Up and Running with Celery and Django (also cron is evil)

Posted on November 30, 2012 by David Rideout (https://www.safaribooksonline.com/blog/author/drideout/) & filed under django (https://www.safaribooksonline.com/blog/category/programming/django-programming/), python (https://www.safaribooksonline.com/blog/category/programming/python-programming/).

The longer I'm a programmer, the lazier I become (http://c2.com/cgi/wiki?LazinessImpatienceHubris). Several years ago I'd have been a giddy schoolgirl if you told me to write a templating engine from scratch. Or authentication, wow—Dealing with HTTP headers and sessions got me so excited!

Nowadays I wonder why things just can't just *work*.

At Safari, there are lots of services with moving parts that need to be scheduled and I've gradually started to *really* dislike cron. Sure it's great for one-off tasks, but handling lots of tasks asynchronously is not one of its strong suits. And really, I'm just too lazy to write the logic to handle failures, redos, and other catch-22's that happen in the pipeline. Instead, I now use a combination of Django and the task queue Celery (http://celeryproject.org/).

Enter Celery and Supervisor *on Ubuntu

Ubuntu is quite nice to work with, as they keep packages relatively up to date. Supervisor? Redis? They just work, almost like *magic*. Here's the steps to get a cron-free world and running in a jif (with a Python virtual environment (http://www.virtualenv.org/en/latest/)):

First, let's install the necessary Ubuntu packages, create a working environment for the project, and get the necessary Python libraries. Let's call the project **Thing**.

```
$ sudo aptitude install supervisor redis-server
$ mkdir thing-project
$ cd thing-project
$ virtualenv --prompt="(thing)" ve
$ . ve/bin/activate
(thing)$ pip install django django-celery redis
```

Now we can start to put the Django pieces together. Start a new Django project called *thing* with an app called *automate* where we'll put our tasks. Also, add a `serverconf/` directory to keep your server/service configs separate.

```
(thing)$ django-admin.py startproject thing # now we have one too many dirs
(thing)$ mv thing/thing/*.* ./thing/
(thing)$ mv thing/manage.py ./
(thing)$ rmdir thing/thing/
(thing)$ python ./manage.py startapp automate &amp;&amp; touch automate/tasks.py
(thing)$ mkdir serverconf
```

Your project should look something like this:

```
/thing-project              # Container directory
    manage.py               # Run Django commands

    /ve                     # Your virtualenv

    /automate               # New app we're starting
        models.py
        tests.py
        views.py
        tasks.py            # Where the magic goes

    /thing
        settings.py         # Project settings

    /serverconf
        # Server Configs go in here, apache, supervisor, etc.
```

Add `automate` to the `INSTALLED_APPS` section in your `settings.py` and be sure to alter your `DATABASES` to use your backend of choice. My `DATABASES` looks like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'thing.db',
        'USER': '',
        'PASSWORD': '',
        'HOST': '',
        'PORT': '',
```

```
    }
}
```

Something to Do

Now let's just create a basic framework that does something, like crawl a web site for content. Modify your `automate/models.py` to look like this:

```
import urllib2

from django.db import models

class WebContent(models.Model):
    # I like timestamps
    timestamp_created = models.DateTimeField(auto_now_add=True)
    timestamp_updated = models.DateTimeField(auto_now=True)

    url = models.CharField(max_length=255)
    content = models.TextField(null=True)

    def update_content(self):
        self.content = urllib2.urlopen(self.url).read()
        self.save()
```

Test it out, it should work just fine:

```
(thing)$ python manage.py syncdb
(thing)$ python manage.py shell
&gt;&gt;&gt; from automate.models import *
&gt;&gt;&gt; rec = WebContent.objects.create(url='https://www.safaribooksonline.com/
&gt;&gt;&gt; rec.update_content()
&gt;&gt;&gt; print rec.content
### Really long dump of web site ###
```

A Real, Grown-up, Cron-like Task

Now we need to start adding the ingredients to turn this into a celery task (the equivalent of a cronjob). First, add `djcelery` to your list of `INSTALLED_APPS` and remember to `(thing)$ manage.py syncdb` as well. Somewhere near the bottom of your `thing/settings.py` , add this:

```
import djcelery

from celery.schedules import crontab

djcelery.setup_loader()

BROKER_URL = &quot;redis://localhost:6379/0&quot;
CELERY_RESULT_BACKEND = &quot;database&quot;
CELERYBEAT_SCHEDULER = &quot;djcelery.schedulers.DatabaseScheduler&q
CELERYBEAT_PIDFILE = '/tmp/celerybeat.pid'
CELERYBEAT_SCHEDULE = {} # Will add tasks later
```

And while we're at it, let's modify the `automate/tasks.py` file, where celery tasks are actually defined:

```
from celery.task import task

from automate.models import WebContent

@task
def update_all_sites():
    for rec in WebContent.objects.all():
        print &amp;quot;Updating site: %s&amp;quot; % rec.url
        rec.update_content()
```

Test it out by running the celery daemon (aka worker). Then queue the task in a separate terminal.

1st terminal:

```
(thing)$ python manage.py celeryd -l INFO
```

Note the following line to show that celery sees the task:

```
[Tasks]
  . automate.tasks.update_all_sites
```

2nd terminal:

```
(thing)$ python manage.py shell
&gt;&gt;&gt; from automate.tasks import *
&gt;&gt;&gt; update_all_sites.apply_async()
&lt;AsyncResult XXXXXXXXXXXXXXXXXXX&gt;
```

Your 1st terminal should have all kinds of awesome things going on:

```
[XXX: INFO/MainProcess] Got task from broker: automate.tasks.update_all_sites[96c453
[XXX: WARNING/PoolWorker-1] Updating site: https://www.safaribooksonline.com/blog
[XXX: INFO/MainProcess] Task automate.tasks.update_all_sites[96c45361-e68c-4e53-91c9
```

Wow, it works! Now update your `CELERYBEAT_SCHEDULE` (like the timing in a cron job) in your `settings.py` to schedule the task.

```
CELERYBEAT_SCHEDULE = {
    # Update web sites every 24h
    &amp;quot;update-web-sites&amp;quot;: {
        &amp;quot;task&amp;quot;: &amp;quot;automate.tasks.update_all_sites&amp;quot;,
        &amp;quot;schedule&amp;quot;: crontab(minute=0, hour=0),
    }
}
```

The Final Piece

The final piece of the puzzle is to set up supervisor so that celery runs automagically alongside Django. Create a log directory called `/var/log/thing`. Your `serverconf/thing-supervisor.conf` should look something like this:

```
;==================================
; celeryd supervisord script for django
; ==================================
;; Queue worker for the web interface.

[program:celery-thing]
command=/path/to/thing-project/ve/bin/python /path/to/thing-project/manage.py celer
directory=/path/to/thing-project
environment=PYTHONPATH='/path/to/thing-project/ve'
user=www-data
numprocs=1
stdout_logfile=/var/log/thing/celeryd.log
stderr_logfile=/var/log/thing/celeryd.log
autostart=true
autorestart=true
startsecs=10
stopwaitsecs=30

; ==========================================
; celerybeat
; ==========================================
[program:celerybeat-thing]
command=/path/to/thing-project/ve/bin/python /path/to/thing-project/manage.py celer
directory=/path/to/thing-project
environment=PYTHONPATH='/path/to/thing-project/ve'
user=www-data
numprocs=1
stdout_logfile=/var/log/thing/celerybeat.log
stderr_logfile=/var/log/thing/celerybeat.log
autostart=true
autorestart=true
startsecs=10
stopwaitsecs = 30
```

Finally, create the symlink so that your *serverconf/thing-supervisor.conf* is loaded when supervisor starts up:

```
$ ln -s /etc/supervisor/conf.d/thing-dev.conf /path/to/thing-project/serverconf/thir
$ service supervisor start
```

There you have it, a complete install without using cron. Now you can go on to do all the cool things that celery supports, i.e. task retries, chaining, etc.

Developing a git workflow (https://www.safaribooksonline.com/blog/2012/11/29/developing-a-git-workflow/)

Introduction to Hive (https://www.safaribooksonline.com/blog/2012/11/30/introduction-to-

hive/)

Tags:

5 Responses to “Up and Running with Celery and Django (also cron is evil)”

Andre

Hi,

you rock! that was exactly what I was looking for the last few days. I found the information on celery and brokers etc., but not as neatly put into one ready to go package as you did!

thanks!

July 28th, 2013 (https://www.safaribooksonline.com/blog/2012/11/30/up-and-running-with-celery-and-django/#comment-359161)

Eduardo Silva

Grate post, I was struggling with django and celery, and found this topic. Please note that your blog’s template seens broken and on the code part and where should appear ” (double quotes) we receive the html name: ".
Thank you!

August 25th, 2013 (https://www.safaribooksonline.com/blog/2012/11/30/up-and-running-with-celery-and-django/#comment-359162)

Some Guy

Agreed, " is appearing several times in the quoted code.

October 8th, 2013 (https://www.safaribooksonline.com/blog/2012/11/30/up-and-running-with-celery-and-django/#comment-359163)

Aditya Darmawan (https://www.facebook.com/andra.yasha)

There are many tutorials about using Celery in Django but this one is the most applicable!

October 28th, 2013 (https://www.safaribooksonline.com/blog/2012/11/30/up-and-running-with-celery-and-django/#comment-359164)

Dhiraj

Do take note of the `user` setting in the supervisord conf file. I got a `permission denied` error because of it! wasted 15 min :(

December 27th, 2013 (https://www.safaribooksonline.com/blog/2012/11/30/up-and-running-with-celery-and-django/#comment-359165)