graphitesend Documentation

Release 0.2.0

Daniel Lawrence <dannyla@linux.com>

December 08, 2013

Contents

CHAPTER 1

About

Graphitesend is a python library that can be used to easily push data into graphite using python.

1.1 Usage Example

Very basic sending of a metric called metric with a value of 45

```
>>> import graphitesend
>>> graphitesend.init()
>>> graphitesend.send('metric', 45)
>>> graphitesend.send('metric2', 55)
```

The above would send the following metric to graphite over the plaintext (default) protocol on port 2003 (default)

```
system.localhostname.metric  45 epoch-time-stamp
system.localhostname.metric2 55 epoch-time-stamp
```

Cleaning up the interface and using a group of cpu to alter the metric prefix

```
>>> import graphitesend
>>> g = graphitesend.init(group='cpu')
>>> g.send('metric', 45)
>>> g.send('metric2', 55)
```

The above would send the following metric to graphite

```
system.localhostname.cpu.metric 45 epoch-time-stamp
system.localhostname.cpu.metric2 55 epoch-time-stamp
```

CHAPTER 2

Using graphitesend from the commandline

A cli script that allows for anything to send metrics over to graphite (not just python).

The usage is very simple you need to give the command a metric and a value.

```
$ graphitesend name.of.the.metric 666
```

Send more* then 1 metric and value

```
$ graphitesend name.of.the.metric 666
$ graphitesend name.of.the.other_metric 2
```

CHAPTER 3

Example Scripts using graphitesend

The github repo of https://github.com/daniellawrence/graphitesend-examples has lots of examples using graphitesend to grab data from your local linux system.

CHAPTER 4

Installation

Stable releases of graphitesnd are best installed via `pip` or `easy_install`.

We recommend using the latest stable version of graphitesnd; releases are made often to prevent any large gaps in functionality between the latest stable release and the development version.

However, if you want to live on the edge, you can pull down the source code from our Git repository, or fork us on Github.

CHAPTER 5

Documentation

Graphitesend is a python library that can be used to easily push data into graphite using python.

5.1 Usage Example

Very basic sending of a metric called metric with a value of 45

```
>>> import graphitesend
>>> graphitesend.init()
>>> graphitesend.send('metric', 45)
>>> graphitesend.send('metric2', 55)
```

The above would send the following metric to graphite over the plaintext (default) protocol on port 2003 (default)

```
system.localhostname.metric  45 epoch-time-stamp
system.localhostname.metric2 55 epoch-time-stamp
```

Cleaning up the interface and using a group of cpu to alter the metric prefix

```
>>> import graphitesend
>>> g = graphitesend.init(group='cpu')
>>> g.send('metric', 45)
>>> g.send('metric2', 55)
```

The above would send the following metric to graphite

```
system.localhostname.cpu.metric 45 epoch-time-stamp
system.localhostname.cpu.metric2 55 epoch-time-stamp
```

5.1.1 Using graphitesend from the commandline

A cli script that allows for anything to send metrics over to graphite (not just python).

The usage is very simple you need to give the command a metric and a value.

```
$ graphitesend name.of.the.metric 666
```

Send more* then 1 metric and value

```
$ graphitesend name.of.the.metric 666
$ graphitesend name.of.the.other_metric 2
```

5.1.2 Example Scripts using graphitesend

The github repo of https://github.com/daniellawrence/graphitesend-examples has lots of examples using graphitesend to grab data from your local linux system.

5.2 Overview and Tutorial

Welcome to GraphiteSend!

This is a quick dive in at some of the features of graphitesend.

5.2.1 What is GraphiteSend?

As the `README` says:

Graphitesend is a python library that can be used to easily push data into graphite using python.

More specifically, Graphite is:

- A common way for you to push all your metrics that your going to gather in python to your graphite server.

The most common usage of this is to either

- quickly put to gether a new script that is going to metrics into graphite.
- extending an oldscript to standarize how to push metrics into graphite.

5.2.2 Hello, `graphite`

Very basic sending of a metric called `hello world` with the current value of 53.

This will make a connection to the a graphite server called (configurable) and pass the following

```
>>> graphitesend.send('hello world', 45)
systems.ubuntu.hello_world 45.000000 1386490491
```

As you can see `graphitesend` has done a few things for you..

- Added a default `prefix` of "systems." to make sure all your metrics land in the same namespace
- Added the `system_name` as the current hostname after the prefix.
- Fixed the space in the metric name
- validated and converted the value into a float
- Used the current timestamp
- Send all the above to the graphitesend on the plain text protocol, default port 2003

5.2.3 Sending Dicts of data

Instead of sending single metrics to the graphite server you can group them up into a `dict` or `list`.

```
>>> graphitesend.send_dict({'hello world': 45, 'goodbye world': 54})
systems.ubuntu.hello_world 45.000000 1386490491
systems.ubuntu.goodbye_world 54.000000 1386490491
```

As long as you keep the format of `{metric: value}` the data will be sent over to graphite.

```
>>> graphitesend.send_dict(
...  {
...   'hello world': 45,
...   'this world': 54
...   'goodbye world': 54
...  }
... )
```

5.2.4 Sending lists of data

You can do the same as sending dicts however by providing a list.

```
>>> graphitesend.send_list([('hello world', 45), ('goodbye world', 54))
systems.ubuntu.hello_world 45.000000 1386490491
systems.ubuntu.goodbye_world 54.000000 1386490491
```

As long as you keep the format of `metric, value, [timestamp]` the data will be sent over to graphite.

The optional timestamp needs to be provided in unix epoch format.

```
>>> graphitesend.send_list(
...  [
...   ('hello world', 45),
...   ('this world', 54),
...   ('goodbye world', 54, 10000)
...  ]
... )
```

5.3 GraphiteSend

class `graphitesend.`**`GraphiteClient`**(*prefix=None*, *graphite_server=None*, *graphite_port=2003*, *debug=False*, *group=None*, *system_name=None*, *suffix=None*, *lowercase_metric_names=False*, *connect_on_create=True*, *fqdn_squash=False*, *dryrun=False*)

Graphite Client that will setup a TCP connection to graphite.

Parameters

- **prefix** (*Default: "systems."*) – string added to the start of all metrics
- **graphite_server** (*Default: graphite*) – hostname or ip address of graphite server
- **graphite_port** (*Default: 2003*) – TCP port we will connect to
- **debug** (*True or False*) – Toggle debug messages
- **group** – string added to after system_name and before metric name

- **system_name** (*Default: current FDQN*) – FDQN of the system generating the metrics
- **suffix** – string added to the end of all metrics
- **lowercase_metric_names** – Toggle the .lower() of all metric names
- **fqdn_squash** (*True or False*) – Change host.example.com to host_example_com
- **dryrun** (*True or False*) – Toggle if it will really send metrics or just return them

It will then send any metrics that you give it via the .send() or .send_dict().

You can also take advantage of the prefix, group and system_name options that allow you to setup default locations where your whisper files will be kept. eg. (where linuxserver is the name of the localhost)

```
>>> init().prefix
systems.linuxserver.

>>> init(system_name='remote_host').prefix
systems.remote_host.

>>> init(group='cpu').prefix
systems.linuxserver.cpu.

>>> init(prefix='apache').prefix
apache.
```

clean_metric_name(*metric_name*)
Make sure the metric is free of control chars, spaces, tabs, etc.

connect()
Make a TCP connection to the graphite server on port self.port

disconnect()
Close the TCP connection with the graphite server.

send(*metric*, *value*, *timestamp=None*)
Format a single metric/value pair, and send it to the graphite server.

Parameters

- **metric** – name of the metric
- **value** – value of the metric
- **timestmap** – epoch time of the event

```
>>> g = init()
>>> g.send("metric", 54)

>>> g = init()
>>> g.send(metric="metricname", value=73)
```

send_dict(*data*, *timestamp=None*)
Format a dict of metric/values pairs, and send them all to the graphite server.

Parameters

- **data** – key,value pair of metric name and metric value
- **timestmap** – epoch time of the event

```
>>> g = init()
>>> g.send_dict({'metric1': 54, 'metric2': 43, 'metricN': 999})
```

send_list(*data*, *timestamp=None*)
Format a list of set's of (metric, value) pairs, and send them all to the graphite server.

Parameters

- **data** – list of key,value pairs of metric name and metric value
- **timestmap** – epoch time of the event

```
>>> g = init()
>>> g.send_list([('metric1', 54), ('metric2', 43, 1384418995), ('metricN', 999)])
```

graphitesend.**cli**()
Allow the module to be called from the cli.

graphitesend.**init**(*init_type='plaintext_tcp'*, **args*, ***kwargs*)
Create the module instance of the GraphiteClient.

graphitesend.**reset**()
disconnect from the graphite server and destroy the module instance.

graphitesend.**send**(**args*, ***kwargs*)
Make sure that we have an instance of the GraphiteClient. Then send the metrics to the graphite server. User consumable method.

graphitesend.**send_dict**(**args*, ***kwargs*)
Make sure that we have an instance of the GraphiteClient. Then send the metrics to the graphite server. User consumable method.

graphitesend.**send_list**(**args*, ***kwargs*)
Make sure that we have an instance of the GraphiteClient. Then send the metrics to the graphite server. User consumable method.

5.4 Bugs/ticket tracker

To file new bugs or search existing ones, you may visit GraphiteSends's Github Issues page. This does require a (free, easy to set up) Github account.

Python Module Index