, encapsulated in a payload object. Each check you wish to perform requires an associated task, but may be performed by multiple nodes. Thus, testing whether your main web site works on port 80 requires a task, as does a test for https on port 443, as they are technically two distinct targets. Specific tasks may have optional tests built into them,for instance a SSL certificate check on a https site.

3.1.2 Task status

A task can be either enabled, disabled, or muted. Disabling a task prevents it from running on nodes, whereas muting a task will still cause nodes to perform it, but alerting will be silenced. Muting can be used for when you still need to monitor a situation, but you don't need to be reminded whenever the test results changes.

3.1.3 Task sensitivity

A task can also have a specific sensitivity set. Sensitivity denotes how failures are treated, and when to alert about state changes:

- **low**: Alerting only happens if all currently active nodes agree that the test has failed, e.g. the service is down completely.
- **default**: Alerting happens if a majority of nodes agree that the test has failed. This is the default behavior and balances out the need for speedy alerting versus the need for fewer false positives.
- **high**: Alerting happens if more than one node sees failures. While more sensitive than the default, it still removes a fair bit of false positives by requiring confirmation of a reported failure by at least one other node.

- **twitchy**: Alerting happens if any node registers a failure. This may be useful for services that have guaranteed service level agreements, but can lead to a lot of false positives.

It should be noted that if you run a setup of Warble with only one, or very few nodes attached, the sensitivity levels may differ very little in terms of when alerting happens, as the definition of quorum changes based on how many active nodes you have at any given time.

3.2 Task Categories

Each task is assigned a task category, which helps you separate tasks into easily recognizable groups and access definitions.

Each task category has a distinct alerting and escalation path, meaning you can assign different teams to different categories, and have alerts go to that team, independent of other task categories. This can be useful for having front-end issues go to a specific team, while back-end issues go to another team.

3.2.1 Task Category Access

Users can be assigned the following access levels to categories, on a per-user basis:

1. Read-only access: The user can read and analyze test results, but cannot edit or remove tasks, nor see the specific payload details (thus, if you add a test with credentials, users with read-only access cannot see the credentials)
2. Read/write access: The user can read, modify, and remove existing tests. They can also add new tests to the category.
3. Admin access: The user can, besides permissions listed above, also modify or remove the category altogether or change its alerting options. This access level should generally be reserved for power users only.

It should be noted that *super users* on the system (such as the account you create at setup) can freely access and modify any aspect of the tasks/categories.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search