metrology Documentation

Release 0.8.0

Timothee Peignier

August 25, 2013

CONTENTS

A library to easily measure what's going on in your python.

Metrology allows you to add instruments to your python code and hook them to external reporting tools like Graphite so as to better understand what's going on in your running python program.

You can report bugs and discuss features on the issues page.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

Either check out Metrology from GitHub or to pull a release off PyPI

```
pip install metrology
```

1.2 Instruments

1.2.1 Gauges

class metrology.instruments.gauge.**Gauge**
A gauge is an instantaneous measurement of a value

```
class JobGauge(metrology.instruments.Gauge):
    def value(self):
        return len(queue)

gauge = Metrology.gauge('pending-jobs', JobGauge())
```

class metrology.instruments.gauge.**PercentGauge**
A percent gauge is a ratio gauge where the result is normalized to a value between 0 and 100.

class metrology.instruments.gauge.**RatioGauge**
A ratio gauge is a simple way to create a gauge which is the ratio between two numbers

1.2.2 Counters

class metrology.instruments.counter.**Counter**
A counter is like a gauge, but you can increment or decrement its value

```
counter = Metrology.counter('pending-jobs')
counter.increment()
counter.decrement()
counter.count
```

count
Return the current value of the counter.

decrement(*value=1*)
Decrement the counter. By default it will decrement by 1.

Parameters **value** – value to decrement the counter.

increment(*value=1*)
Increment the counter. By default it will increment by 1.

Parameters **value** – value to increment the counter.

1.2.3 Derive

class metrology.instruments.derive.**Derive**(*average_class=<class 'metrology.stats.ewma.EWMA'>*)
A derive is like a meter but accepts an absolute counter as input.

derive = Metrology.derive('network.io') derive.mark() derive.count

mark(*value=1*)
Record an event with the derive.

Parameters **value** – counter value to record

1.2.4 Meters

class metrology.instruments.meter.**Meter**(*average_class=<class 'metrology.stats.ewma.EWMA'>*)
A meter measures the rate of events over time (e.g., "requests per second"). In addition to the mean rate, you can also track 1, 5 and 15 minutes moving averages

```
meter = Metrology.meter('requests')
meter.mark()
meter.count
```

count
Returns the total number of events that have been recorded.

fifteen_minute_rate
Returns the fifteen-minute average rate.

five_minute_rate
Returns the five-minute average rate.

mark(**args*, ***kwargs*)
Record an event with the meter. By default it will record one event.

Parameters **value** – number of event to record

mean_rate
Returns the mean rate of the events since the start of the process.

one_minute_rate
Returns the one-minute average rate.

1.2.5 Histograms

class metrology.instruments.histogram.**Histogram**(*sample*)
A histogram measures the statistical distribution of values in a stream of data. In addition to minimum, maximum, mean, it also measures median, 75th, 90th, 95th, 98th, 99th, and 99.9th percentiles

```
histogram = Metrology.histogram('response-sizes')
histogram.update(len(response.content))
```

Metrology provides two types of histograms: uniform and exponentially decaying.

count
Return number of values.

max
Returns the maximun value.

mean
Returns the mean value.

min
Returns the minimum value.

stddev
Returns the standard deviation.

variance
Returns variance

class metrology.instruments.histogram.**HistogramExponentiallyDecaying**
A exponentially decaying histogram produces quantiles which are representative of approximately the last five minutes of data. Unlike the uniform histogram, a biased histogram represents recent data, allowing you to know very quickly if the distribution of the data has changed.

class metrology.instruments.histogram.**HistogramUniform**
A uniform histogram produces quantiles which are valid for the entirely of the histogram's lifetime. It will return a median value, for example, which is the median of all the values the histogram has ever been updated with.

Use a uniform histogram when you're interested in long-term measurements. Don't use one where you'd want to know if the distribution of the underlying data stream has changed recently.

1.2.6 Timers and utilization timers

class metrology.instruments.timer.**Timer**(*histogram=<class 'metrology.instruments.histogram.HistogramExponentiallyDecaying'>*)
A timer measures both the rate that a particular piece of code is called and the distribution of its duration

```
timer = Metrology.timer('responses')
with timer:
    do_something()
```

count
Returns the number of measurements that have been made.

fifteen_minute_rate
Returns the fifteen-minute average rate.

five_minute_rate
Returns the five-minute average rate.

max
Returns the maximum amount of time spent in the operation.

mean
Returns the mean time spent in the operation.

mean_rate
Returns the mean rate of the events since the start of the process.

min
Returns the minimum amount of time spent in the operation.

one_minute_rate
Returns the one-minute average rate.

stddev
Returns the standard deviation of the mean spent in the operation.

update(*duration*)
Records the duration of an operation.

class metrology.instruments.timer.**UtilizationTimer**(*histogram=<class 'metrology.instruments.histogram.HistogramExponentiallyDecaying'>*)
A specialized timer that calculates the percentage of wall-clock time that was spent

```
utimer = Metrology.utilization_timer('responses')
with utimer:
  do_something()
```

count
Returns the number of measurements that have been made.

fifteen_minute_rate
Returns the fifteen-minute average rate.

fifteen_minute_utilization
Returns the fifteen-minute average utilization as a percentage.

five_minute_rate
Returns the five-minute average rate.

five_minute_utilization
Returns the five-minute average utilization as a percentage.

max
Returns the maximum amount of time spent in the operation.

mean
Returns the mean time spent in the operation.

mean_rate
Returns the mean rate of the events since the start of the process.

mean_utilization
Returns the mean (average) utilization as a percentage since the process started.

min
Returns the minimum amount of time spent in the operation.

one_minute_rate
Returns the one-minute average rate.

one_minute_utilization
Returns the one-minute average utilization as a percentage.

stddev
Returns the standard deviation of the mean spent in the operation.

1.2.7 Health Checks

class metrology.instruments.healthcheck.**HealthCheck**

A health check is a small self-test to verify that a specific component or responsibility is performing correctly

```
class DatabaseHealthCheck(metrology.healthcheck.HealthCheck):
    def __init__(self, database):
        self.database = database

    def check(self):
        if database.ping():
            return True
        return False

health_check = Metrology.health_check('database', DatabaseHealthCheck(database))
health_check.check()
```

check()

Returns True if what is being checked is healthy

1.2.8 Profilers

class metrology.instruments.profiler.**Profiler**(*frequency=None*, *histogram=<class 'metrology.instruments.histogram.HistogramExponentiallyDecaying'>*)

A profiler measures the distribution of the duration passed in a every part of the code

```
profiler = Metrology.profiler('slow-code')
with profiler:
    run_slow_code()
```

> **Warning:** This instrument does not yet work on Windows, and it doesn't run on Python 3

update(*key*, *duration*)

Records the duration of a call.

1.3 Reporters

1.3.1 Graphite

class metrology.reporter.graphite.**GraphiteReporter**(*host*, *port*, ***options*)

A graphite reporter that send metrics to graphite

```
reporter = GraphiteReporter('graphite.local', 2003)
reporter.start()
```

Parameters

- **host** – hostname of graphite
- **port** – port of graphite
- **interval** – time between each reporting
- **prefix** – metrics name prefix

1.3.2 Logging

class metrology.reporter.logger.**LoggerReporter**(*logger=<module 'logging' from '/usr/lib/python2.7/logging/__init__.pyc'>*, *level=20*, ***options*)

A logging reporter that write metrics to a logger

```
reporter = LoggerReporter(level=logging.DEBUG, interval=10)
reporter.start()
```

Parameters

- **logger** – logger to use
- **level** – logger level
- **interval** – time between each reporting
- **prefix** – metrics name prefix

1.3.3 Librato

class metrology.reporter.librato.**LibratoReporter**(*email*, *token*, ***options*)

A librato metrics reporter that send metrics to librato

```
reporter = LibratoReporter("<email>", "<token>", source="front.local")
reporter.start()
```

Parameters

- **email** – your librato email
- **token** – your librato api token
- **source** – source of the metric
- **interval** – time between each reporting
- **prefix** – metrics name prefix
- **filters** – allow given keys to be send
- **excludes** – exclude given keys to be send

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

m