How to use Celery+RabbitMQ

Nick Thompson

November 22, 2015

Getting started:

```
$ git clone https://github.com/NAThompson/learn_celery.git
$ cd learn_celery
$ pip3 install -r requirements.txt
```

What will we attempt to accomplish?

- Link Celery+RabbitMQ into Django ecosystem

What will we attempt to accomplish?

- Link Celery+RabbitMQ into Django ecosystem
- Use tasks to keep page loads from timing out

What will we attempt to accomplish?

- Link Celery+RabbitMQ into Django ecosystem
- Use tasks to keep page loads from timing out
- Associate a running job with a progress bar

What will we attempt to accomplish?

- Link Celery+RabbitMQ into Django ecosystem
- Use tasks to keep page loads from timing out
- Associate a running job with a progress bar
- Link CPU cycles to billing

What will we attempt to accomplish?

- Link Celery+RabbitMQ into Django ecosystem
- Use tasks to keep page loads from timing out
- Associate a running job with a progress bar
- Link CPU cycles to billing
- Autoscale nodes based on queue length

Why Celery + RabbitMQ

- Job scheduling is boring, and it’s a pain.

Why Celery + RabbitMQ

- Job scheduling is boring, and it's a pain.
- So you certainly don't want to learn *two* job schedulers

Why Celery + RabbitMQ

- Job scheduling is boring, and it's a pain.
- So you certainly don't want to learn *two* job schedulers
- You never want to be in a spot where you have to swap out your job scheduler

Why Celery + RabbitMQ

- Job scheduling is boring, and it's a pain.
- So you certainly don't want to learn *two* job schedulers
- You never want to be in a spot where you have to swap out your job scheduler
- Therefore, use the most well-tested and popular stack

What is RabbitMQ?

- RabbitMQ is an implementation of the *advanced messaged queuing protocol* (AMQP), a standardized way to pass data between applications.
- The specification of AMQP is 300 pages long, but it's basically an email server protocal for binary applications

Warning!

- Celery as an API is a bit baffling

Warning!

- Celery as an API is a bit baffling
- Many people contend that using RabbitMQ directly leads to less astonishment

Warning!

- Celery as an API is a bit baffling
- Many people contend that using RabbitMQ directly leads to less astonishment
- The author has never tried RabbitMQ directly, but has been astonished many times by Celery. So keep this in mind as an option.

How do I set up RabbitMQ?

```
$ sudo apt-get install -y rabbitmq-server
$ sudo rabbitmqctl status
$ sudo lsof -i :5672 # 5672 is the well-known port of AMQP
```

If your server still isn't up, or you need to restart it:

```
$ sudo rabbitmq-server -detached
```

Testing RabbitMQ

We want to get the minimal working RabbitMQ working on localhost:

```
$ rabbitmqctl list_queues
Listing queues ...
$ python3 basic_rabbitmq/send_message.py
We have sent the message 'Hello World' to RabbitMQ
$ rabbitmqctl list_queues
Listing queues ...
hello 1
$ python3 basic_rabbitmq/send_message.py
We have sent the message 'Hello World' to RabbitMQ
$ rabbitmqclt list_queues
Listing queues ...
hello 2
```

Receive the messages:

```
learn_celery$ python3 basic_rabbitmq/receive_message.py
Waiting for messages. To exit press CTRL+C
[x] Received (b'Hello World',)
[x] Received$ (b'Hello World',)
learn_celery rabbitmqctl list_queues
Listing queues ...
hello 0
```

Deleting a queue

As you learn RabbitMQ, you start generating meaningless queues that you might want to get rid of. Here's how to delete them:

```
learn_celery$ rabbitmqctl list_queues
Listing queues ...
hello 0
(learn_celery) ~/learn_celery$ rabbitmqctl stop_app
Stopping node 'rabbit@nthompson-Precision-M6700' ...
(learn_celery) ~/learn_celery$ rabbitmqctl reset
Resetting node 'rabbit@nthompson-Precision-M6700' ...
(learn_celery) ~/learn_celery$ rabbitmqctl start_app
Starting node 'rabbit@nthompson-Precision-M6700' ...
(learn_celery) ~/learn_celery$ rabbitmqctl list_queues
Listing queues ...
```

On to Celery

- You could connect to your RabbitMQ directly, without Celery, but there's a lot of boilerplate.
- Celery makes a nice glue between your Django and Rabbitmq.

Celery Basic Example

```
$ cd basic_celery
$ celery --app=task work --loglevel=info &
$ python3 -q
>>> from tasks import add
>>> r = add.delay(12, 12)
[2015-11-05 14:16:07,806: INFO/MainProcess] Received task: tasks.add[4f
[2015-11-05 14:16:07,816: INFO/MainProcess] Task tasks.add[4f2c482f-5e6
>>> r.status
'SUCCESS'
>>> r.result
24
```

Easy peasy.

Celery Basic Example

Celery will start some queues in RabbitMQ after startup:

```
# rabbitmqctl list_queues
Listing queues ...
celery0
celery@nthompson-Precision-M6700.celery.pidbox0
celeryev.39a8ba96-9ed0-4b31-83a3-0728cb6273c50
```

Multiple workers

Celery has support for multiple workers:

```
celery --app=tasks worker --loglevel=INFO --concurrency=`nproc`
-------------- celery@nthompson-Precision-M6700 v3.1.19 (Cipater)
---- **** -----
--- * ***  * -- Linux-3.19.0-32-generic-x86_64-with-Ubuntu-15.04-vivid
-- * - **** ---
- ** ---------- [config]
- ** ---------- .> app:         tasks:0x7feabb181128
- ** ---------- .> transport:   amqp://guest:**@localhost:5672//
- ** ---------- .> results:     rpc://
- *** --- * --- .> concurrency: 8 (prefork)
-- ******* ----
--- ***** ----- [queues]
-------------- .> celery           exchange=celery(direct) key=celery
```

The default concurrency is nproc.

Embedding Celery into Django

It's not a huge jump from this to embedding in Django:

- Specify the serialization types you'll accept in settings.py:

```
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
```

Embedding Celery into Django

It's not a huge jump from this to embedding in Django:

- Specify the serialization types you'll accept in settings.py:

```
CELERY_ACCEPT_CONTENT = ['json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
```

- Specify the broker url in settings.py

```
BROKER_URL = 'amqp://guest:guest@localhost//'
```

Embedding Celery into Django

- Start a app and add some tasks to it:

```
$ ./manage.py startapp jobs
$ touch jobs/tasks.py # Define a task in here. Must be named tasks.
```

Embedding Celery into Django

- Start a app and add some tasks to it:

```
$ ./manage.py startapp jobs
$ touch jobs/tasks.py # Define a task in here. Must be named tasks.
```

- Add your app to INSTALLED_APPS in settings.py, and

```
echo "from .celery import app as celery_app  # noqa" >> __init__.py
```

Embedding Celery into Django

- Start a app and add some tasks to it:

```
$ ./manage.py startapp jobs
$ touch jobs/tasks.py # Define a task in here. Must be named tasks.
```

- Add your app to INSTALLED_APPS in settings.py, and

```
echo "from .celery import app as celery_app  # noqa" >> __init__.py
```

- Now start up celery:

```
$ celery --app=celery_in_django worker --loglevel=INFO &
```

You should see your task registered in the logs.

Minimal Django+Celery Example:

Assuming you have RabbitMQ up, then:

```
learn_celery$ git checkout basic_async_task
learn_celery$ cd django_example
django_example$ celery --app=celery_in_django worker --loglevel=info &
django_example$ ./manage.py runserver
```

Now navigate to 127.0.0.1:8000, and start hitting buttons.

Not-as-minimal Django+Celery Example

Problems with the minimal example:

- The user gets no feedback when he's pressed a button; as such might press it 40 times

Not-as-minimal Django+Celery Example

Problems with the minimal example:

- The user gets no feedback when he's pressed a button; as such might press it 40 times
- The user doesn't know when the task is done

Not-as-minimal Django+Celery Example

Problems with the minimal example:

- The user gets no feedback when he's pressed a button; as such might press it 40 times
- The user doesn't know when the task is done
- The user doesn't know how much to estimate how long the job will run

Take 1: Store tasks in the session

This is a low-effort, effective for some tasks method:

- Store task id in request.session; the use the task id as an argument to celery.result.AsyncResult constructor.

Take 1: Store tasks in the session

This is a low-effort, effective for some tasks method:

- Store task id in request.session; the use the task id as an argument to celery.result.AsyncResult constructor.
- Then we make buttons available based on task status:

```
$ git checkout task_in_session
$ cd django_example
$ ./manage.py runserver
```

Take 1: Store tasks in the session

Note that by default, the task state moves from 'PENDING' directly to 'FINISHED'.
This can be remedied by putting the following in settings.py:

```
CELERY_TRACK_STARTED = True
```

Take 1: Store tasks in the session

- This is not a great design, as the session should expire at browser close, and we might want to run jobs for longer than a session.

Take 1: Store tasks in the session

- This is not a great design, as the session should expire at browser close, and we might want to run jobs for longer than a session.
- Hence we need to store task id's in our database.

Take 1: Store tasks in the session

- This is not a great design, as the session should expire at browser close, and we might want to run jobs for longer than a session.
- Hence we need to store task id's in our database.
- Note: This will require user logins, the code is going to get more complex!

Take 2: Store task in the Database

One we have user logins, we can associate each user with a list of tasks.

```
$ git checkout tasks_in_db
$ basic_celery/kill_celery.sh
$ cd django_example
$ ./start_celery.sh
$ ./manage.py runserver
```

This will show us what time we started the task, as well as show when the task is started and finished (but no timestamps yet).

Progress Reporting

An accepted method of doing progress reporting in Celery is the use of custom states. Let's look at the code:

```
$ git checkout progress_bar
$ basic_celery/kill_celery.sh
$ cd django_example
$ ./start_celery.sh
$ ./manage.py runserver
```

Progress Bars

If the progress bar is moving too slowly, we might want to terminate the running job. Easy: Link it to a post request, user the celery “revoke” function:

```
$ git checkout revoke
$ basic_celery/kill_celery.sh
$ cd django_example
$ ./start_celery.sh
$ ./manage.py runserver
```

Job metadata

Note that the job metadata is being printed to stdout by Celery:

```
[2015-11-22 19:13:35,024: INFO/MainProcess]
Task jobs.tasks.fft_random[123e5a8d-1cda-457c-9082-810d4f163572]
succeeded in 13.250278769002762s: 0.20759175911474792
```