EC2 Django Deployment – Step-by-Step

Jeffrey L. Eppinger
Carnegie Mellon University
October 12, 2022

There are 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 `image-example` from class.

Provision the EC2 Instance

Version 2<u>2</u>.04 has some big differences. Use the dropdown menu to select version 2<u>0</u>.04.

Go to `aws.amazon.com`:

- Create account
- Create an EC2 instance
    - Choose Ubuntu x-86 version 20.04
    - T2.micro (whichever size is free)
    - Review & Launch
    - Create Key Pair, download as <name>
        - This will download a PEM file you can use to connect (option #2 below)
- Note: the EC2 console can be used to stop/start your EC2 instance, to find its IP address and its security group, and even to connect to the instance.

Connecting to your EC2 instance

Here are three options you can use to access a shell on your EC2 instance:

Just works

1. In the EC2 console on aws.amazon.com, select your instance and click on Connect => “EC2 Instance Connect”. This lets you use a shell from your browser.

Easier to re-connect

2. Connect to your instance from using SSH (on your laptop) and the key in your downloaded PEM file:
   a) Remove write access to the PEM file (on MAC: `chmod 400 <name>.pem`)
   b) Then: `ssh -i <name>.pem ubuntu@<ip-address>`

Many steps but easiest reconnect & repo cloning

3. Connect to your instance with SSH using your laptop’s public key:
   a) Set up SSH authentication for GitHub as per our Git Quickstart guide.
   b) Connect to your instance using options #1 or #2
      - Using vim (or other editor), add your SSH public key (as an additional line) to your instance’s `~/.ssh/authorized_keys` file
      - Exit the instance’s shell
   c) On your laptop, create (or add to) your `~/.ssh/config` file with these lines:

      ```
      Host <nickname>
        Hostname <ip-address>
        User ubuntu
        ForwardAgent yes
      ```

      Note: you replace <nickname> with a nickname of your choosing.
   d) Now reconnect to your instance this way: `ssh <nickname>`
      - Test that GitHub recognizes your forwarded identity on your EC2 instance using: `ssh -T git@github.com`
      - If this command does not show your identity, return to your computer and run `ssh-add`. Then reconnect to your instance and see if the above command now works.

Install Python and Django

In shell on EC2 Instance, run the following commands:

```
sudo apt update
sudo apt upgrade
sudo -H apt install python3-pip
sudo -H pip3 install django             (or -r requirements.txt)
sudo reboot
```

(For the instructions, below, you cannot use a virtual environment.)

Allow network access to port 8000

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

- Select your server instance
- Under the Security tab, click on your security group
- Select Security Groups → <your security group>
    - Click "Inbound rules" Tab
    - Click "Edit inbound rules" → Add Rule
    - Add Custom TCP Rule for Port 8000 with Source 0.0.0.0/0
    - Save

Careful not to delete Port 22.
That's for your SSH connections.

Note: it can take a few minutes for AWS to enable port 8000

Install Test Application

If using SSH credentials with GitHub use this URL:

```
git@github.com:cmu-webapps/image-example.git
```

Reconnect to your EC2 Instance:

```
git clone https://github.com/cmu-webapps/image-example.git
cd image-example
python3 manage.py makemigrations picture_list
python3 manage.py migrate
```

Now you'll need to edit a your `settings.py`. You can use `vim` or `Emacs`, etc. The `vim` editor is already installed. Here's a nice quick reference: http://vim.rtorr.com.

<edit> `webapps/settings.py`
- Add your IP address to `ALLOWED_HOSTS` list
- Save the file and exit the editor

```
python3 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
- Upload a picture so that the images folder is created

Install Apache HTTP Server

In shell on EC2 Instance, stop Django (type Control-C) and then:

```
sudo apt install apache2
sudo apt install libapache2-mod-wsgi-py3
```

In EC2 Console:

- Select your security group
  - Click "Edit inbound rules"
  - Click "Edit inbound rules" → Add Rule → Add TCP Rule for HTTP (Port 80)
  - Save

You may remove port 8000 or keep it open if you wish to continue to "runserver" from the command line for debugging purposes.

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

- You should see the Apache splash screen

Configure Apache to Serve Django App

In the 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 (or delete) default mapping for "/" url – it's around line 159:

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

  - Insert alias for "/" url:

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

  - Add permissions for example project directory:

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

  - Save the file

- In shell on EC2 Instance, fix permissions on the directories and files and restart Apache:

```
cd ~/image-example
sudo chgrp -R www-data .      <= Note the “.”
chmod -R g+w .                <= Note the “.”
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. If you did not create an image when testing your app running on port 8000 with the development environment, you will need to create the media folder and then run the `chgrp` and `chmod` commands on the media folder. (The commands above apply to the media folder because the example creates the media folder in the project directory.)

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.

(Continue to configure static files)

Configure Apache to Serve Static Files

Django is designed so as not to serve your static files. The procedure is to collect all your static files into a new directory and then use Apache (or some other mechanism) to serve these files.

<edit> `webapps/settings.py`

- Add a line (near the bottom) to specify where to collect the static files:

```
STATIC_ROOT = BASE_DIR / 'collected_static'
```

- Save the file and exit the editor

Then instruct Django to collect the static files:

```
python3 manage.py collectstatic
```

Check out the `collected_static` folder. In there, you'll see all your static files.

Note: If you have just one application, you can skip running "collectstatic" and just alias /static to your app's static folder and set the access parameters for this folder (e.g., you can just use /home/ubuntu/<andrewid>/hw7/socialnetwork/static for alias and directory).

Now edit the Apache config file:

```
sudo <edit> /etc/apache2/apache2.conf
```

- Add alias and permissions for static folder:

```
Alias /static /home/ubuntu/image-example/collected_static

<Directory /home/ubuntu/image-example/collected_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 the shell on EC2 Instance:

```
sudo apt install mysql-server
sudo apt install libmysqlclient-dev
sudo -H pip3 install mysqlclient

sudo mysql
      create user ''@'localhost' identified by '';
      grant all privileges on *.* to ''@'localhost';
      quit;
```

You may specify a username and password if you wish.

If you set username and password, use this command:
```
mysql -u <user> -p
```
to be prompted for your password.

```
mysql
      create database django character set utf8mb4;
      quit;
```

Oddly, MySQL's utf8 character set only handles 3-byte Unicode. They have added utf8mb4 to handle 4-byte.

```
cd image-example
<edit> webapps/settings.py
```

- Change DB config to use MySQL:

```
DATABASES = {
    'default': {
        'OPTIONS': {'charset': 'utf8mb4'},
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django',
        'USER': '',
        'PASSWORD': '',

    }
}
```

If you set username and password, enter them here.

```
python3 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
      use django;
      show tables;
      select * from picture_list_item;
      quit;
```

If you set username and password, use this command:
```
mysql -u <user> -p
```
to be prompted for your password.
To use full Unicode from the mysql command line to run (in mysql):
```
set charset utf8mb4;
```

Removing Secrets from your Repo

You should not store secret data in your repo, such as passwords and other encryption keys. To for EC2 deployments, you should put the secrets in a `config.ini` file like this:

```
[Django]
secret=p8xyz5&fxyzhyb5fxyz-@t#g!2=_yh_#^0y_9xyzk7+tgq+ts
```

Note that the % character is special in `.ini` files, so if you have any `%` characters in your secret, you must double it (use %%).

Then read the secrets in from your `config.ini` file using the Python `ConfigParser` class. For Django's `SECRET_KEY` in `settings.py`, it would look like this:

```
from configparser import ConfigParser
...
CONFIG = ConfigParser()
CONFIG.read(BASE_DIR / "config.ini")
...
SECRET_KEY = CONFIG.get("Django", "secret")
```

Do the same for any other secrets, as well as changeable configuration parameter that you don't want to hard code into your Python files, such as database usernames and passwords, authentication keys to cloud-based services, etc. You can have multiple sections in your .ini file, but starting each section with a new header in square brackets, like this:

```
[MYSQL]
user=root
password=monkeybreath
```

Since the `config.ini` file, must not be in your repo, we document its format with a "sample" file, such as `config.ini.sample`, which doesn't contain the secrets. Example:

```
# To generate a new secret key, use get_random_secret_key():
#    from django.core.management.utils import get_random_secret_key
#    print(get_random_secret_key())
# Warning: The % character is special to ConfigParser - use %%
# Warning: Changing secret keys invalidates existing sessions,
#          so may need to delete your DB tables and re-migrate

[Django]
secret=
```

Additional notes:

To upload files from your laptop to your server (e.g., a config.ini file), you can use `sftp`:

```
sftp –i <name>.pem ubuntu@<ip-address>        (or sftp <nickname>)
    cd <andrewid>/hw7
    put config.ini
    exit
```

To refresh your deployed app with changes that have been pushed to GitHub from elsewhere, simply run:

```
git pull
sudo apache2ctl restart
```

If you've added files to your repo and they have arrived on EC2 because of your pull, you will need to set their file permissions so they are readable by Apache by re-running:

```
cd ~
sudo chgrp -R www-data image-example
chmod -R g+w image-example
```

If you need to delete all the data in your MySQL database, delete and remigrate with:

```
mysql
    drop database django;
    create database django character set utf8mb4;
    quit;
python3 manage.py migrate
```

If you need to completely reset your migrations:

```
mysql
    drop database django;
    create database django character set utf8mb4;
    quit;
rm -fr picture_list/migrations
python3 manage.py makemigrations picture_list
python3 manage.py migrate
```

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

```
rm db.sqlite3
rm -fr picture_list/migrations
python3 manage.py makemigrations picture_list
python3 manage.py migrate
```

(Be careful with the `-fr` option on the `rm` command. It recursively deletes files without any prompting, so you need to be careful to get it right. But it should be OK since all your code is safely in your repo on GitHub.)