kaneda

Release 0.4

May 25, 2016

Contents

Kaneda is a Python library that allows to get, report and store events and metrics of your applications. It provides a several builtin Metrics methods in order to store any amount of data that you want to then analyze it or for performance studies

Example:

```
from kaneda import Metrics
metrics = Metrics(...)
metrics.event('welcome', 'Kaneda is cool')
metrics.gauge('answer_of_life', 42)
```

CHAPTER 1

Usage

You need to install *Kaneda* package:

```
pip install git+https://gitlab.apsl.net/apsl/kaneda.git
```

1.1 Backend reporting

You need a backend in order to keep data in a persistent storage. You can use builtin Backends or define your custom backend subclassing the `BaseBackend` class.

The following example it shows how to send metrics with Elasticsearch as a backend:

```
from kaneda.backend import ElasticsearchBackend
from kaneda import Metrics

backend = ElasticsearchBackend(index_name='myindex', app_name='myapp', host='localhost',
                               port=9200, user='kaneda', password='kaneda')
metrics = Metrics(backend=backend)
metrics.gauge('answer_of_life', 42)
```

A backend class can also be instantiated passing a previously defined connection client. This is specially useful when you want to use a tuned connection:

```
client = Elasticsearch(['localhost'], port=9200, http_auth=('kaneda', 'kaneda'), timeout=0.3)
backend = ElasticsearchBackend(index_name='myindex', app_name='myapp', client=client)
```

1.2 Asynchronous reporting

Depending the selection of the backend the process of reporting metrics could be "slow" if the response time of your application is critical (e.g: a website). Furthermore if your application doesn't need the see the reported metrics in real time you probably have to consider to using asynchronous reporting. With this system you are allowed to send a metric report in background without adding too much overhead.

To use this system you need to install a queue system and use one of the builtin Kaneda *Queues* classes. To setup Kaneda in async mode follow these steps.

1. Install and configure your queue system (e.g: *RQ*).

```
pip install rq
```

2. Setup your backend configuration in new file named `kanedasettings.py`.

```
BACKEND = 'kaneda.backends.ElasticsearchBackend'
ELASTIC_INDEX_NAME = 'myindex'
ELASTIC_APP_NAME = 'myapp'
ELASTIC_HOST = 'localhost'
ELASTIC_PORT = 9200
ELASTIC_USER = 'kaneda'
ELASTIC_PASSWORD = 'kaneda'
```

3. Run the worker

```
rqworker
```

Now you can use Kaneda with the same *Metrics* API:

```
from kaneda.queues import RQQueue
from kaneda import Metrics

queue = RQQueue(redis_url='redis://localhost:6379/0')
metrics = Metrics(queue=queue)
metrics.gauge('answer_of_life', 42)
```

As in the backend example it can be used passing a queue client:

```
q = Queue(queue_name, connection=Redis())
queue = RQQueue(queue=q)
```

Also you are able to specify a Redis connection url (or a broker url if you use *Celery*). Notice this allows you to run the worker on a different server.

CHAPTER 2

Metrics

class kaneda.base.**Metrics**(*backend=None*, *queue=None*)

Metrics reporting class

Parameters

- **backend** – instance of kaneda.backends. It is the responsible to store the reported data.
- **queue** – instance of kaneda.queues. It is the responsible to store the reported data asynchronously.

If none of the parameters are passed it tries get the backend from kaneda settings file.

custom(*name*, *metric*, *value*, *tags=None*, *id_=None*)

Send a custom metric report.

```
>>> metrics.custom('hotel.response_data', metric='xml', value={'status': 'ok', 'xml': ...},
```

decrement(*name*, *tags=None*)

Decrement a counter.

```
>>> metrics.decrement('hotel.occupation')
```

event(*name*, *text*, *tags=None*)

Record an event.

```
>>> metrics.event('user.signup', 'New user registered')
```

gauge(*name*, *value*, *tags=None*)

Record the value of a gauge.

```
>>> metrics.gauge('users.notifications', 13, tags=['new_message', 'follow_request'])
```

increment(*name*, *tags=None*)

Increment a counter.

```
>>> metrics.increment('user.profile.views')
```

timed(*name=None*, *tags=None*, *use_ms=None*)

Measure the amount of time of a function (using a decorator) or a piece of code (using a context manager). If name is not provided while using the decorator it will be used the name of the module and the function.

```
# With decorator
@metrics.timed('request.response_time')
def perform_request(params):
    pass
```

```
        # With context manager
        with metrics.timed('request.response_time'):
            pass
```

timing(*name*, *value*, *tags=None*)
Record a timing.

```
        >>> metrics.timing('hotel.availability.request_time', 4)
```

CHAPTER 3

Backends

Kaneda provides builtin backends to store metrics and events in a persistent storage. If you want to use your custom backend you need to subclass `BaseBackend` and implement your custom `report` method which is the responsible to store the metrics data.

3.1 Elasticsearch

Elasticsearch is a search based NoSQL database that works very well with metrics data. It provides powerful tools to analyze data and build real-time dashboards easily with Kibana.

Note: Before using Elasticesearch as backend you need to install Elasticsearch Python client:

```
pip install elasticsearch
```

class `kaneda.backends.`**`ElasticsearchBackend`**(*index_name*, *app_name*, *client=None*, *connection_url=None*, *host=None*, *port=None*, *user=None*, *password=None*, *timeout=0.3*)

Elasticsearch backend.

Parameters

- **`index_name`** – name of the Elasticsearch index used to store metrics data. Default name format will be index_name-YYYY.MM.DD.
- **`app_name`** – name of the app/project where metrics are used.
- **`client`** – client instance of Elasticsearch class.
- **`connection_url`** – Elasticsearch connection url (https://user:secret@localhost:9200). It can be used passing a single connection_url (a string) or passing multiple connection_urls (a list).
- **`host`** – server host. It can be used passing a single host (a string) or passing multiple hosts (a list).
- **`port`** – server port.
- **`user`** – HTTP auth username.
- **`password`** – HTTP auth password.
- **`timeout`** – Elasticsearch connection timeout (seconds).

3.2 MongoDB

MongoDB is a document oriented NoSQL database. Is a great tool to store metrics as it provides a powerful aggregation framework to perform data analysis.

Note: Before using MongoDB as backend you need to install MongoDB Python client:

```
pip install pymongo
```

class kaneda.backends.**MongoBackend**(*db_name*, *collection_name*, *client=None*, *connection_url=None*, *host=None*, *port=None*, *timeout=300*)

MongoDB backend.

Parameters

- **db_name** – name of the MongoDB database.
- **collection_name** – name of the MongoDB collection used to store metric data.
- **client** – client instance of MongoClient class.
- **connection_url** – Mongo connection url (mongodb://localhost:27017/).
- **host** – server host.
- **port** – server port.
- **timeout** – MongoDB connection timeout (milliseconds).

3.3 RethinkDB

RethinkDB is an open source scalable, distributed NoSQL database built for realtime applications.

Note: Before using RethinkDB as backend you need to install RethinkDB Python client:

```
pip install rethinkdb
```

class kaneda.backends.**RethinkBackend**(*db*, *table_name=None*, *connection=None*, *host=None*, *port=None*, *user=None*, *password=None*, *timeout=0.3*)

RethinkDB backend.

Parameters

- **db** – name of the RethinkDB database.
- **table_name** – name of the RethinkDB table. If this is not provided, it will be used the name of the metric.
- **host** – server host.
- **port** – server port.
- **user** – auth username.
- **password** – auth password.
- **timeout** – RethinkDB connection timeout (seconds).

3.4 Logger

You can use a logger instance of the logging library from the Python standard lib. Useful for debugging.

class kaneda.backends.**LoggerBackend**(*logger=None*, *filename=''*)
Logger backend.

Parameters

- **logger** – logging instance.
- **filename** – name of the file where logger will store the metrics.

3.5 InfluxDB

InfluxDB is an open source time series database with no external dependencies. It's useful for recording metrics, events, and performing analytics.

Note: Before using InfluxDB as backend you need to install InfluxDB Python client:

```
pip install influxdb
```

Warning: InfluxDB can store other type of data besides time series. However it has some restrictions:

- Metrics *tags* field can't be a `list` only a `dict`:

```
# bad
metrics.timing('user.profile_load_time', 230, tags=['login', 'edit_profile'])

# good
metrics.timing('user.profile_load_time', 230, tags={'from': 'login', 'to': 'edit_profile'})
```

- *Custom* metric *value* field can't be a `list` nor a nested `dict`:

```
# bad
metrics.custom('zone.search', metric='query_time', value={'times': [120, 230]})
metrics.custom('zone.search', metric='query_time', value={'times': {'start': 120}, {'end': 230}

# good
metrics.custom('zone.search', metric='query_time', value={'start_time': 120, 'end_time': 230})
```

class kaneda.backends.**InfluxBackend**(*database*, *client=None*, *connection_url=None*, *host=None*, *port=None*, *username=None*, *password=None*, *timeout=0.3*)
InfluxDB backend.

Parameters

- **database** – name of the InfluxDB database.
- **client** – client instance of InfluxDBClient class.
- **connection_url** – InfluxDB connection url (influxdb://username:password@localhost:8086/databasename).
- **host** – server host.

- **port** – server port.
- **username** – auth username.
- **password** – auth password.
- **timeout** – InfluxDB connection timeout (seconds).

CHAPTER 4

Queues

Kaneda provides builtin queues to store metrics and events to perform *asynchronous reporting*. If you want to use your custom asynchronous queue system you need to subclass `BaseQueue` and implement your custom `report` method which is the responsible to pass metrics data to a job queue.

4.1 Celery

Celery is a simple, flexible and reliable distributed system to process vast amounts of messages. It can be configured using various broker systems such Redis or RabbitMQ.

Note: Before using Celery as async queue you need to install Celery library:

```
pip install celery
```

class `kaneda.queues.`**`CeleryQueue`**(*app=None*, *broker=None*, *queue_name=''*)

Celery queue.

Parameters

- **`app`** – app instance of Celery class.
- **`broker`** – broker connection url where Celery will attend the async reporting requests.
- **`queue_name`** – name of the queue being used by the Celery worker process.

To run the worker execute this command:

```
celery -A kaneda.tasks.celery worker
```

4.2 RQ

RQ (Redis Queue) is a simple Python library for queueing jobs and processing them in the background with workers. It uses Redis as main broker system.

Note: Before using RQ as async queue you need to install RQ and Redis library:

```
pip install redis
pip install rq
```

To run the worker execute this command:

```
rqworker [queue]
```

The default queue is "kaneda".

class kaneda.queues.**RQQueue**(*queue=None*, *redis_url=None*, *queue_name='kaneda'*)
RQ queue

Parameters

- **queue** – queue instance of RQ class.
- **redis_url** – Redis connection url where RQ will attend the async reporting requests.
- **queue_name** – name of the queue being used by the RQ worker process.

CHAPTER 5

Settings

Kaneda can be used with a settings file as the same way to use with *Django*. Simply define a `kanedasettings.py` file with the backend or queue settings. Alternatively you can define the environment variable *DEFAULT_SETTINGS_ENVAR* pointing to the desired settings filename.

With this you will be able to use *Metrics* class without passing parameters:

```
from kaneda import Metrics

metrics = Metrics()
metrics.gauge('answer_of_life', 42)
```

5.1 Backends settings

5.1.1 General

BACKEND Class name of the backend. Available options are:

- `kaneda.backends.ElasticsearchBackend`
- `kaneda.backends.MongoBackend`
- `kaneda.backends.LoggerBackend`
- `kaneda.backends.RethinkBackend`
- `kaneda.backends.InfluxBackend`

5.1.2 Elasticsearch

ELASTIC_INDEX_NAME Name of the Elasticsearch index used to store metrics data. Default name format will be app_name-YYYY.MM.DD.

ELASTIC_APP_NAME Name of the app/project where metrics are used.

ELASTIC_CONNECTION_URL Elasticsearch connection url (https://user:secret@localhost:9200).

ELASTIC_HOST Server host.

ELASTIC_PORT Server port.

ELASTIC_USER HTTP auth username.

ELASTIC_PASSWORD HTTP auth password.

ELASTIC_TIMEOUT Elasticsearch connection timeout (seconds).

5.1.3 MongoDB

MONGO_DB_NAME Name of the MongoDB database.

MONGO_COLLECTION_NAME Name of the MongoDB collection used to store metric data.

MONGO_CONNECTION_URL Mongo connection url (mongodb://localhost:27017/).

MONGO_HOST Server host.

MONGO_PORT Server port.

MONGO_TIMEOUT MongoDB connection timeout (milliseconds).

5.1.4 RethinkDB

RETHINK_DB Name of the RethinkDB database

RETHINK_TABLE_NAME Name of the RethinkDB table. If this is not provided, it will be used the name of the metric.

RETHINK_HOST Server host.

RETHINK_PORT Server port.

RETHINK_USER Auth username.

RETHINK_PASSWORD Auth password.

RETHINK_TIMEOUT RethinkDB connection timeout (seconds).

5.1.5 InfluxDB

INFLUX_DATABASE Name of the InfluxDB database.

INFLUX_CONNECTION_URL InfluxDB connection url (influxdb://username:password@localhost:8086/databasename).

INFLUX_HOST Server host.

INFLUX_PORT Server port.

INFLUX_USERNAME Auth username.

INFLUX_PASSWORD Auth password.

INFLUX_TIMEOUT InfluxDB connection timeout (seconds).

5.1.6 Logger

LOGGER_FILENAME Name of the file where logger will store the metrics.

5.2 Queues settings

5.2.1 General

QUEUE Class name of the queue. Available options are:

- `kaneda.backends.CeleryQueue`
- `kaneda.backends.RQQueue`

5.2.2 Celery

CELERY_BROKER Broker connection url.

CELERY_QUEUE_NAME Name of the Celery queue.

5.2.3 RQ

RQ_REDIS_URL Redis connection url.

RQ_QUEUE_NAME Name of the RQ queue.

CHAPTER 6

Django Setup

Kaneda can be use with Django as a mechanism to reporting metrics and events.

1. Add `django_kaneda` to `INSTALLED_APPS` in `settings.py`.
2. Set `KANEDA_BACKEND` and the properly configuration of your selected backend in `settings.py`. If you want to use Elasticsearch our configuration will be something like this:

```
    KANEDA_BACKEND = 'kaneda.backends.ElasticsearchBackend'
    KANEDA_ELASTIC_INDEX_NAME = 'kaneda'
    KANEDA_ELASTIC_APP_NAME = 'YouProject'
    KANEDA_ELASTIC_HOST = 'localhost'
    KANEDA_ELASTIC_PORT = 9200
    KANEDA_ELASTIC_USER = 'user'
    KANEDA_ELASTIC_PASSWORD = 'pass'
```

Alternatively you can set `KANEDA_QUEUE` to specify a *queue* configuration to use Kaneda in *async mode*:

```
KANEDA_BACKEND = 'kaneda.queues.CeleryQueue'
KANEDA_CELERY_BROKER = 'redis://localhost:6379/0'
```

With this, you can use Kaneda in everyplace of your Django project:

```
from django_kaneda import metrics


class UserProfileView(TemplateView):
    template_name = 'user/profile.html'

    @metrics.timed('user_profile.time')
    def get(self, request, *args, **kwargs):
        metrics.increment('user_profile.views')
        return super(UserProfileView, self).get(request, *args, **kwargs)
```

6.1 Debug mode

You can use Kaneda in debug mode with a logger as backend. Simply set `KANEDA_DEBUG` to *True* to report everything to a logger instead a persistent backend. Furthermore, you can set a previously defined logger on `settings.py` and use as your debug logger:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
```

```
    'formatters': {
        'with_timestamp': {
            'format': '%(asctime)s - %(name)s - %(message)s'
        }
    },
    'handlers': {
        'file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': '/tmp/kaneda-demo.log',
            'formatter': 'with_timestamp'
        },
    },
    'loggers': {
        'kaneda.demo': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}

KANEDA_DEBUG = True
KANEDA_LOGGER = 'kaneda.demo'
```

Alternatively you can set `KANEDA_LOGGER_FILENAME` instead of `KANEDA_LOGGER` to store the reporting results in a specific filename.

6.2 Available settings

6.2.1 Elasticsearch

KANEDA_ELASTIC_INDEX_NAME (='kaneda') Name of the Elasticsearch index used to store metrics data. Default name format will be app_name-YYYY.MM.DD.

KANEDA_ELASTIC_APP_NAME (='default') Name of the app/project where metrics are used.

KANEDA_ELASTIC_CONNECTION_URL (=None) Elasticsearch connection url (https://user:secret@localhost:9200).

KANEDA_ELASTIC_HOST (=None) Server host.

KANEDA_ELASTIC_PORT (=None) Server port.

KANEDA_ELASTIC_USER (=None) HTTP auth username.

KANEDA_ELASTIC_PASSWORD (=None) HTTP auth password.

KANEDA_ELASTIC_TIMEOUT (=0.3) Elasticsearch connection timeout (seconds).

6.2.2 MongoDB

KANEDA_MONGO_DB_NAME (='kaneda') Name of the MongoDB database.

KANEDA_MONGO_COLLECTION_NAME (='default') Name of the MongoDB collection used to store metric data.

KANEDA_MONGO_CONNECTION_URL (=None) Mongo connection url (mongodb://localhost:27017/).

KANEDA_MONGO_HOST (=None) Server host.

KANEDA_MONGO_PORT (=None) Server port.

KANEDA_MONGO_TIMEOUT (=300) MongoDB connection timeout (milliseconds).

6.2.3 RethinkDB

KANEDA_RETHINK_DB (='kaneda') Name of the RethinkDB database

KANEDA_RETHINK_TABLE_NAME (=None) Name of the RethinkDB table. If this is not provided, it will be used the name of the metric.

KANEDA_RETHINK_HOST (=None) Server host.

KANEDA_RETHINK_PORT (=None) Server port.

KANEDA_RETHINK_USER (=None) Auth username.

KANEDA_RETHINK_PASSWORD (=None) Auth password.

KANEDA_RETHINK_TIMEOUT (=0.3) RethinkDB connection timeout (seconds).

6.2.4 InfluxDB

KANEDA_INFLUX_DATABASE (='kaneda') Name of the InfluxDB database.

KANEDA_INFLUX_CONNECTION_URL (=None) InfluxDB connection url (influxdb://username:password@localhost:8086/databasename).

KANEDA_INFLUX_HOST (=None) Server host.

KANEDA_INFLUX_PORT (=None) Server port.

KANEDA_INFLUX_USERNAME (=None) Auth username.

KANEDA_INFLUX_PASSWORD (=None) Auth password.

KANEDA_INFLUX_TIMEOUT (=0.3) InfluxDB connection timeout (seconds).

6.2.5 Celery

KANEDA_CELERY_BROKER (='') Broker connection url.

KANEDA_CELERY_QUEUE_NAME (='') Name of the Celery queue.

6.2.6 RQ

KANEDA_RQ_REDIS_URL (='') Redis connection url.

KANEDA_RQ_QUEUE_NAME (='kaneda') Name of the RQ queue.

6.2.7 Debug

KANEDA_DEBUG (=True) Use Kaneda in debug mode.

KANEDA_LOGGER (=None) Name of a previously defined logger, to use in debug mode.

KANEDA_LOGGER_FILENAME (=None) Name of the file where logger will store the metrics, to use in debug mode.

CHAPTER 7

Changelog

7.1 0.4 (2016-04-28)

7.1.1 Add support to run Kaneda in asynchronous mode.

- Add Celery and RQ queue classes and tasks with backend reporting.
- Add the option to configure Kaneda with a settings file.
- Add utils.py file to handle the settings.
- Add asynchronous reporting, settings file and queue classes documentation.
- Add integration and benchmark testing. Improve unit tests.

7.1.2 Add Django support.

- Move from django-kaneda project.
- Add integration tests.
- Improve Django documentation

7.2 0.3.1 (2016-03-29)

- ElasticBackend now can be configured passing a list of 'hosts' or a list of 'connection_urls'.

7.3 0.3 (2016-03-28)

- Add LoggerBackend.
- Add exeception treatment on backend report methods.
- Add connection timeout for storage backends.
- Storage backends now support passing client or connection url as parameters instead all parameters.
- Add support to use django-kaneda on debug mode.

7.4 0.2 (2016-02-17)

- Refactor backend payload build method to unpack value field when is a dict.

7.5 0.1 (2016-01-13)

- Initial release.

C

D

E

G

I

L

M

R

T