How to Deploy Django Applications on AWS EC2 Using Apache

Tahseen Rahman
May 28, 2019 · 8 min read

Cloud is about how you do computing, not where you do computing.

What is the end goal?

The end goal is to connect the Django Web Server Gateway Interface (WSGI) to Apache server to execute python code in Django application, in a production environment using AWS EC2 as a cloud service.

Why use the Apache Server?

Using Apache as a web server for production has its pros, such as security and handling traffic effectively, and to keep the server always up and running.

Outline:

1) Launching an EC2 instance
2) Connecting the EC2 instance on local machine
3) Configure EC2 instance (Install Apache2, pip, apache module for wsgi, and virtual environment)
4) Clone the Django Application Repository.
5) Create a virtual Environment
6) Install the necessary packages
7)Test run the application
8) Configure Apache to point to our django application

Let's get the basics out of the way.

Photo by Stanley Dai on Unsplash

What's the Django Framework?

Django is a Python web-framework used to develop web applications. It is used to combine HTML/CSS/Javascript files for front-end, and python for the back-end. Its inbuilt lightweight server interacts with WSGI to serve files.

What is Web Server Gateway Interface?

Web Server Gateway Interface (WSGI)is just an ***interface*** specification with which, the ***server and application communicate.***
A WSGI server (meaning WSGI compliant) only ***receives the request from the client, passes it to the application and then sends the response*** returned by the application to the client. It is only used for Python Programming Language.

An HTTP web server's primary task is to receive a request, and serve HTTP files in response. **Apache** is an example of an HTTP web server.
The caveat here is that that Apache at its core, is designed to serve only HTTP requests. This is where different modules for Apache comes in. These modules help Apache perform various functions.

A module called **mod-wsgi** will be used to help **Apache communicate with WSGI interface** in our Django application.

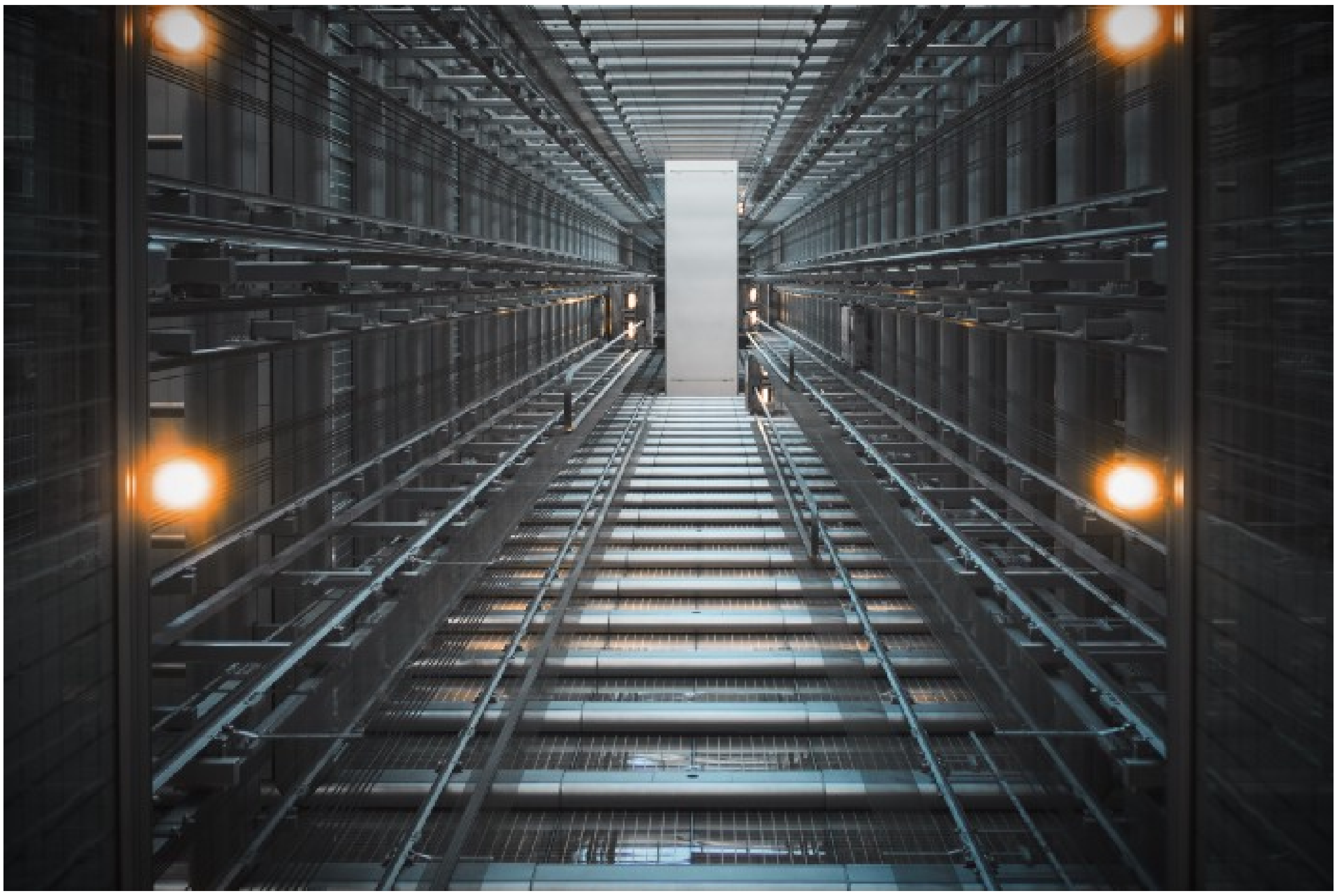


Photo by Robert V. Ruggiero on Unsplash

Why Apache?

There are several other web servers available with the most prominent one being Nginx(Engine-X). **So why choose Apache?**

- Nginx is event-driven while **Apache uses one thread per request**. This theoretically makes Nginx faster but real-world performance between both shows a negligible difference.
- The tests show that in one second Nginx might be able to serve 15000 static files whereas Apache may only serve 3000 with the same hardware.
  Django, on the other hand, might only be able to serve 50 requests in one second. So, if it takes Apache 0.3 ms to proxy a request to Django app, the Django app might need 20ms to respond to the request. In the same scenario, Nginx might take 0.06ms to proxy a request to Django app. So the total time for Apache comes out to be 20.3ms whereas Nginx takes 20.06ms which is very close to one another. A human might not even notice this slight difference.
- Apache has better documentation support as compared to Nginx.
- Apache is an older player, free and open source for more than two decades now. This has gathered support from a huge community comprising of developers.
- Apache has support for a lot of custom modules for security, request handling, SSL termination, etc which in turn helps in faster and efficient development.
- If one is already familiar with Apache, then it’s a better choice and will be less time-consuming.
- Both Nginx or Apache can be used to achieve our goal, it depends on one’s needs and ease of use.

AWS Elastic Compute Cloud (EC2)

EC2 is a web service interface that provides virtual machines which can be tailored to our needs in AWS cloud. It is designed for developers to have complete control over web-scaling and computing resources.

. . .

Launching an AWS EC2 Instance

To use AWS services, one needs to sign up on aws.awazon.com.
After completing the sign-up process and logging in, navigate to the services page and choose *EC2* from the *Compute* sub-menu.

In the EC2 dashboard, the ***Launch Instance*** portal will set up a new instance.

- Choose an Amazon Machine Image(AMI)
  In the free tier option,
  Ubuntu Server 18.04 LTS (HVM), SSD Volume Type — ami-0a313d6098716f372 (64-bit x86) is used for the purpose of this article.
- Configure instance types and Security Groups. Make sure to add at least the local machine's IP to access the EC2 machine's public DNS.
  If needed, Security Group can be configured to allow requests from anywhere.
- A key pair is generated after launching the instance. This key-pair file is to be downloaded on the local system as it helps in connecting to EC2 using SSH.

. . .

Connecting to EC2

Copy the SSH command, which will be used to access the EC2 instance, using the terminal.

Windows users might have to use **Putty** to connect using SSH.

- In the local machine, ***navigate to the directory in the terminal where the key pair file was downloaded and is currently present***.
- In the terminal ***paste the SSH command*** that was copied previously and ***press yes*** if prompted with some access permission(*which is basically to add your local machine as a trusted host)*.

. . .

Setting up the EC2 instance

The Ubuntu EC2 instance needs to be configured to achieve the goal.
Type the following two commands.

```
sudo apt-get update
```

```
sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3
```

The first command is to update ***apt package manager*** in Ubuntu. The second command installs apache2, pip3, and mod-wsgi(module used in Apache, for running Django server, and will be enabled automatically upon installation).

Here sudo is used to provide root access whenever necessary.

Install virtualenv using the following command.

```
sudo pip3 install virtualenv
```

A **virtual environment** is a tool that helps to keep dependencies required by different projects separate, by creating isolated python **virtual environments** for them, and it doesn't affect the dependencies installed on the OS level.

- ***Create a directory called django*** using the command: `mkdir django`
- Use `cd django` command to change the current directory.
  Here we will set up our virtual environment and also clone our Django application repository.

Photo by Georgie Cobbs on Unsplash

The following commands will create **the virtual environment with python3, and clone the GitHub repository.**
In this article, the repository name is taken as "myproject".

```
virtualenv myprojectenv
git clone "https://github.com/<username>/<repository name>.git"
```

> *GitHub is a web-based version-control and collaboration platform for software developers.*

. . .

Running the Django development server

Activate the virtual environment with the command:

```
source myprojectenv/bin/activate
```

Use the following commands to edit the ***settings.py*** file:

```
cd myproject

sudo vi myproject/settings.py
```

> The second command will open up the settings.py file in *vim* editor.
> Press **I** to insert in the file, and insert the following line at the end of the file.
>
> ```
> STATIC_ROOT = os.path.join(BASE_DIR, "static/")
> ```
>
> *This line specifies the directory where all the static files are located.*
>
> **Note: Add instance's DNS name/IP to "Allowed Hosts" in settings.py**
>
> ```
> ALLOWED_HOSTS=['EC2_DNS_NAME']
> ```
>
> Press *Esc* key and type ***:wq*** to save the file.

To install the necessary requirements, run the following command:

```
pip install -r requirements.txt
```

Note: A repository may or may not contain requirements file. If not, please install the necessary dependencies to run the Django app without any errors. The most important dependency would be Django which can be installed using pip.

To install Django, run the following command:

```
pip install django
```

Use the following commands to migrate the database, get static files ready and to test run the Django inbuilt development server.

```
python manage.py makemigrations

python manage.py migrate

python manage.py collectstatic

python manage.py runserver 0.0.0.0:8000
```

In a browser, navigate to public DNS of the instance, make sure to append port "8000" at the end and if everything was properly configured, it will show the index page of the

application.
For example: www.ec2dnsname.com:8000

This might work for some time now, but as soon as the SSH connection is broken/closed, or the instance is rebooted, the server will stop and thus will not respond to requests.

To resolve this issue, we use the Apache server. It means that the Apache server always keeps running in the background, and keeps on responding to users requesting for data and files. This is where the robustness of the Apache server outshines the development server of Django.

. . .

Apache Server Configuration

Let's edit the file where we will define our virtual host.
Type the following command:

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

To open the file in vim editor. Press ***I*** to start editing the file.

Replace the file with the following configuration:

```
<VirtualHost *:80>

ServerAdmin webmaster@example.com

DocumentRoot /home/ubuntu/django/myproject

ErrorLog ${APACHE_LOG_DIR}/error.log

CustomLog ${APACHE_LOG_DIR}/access.log combined

Alias /static /home/ubuntu/django/myproject/static

<Directory /home/ubuntu/django/myproject/static>

Require all granted

</Directory>
```

```
<Directory /home/ubuntu/django/myproject/myproject>

<Files wsgi.py>

Require all granted

</Files>

</Directory>

WSGIDaemonProcess myproject python-
path=/home/ubuntu/django/myproject python-
home=/home/ubuntu/django/myprojectenv

WSGIProcessGroup myproject

WSGIScriptAlias / /home/ubuntu/django/myproject/myproject/wsgi.py

</VirtualHost>
```

> · The first line specifies that **Apache will listen to port 80**(default HTTP), and serve the files specified.
>
> · Document root specifies the location of the application files.
>
> · The directory tags are used to give Apache server the permission to access the respective directory and its files.
>
> · The WSGI settings are required for the Django server to work.
>
> · Please make sure the paths are correct and according to naming of directories and files in the Django project.

Apache Virtual Hosts are used to run more than one web site(domain) using a single IP address. In other words, multiple web sites(domains) but a single server.

```
cd /home/ubuntu/django/myproject
```

Use the above command and ***navigate to the django project directory***. Use the following commands to give the required permissions.

```
chmod 664 db.sqlite3
```

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

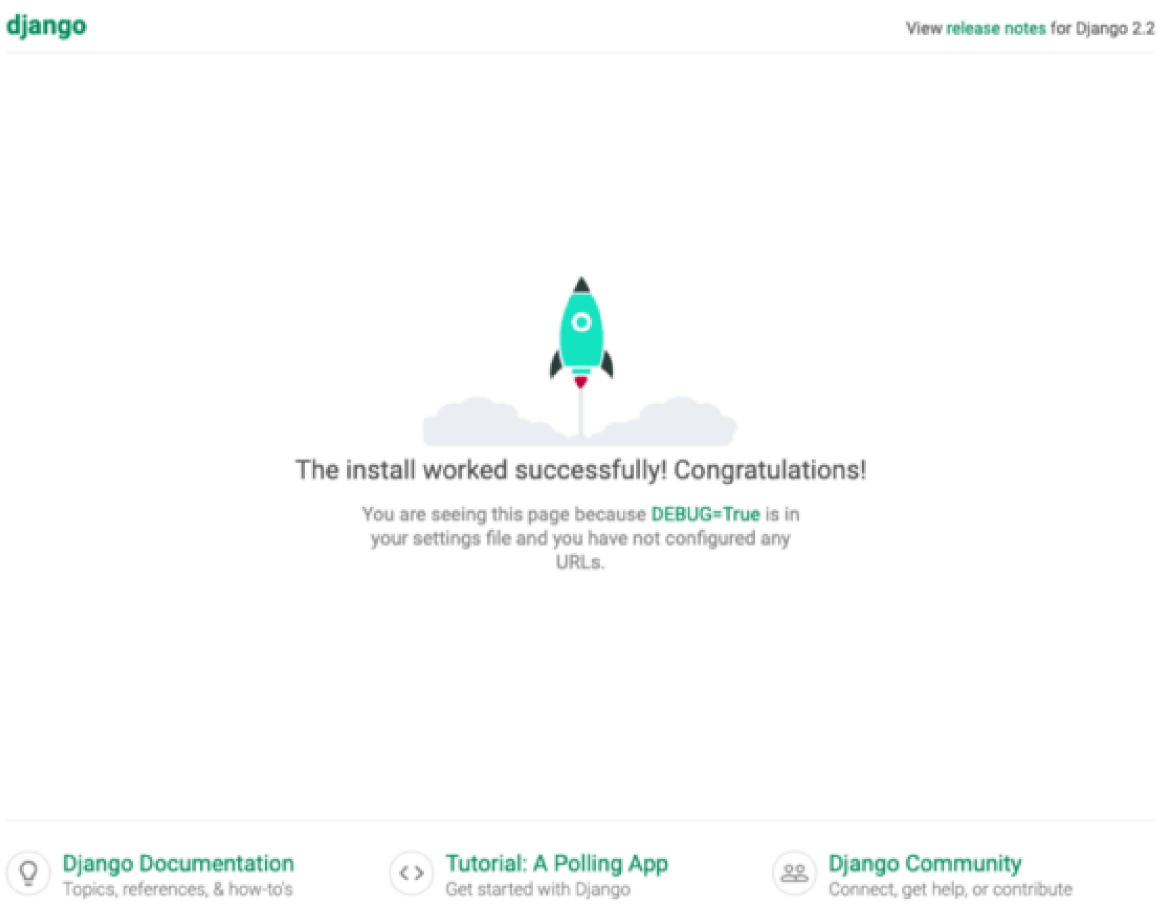
With all this done, **restart the apache2 server** by using the following command:

```
sudo service apache2 restart
```

. . .

It works!

Now ***navigate to public DNS of the instance in the browser*** and one shall see the Django application running if everything was set up correctly, the page of victory.


django
View release notes for Django 2.2
The install worked successfully! Congratulations!
You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

Django default home page

. . .

This was just an example of how to setup **Django with Apache** on cloud service. There are a lot of other factors that come into play when deploying a production server.

One of the most important areas of attention is **Security.**
To make the server more secure, one can limit it's access to a few IP addresses which should be ideal for development and testing.

For all of these purposes, AWS tools like Security Groups and IAM users can come handy.

One can also use AWS beanstalk service to deploy Django apps in an easier way.

Reference: Django documentation

Summary

Let's have a quick recap.

To begin with, an AWS-EC2 instance is spawned.
With the key-pair file, the connection to the instance was made using SSH.

Dependencies such as *Apache*, *pip*, *mod-wsgi*, and *virtualenv* are installed on the EC2 instance.

After creating a virtual environment, the repository is cloned, and django app dependencies are installed. There on, Django development server is run.

If everything works fine, Apache is configured by changing the *000-default.conf* file. The files of the app are given the required permissions and the Apache server is restarted.

. . .

If you enjoyed this article, don't forget to give it a clap!

9,546

For more articles like this, follow our blog, and stay updated with our posts on social media.