Mark Gituma Follow
Dec 19, 2017 · 4 min read

Using Django 2 with Celery and Redis

The current Django version 2.0 brings about some significant changes; this includes a lack of support for python2. Thus, the focus of this tutorial is on using python3 to build a Django application with celery for asynchronous task processing and Redis as the message broker. The code for this tutorial can by downloaded directly from my github account.

Preparation

Python3

As Django 2 uses python3, we need to make sure we install it. As I am working on a Mac, the command used for installation is:

```
$ brew install python3
```

This installs python3 as well as well as pip3. However, the global python version still points to python2:

```
$ python --version
Python 2.7.10
```

Virtualenv

In order to define the python version for the project, the virtualenvwrapper package is be used. This means that, depending on the project we can switch between different versions of python . To use virtualenvwrapper (assuming it has been installed), the following command needs to be run:

```
$ mkvirtualenv --python=`which python3` <env_name>
```

Where `<env_name>` should be replaced with the name of the environment. The expression `which python` resolves to the path where python3 has been installed i.e. `/user/local/bin/python3` .The virtual environment can be activated by running `workon <env_name>` . Within the virtual environment, the python version is:

```
$ python --version
Python 3.6.3
```

Pip packages

The required python packages within the virtual environment can be installed by running:

```
$ pip install Django==2.0
$ pip install Celery==4.1.0
$ pip install redis==2.10.6
```

It's good to explicitly specify the package versions as will lead to a codebase that's easier to maintain due to being predictable as per the 12 factor app manifesto.

Redis

As celery requires a message broker, we need to set one up. Redis is one of the easiest brokers to configure and be done by running the following commands (for ubuntu please refer to the following article):

```
$ brew install redis
$ brew services start redis
```

That's all we need to get Redis going, by default it runs in `localhost` on port `6379` and is what we will use in our `settings.py` file.

Setup

Setting up the python project can be found in the official Django documentation, so the steps won't be repeated. It's assumed the following directory structure has been created:

```
<mysite>/
├── <mysite>
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

Where `<mysite>` should be replaced with the actual project name.

celery.py

The `<mysite>/<mysite>/celery.py` file then needs to be created as is the recommended way that defines the Celery instance. The file should have the following configuration:

```
# http://docs.celeryproject.org/en/latest/django/first-
steps-with-django.html

from __future__ import absolute_import, unicode_literals
import os
from celery import Celery
```

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'<mysite>.settings')

app = Celery('<mysite>')
app.config_from_object('django.conf:settings',
namespace='CELERY')
app.autodiscover_tasks()


@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

__init__.py

In order to ensure that the app get's loaded when django starts, the `celery.py` file needs to be imported in `<mysite>/<mysite>/__init__.py` file:

```
from __future__ import absolute_import, unicode_literals

# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app

__all__ = ['celery_app']
```

Settings.py

In the `<mysite>/<mysite>/settings.py` file, we need to configure celery by adding the following variables:

```
# Celery application definition
CELERY_BROKER_URL = 'redis://localhost:6379'
CELERY_RESULT_BACKEND = 'redis://localhost:6379'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TASK_SERIALIZER = 'json'
```

This shows that redis is been used as a broker running on `localhost` on port `6379` . It accepts the `application/json` content type with `json` format for serialization.

Quick Testing

In order to test our setup in the virtual environment, run:

```
celery -A <mysite> worker -l info
```

The celery worker should be running and should be connected to the redis host on `redis://localhost:6379//` .

In a separate terminal but within the same folder, activate the virtual environment i.e. `workon <env_name>` and then run:

```
$ python manage.py shell
```

This activates the python interpreter which has the Django specific project environmental variables loaded. From within the interpreter run:

```
from django_2_celery.celery import debug_task
debug_task.delay()

<AsyncResult: c600110a-2ec1-4644-ab3d-1528f516bfed>
```

Where we import the `debug_task` which was defined in `<mysite>/<mysite>/celery.py` . The function is then executed asynchronously using the message broker, and then returns an asynchronous object which can be used to get the function result as well as check if the task has completed.

In the original terminal, the result of the `debug_task` task is:

```
[2017-12-18 19:15:35,120: INFO/MainProcess] Received task:
django_2_celery.celery.debug_task[c600110a-2ec1-4644-ab3d-
1528f516bfed]
[2017-12-18 19:15:35,133: WARNING/ForkPoolWorker-2]
Request: <Context: {'lang': 'py',.........
[2017-12-18 19:15:35,155: INFO/ForkPoolWorker-2] Task
django_2_celery.celery.debug_task[c600110a-2ec1-4644-ab3d-
1528f516bfed] succeeded in 0.02517244400223717s: None
```

Conclusion

This gives a simplified way on how to run Django 2 with Celery in a development environment. A more complicated example will specify tasks in different apps to perform more complex operations. Such a setup is outside the scope of this tutorial, but there are enough online resources to get going.

If you like the post and want to be notified of new blogs, follow me on twitter @MarkGituma.

Credits

- https://medium.com/@yehandjoe/celery-4-periodic-task-in-django-9f6b5a8c21c7
- https://www.codementor.io/uditagarwal/asynchronous-tasks-using-celery-with-django-du1087f5k

- https://www.codingforentrepreneurs.com/blog/celery-redis-django/