EC2 Django Deployment – Step-by-Step

Jeffrey L. Eppinger
Carnegie Mellon University
Updated February 23, 2017

There are probably many ways to configure an EC2 instance to deploy your Django app, but since many people seem to be having trouble finding a consistent set of tutorials, here's a step-by-step guide, at least of how I do it. In this example, I deploy the `addrbook-example` from class.

Provision the EC2 Instance

Go to `aws.amazon.com`:

- Create account
- Create an EC2 instance
  - Choose Ubuntu (current version is 16)
  - T2.micro (whichever size is free)
  - Review & Launch
  - Create Key Pair, download as <name>
- Note: the EC2 console can be used to stop/start your EC2 instance, to find its IP address and its security group, etc.

Connect to your instance with SSH:

- On MAC: `chmod 400 <name>.pem`
- `ssh –i <name>.pem ubuntu@<ip-address>`

Add hostname to hosts file to suppress sudo warnings (optional)

In shell on EC2 Instance, find your "hostname":

- `echo $(hostname)`

Remember the value of the host name for use in the step (Control-C/Command-C, etc):

- The hostname will be something like: `ip-nnn-nn-n-nn`

Then in shell on EC2 Instance, <edit> `/etc/hosts` using `vim` or `Emacs`, etc. The vim editor is already installed. Here's a nice quick reference: http://vim.rtorr.com.

- `sudo <edit> /etc/hosts`
  - Add the hostname to the end of the first line of the file, and save file

```
127.0.0.1 localhost ip-nnn-nn-nn
```

Install Django, Test Sample App

In shell on EC2 Instance:

- `sudo apt-get update`
- `sudo apt-get dist-upgrade`
- `sudo apt-get install python-pip`
- `sudo pip install --upgrade pip` ← Note: the double hyphen
- `sudo pip install django==1.10` ← Note: the double equals
- `sudo reboot`

Allow network access to port 8000

While it's rebooting, in EC2 Console (aws.amazon.com):

- Select your server instance
- Note your instance's security group
- Select Security Groups → <your security group>
    - Click "Inbound" Tab
    - Click Edit → Add Rule
    - Add Custom TCP Rule for Port 8000
    - Save

Note: it can take several minutes (up to five) for AWS to nable port 8000

- `git clone https://github.com/CMU-Web-Application-Development/addrbook-example.git`

- `cd addrbook-example.git`
- `python manage.py migrate`
- `python manage.py runserver 0.0.0.0:8000`

In web browser, visit `http://<ip-address>:8000`

- You should see the class example running, using SQLite for the DB

Install Apache HTTP Server

In shell on EC2 Instance:

- `sudo apt-get install apache2`
- `sudo apt-get install libapache2-mod-wsgi`

In EC2 Console:

- Select your security group
  - Click Edit
  - Change the Custom TCP Rule to HTTP (to enable port 80 instead)
  - Save

In web browser, visit `http://<ip-address>`

- You should see the Apache splash screen

Configure Apache to Serve Django App

In shell on EC2 Instance, edit the Apache config file using emacs or vim or some other editor:

- `sudo <edit> /etc/apache2/apache2.conf`
  - Comment out default mapping for "/" url:

```
#<Directory />
#   Options FollowSymLinks
#   AllowOverride None
#   Require all denied
#</Directory>
```

  - Insert alias for "/" url:

```
WSGIScriptAlias / /home/ubuntu/addrbook-example/webapps/wsgi.py
WSGIPythonPath /home/ubuntu/addrbook-example
```

  - Add permissions for example project directory:

```
<Directory /home/ubuntu/addrbook-example>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
```

  - Save the file

- In shell on EC2 Instance, restart Apache and fix permissions on the directories and files:
    - `cd ~`
    - `sudo chgrp –R www-data addrbook-example`
    - `chmod –R g+w addrbook-example`
    - `sudo apache2ctl restart`

In web browser, visit `http://<ip-address>`

- You should now see example app running under Apache but static files are not working

A note on file permissions: The Apache server handles requests in processes running under a special user called `www-data`. The `chgrp` command, above, puts all your Django project's files into the `www-data` group, allowing Apache access. The `chmod` command gives write permission to the group.

A note on debugging: The Apache server will print errors out in a file called `/var/log/apache2/error.log`. Also, any print statements you've put in your Python code will show up in this error log.

Configure Apache to Serve Static Files

In shell on EC2 Instance, edit the Apache config file using emacs or vim or some other editor:

- `sudo <edit> /etc/apache2/apache2.conf`
    - Add alias and permissions for static folder:

```
Alias /static /home/ubuntu/addrbook-example/addrbook/static

<Directory /home/ubuntu/addrbook-example/addrbook/static>
    Order allow,deny
    Allow from all
</Directory>
```

- `sudo apache2ctl restart`

In web browser, visit `http://<ip-address>`

- Static files should now work

Install and Configure MySQL

In shell on EC2 Instance:

- `sudo apt-get install mysql-server` ← When prompted, set password
- `sudo apt-get install python-dev libmysqlclient-dev`
- `sudo pip install MySQL-python`

- `mysql -u root -p`
  - `mysql> create database addrbook;`
  - `mysql> quit;`

- `<edit> addrbook-example/webapps/settings.py`
  - Change DB config to use MySQL:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'addrbook',
        'USER': 'root',
        'PASSWORD': '<your password>',
    }
}
```

- `python manage.py migrate`

- `sudo apache2ctl restart`

In web browser, visit `http://<ip-address>`

- Model data should now be stored in the database.

To view the data, in the shell on the EC2 Instance:

- `mysql -u root -p`
  - `mysql> use addrbook;`
  - `mysql> show tables;`
  - `mysql> select * from addrbook_entry;`
  - `mysql> select * from auth_user;`
  - `mysql> quit;`

Additional notes:

Many people have migration problems that they only discover when they try to rebuild their database. You may need to remove your old migrations and re-make new ones. If you have unexplainable error messages from the Django ORM, whether in the cloud or locally, try this sequence of commands to completely start from a clean DB with new migrations is:

- `mysql -u root -p`
    - `mysql> drop database addrbook;`
    - `mysql> create database addrbook;`
    - `mysql> quit;`

- Then in the shell:
    - `cd ~/addrbook-example`
    - `rm addrbook/migrations/0*`
    - `python manage.py makemigrations`
    - `python manage.py migrate`

Note: If your DB is in SQLite, you can completely reset your migrations this way:

- `cd …/addrbook-example`
- `rm db.sqlite3`
- `rm addrbook/migrations/0*`
- `python manage.py makemigrations`
- `python manage.py migrate`