Django Deployment Cheat Sheet (Apache and mod_wsgi on Ubuntu 16.04)

1. sudo apt-get update
2. sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3

Configure a Python Virtual Environment

1 sudo pip3 install virtualenv

Create a directory where you wish to keep your project and move into the directory:

mkdir ~/myproject

cd ~/myproject

*Within the project directory, create a Python virt*ual environment by typing:

virtualenv myproject

source myprojectenv/bin/activate

Install Django into virtual environment with the local instance of pip:

pip install django==1.11.7

Create and Configure a New Django Project

django-admin.py startproject myproject .

…...Note the dot at the end that tells Django to create the files in the current directory

Adjust the Project Settings

1. **nano myproject/settings.py**

2. Begin by finding the ALLOWED_HOSTS line. Inside the square brackets, enter your server's public IP address, domain name or both. Each value should be wrapped in quotes and separated by a comma like a normal Python list: ***ALLOWED_HOSTS = ["server_domain_or_IP"]***

Configure STATIC_ROOT

```
STATIC_URL = '/static/'

STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

…………..Save and close the file when you are finished…………

Complete Initial Project Setup

```
cd ~/myproject

pyhon3 manage.py makemigrations

python3 manage.py migrate

python3 manage.py createsuperuser

python3 manage.py collectstatic
```

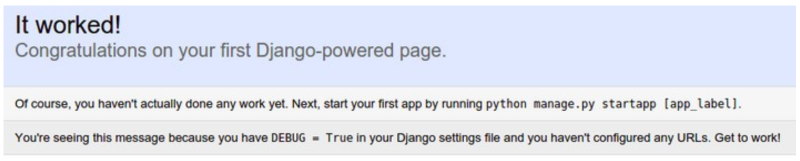
Now, we can adjust our firewall settings to allow traffic to our Django development server, which we'll run on port 8000 if you have UFW firewall eanbled

Allow connections to the development server by typing:

```
sudo ufw allow 8000
```

Finally, you can test your project by starting up the Django development server with this command: *python3 manage.py runserver 0.0.0.0:8000*

Go to http://server_domain_or_IP:8000 to see Django default page

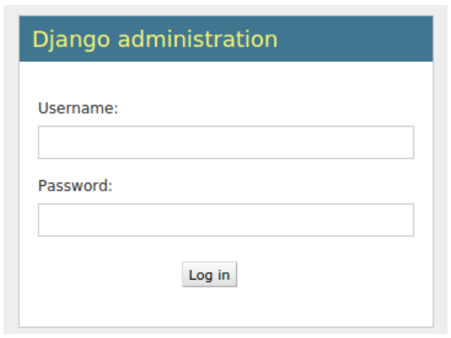


It worked!
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running python manage.py startapp [app_label].

You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

If you append /admin to the end of the URL in the address bar, you will be prompted for the administrative username and password you created with the createsuperuser command:



When you are finished exploring, hit CTRL-C in the terminal window to shut down the development server.

We're now done with Django for the time being, so we can back out of our virtual environment by typing: *deactivate*

Configure Apache

Now that your Django project is working, we can configure Apache as a front end. Client connections that it receives will be translated into the WSGI format that the Django application expects using the mod_wsgi module. This should have been automatically enabled upon installation earlier.

To configure the WSGI pass, we'll need to edit the default virtual host file:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

We can keep the directives that are already present in the file. We just need to add some additional items. To start, let's configure the static files. We will set up the alias and then grant access to the directory in question with a directory block:

```
<VirtualHost *:80>

    Alias /static /home/sammy/myproject/static

       <Directory /home/sammy/myproject/static>
            Require all granted
       </Directory>

</VirtualHost>
```

Next, we'll grant access to the wsgi.py file within the second level project directory where the Django code is stored.

```
<VirtualHost *:80>

   . . .

    Alias /static /home/sammy/myproject/static

    <Directory /home/sammy/myproject/static>
        Require all granted
    </Directory>

    <Directory /home/sammy/myproject/myproject>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>
</VirtualHost>
```

We'll use daemon mode to run the WSGI process by using the ***WSGIDaemonProcess*** directive to set this up.

This directive takes an ***arbitrary name for the process***. We'll use myproject to stay consistent. Afterwards, ***we set up the Python home*** where Apache can find all of the components that may be required. Since we used a virtual environment, we can ***point this directly to our base virtual environment directory***. Afterwards, we set the ***Python path to point to the base of our Django project***.

Next, we need to specify the ***process group***. This should ***point to the same name*** we selected for the ***WSGIDaemonProcess directive (myproject in our case)***. Finally, we need to set the script alias so that Apache will pass requests for the root domain to the `wsgi.py` file:

```
<VirtualHost *:80>
    . . .

    Alias /static /home/sammy/myproject/static
    <Directory /home/sammy/myproject/static>
        Require all granted
    </Directory>

    <Directory /home/sammy/myproject/myproject>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess myproject python-home=/home/sammy/myproject/myprojectenv
                  python-path=/home/sammy/myproject
    WSGIProcessGroup myproject
    WSGIScriptAlias / /home/sammy/myproject/myproject/wsgi.py

</VirtualHost>
```

Save and close

Wrapping Up Some Permissions Issues

If you are using the SQLite database, you need to allow the Apache process access to this file. Change the permissions so that the group owner of the database can read and write. The database file is called `db.sqlite3` *by default and it should be located in your base project directory:*

```
chmod 664 ~/myproject/db.sqlite3
```

we *need to give the group Apache runs under, the www-data group, group ownership of the file:*

```
sudo chown :www-data ~/myproject/db.sqlite3
```

In order to write to the file, we also need to give the Apache group ownership over the database's parent directory:

```
sudo chown :www-data ~/myproject
```

Adjust our firewall again. We no longer need port 8000 open since we are proxying through Apache, so we can remove that rule. We can then add an exception to allow traffic to the Apache process:

```
sudo ufw delete allow 8000
```

```
sudo ufw allow 'Apache Full'
```

Check your Apache files to make sure you did not make any syntax errors:

```
sudo apache2ctl configtest
```

if you see.. Syntax OK at the end of the output then all has gone well

restart the Apache service to implement the changes you made. Restart Apache by typing:

```
sudo systemctl restart apache2
```

You should now be able to access your Django site by going to your server's domain name or IP address without specifying a port. The regular site and the admin interface should function as expected.

If you have a domain name for your application, the easiest way to secure your application is with a free SSL certificate from Let's Encrypt. Follow our Let's Encrypt guide for Apache on 16.04 to learn how to set this up.

If you **do not** have a domain name for your application and are using this for your own purposes or for testing, you can always create a self-signed certificate. You can learn how to set this up with our guide on creating self-signed SSL certificates for Apache on Ubuntu 16.04.