Flask-Notifications Documentation

Release 0.1

CERN

Jun 04, 2017

Contents

-- *coding: utf-8* -- # # This file is part of Flask-Notifications # Copyright (C) 2015 CERN. # # Flask-Notifications is free software; you can redistribute it and/or modify # it under the terms of the Revised BSD License; see LICENSE file for # more details.

Flask-Notifications is a Flask extension that provides a generic real-time notification framework.

CHAPTER 1

Contents

Installation

Flask-Notifications is on PyPI so all you need is:

```
$ pip install Flask-Notifications
```

The development version can be downloaded from its page at GitHub.

```
$ git clone https://github.com/inveniosoftware/flask-notifications.git
$ cd flask-notifications
$ python setup.py develop
$ ./run-tests.sh
```

Requirements

Flask-Notifications has the following dependencies:

- Flask
- Flask-CeleryExt
- Redis
- Gevent
- Blinker
- Sse
- six

Flask-Notifications requires Python version 2.6, 2.7 or 3.3+.

Usage

This guide assumes that you have successfully installed `Flask-Notifications` package already. If not, please follow the *Installation* instructions first.

Building a simple notification system

`Flask-Notifications` provides a simple API to build your own real-time notification system. In this guide, we will see how to build such a system easily in a few steps.

First, we create the `Flask` application and initialise the Notifications extension. `Flask-Notifications` depends upon Celery and Redis. The first one is used for task processing and the second one for the Pub/Sub primitives. Then, we reuse Redis as a broker too.

In case you want to use another broker as *RabbitMQ*, you can implement the Pub/Sub or Fan-Out pattern by yourself by extending the `Backend` type.

```
from flask import Flask
from flask_notifications import Notifications

app = Flask(__name__)
notifications = Notifications(app=app)
```

or:

```
from flask import Flask
from flask_notifications import Notifications

app = Flask(__name__)
notifications = Notifications()
notifications.init_app(app=app)
```

or:

```
from flask import Flask
from flask_notifications import Notifications

# Corresponding information for brokers and Celery
config = {...}
celery = FlaskCeleryExt(app).celery
redis = StrictRedis(host=redis_host)

app = Flask(__name__)
notifications = Notifications()
notifications.init_app(app=app, celery=celery, broker=redis)
```

Now, we create a `EventHub`. A hub is composed of a filter and a list of consumers. When an event is sent to the hub, the filter is applied to that event. If it passes, it is sent to all the registered consumers.

An `EventHub` requires a label as a parameter. This label cannot be randomized. In order to make a reference to a hub, one should get first his identifier, which is not the same as the label.

```
test_hub = notifications.create_hub("TestHub")
test_hub_id = user_hub.hub_id
```

The next step is to set up our hub. Let's say we want to aggregate in that hub all the events with the "test" type and which are sent from now on.

```
import datetime
now = datetime.now()

from flask_notifications.filters.with_event_type import WithEventType
from flask_notifications.filters.before_date import BeforeDate

event_hub.filter_by(
    WithEventType("test") & Not(BeforeDate(now))
)
```

This creates a composed filter with those requirements. Any `EventFilter` can be composed using the bitwise operators (`&`, `|` and `^`) -it's not possible to use the logical operators `and`, `or` and `xor` because Python2.7 does not allow to override his behaviour-.

Now, we register some consumers to our hub.

```
@event_hub.register_consumer(celery_task_name="app.write_to_file")
def write_to_file(event, *args, **kwargs):
    with open("events.log", "a+w") as f:
        f.write(str(event))


push_consumer = PushConsumer(redis, event_hub_id)
event_hub.register_consumer(push_consumer)
```

When registering a function using the decorator, it is very important to specify the `celery_task_name` relatively to your application to help the workers to detect the function. More information here.

If you feel like to write a complex consumer, you can extend the the `Consumer` interface. Also, this interface has some hooks. One before consuming the event and the other after. This is very handy when you want to confirm that an event has been stored and hence send it to a database to persist it.

The only missing step is to send notifications.

```
event = Event(None, "test", "This event will pass the filter",
              "This is the body of the test", sender="system")
notifications.send(event.to_json())

event = Event(None, "system", "This event will not pass the filter",
              "This is the body of the test", sender="system")
notifications.send(event.to_json())
```

`Event` is a dictionary with a predefined model. If you would like to add your own fields and filter them, you just need to add the field to the `Event` and create a new filter by extending `EventFilter`.

You should now be able to emulate this example in your own Flask applications. For more information, please read the *Architecture* guide, check the *Predefined Consumers* section, the *Configuration* and peruse the *API*.

Architecture

The application is composed of two main parts: the main program and the workers. These workers will process asynchronously all the consumers in the hubs.

As we are defining functions using the `@register_consumer` decorator, the workers need to know and register this function as well. Therefore, a worker imports the main program and compiles it. It is very important not to have any randomized value because they won't match neither in the main application nor the program.

If you are going to use predefined consumers, you need to add the necessary configuration values to the Flask configuration.

Configuration

`Flask-Notifications` only needs one parameter in the Flask configuration: **BACKEND**. This option points to the Python path of a subclass of `Backend`. By default, it uses `RedisBackend`, but you can add your own implementation of `Backend` using other brokers like RabbitMQ. You just need to make you sure that the option has the right path to the class in order to be imported by the Notifications module.

```
config = {
    ...,
    # Default option
    "BACKEND": "flask_notifications.pubsub.redis_pubsub.RedisPubSub",
}
```

Also, `Flask-Notifications` uses the **JSON** serializer and deserializer to pass the events to the consumers. So, it is important that you allow the json serializer in the Celery configuration by using the following options (you can add any serializer that you want, the important thing is to enable the json serializer):

```
config = {
    ...,
    "CELERY_ACCEPT_CONTENT": ["application/json"],
    "CELERY_TASK_SERIALIZER": "json",
    "CELERY_RESULT_SERIALIZER": "json",
}
```

Predefined Consumers

The predefined consumers exist to fulfil simple needs like sending an email or writing a log. You can use them in your code by importing and registering them.

For more complex consumers, you may create your own by extending a predefined consumer or creating a new one extending `Consumer`.

Current predefined consumers:

class `push_consumer.`**`PushConsumer`**

class `log_consumer.`**`LogConsumer`**

class `flaskemail_consumer.`**`FlaskEmailConsumer`**

class `flaskmail_consumer.`**`FlaskMailConsumer`**

API

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

Flask extension

Decorators

`flask_notifications.event_hub.`**`register_consumer`**`()`

Proxies

`flask_notifications.event_hub.`**`current_notifications`**
`Root of the notification extension.`

Changes

Version 0.1.0 (released TBD)

- Initial public release.

Contributing

Bug reports, feature requests, and other contributions are welcome. If you find a demonstrable problem that is caused by the code of this library, please:

1. Search for already reported problems.
2. Check if the issue has been fixed or is still reproducible on the latest *master* branch.
3. Create an issue with **a test case**.

If you create a feature branch, you can run the tests to ensure everything is operating correctly:

```
$ ./run-tests.sh
```

License

Flask-Notifications is free software; you can redistribute it and/or modify it under the terms of the Revised BSD License quoted below.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an Intergovernmental Organization or submit itself to any jurisdiction.

CHAPTER 2

Authors

Flask-Notifications is developed for use in Invenio digital library software.

Contact Invenio at info@inveniosoftware.org

- Jorge Vicente Cantero <jorgevc@fastmail.es>

Python Module Index

f

Index