WSCelery Documentation

Release 0.2.0

Antonis Kalou

Jul 04, 2017

Contents

Real time celery monitoring using websockets. Inspired by flower.

CHAPTER 1

Contents

Installation

PyPI version (recommended):

```
$ pip install wscelery
```

Development version:

```
$ pip install https://github.com/johan-sports/wscelery/zipball/master
```

Usage

Launch the websocket listener on port 8001:

```
$ wscelery --port=8001
```

Or launch from celery:

```
$ celery wscelery -A proj --address=127.0.0.1 --port=8001
```

Broker URL and other configuration options can be passed through standard Celery options:

```
$ celery wscelery -A proj --broker=amqp://guest:guest@localhost:5672//
```

Configuration

WSCelery can be configured from the command line:

```
$ wscelery --allow-origin=.*
```

Or using environment variables. All options are configured with a *WSCELERY_* prefix.

```
$ export WSCELERY_PORT=8001
$ wscelery  # will use port 8001
```

Options

Standard celery configuration options can be overriden using environment variables or command line options. See the Celery reference for a complete list of celery options.

Celery command line options can be passed to wscelery too. E.g.

```
$ wscelery --broker=amqp://guest:guest@10.9.3.123:5672
```

For a full list of celery options see:

```
$ celery --help
```

For a full list of wscelery specific options see:

```
$ wscelery --help
```


address

Run the websocket server on the given address. (Defaults to *127.0.0.1*)

port

Run the websocket server on a given port. (Defaults to *1337*)

allowed_origin

A regex of origins allowed to access the websocket. (Defaults to current host)

debug

Run wscelery in debug mode. **Do not use in production.**

Examples

All following examples are also available in the project examples directory.

Setup

To run these examples we must first create a project with some celery tasks.

Listing 1.1: tasks.py

```
from celery import Celery

app = Celery('tasks', broker='amqp://guest:guest@localhost:5672')


@app.task
def add(x, y):
    return x + y
```

This assumes that RabbitMQ is running in on `localhost:5672`.

We will also define an endpoint for triggering tasks. In this example we are using flask, but any other web framework/library will work fine.

To install application dependencies run `pip install -r examples/celery_app/requirements.txt` in the project root folder.

Listing 1.2: app.py

```
from flask import Flask, jsonify
from flask_cors import CORS

import tasks

app = Flask(__name__)
# Allow any origin
CORS(app)


@app.route('/')
def index():
    return 'Its working!'


@app.route('/add/<int:x>/<int:y>', methods=['POST'])
def add(x, y):
    task = tasks.add.delay(x, y)
    return jsonify({'task_id': task.id})


if __name__ == '__main__':
    app.run()
```

Start the web server with

```
$ python app.py
```

To test that the API is working, trigger a task

```
$ curl -X POST http://localhost:5000/add/1/2
{
  "task_id": "1ee8e9bf-17b9-4fef-90ca-42c0c5880f13"
}
```

We must also start a celery worker to process the task

```
$ celery worker -A tasks
```

Lastly, run wscelery on localhost and allow all origins:

```
$ celery wscelery --allow-origin=.*
```

Javascript

This code is intended to run in the browser. It will trigger the add task for given user input and report the finished status.

First we define a basic HTML file with a form and load jQuery:

Listing 1.3: index.html

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>WSCelery Client</title>

        <script
            src="https://code.jquery.com/jquery-3.2.1.min.js"
            integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
            crossorigin="anonymous"></script>
        <script src="client.js" type="text/javascript"></script>
    </head>
    <body>
        <form action="" id="add">
            <input name="x" type="number" required />
            <input name="y" type="number" required />
            <input type="submit" value="Add" />
        </form>

        <p id="status"></p>
    </body>
</html>
```

When the form is submitted a request is made to the web API to start the task. We then open a connection to wscelery and handle different message types reporting the current status.

Listing 1.4: client.js

```
window.onload = function() {
  function openSocket(taskId) {
    // Connect websocket
    var taskSocket = new WebSocket('ws://localhost:1337/' + taskId);
```

```
    $('p#status').text('Opened websocket, processing...');
    taskSocket.onmessage = function(event) {
      var msg = JSON.parse(event.data);

      switch(msg.type) {
      case 'task-succeeded':
        $('p#status').text('Task succeeded with result: ' + msg.result + ' Elapsed: '␣
↪+ msg.runtime);
        taskSocket.close();
        break;
      case 'task-retried': // fallthrough
      case 'task-failed':
        $('p#status').text('Task failed with exception: ' + msg.exception);
        taskSocket.close();
        break;
      case 'task-rejected': // fallthrough
      case 'task-revoked':
        taskSocket.close();
        break;
      default: // ignore
        break;
      }
    };

    taskSocket.onerror = function() {
      $('p#status').text('Websocket error!');
    };
  }

  $('form#add').submit(function(event) {
    var formData = new FormData(event.target);
    var x = formData.get('x');
    var y = formData.get('y');
    // Create task
    $.ajax({
      url: 'http://localhost:5000/add/' + x + '/' + y,
      type: 'POST',
      success: function(data) {
        $('p#status').text('Received task with ID:', data.task_id);
        openSocket(data.task_id);
      },
      error: function() {
        $('p#status').text('Request to web API failed.');
      }
    });
    event.preventDefault();
  });
};
```

API Reference

WSCelery provides a websocket connection under `ws://my-domain.com/<task-id>`. Once a connection is established the server sends status updates for the task with ID *<task-id>* to the client.

It is the client's responsibility to close the connection.

Events

The received events mirror those described in the Celery monitoring reference with some keys excluded. Events are sent through the websocket as JSON and have the following structure:

- *task-sent*
- *task-received*
- *task-started*
- *task-succeeded*
- *task-failed*
- *task-rejected*
- *task-revoked*
- *task-retried*

task-sent

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-sent",
  "name": "myapp.add",
  "retries": 0,
  "eta": 32,
  "routing_key": "default",
  "root_id": 12,
  "parent_id": 15
}
```

task-received

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-received",
  "timestamp": 1494943644.786262,
  "local_received": 1494947444.446089,
  "utcoffset": -2,
  "retries": 1,
  "root_id": 12,
  "parent_id": 15
}
```

task-started

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-received",
  "timestamp": 1494943644.786262,
```

```
  "local_received": 1494947444.446089,
  "utcoffset": -2
}
```

task-succeeded

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-succeeded",
  "timestamp": 1494943644.786262,
  "local_received": 1494947444.446089,
  "utcoffset": -2,
  "result": "42",
  "runtime": 5.227228619001835
}
```

task-failed

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-failed",
  "timestamp": 1494943644.786262,
  "local_received": 1494947444.446089,
  "utcoffset": -2,
  "traceback": "...",
  "exception": "ValueError('oops')",
}
```

task-rejected

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-rejected",
  "requeued": true,
}
```

task-revoked

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-revoked",
  "terminated": true,
  "signum": 3,
  "expired": false
}
```

task-retried

```
{
  "uuid": "bbef09c9-aff2-4f51-8238-d594fe16bc66",
  "type": "task-retried",
  "timestamp": 1494943644.786262,
  "local_received": 1494947444.446089,
  "utcoffset": -2,
  "exception": "ValueError('oops')",
  "traceback": "...",
}
```

Nginx Usage

The following is a minimal nginx configuration:

```
server {
  listen 80;
  server_name wscelery.johan-sports.com;
  charset utf-8;

  location / {
     proxy_pass http://localhost:1337;
     proxy_redirect off;
     proxy_http_version 1.1;
     proxy_set_header Host $host;
     proxy_set_header Upgrade $http_upgrade;
     proxy_set_header Connection $connection_upgrade;
  }
}
```

Docker Usage

WSCelery is has automatic builds on DockerHub.

Pull the image and start the container

```
$ docker pull johansports/wscelery
$ docker run -p=1337:1337 -d johansports/wscelery
```

You can also specify environment variables

```
$ docker run -e"BROKER_URL=amqp://guest:guest@localhost:5672" -e"WSCELERY_ALLOW_
↪ORIGIN=.*" run -d johansports/wscelery
```

Caveats

WSCelery is still in its early days and thus has some caveats:

- TLS encryption not (yet) supported

CHAPTER 2

License

WSCelery is Open Source and is licensed under the MIT License.