EC2 Django Deployment – Step-by-Step

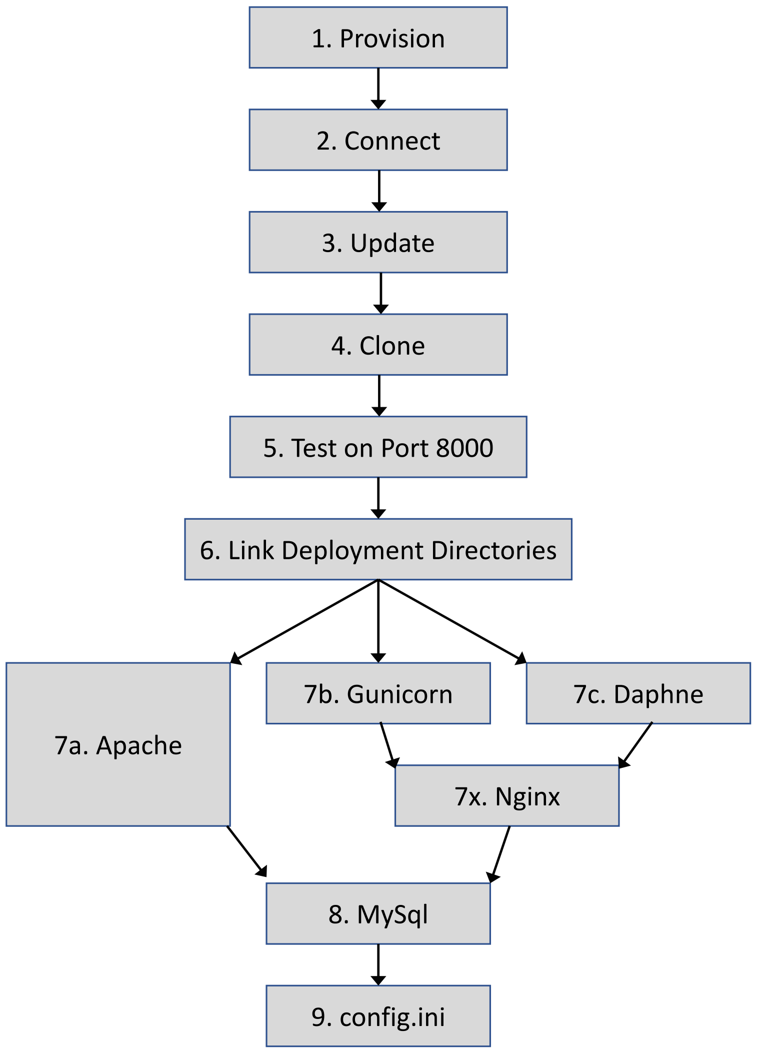
Web Application Development Course
Carnegie Mellon University
February 26, 2024

There are many ways to configure an EC2 instance to deploy your Django app. See the Django documentation for a discussion of the many options:

- https://docs.djangoproject.com/en/4.2/howto/deployment/

This guide outlines three options for deploying the class examples or your homework. You must use the Daphne option when deploying applications that use WebSockets. For non-WebSockets applications, it will be simpler to deploy using Apache or Gunicorn.

Here is an outline of the steps you need to deploy your application using this guide.



1. Provision the EC2 Instance

Version 22.04 has some big differences. Use the dropdown menu to select version 20.04.

Go to `aws.amazon.com`:

- Create account
- Create an EC2 Ubuntu Instance. Use version 20.04.
  - Review & Launch
  - Create Key Pair, download as <name>
    - This will download a PEM file you can use to connect (option #2 below)
- Enable ports 80 and 8000
  - 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 HTTP Rule (Port 80) with Source 0.0.0.0/0
    - 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: 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.

2. Connecting to your EC2 instance

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

Just works

a. 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

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

Many steps but easiest reconnect & repo cloning

c. Connect to your instance with SSH using your laptop's public key:
   1. Set up SSH authentication for GitHub as per our Git Quickstart guide.
   2. Connect to your instance using options 2a or 2b, above.
      - 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
   3. On your laptop, create (or add to) your `~/.ssh/config` file with these lines:

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

   4. 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.

3. Update the Software on your EC2 Instance

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 -U pip
sudo reboot
```

(It's easier not to use a virtual environment on the EC2 instance. The deployment software will use the normal environment and you're not running software for other course on this server.)

4. Cloning Repos

Just works

If you connected to your EC2 instance using methods 2a or 2b, you can use either:

- Option 1: Use an HTTPS URL (https://github.com/cmu-webapps/...). You will be prompted for your GitHub username and "password". You must use a personal token for the password.
- Option 2: Create a new public/private key pair on the your EC2 instance and register the public key as (another one of) your public key(s) on GitHub. (See the SSH Credentials Section in the Git Quickstart guide for the steps.)

If you set up Agent Forwarding when you connected to your EC2 instance using method #2c above, you can use:

- Option 3: `git clone` commands with SSH paths (git@github.com:cmu-webapps/...)

After setup, easiest to use if you access GitHub often

Examples:

```
git clone https://github.com/cmu-webapps/image-example.git
git clone https://github.com/cmu-webapps/websockets-example.git
git clone https://github.com/cmu-webapps/<andrewid>.git
```

or

```
git clone git@github.com:cmu-webapps/image-example.git
git clone git@github.com:cmu-webapps/websockets-example.git
git clone git@github.com:cmu-webapps/<andrewid>.git
```

5. Test on Port 8000

1. Set the current working directory to the folder for the Django project you want to run.

   Examples:

   ```
   cd image-example
   ```

   or

   ```
   cd <andrewid>/hw7
   ```

2. Install Django and any other Python packages the project uses.

   Examples:

   ```
   sudo -H pip3 install django
   ```

   or

   ```
   sudo -H pip3 install -r requirements.txt
   ```

3. Initialize the database:

   ```
   python3 manage.py makemigrations
   python3 manage.py migrate
   ```

4. Add your IP address to ALLOWED_HOSTS in settings.py, if necessary.

5. Start the server. (Don't forget 0.0.0.0 so it listens for external requests.)

   ```
   python3 manage.py runserver 0.0.0.0:8000
   ```

6. In your web browser, visit `http://<ip-address>:8000`
   - You should see the Django application running, using SQLite for the DB

7. Stop Django (type Control-C).

6. Link Deployment Folders

To simplify the steps in the various deployment options discussed the Sections 7, below, we will make symbolic links to your Django application folder and your static folder (or folders). This will allow you to copy and paste the configurations in Sections 7 with less editing.

1. Create a symbolic link to your Django project folder. This is the folder that contains manage.py. The link must be called "project".

   ```
   cd ~
   ln -s <project-folder> project
   ```

   For a course example, such as the image-example, the `<project-folder>` would be `image-example`

   For HW7, `<project-folder>` would be `<andrewid>/hw7`

2. Create a symbolic link to your static file folder.

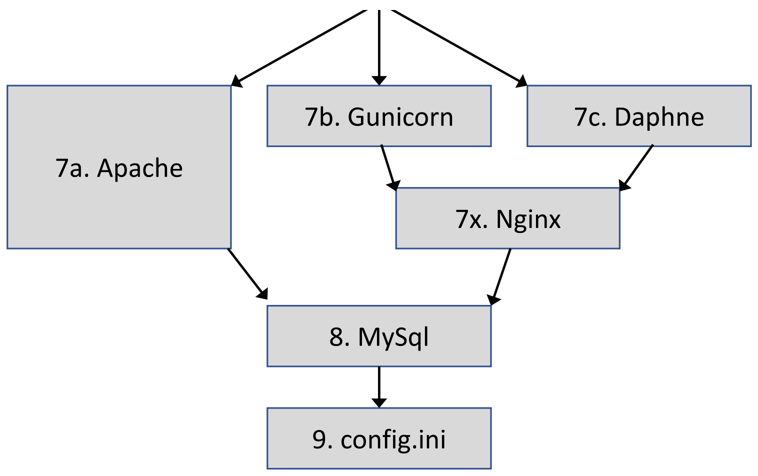
   ```
   cd ~
   ln -s <project-folder>/<appname>/static
   ```

   Or, if you have multiple applications in your project, you should create a `static` folder and then link each applications' static subfolders into this folder. For example, the image-example has the `picture_list` application as well as the `welcome` application. So it would be like this:

   ```
   mkdir ~/static
   cd ~/static
   ln -s ~/image-example/picture_list/static/picture_list
   ln -s ~/image-example/welcome/static/welcome
   ```

If done correctly, running `ls ~/project` should show the files at the top of your project folder and `ls ~/static` should show your static files (or if you repeated the appname in the applications' static folders, then `ls ~/static/<appname>` should show your static files).

7. Choose your deployment server software



You will need to select deployment servers. The three options covered in this guide are:

7a	Apache HTTP Server with the mod_wsgi extension as the WSGI application server
7b + 7x	Gunicorn as the WSGI application server running behind the Nginx web server
7c + 7x	Daphne as the ASGI application server running behind the Nginx web server

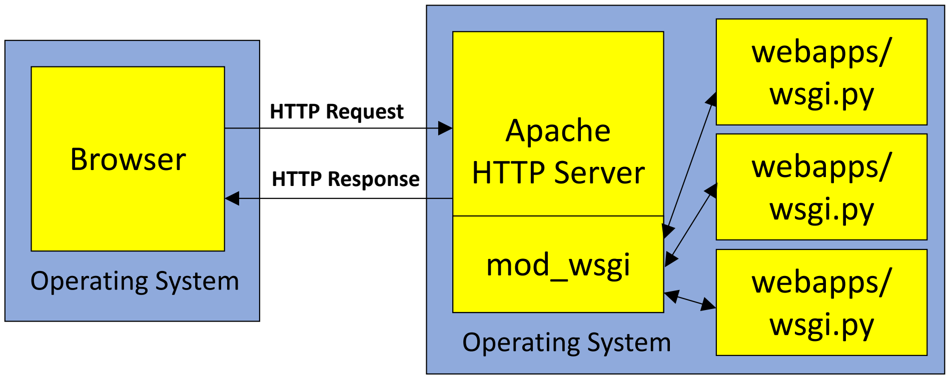
Choose one of these options. For non-WebSockets applications, we typically use (7a) or (7b). For applications using WebSockets, you must use (7c). We expect to update this guide soon to provide section 7c (the instructions for using Daphne for websockets).

For documentation on Django deployment, see:

- https://docs.djangoproject.com/en/4.2/howto/deployment/

7a. Apache HTTP Server + mod_wsgi

One option for deployment server is to use Apache to listen on port 80 for incoming HTTP (or HTTPS) requests. The Apache server will directly handle static files and will forward other requests to Django running under the mod_wsgi which acts as the WSGI Application Server.



7a.1 Install Apache and mod_wsgi

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

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

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

- You should see the Apache splash screen

7a.2 Configure Apache

Create a configuration file for Django:

```
sudo vim /etc/apache2/sites-available/django.conf
```

Insert (using copy/paste) the text on the next page into this new file.

Helpful `vim` commands:	
`i`	insert mode (use arrow keys to move, use <esc> to exit)
`:x` <ret>	save file and exit vim

```
User ubuntu
Group ubuntu

WSGIPythonPath /home/ubuntu/project

<VirtualHost *:80>
    WSGIScriptAlias / /home/ubuntu/project/webapps/wsgi.py

    <Directory /home/ubuntu/project>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    Alias /static /home/ubuntu/static

    <Directory /home/ubuntu/static>
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Download text of this config file from here

Then disable the Apache default configuration (by removing the link in sites-enabled) and enable the Django configuration (by creating a link in sites-enabled):

```
cd /etc/apache2/sites-enabled
sudo rm 000-default.conf
sudo ln -s ../sites-available/django.conf
```

Finally, comment out permissions for the root path in the main Apache config file (near line 159):

```
sudo vim /etc/apache2/apache2.conf

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

Helpful `vim` command:
`:159` <return>   go to line 159

Then restart Apache:

```
sudo apache2ctl restart
```

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

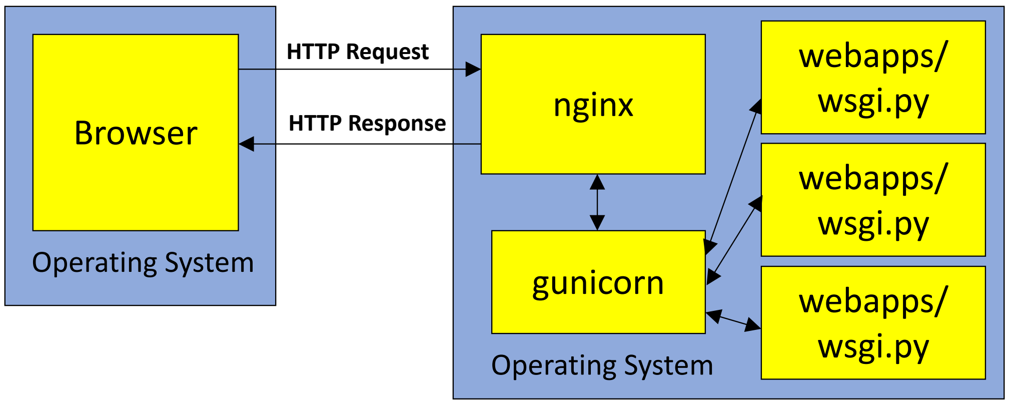
- You should now see example app running under Apache.

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.

Next:
Go to
Section 8
(MySQL)

7b Nginx server + Gunicorn

One option for deployment server is to use nginx to listen on port 80 for incoming HTTP (or HTTPS) requests. The Nginx server will directly handle static files and will forward other requests to Django running under the Gunicorn WSGI Application Server.



7b.1 Install and Test Gunicorn

Install Gunicorn:

```
sudo -H pip3 install gunicorn
```

Now we'll ensure that gunicorn is working and serve our app using the following command:

```
cd ~/project
gunicorn webapps.wsgi:application --bind 0.0.0.0:8000
```

Again, visit `http://<ip-address>:8000` to verify your application is running.

Note that the static files will not be served. However, they may be cached in your browser. (So if you hard refresh your page, your CSS and any JS code won't work.).

Type CTRL-C to stop Gunicorn.

7b.2 Daemonizing gunicorn using supervisord

We need to daemonize the `gunicorn_cfg` so that it will start up everytime the server reboots. For this guide, we will be using `supervisord` to take care of this:

First install `supervisord` using the command:

```
sudo apt-get install supervisor
```

All supervisor configurations (.conf files) are kept in `/etc/supervisor/conf.d/`

```
cd /etc/supervisor/conf.d/
```

Helpful `vim` commands:

`i`	insert mode (use arrow keys to move, use <esc> to exit)
`:x` <ret>	save file and exit vim

Let's create a service:

```
sudo vim django.conf
```

Press 'i' to insert text and copy-paste the following:

```
[program:django]
directory=/home/ubuntu/project
autostart=true
autorestart=true
user=ubuntu

# Project
PROJECT_DIR=/home/ubuntu/project
DJANGO_SETTINGS_MODULE=webapps.settings

export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$PROJECT_DIR:$PYTHONPATH

DJANGO_ARGS="--name=django --workers=3 --bind=localhost:8000"
LOG_ARGS="--log-level=debug --log-file=/home/ubuntu/wsgi.log"

command=gunicorn webapps.wsgi:application $DJANGO_ARGS $LOG_ARGS
```

Download text of this config file from here

Save the configuration and exit. Tell the supervisor to update its registry of services:

```
sudo supervisorctl reread
sudo supervisorctl update
```

You can check its status by the following command:

```
sudo supervisorctl status django:*
```

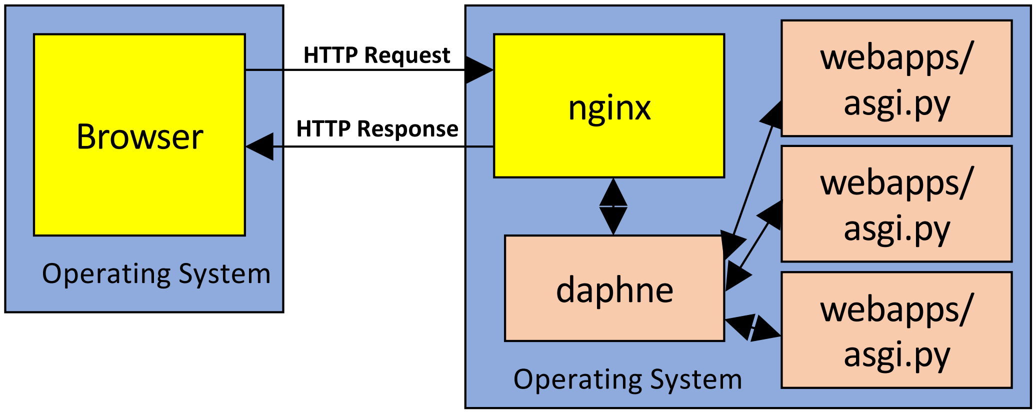
And you can restart Gunicorn, which you'll need to do if you make changes to your Django app:

```
sudo supervisorctl restart django:*
```

Go to Section 7x (Nginx)

7c. Nginx + Daphne (required for WebSocket/Channels deployments)

For web applications that use ASGI, we will use an ASGI server (*Daphne*) in combination with Nginx for deployment. Think of *Daphne* as a replacement for *Gunicorn* (in section 7b) for applications that use ASGI (for WebSocket apps) instead of WSGI (for traditional web apps).



Note: Daphne can be used for traditional (WSGI) web apps as well! It supports both ASGI & WSGI apps, so this guide will also work for apps that do not use WebSockets.

7c.1 Install and test Daphne

You probably installed Daphne when you installed the packages to test on port 8000, but if not:

```
sudo -H pip3 install daphne
```

Now we'll ensure that Daphne works and serve our app using the following command:

```
cd ~/project
daphne webapps.asgi:application -b 0.0.0.0 -p 8000
```

Visit `http://<ip-address>:8000` to verify that the application is running.

Note that the static files will not be served yet. (So you're CSS and any JS code won't work.)

Type Ctrl-C to stop Daphne.

If running WebSockets, see Section 11 for additional configuration to support communication among multiple processes.

7c.2 Daemonizing Daphne using supervisord

First install `supervisord` using the command

```
sudo apt install supervisor
```

All supervisor configurations (.conf files) are kept in `/etc/supervisor/conf.d/`

```
cd /etc/supervisor/conf.d/
```

Let's create a configuration to run make Daphne run Django:

```
sudo vim django.conf
```

Download text of this config file from here

```
[fcgi-program:django]
# TCP socket used by Nginx backend upstream
socket=tcp://localhost:8000

user=ubuntu
directory=/home/ubuntu/project

# Each process needs to have a separate socket file, so we use process_num
DJANGO_ARGS="-u /run/daphne/daphne%(process_num)d.sock --fd 0"
LOG_ARGS="--access-log - --proxy-headers"
command=daphne webapps.asgi:application -u /run/daphne/daphne%(process_num)d.sock
--fd 0 --access-log - --proxy-headers

# Number of processes to startup, roughly the number of CPUs you have
numprocs=3

# Give each process a unique name so they can be told apart
process_name=asgi%(process_num)d

# Choose where you want your log to go
stdout_logfile=/home/ubuntu/asgi.log
redirect_stderr=true
autostart=true
autorestart=true
```

Create the folder for Daphne sockets to use for interprocess communication:

```
sudo mkdir /run/daphne
sudo chown ubuntu /run/daphne
```

Save the configuration and exit. Tell the supervisor to update its registry of services:

```
sudo supervisorctl reread
sudo supervisorctl update
```

You can check its status by the following command:

```
sudo supervisorctl status django:*
```

You can restart Daphne, which you need to do when changing your Django app:

```
sudo supervisorctl restart django:*
```

7x. Install Nginx Server

If you're using Gunicorn or Daphne as the Application Server to run your Django project, follow the instructions in this section to use Nginx to listen on port 80 for incoming requests and to route them to either Gunicorn or Daphne, as well as to handle requests for static file.

(If you are installed Apache, using the instructions in Section 7a, skip this section.)

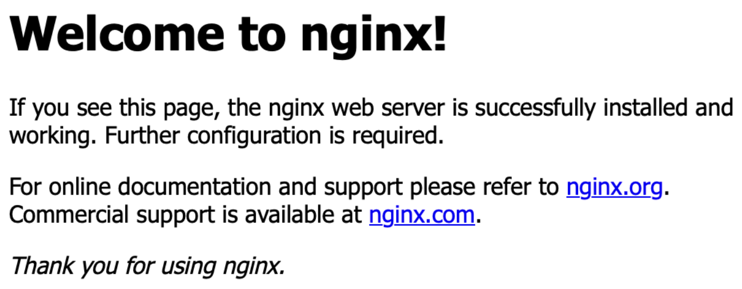
7x.1 Install Nginx Server

In shell on EC2 Instance:

```
sudo apt install nginx
```

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

- You should see the Nginx splash screen



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

7x.2 Configure Nginx to Serve Django App and static files

Before we setup nginx for our app, we must get rid of the default nginx config.

```
sudo rm /etc/nginx/sites-enabled/default
```

Now let's setup our own config. Paste the text in the box below into this file. Save and exit.

```
sudo vim /etc/nginx/sites-available/django
```

```
upstream django_app {
    server localhost:8000;
}

server {
    listen 80;
    server_name my-domain-name.com; # optional (if you have a domain name)

    location /static/ {
        alias /home/ubuntu/static/;
    }

    location / {
        try_files $uri @proxy_to_app;
    }

    location @proxy_to_app {
        proxy_pass http://django_app;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        # More info: http://en.wikipedia.org/wiki/X-Forwarded-For
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Host $server_name;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Download text of this config file from here

Now we need to create a symbolic link to this file in the sites-enabled folder:

```
cd /etc/nginx/sites-enabled
sudo ln -s ../sites-available/django
```

Restart Nginx:

```
sudo service nginx restart
```

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

- You should now see your Django app running via Nginx
- Static files should work

8. Install and Configure MySQL

In the shell on EC2 Instance:

```
   sudo apt install pkg-config
   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;

   vim ~/project/webapps/settings.py
```

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

- 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

   <restart application server>
```

If Apache use:

```
  sudo apache2ctl restart
```

If Gunicorn or Daphne use:

```
  sudo supervisorctl restart django:*
```

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 <table_name>;
         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;
```

9. 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=
```

10. 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
<restart application server>
```

If Apache use:
```
  sudo apache2ctl restart
```
If Gunicorn or Daphne use:
```
  sudo supervisorctl restart django:*
```

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
```

For HW7, replace `picture_list` with `socialnetwork`

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

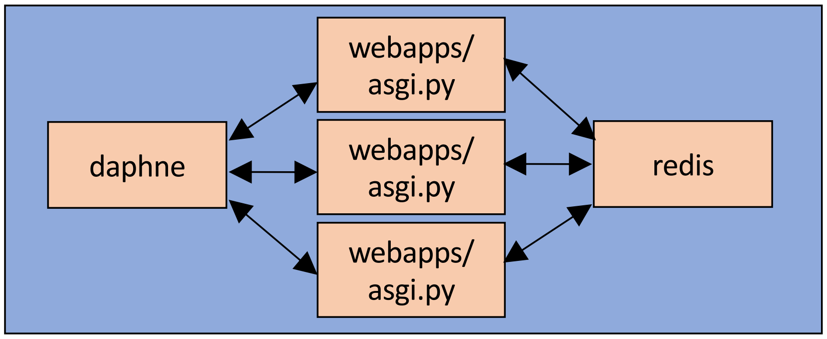
```
rm db.sqlite3
rm -fr picture_list/migrations
python3 manage.py makemigrations picture_list
python3 manage.py migrate
```

(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.)

11. Additional Configuration Steps for WebSockets Deployment

When running with the Django development server, there is only one process running your web application so we can use the Channels `InMemoryChannelLayer`.

When deploying a WebSockets application running with multiple Django processes, you need to use a Channels backend which will support distribution of the WebSockets messages across all the Django processes. We recommend using the `RedisChannelLayer` and running the Redis server.



Install the Redis server (which will also start it):

```
    sudo apt install redis-server
```

Install the Channels backend for Redis:

```
    sudo -H pip3 install channels_redis
```

Modify your settings.py file to configure Channels to use Redis:

```
    CHANNEL_LAYERS = {
        "default": {
            "BACKEND": "channels_redis.core.RedisChannelLayer",
            "CONFIG": {
                "hosts": [("localhost", 6379)],
            },
        },
    }
```