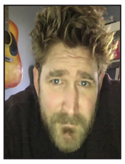
iVoyeur
Graphios

DAVE JOSEPHSEN



Dave Josephsen is the sometime book-authoring developer evangelist at Librato.com. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

Hello again, intrepid reader. It seems like only yesterday that we were talking about implementing spread data in Graphite [1]. In that article I alluded to a tool called Graphios, with which you may not be familiar. More specifically, I was using it to collect metrics data from Nagios and inject it into Graphite, but I was a little short on the details surrounding that particular tool. This month I thought I'd correct that by taking some time to elucidate on Graphios.

Really, there are two very good reasons I want to talk about Graphios. The first is that although Graphios is not very widely used today (compared, to say, PNP4Nagios), it is certainly the easiest way to connect Nagios to systems like StatsD, Graphite, and Librato. Indeed, if you need to emit metric data from Nagios to one of the newer time-series analysis systems like InfluxDB or OpenTSDB, Graphios (via StatsD) is pretty much your only option besides coding up something yourself.

The second reason I want to talk about Graphios is that its creator Shawn Sterling and I recently spent the better part of several months ripping out its Graphite-specific backend and replacing it with a modular framework into which any sort of graphing system can be plugged. And I think you'll agree that tooting one's own horn is an excellent and time-honored reason to talk about anything in general.

As a result, Graphios works as a glue layer between Nagios and any Metrics System that supports the Carbon, StatsD, or Librato API protocols (which is to say, pretty much every metrics system today). As depicted in Figure 1, Graphios uses the `host_perfdata` and `service_perfdata` hooks (defined in your nagios.cfg) to read metric data from your perfdata log, and handles formatting and sending it to systems like Librato, StatsD, and collectd.

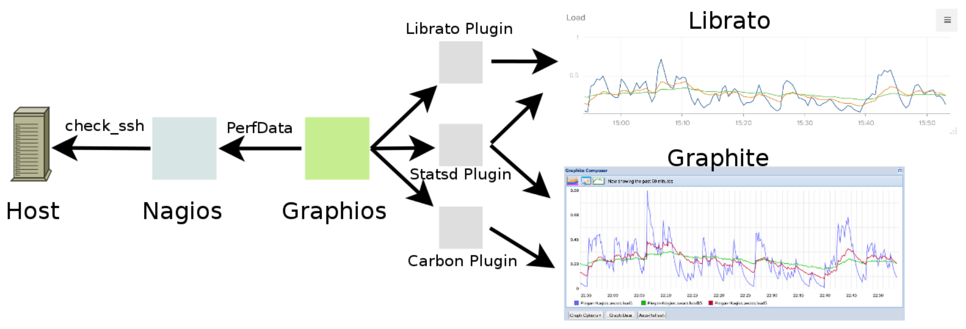


Figure 1: Graphios is a glue layer between Nagios and many different metrics systems.

Installation

Graphios is a Python program, so the easiest way to install it is with pip:

```
pip install graphios
```

It's also pretty easy to install Graphios manually. First, get the most recent version from Git with the following:

```
git clone https://github.com/shawn-sterling/graphios.git
```

Then copy the various files to the appropriate locations:

```
mkdir -p /etc/graphios
cp graphios*.py /usr/local/bin
cp graphios.cfg /etc/graphios/
```

Configuration Requirements

To get Graphios up and running, you'll need to manually configure three things:

- The Nagios config files that deal with host and service checks
- The nagios.cfg file
- The graphios.cfg file

If you use `pip install graphios`, the setup.py script will attempt to detect and automatically add a commented-out configuration to your nagios.cfg. The setup script does a pretty good job of this on all but the most bespoke Nagios setups (simply uncomment and restart Nagios), but given the configuration flexibility of Nagios, it's possible you'll need to manually intervene and modify the nagios.cfg yourself.

What's in a Name?

Nagios is a standalone monolithic system in that it assumes its check-command output will never be exported, that no system but Nagios will ever need to process it. So Nagios services generally have very simple names like PING or LOAD. In Nagios, it should be obvious to the operator what those names refer to because all of the context is inside the Nagios UI.

Graphing systems like Graphite, however, are not monolithic; they're designed to work alongside other monitoring systems and data collectors. Therefore they necessarily assume that all data is sourced externally (everything comes from some other monitoring system), and as a result they use dot-delineated, hierarchical metric names like Nagios.dc4.dbserver12.LOAD. In Graphite, a name like LOAD doesn't tell the operator anything about what system the metric refers to, much less how it was collected.

To be an effective glue layer, Graphios gives you a simple, transparent means to translate Nagios's simple, monolithic service names into context-rich hierarchical names that can be used by systems like Librato and Graphite. Specifically, Graphios can read metric prefixes and suffixes out of your Nagios service and host definitions using custom attributes. For example, a typical Nagios service description, excluding the minutiae normally packed into upper-level templates, looks like this:

```
define service{
  use                    generic-service
  hostname               box1,box2,box3
  service_description    SSH
  check_command          check_ssh }
```

The output of the `check_ssh` plugin looks like this:

```
SSH OK - OpenSSH_5.9p1 Debian-5ubuntu1 (protocol 2.0) |
time=0.009549s;;;0.000000;10.
```

Everything after the pipe is performance data [2]; these are the metrics Graphios exports. In this case, we have a single metric called time, which measures the response time of the SSH port (in this case, the SSH port responded to the check_ssh plugin in 0.009549 seconds). Graphios automatically prefixes the metric name with the hostname, so without doing anything at all, our metric name becomes:

```
box1.time
```

As I've already observed above, box1.time isn't a particularly meaningful metric name, so we can tell Graphios to put some additional context in front of this metric name by inserting a `_graphiteprefix` custom attribute into the service definition like so:

```
define service{
  use                    generic-service
  hostname               box1,box2,box3
  service_description    SSH
  check_command          check_ssh
      _graphiteprefix            nagios.dc1 }
```

Graphios will now prepend this prefix to the metric name, making it:

```
nagios.dc1.box1.time
```

This is a little bit better, but we can insert some additional context about the service between the hostname and the metric name using a `_graphitepostfix` custom attribute in our service configuration like so:

```
define service{
  use                    generic-service
  hostname               box1,box2,box3
  service_description    SSH
  check_command          check_ssh
      _graphiteprefix          nagios.dc1
      _graphitepostfix                 sshd.rt }
```

Graphios will now insert this string between the host and metric name, making it:

```
nagios.dc1.box1.sshd.rt.time
```

Now we have a pretty decent metric name for use with systems like Graphite and StatsD.

Configuring Nagios Perfdata Hooks

Next we need to configure Nagios to export performance data to a log file in a format that Graphios can understand. If you installed Graphios using `pip install graphios`, check the bottom of your nagios.cfg file for a block of configuration that begins:

```
# ###### AUTO-GENERATED GRAPHIOS CONFIGS
```

If you aren't already using Nagios perfdata hooks for something else, that is, if your currently running Nagios configuration *contains* `process_performance_data=0`, then you can simply uncomment this configuration block and restart Nagios.

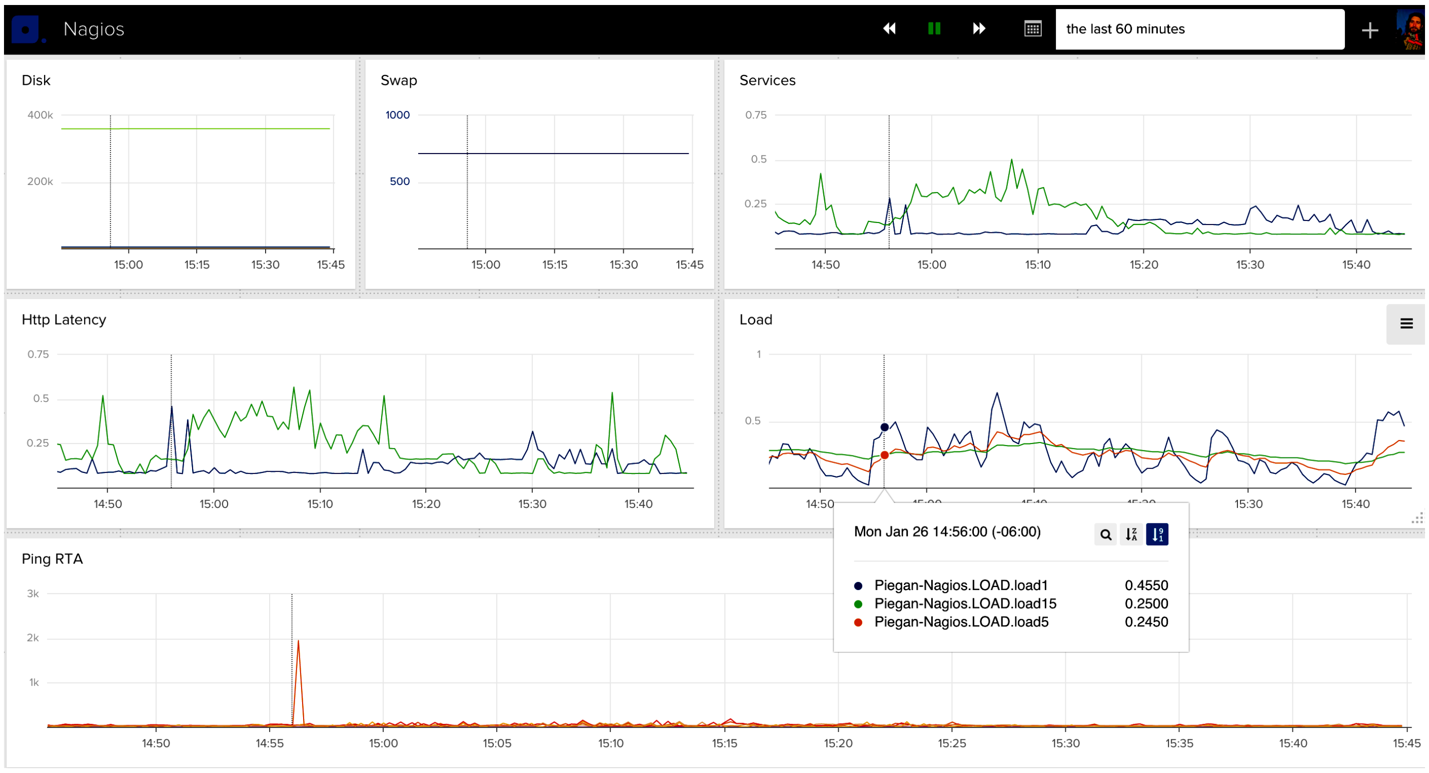
If you're already using Nagios perfdata hooks for something like PNP4Nagios, or one of the other RRDtool-based graphing systems, chances are you can safely run both Graphios and your current tool set at the same time. Refer to the Graphios documentation [2] for instructions on how to set this up. You should also consult the Graphios setup docs if you don't see the auto-generated Graphios config at the bottom of your nagios.cfg, or if you didn't use pip to install.

Once you've configured Nagios to emit performance data, restart the Nagios daemon and verify that it's writing a log file to the Graphios spool directory (named by the service_perfdata_file attribute in your nagios.cfg) with a name like service-perfdata .1418637947. The file should contain lines that look like this:

```
DATATYPE::SERVICEPERFDATA  TIMET::1418637938  HOSTNAME::box1
SERVICEDESC::SSH  SERVICEPERFDATA::time=0.066863s;;;0.000000;10.000000
SERVICECHECKCOMMAND::check_ssh  HOSTSTATE::U  HOSTSTATETYPE::HARD
SERVICESTATE::OK  SERVICESTATETYPE::HARD  GRAPHITEPREFIX::nagios.dc1
     GRAPHITEPOSTFIX::sshd.rta
```

(Finally) Configure Graphios

Graphios installs its config file in /etc/graphios/graphios.cfg by default. This file is very well commented and, by and large, self-explanatory. There is a global configuration section and one section for each backend plugin that Graphios can write to. Plugins are generally enabled by setting their enable line to True and configuring the required attributes for the plugin. Here, for example. is a working configuration for Librato:



Mon Jan 26 14:56:00 (-06:00)
● Piegan-Nagios.LOAD.load1 0.4550
● Piegan-Nagios.LOAD.load15 0.2500
● Piegan-Nagios.LOAD.load5 0.2450

Figure 2: Lovely data, as if from heaven

```
enable_librato=True
librato_email = dave@librato.com
librato_token =
ecb79ff8a82areallylonggreatbigstringtokenything6b8cb77e8b5
librato_whitelist=["load","rta","swap"]
```

The whitelist attribute bears mentioning since, without it, Graphios would emit all performance data measured by Nagios to Librato, which could become expensive. As configured above, only metrics matching the regular expressions “load”, “rta”, and “swap” will be emitted to Librato. Here’s a working configuration for StatsD:

```
enable_statsd=True
statsd_servers = 192.168.1.87:8125
```

You may enable multiple backend plugins (Librato AND StatsD) and even multiple comma-separated instances of the same backend plugin (four different StatsD servers and a carbon server), and Graphios will happily parse out and emit your Nagios Metrics to each backend system in turn. At this point you can run Graphios from the command line and see whether everything works as expected:

```
graphios.py --verbose
```

Now you should start seeing something like what’s found in Figure 2, beautiful metrics data magically appearing in your metrics backend of choice.

Daemonizing Graphios

Graphios ships with init scripts for Debian and RPM-based systems, and these were installed automatically if you ran `pip install graphios` on a compatible system.

So How Does This Work Again?

Although its configuration necessarily borders on complex, Graphios is conceptually a very simple special-purpose log parser. It runs as a daemon, waking up on a configurable interval, checking for new performance data logs exported by Nagios, and processing them.

As I’ve already quite proudly mentioned, Graphios has a modular backend model that allows it to write to multiple metrics systems. When Graphios finds a new performance data file, it parses metrics out of it, computes appropriate metric names for the enabled backend plugins, and then it emits the metrics to each backend metrics system as required.

If you’re running Nagios today, and you’re still trapped in the RRDtool era, you owe it to yourself to install Graphios and experience the future of scalable metrics analysis systems like Graphite, InfluxDB, and OpenTSDB. One of the nicest features of Graphios for me has been its support for running multiple backends in parallel. Graphios makes it painless and simple to spin up and test new metrics systems, or combinations of metrics systems, without interrupting your production metric streams. I hope you find it as useful as I have.

Take it easy.

References

[1] Dave Josephsen, “iVoyeur: Spreading,” *;login:*, vol. 40, no. 1, February 2015 (USENIX): https://www.usenix.org/publications/login/feb15/josephsen.

[2] https://github.com/shawn-sterling/graphios/blob/master/README.md.