pyformance Documentation

Release 0.3.4

Omer Getrel

Oct 04, 2017

Contents

1 Manual **3**
- 1.1 Installation . . . 3
- 1.2 Usage . . . 3
- 1.3 API-Reference . . . 5

2 Indices and tables **11**

3 Contributors **13**

Python Module Index **15**

This package is a Python port of the core portion of a Java Metrics library by Coda Hale (http://metrics.codahale.com/), with inspiration by YUNOMI - Y U NO MEASURE IT (https://github.com/richzeng/yunomi).

PyFormance is a toolset for performance measurement and statistics, with a signaling mechanism that allows to issue events in cases of unexpected behavior.

The following metrics classes are available.

Gauge

A gauge metric is an instantaneous reading of a particular value.

Counter

Simple interface to increment and decrement a value. For example, this can be used to measure the total number of jobs sent to the queue, as well as the pending (not yet complete) number of jobs in the queue. Simply increment the counter when an operation starts and decrement it when it completes.

Meter

Measures the rate of events over time. Useful to track how often a certain portion of your application gets requests so you can set resources accordingly. Tracks the mean rate (the overall rate since the meter was reset) and the rate statistically significant regarding only events that have happened in the last 1, 5, and 15 minutes (Exponentally weighted moving average).

Histogram

Measures the statistical distribution of values in a data stream. Keeps track of minimum, maximum, mean, standard deviatoin, etc. It also measures median, 75th, 90th, 95th, 98th, 99th, and 99.9th percentiles. An example use case would be for looking at the number of daily logins for 99 percent of your days, ignoring outliers.

Timer

A useful combination of the Meter and the Histogram letting you measure the rate that a portion of code is called and a distribution of the duration of an operation. You can see, for example, how often your code hits the database and how long those operations tend to take.

CHAPTER 1

Manual

Installation

The package is available on the Python packaging index (pypi).

```
pip install pyformance
```

The source repository is on GitHub: https://github.com/omergertel/pyformance

Usage

Reporters

A simple call which will periodically push out your metrics to [Hosted Graphite](https://www.hostedgraphite.com/) using the HTTP Interface.

```
registry = MetricsRegistry()
#Push metrics contained in registry to hosted graphite every 10s for the account␣
↪specified by Key
    reporter = HostedGraphiteReporter(registry, 10, "XXXXXXXX-XXX-XXXXX-XXXX-
↪XXXXXXXXXX")
# Some time later we increment metrics
histogram = registry.histogram("test.histogram")
histogram.add(0)
    histogram.add(10)
    histogram.add(25)
```

Advanced

Decorators

The simplest and easiest way to use the PyFormance library.

Counter

You can use the 'count_calls' decorator to count the number of times a function is called.

```
>>> from pyformance import counter, count_calls
>>> @count_calls
... def test():
...     pass
...
>>> for i in range(10):
...     test()
...
>>> print counter("test_calls").get_count()
10
```

Timer

You can use the 'time_calls' decorator to time the execution of a function and get distributtion data from it.

```
>>> import time
>>> from pyformance import timer, time_calls
>>> @time_calls
... def test():
...     time.sleep(0.1)
...
>>> for i in range(10):
...     test()
...
>>> print timer("test_calls").get_mean()
0.100820207596
```

With statement

You can also use a timer using the with statement

```
>>> from time import sleep
>>> from pyformance import timer
>>> with timer("test").time():
...    sleep(0.1)
>>> print timer("test_calls").get_mean()
0.10114598274230957
```

Regex Grouping

Useful when working with APIs. A RegexRegistry allows to group API calls and measure from a single location instead of having to define different timers in different places.

```
>>> from pyformance.registry import RegexRegistry
>>> reg = RegexRegistry(pattern='^/api/(?P<model>)/\d+/(?P<verb>)?$')
>>> def rest_api_request(path):
...     with reg.timer(path).time():
...         # do stuff
>>> print reg.dump_metrics()
```

API-Reference

Registry

class pyformance.registry.**MetricsRegistry**(*clock=<module 'time' from '/home/docs/checkouts/readthedocs.org/user_builds/pyformance/envs/latest/dynload/time.so'>*)

A single interface used to gather metrics on a service. It keeps track of all the relevant Counters, Meters, Histograms, and Timers. It does not have a reference back to its service. The service would create a L{MetricsRegistry} to manage all of its metrics tools.

add(*key*, *metric*)

Use this method to manually add custom metric instances to the registry which are not created with their constructor's default arguments, e.g. Histograms with a different size.

Parameters

- **key** (*C{str}*) – name of the metric
- **metric** – instance of Histogram, Meter, Gauge, Timer or Counter

counter(*key*)

Gets a counter based on a key, creates a new one if it does not exist.

Parameters **key** (*C{str}*) – name of the metric

Returns L{Counter}

dump_metrics()

Formats all of the metrics and returns them as a dict.

Returns C{list} of C{dict} of metrics

get_metrics(*key*)

Gets all the metrics for a specified key.

Parameters **key** (*C{str}*) – name of the metric

Returns C{dict}

histogram(*key*)

Gets a histogram based on a key, creates a new one if it does not exist.

Parameters **key** (*C{str}*) – name of the metric

Returns L{Histogram}

meter(*key*)

Gets a meter based on a key, creates a new one if it does not exist.

Parameters **key** (*C{str}*) – name of the metric

Returns L{Meter}

timer(*key*)

Gets a timer based on a key, creates a new one if it does not exist.

Parameters **key** (*C{str}*) – name of the metric

Returns L{Timer}

class pyformance.registry.**RegexRegistry**(*pattern=None*, *clock=<module 'time' from '/home/docs/checkouts/readthedocs.org/user_builds/pyformance/envs/latest/lib/dynload/time.so'>*)

A single interface used to gather metrics on a service. This class uses a regex to combine measures that match a pattern. For example, if you have a REST API, instead of defining a timer for each method, you can use a regex to capture all API calls and group them. A pattern like '^/api/(?P<model>)/d+/(?P<verb>)?$' will group and measure the following:

/api/users/1 -> users /api/users/1/edit -> users/edit /api/users/2/edit -> users/edit

pyformance.registry.**count_calls**(*fn*)

Decorator to track the number of times a function is called.

Parameters **fn** (*C{func}*) – the function to be decorated

Returns the decorated function

Return type C{func}

pyformance.registry.**hist_calls**(*fn*)

Decorator to check the distribution of return values of a function.

Parameters **fn** (*C{func}*) – the function to be decorated

Returns the decorated function

Return type C{func}

pyformance.registry.**meter_calls**(*fn*)

Decorator to the rate at which a function is called.

Parameters **fn** (*C{func}*) – the function to be decorated

Returns the decorated function

Return type C{func}

pyformance.registry.**time_calls**(*fn*)

Decorator to time the execution of the function.

Parameters **fn** (*C{func}*) – the function to be decorated

Returns the decorated function

Return type C{func}

Meters

class pyformance.meters.gauge.**CallbackGauge**(*callback*)

A Gauge reading for a given callback

get_value()

returns the result of callback which is executed each time

class pyformance.meters.gauge.**Gauge**

A base class for reading of a particular.

For example, to instrument a queue depth:

class QueueLengthGaguge(Gauge):

def __init__(self, queue): super(QueueGaguge, self).__init__() self.queue = queue

def get_value(self): return len(self.queue)

get_value()
A subclass of Gauge should implement this method

class pyformance.meters.gauge.**SimpleGauge**(*value=nan*)
A gauge which holds values with simple getter- and setter-interface

get_value()
getter returns current value

set_value(*value*)
setter changes current value

class pyformance.meters.counter.**Counter**
An incrementing and decrementing metric

clear()
reset counter to 0

dec(*val=1*)
decrement counter by val (default is 1)

get_count()
return current value of counter

inc(*val=1*)
increment counter by val (default is 1)

class pyformance.meters.timer.**Timer**(*threshold=None*, *size=1028*, *alpha=0.015*, *clock=<module 'time' from '/home/docs/checkouts/readthedocs.org/user_builds/pyformance/envs/latest/lib/pytho dynload/time.so'>*, *sink=None*, *sample=None*)
A timer metric which aggregates timing durations and provides duration statistics, plus throughput statistics via Meter and Histogram.

clear()
clear internal histogram and meter

get_count()
get count from internal histogram

get_fifteen_minute_rate()
get 15 rate from internal meter

get_five_minute_rate()
get 5 minute rate from internal meter

get_max()
get max from internal histogram

get_mean()
get mean from internal histogram

get_mean_rate()
get mean rate from internal meter

get_min()
get min from internal histogram

get_one_minute_rate()
get 1 minut rate from internal meter

get_snapshot()
get snapshot from internal histogram

get_stddev()
get stddev from internal histogram

get_sum()
get sum from internal histogram

get_var()
get var from internal histogram

time(**args*, ***kwargs*)
Parameters will be sent to signal, if fired. Returns a timer context instance which can be used from a with-statement. Without with-statement you have to call the stop method on the context

class pyformance.meters.histogram.**Histogram**(*size=1028*, *alpha=0.015*, *clock=<module 'time' from '/home/docs/checkouts/readthedocs.org/user_builds/pyformance/envs/lat dynload/time.so'>*, *sample=None*)
A metric which calculates the distribution of a value.

add(*value*)
Add value to histogram

clear()
reset histogram to initial state

get_count()
get current value of counter

get_max()
get current maximum

get_mean()
get current mean

get_min()
get current minimum

get_snapshot()
get snapshot instance which holds the percentiles

get_stddev()
get current standard deviation

get_sum()
get current sum

get_var()
get current variance

class pyformance.stats.snapshot.**Snapshot**(*values*)
This class is used by the histogram meter

get_75th_percentile()
get current 75th percentile

get_95th_percentile()
get current 95th percentile

get_999th_percentile()
get current 999th percentile

get_99th_percentile()
get current 99th percentile

get_median()
get current median

get_percentile(*percentile*)
get custom percentile

Parameters **percentile** – float value between 0 and 1

get_size()
get current size

Reporters

class pyformance.reporters.console_reporter.**ConsoleReporter**(*registry=None*, *reporting_interval=30*, *stream=<open file '<stderr>', mode 'w'>*, *clock=None*)
Show metrics in a human readable form. This is useful for debugging if you want to read the current state on the console.

class pyformance.reporters.carbon_reporter.**CarbonReporter**(*registry=None*, *reporting_interval=5*, *prefix=''*, *server='0.0.0.0'*, *port=2003*, *socket_factory=<class 'socket._socketobject'>*, *clock=None*, *pickle_protocol=False*)
Carbon is the network daemon to collect metrics for Graphite

class pyformance.reporters.carbon_reporter.**UdpCarbonReporter**(*registry=None*, *reporting_interval=5*, *prefix=''*, *server='0.0.0.0'*, *port=2003*, *socket_factory=<class 'socket._socketobject'>*, *clock=None*, *pickle_protocol=False*)
The default CarbonReporter uses TCP. This sub-class uses UDP instead which might be unreliable but it is faster

class pyformance.reporters.newrelic_reporter.**NewRelicReporter**(*license_key*, *registry=None*, *name='build-6074678-project-22200-pyformance'*, *reporting_interval=5*, *prefix=''*, *clock=None*)
Reporter for new relic

agent_data
Return the agent data section of the NewRelic Platform data payload

Return type dict

class pyformance.reporters.influx.**InfluxReporter**(*registry=None*, *reporting_interval=5*, *prefix=''*, *database='metrics'*, *server='127.0.0.1'*, *username=None*, *password=None*, *port=8086*, *protocol='http'*, *autocreate_database=False*, *clock=None*)

InfluxDB reporter using native http api (based on https://influxdb.com/docs/v1.1/guides/writing_data.html)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Contributors

Special thanks to the help of these people:

- Henning Schroeder (hajs)

Python Module Index

p

pyformance.meters.counter, 7
pyformance.meters.gauge, 6
pyformance.meters.histogram, 8
pyformance.meters.timer, 7
pyformance.registry, 5
pyformance.reporters.carbon_reporter, 9
pyformance.reporters.console_reporter, 9
pyformance.reporters.influx, 10
pyformance.reporters.newrelic_reporter, 9
pyformance.stats.snapshot, 8

Index

A

add() (pyformance.meters.histogram.Histogram method), 8
add() (pyformance.registry.MetricsRegistry method), 5
agent_data (pyformance.reporters.newrelic_reporter.NewRelicReporter attribute), 9

C

CallbackGauge (class in pyformance.meters.gauge), 6
CarbonReporter (class in pyformance.reporters.carbon_reporter), 9
clear() (pyformance.meters.counter.Counter method), 7
clear() (pyformance.meters.histogram.Histogram method), 8
clear() (pyformance.meters.timer.Timer method), 7
ConsoleReporter (class in pyformance.reporters.console_reporter), 9
count_calls() (in module pyformance.registry), 6
Counter (class in pyformance.meters.counter), 7
counter() (pyformance.registry.MetricsRegistry method), 5

D

dec() (pyformance.meters.counter.Counter method), 7
dump_metrics() (pyformance.registry.MetricsRegistry method), 5

G

Gauge (class in pyformance.meters.gauge), 6
get_75th_percentile() (pyformance.stats.snapshot.Snapshot method), 8
get_95th_percentile() (pyformance.stats.snapshot.Snapshot method), 8
get_999th_percentile() (pyformance.stats.snapshot.Snapshot method), 8
get_99th_percentile() (pyformance.stats.snapshot.Snapshot method), 8
get_count() (pyformance.meters.counter.Counter method), 7
get_count() (pyformance.meters.histogram.Histogram method), 8
get_count() (pyformance.meters.timer.Timer method), 7
get_fifteen_minute_rate() (pyformance.meters.timer.Timer method), 7
get_five_minute_rate() (pyformance.meters.timer.Timer method), 7
get_max() (pyformance.meters.histogram.Histogram method), 8
get_max() (pyformance.meters.timer.Timer method), 7
get_mean() (pyformance.meters.histogram.Histogram method), 8
get_mean() (pyformance.meters.timer.Timer method), 7
get_mean_rate() (pyformance.meters.timer.Timer method), 7
get_median() (pyformance.stats.snapshot.Snapshot method), 9
get_metrics() (pyformance.registry.MetricsRegistry method), 5
get_min() (pyformance.meters.histogram.Histogram method), 8
get_min() (pyformance.meters.timer.Timer method), 7
get_one_minute_rate() (pyformance.meters.timer.Timer method), 7
get_percentile() (pyformance.stats.snapshot.Snapshot method), 9
get_size() (pyformance.stats.snapshot.Snapshot method), 9
get_snapshot() (pyformance.meters.histogram.Histogram method), 8
get_snapshot() (pyformance.meters.timer.Timer method), 7
get_stddev() (pyformance.meters.histogram.Histogram method), 8
get_stddev() (pyformance.meters.timer.Timer method), 7

get_sum() (pyformance.meters.histogram.Histogram method), 8
get_sum() (pyformance.meters.timer.Timer method), 8
get_value() (pyformance.meters.gauge.CallbackGauge method), 6
get_value() (pyformance.meters.gauge.Gauge method), 6
get_value() (pyformance.meters.gauge.SimpleGauge method), 7
get_var() (pyformance.meters.histogram.Histogram method), 8
get_var() (pyformance.meters.timer.Timer method), 8

H

hist_calls() (in module pyformance.registry), 6
Histogram (class in pyformance.meters.histogram), 8
histogram() (pyformance.registry.MetricsRegistry method), 5

I

inc() (pyformance.meters.counter.Counter method), 7
InfluxReporter (class in pyformance.reporters.influx), 10

M

meter() (pyformance.registry.MetricsRegistry method), 5
meter_calls() (in module pyformance.registry), 6
MetricsRegistry (class in pyformance.registry), 5

N

NewRelicReporter (class in pyformance.reporters.newrelic_reporter), 9

P

pyformance.meters.counter (module), 7
pyformance.meters.gauge (module), 6
pyformance.meters.histogram (module), 8
pyformance.meters.timer (module), 7
pyformance.registry (module), 5
pyformance.reporters.carbon_reporter (module), 9
pyformance.reporters.console_reporter (module), 9
pyformance.reporters.influx (module), 10
pyformance.reporters.newrelic_reporter (module), 9
pyformance.stats.snapshot (module), 8

R

RegexRegistry (class in pyformance.registry), 5

S

set_value() (pyformance.meters.gauge.SimpleGauge method), 7
SimpleGauge (class in pyformance.meters.gauge), 7
Snapshot (class in pyformance.stats.snapshot), 8

T

time() (pyformance.meters.timer.Timer method), 8
time_calls() (in module pyformance.registry), 6
Timer (class in pyformance.meters.timer), 7
timer() (pyformance.registry.MetricsRegistry method), 5

U

UdpCarbonReporter (class in pyformance.reporters.carbon_reporter), 9