timeexecution Documentation

Release

Niels Lensink

May 16, 2018

Contents

Contents:

CHAPTER 1

Installation

At the command line:

```
$ pip install timeexecution
```

CHAPTER 2

Usage

To use this package you decorate the functions you want to time its execution. Every wrapped function will create a metric consisting of 3 default values:

- *name* - The name of the series the metric will be stored in
- *value* - The time it took in ms for the wrapped function to complete
- *hostname* - The hostname of the machine the code is running on

See the following example

```
from time_execution import settings, time_execution
from time_execution.backends.influxdb import InfluxBackend
from time_execution.backends.elasticsearch import ElasticsearchBackend

# Setup the desired backend
influx = InfluxBackend(host='influx', database='metrics', use_udp=False)
elasticsearch = ElasticsearchBackend('elasticsearch', index='metrics')

# Configure the time_execution decorator
settings.configure(backends=[influx, elasticsearch])

# Wrap the methods where u want the metrics
@time_execution
def hello():
    return 'World'

# Now when we call hello() and we will get metrics in our backends
hello()
```

This will result in an entry in the influxdb

```
[
    {
        "name": "__main__.hello",
        "columns": [
```

(continues on next page)

(continued from previous page)

```
            "time",
            "sequence_number",
            "value",
            "hostname",
        ],
        "points": [
            [
                1449739813939,
                1111950001,
                312,
                "machine.name",
            ]
        ]
    }
]
```

And the following in Elasticsearch

```
[
    {
        "_index": "metrics-2016.01.28",
        "_type": "metric",
        "_id": "AVKIp9DpnPWamvqEzFB3",
        "_score": null,
        "_source": {
            "timestamp": "2016-01-28T14:34:05.416968",
            "hostname": "dfaa4928109f",
            "name": "__main__.hello",
            "value": 312
        },
        "sort": [
            1453991645416
        ]
    }
]
```

It's also possible to decorate coroutines or awaitables in Python >=3.5.

For example:

```
import asyncio
from time_execution import time_execution_async

# ... Setup the desired backend(s) as described above ...

# Wrap the methods where you want the metrics
@time_execution_async
async def hello():
    await asyncio.sleep(1)
    return 'World'

# Now when we schedule hello() we will get metrics in our backends
loop = asyncio.get_event_loop()
loop.run_until_complete(hello())
```

2.1 Hooks

time_execution supports hooks where you can change the metric before its being sent to the backend.

With a hook you can add additional and change existing fields. This can be useful for cases where you would like to add a column to the metric based on the response of the wrapped function.

A hook will always get 3 arguments:

- *response* - The returned value of the wrapped function
- *exception* - The raised exception of the wrapped function
- *metric* - A dict containing the data to be send to the backend
- *func_args* - Original args received by the wrapped function.
- *func_kwargs* - Original kwargs received by the wrapped function.

From within a hook you can change the *name* if you want the metrics to be split into multiple series.

See the following example how to setup hooks.

```
# Now lets create a hook
def my_hook(response, exception, metric, func_args, func_kwargs):
    status_code = getattr(response, 'status_code', None)
    if status_code:
        return dict(
            name='{}.{}'.format(metric['name'], status_code),
            extra_field='foo bar'
        )

# Configure the time_execution decorator, but now with hooks
settings.configure(backends=[backend], hooks=[my_hook])
```

2.2 Manually sending metrics

You can also send any metric you have manually to the backend. These will not add the default values and will not hit the hooks.

See the following example.

```
loadavg = os.getloadavg()
write_metric('cpu.load.1m', value=loadavg[0])
write_metric('cpu.load.5m', value=loadavg[1])
write_metric('cpu.load.15m', value=loadavg[2])
```

2.3 Custom Backend

Writing a custom backend is very simple, all you need to do is create a class with a *write* method. It is not required to extend *BaseMetricsBackend* but in order to easily upgrade I recommend u do.

```
from time_execution.backends.base import BaseMetricsBackend
```

(continues on next page)

(continued from previous page)

```
class MetricsPrinter(BaseMetricsBackend):
    def write(self, name, **data):
        print(name, data)
```

CHAPTER 3

API

Contents:

3.1 time_execution

3.1.1 time_execution.backends

time_execution.backends.base module

Base metrics backend

class time_execution.backends.base.**BaseMetricsBackend**

Bases: object

write(*name*, ***data*)

time_execution.backends.elasticsearch module

class time_execution.backends.elasticsearch.**ElasticsearchBackend**(*hosts=None*, *index='metrics'*, *doc_type='metric'*, *index_pattern='{index}-{date:%Y.%m.%d}'*, **args*, ***kwargs*)

Bases: *time_execution.backends.base.BaseMetricsBackend*

bulk_write(*metrics*)

Write multiple metrics to elasticsearch in one request

Parameters **metrics** (*list*) – data with mappings to send to elasticsearch

get_index()

write(*name*, ***data*)
Write the metric to elasticsearch

Parameters

- **name** (*str*) – The name of the metric to write
- **data** (*dict*) – Additional data to store with the metric

time_execution.backends.influxdb module

class time_execution.backends.influxdb.**InfluxBackend**(***kwargs*)
Bases: *time_execution.backends.base.BaseMetricsBackend*

write(*name*, ***data*)

3.1.2 time_execution.decorator module

Time Execution decorator

class time_execution.decorator.**time_execution**(*func=None*, ***params*)
Bases: fqn_decorators.decorators.Decorator

after()
Allow performing an action after the function is called.

before()
Allow performing an action before the function is called.

get_exception()
Retrieve the exception

time_execution.decorator.**write_metric**(*name*, ***metric*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Settings

Package configuration is done through using package settings:

```
from time_execution import settings
settings.configure()
```

Parameters accepted by *configure* method are described bellow.

5.1 backends

Optional parameter, equals to empty list by default, accepts a list of *time_execution.backends* instances.

5.2 hooks

Optional parameter, equals to empty list by default, accepts the list of callable, see *Hooks*.

5.3 duration_field

Optional parameter, equals to *"value"* by default.

5.4 origin

Optional parameter, equals to *None* by default. If specified, then sent metrics will have *"origin"* attribute, which can be used to identify origin of the metric, when *time_execution* package is used in multiple applications.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index