DEPLOY LIKE A PRO - DJANGO 5.2 DEPLOYMENT ON LINUX

Architecture Overview

Understanding the nature of your Django application is crucial when selecting the appropriate hosting infrastructure. For simpler applications with minimal complexity and no immediate need for scalability, shared hosting may suffice. However, as your application grows in complexity and requires greater control or scalability, transitioning to a Virtual Machine (VM) or Virtual Private Server (VPS) becomes essential.

In most cases, if your application has specific dependencies or unique configuration requirements, a Virtual Machine (VM) or Virtual Private Server (VPS) is the more practical choice. **Shared hosting environments rarely support Django natively, as they often lack essential tools like WSGI server integration, virtual environments, and full control over server configurations.** As such, a VM or VPS becomes a necessity for most Django deployments.

To successfully deploy a Django application on a Virtual Machine (VM), you'll need to set up several essential components that work together to serve your application reliably and securely. At the end of this documentation(Appendix), there will be a checklist that you can run against your deployment process to ensure that you are set up for success.

Key Components for Django Deployment on a Virtual Machine

To deploy a Django application on a Virtual Machine, you'll need to configure the following core components:

1. Linux Server (Host Machine)

This is the operating system running on your Virtual Machine. Most Django deployments use a Linux distribution like Ubuntu or Debian due to their stability, package availability and community support. This server acts as the foundation for installing and managing all other components.

2. Web Server (e.g., Nginx or Apache)

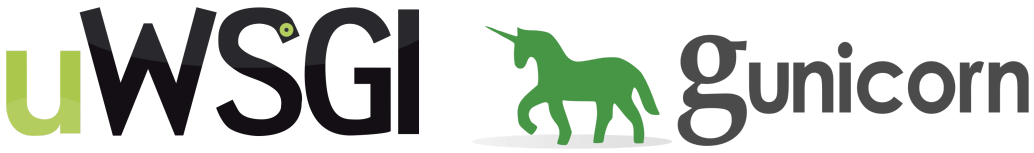
The web server handles incoming HTTP requests from clients (e.g., browsers) and forwards them to the Django application via the WSGI server. It also serves static files, manages SSL/TLS encryption, and handles load balancing if needed. Nginx is a popular choice due to its speed and ease of configuration.

3. Database (e.g., PostgreSQL or MySQL)

Django supports multiple databases, but PostgreSQL is widely recommended for its advanced features and performance. The database stores all persistent data for your application, such as user accounts, posts, and transactions.

4. WSGI Server (e.g., Gunicorn or uWSGI)

The WSGI server acts as a bridge between the web server and your Django application. It runs your Django app as a Python process and responds to requests passed from the web server. Gunicorn is a popular choice for its simplicity and solid performance with Django.



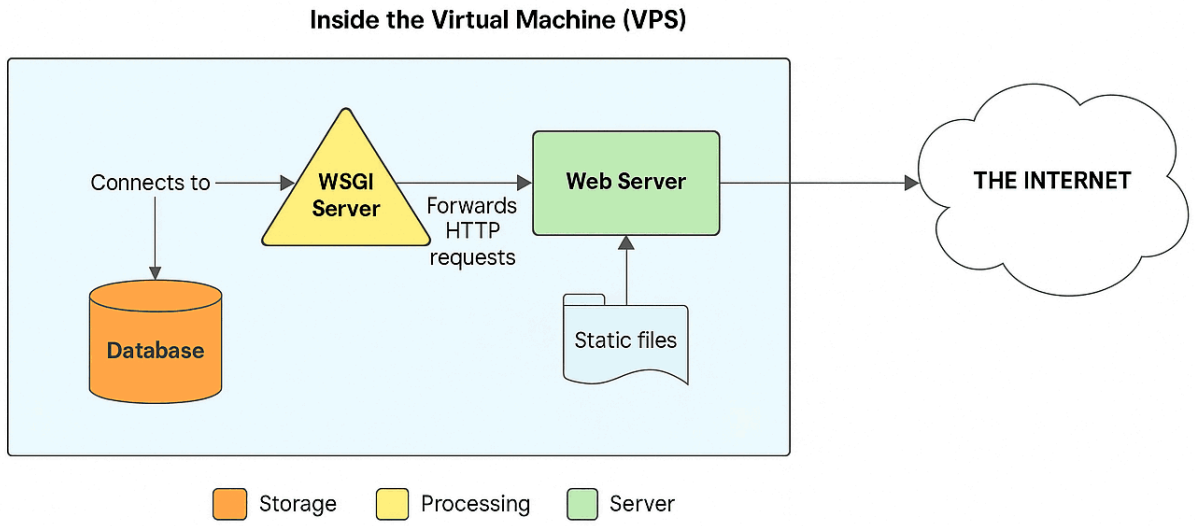
5. systemd Service

systemd is a system and service manager for Linux. Creating a systemd service file allows your WSGI server (like Gunicorn) to start automatically on boot and be managed using standard service commands (start, stop, restart, etc.). This ensures that your application runs reliably and can recover from reboots or crashes.

Architecture Types | Setups

1. Single Machine Deployment

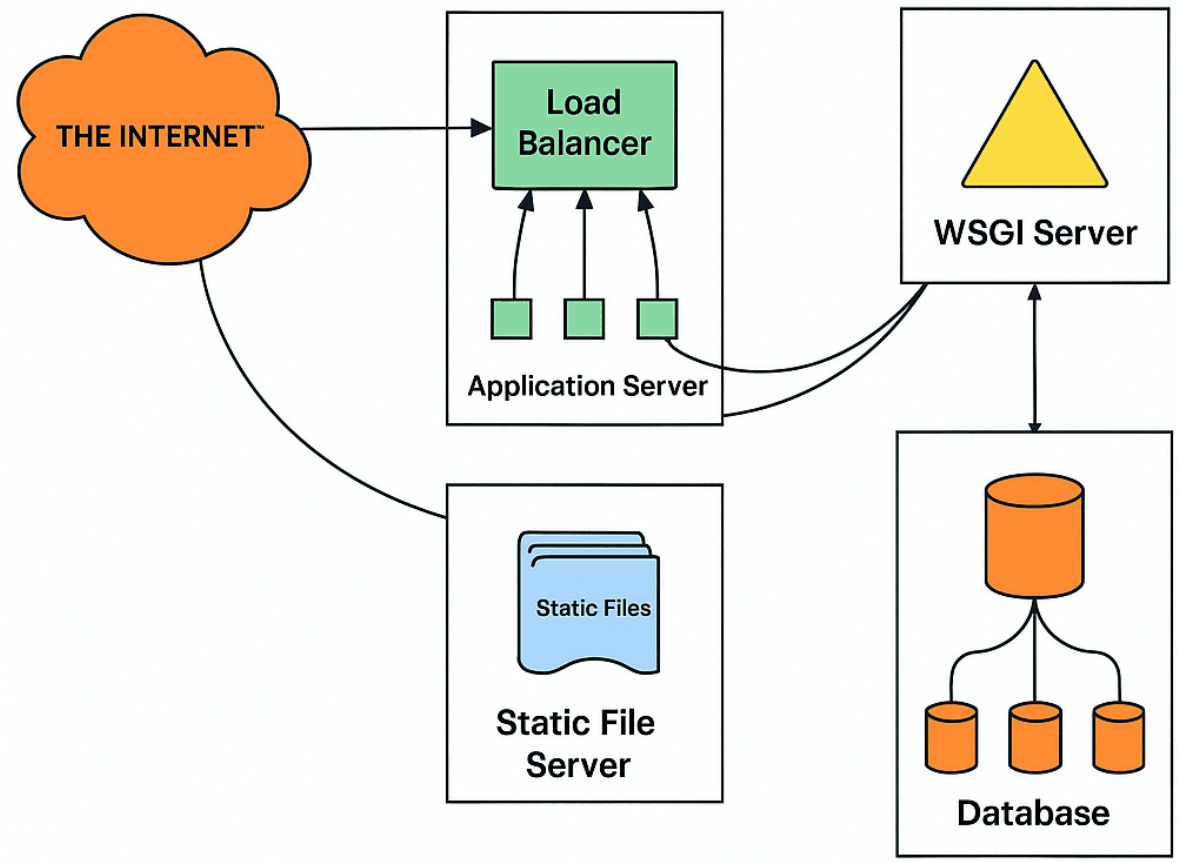
Below is a simple architecture whereby everything has been set up in a single machine.



The diagram below illustrates a basic Django deployment setup where all components—including the **database, WSGI server, web server, and static files**—reside within a single Linux-based Virtual Private Server (VPS). This is a common and cost-effective architecture for small to medium applications.

2. Distributed deployment (Multi-Server Deployment)

This architecture separates different responsibilities into dedicated servers or services. This is typically used for medium to large-scale Django applications that need to scale horizontally, support high availability, or isolate workloads for performance and security.



Components Overview:

1. **Load Balancer (Top-left box connecting to multiple boxes)**
    - Distributes incoming traffic to multiple application servers.
    - Ensures high availability and load management.
    - Can be set up using Nginx, HAProxy, or cloud-native solutions like AWS ELB or GCP Load Balancer.
2. **Application Servers (Middle-top section)**
    - These servers run Django apps behind a WSGI server (e.g., Gunicorn or uWSGI).
    - Each one can handle a portion of the incoming requests in parallel.
    - Helps scale your backend horizontally.
3. **WSGI Server (Right top triangle)**

- Interfaces between the web server (e.g., Nginx) and Django application.
- Manages Python processes to serve dynamic content.

4. **Static File Server (Bottom-left box)**
    - Hosts static files (CSS, JS, images) independently.
    - Can be served via Nginx, AWS S3, or other CDN-backed systems for performance.
5. **Database Server (Bottom-right)**
    - Centralized database accessed by all app servers.
    - Can be a managed service like PostgreSQL, MySQL, or cloud-hosted databases.
    - The additional small cylinders below represent replicas or backups for redundancy and read scaling.

Virtual Machine Providers

When deploying a Django application to a Virtual Machine, choosing the right provider is essential. Below are some of the most popular VM providers that offer flexible infrastructure, reliable performance, and global availability:

Accessing the Virtual Machine through Secure Shell(ssh)

Secure Shell (SSH) is a cryptographic network protocol that provides a secure way to access and manage your virtual machine remotely. This section covers how to connect to your VM using SSH from various operating systems.

Prerequisites

Before connecting via SSH, ensure you have:

- The IP address or hostname of your VM
- Valid user credentials (username and password or SSH key)
- SSH client installed on your local machine
- Network connectivity to your VM (proper firewall rules and network configuration)

Connecting from Linux/macOS

Linux and macOS systems typically include an SSH client by default.

1. Open a terminal window
2. Use the following command format:

   ```
   ssh username@vm_ip_address
   ```

   Example:

   ```
   ssh admin@192.168.1.100
   ```

3. If this is your first connection, you'll see a message about the host authenticity:

```
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
ECDSA key fingerprint is SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.Are
you sure you want to continue connecting (yes/no/[fingerprint])?
```

Type `yes` to continue.

4. Enter your password when prompted.

Connecting from Windows

Windows users have several options:

Option 1: Using PowerShell (Windows 10/11)

1. Open PowerShell
2. Use the same command syntax:

```
ssh username@vm_ip_address
```

Option 2: Using PuTTY

1. Download and install PuTTY from https://www.putty.org/
2. Launch PuTTY
3. Enter the VM's IP address in the "Host Name" field
4. Ensure the connection type is set to SSH (port 22 by default)
5. Click "Open"
6. Enter your credentials when prompted

SSH Key Authentication (Recommended)

For better security, consider using SSH key pairs instead of passwords:

1. Generate a key pair on your local machine:

```
ssh-keygen -t rsa -b 4096
```

2. Copy the public key to your VM:

```
ssh-copy-id username@vm_ip_address
```

3. Now you can log in without a password:

```
ssh username@vm_ip_address
```

Common SSH Options

- Specify a different port (if your VM uses non-standard SSH port):

```
ssh -p 2222 username@vm_ip_address
```

- Enable verbose output for debugging:

```
ssh -v username@vm_ip_address
```

- Run a single command remotely:

```
ssh username@vm_ip_address "command_to_run"
```

Troubleshooting

If you encounter connection issues:

1. Verify the VM's IP address is correct
2. Check that the SSH service is running on the VM
3. Confirm network connectivity (ping the VM)
4. Verify firewall rules allow SSH traffic (port 22 by default)
5. Check `/var/log/auth.log` or `/var/log/secure` on the VM for authentication errors

Security Best Practices

1. Disable root login over SSH
2. Use SSH key authentication instead of passwords
3. Change the default SSH port (reduces brute force attacks)
4. Implement fail2ban to block repeated login attempts
5. Keep your SSH client and server software updated

Checklist for the first actions in a Virtual Machine

When you first access a virtual machine (VM), there are several important initial actions you should take to secure, configure, and optimize it for your needs. Below is a checklist of recommended first steps:

1. Update the System

Ensure all packages and security patches are up to date to prevent vulnerabilities.

- **Debian/Ubuntu-based systems:**

```
sudo apt update && sudo apt upgrade -y
```

- **RHEL/CentOS/Fedora:**

```
sudo dnf upgrade -y  # or `sudo yum upgrade -y` for older versions
```

- **SUSE/openSUSE:**

```
sudo zypper refresh && sudo zypper update -y
```

2. Create a New User (Avoid Using `root`)

Running as `root` is risky. Instead, create a new user with `sudo` privileges.

```
sudo adduser yourusername
sudo usermod -aG sudo yourusername  # Debian/Ubuntu
sudo usermod -aG wheel yourusername # RHEL/CentOS
```

Then, log out and log back in as the new user.

3. Secure SSH Access

- **Disable root login over SSH:**

  Edit `/etc/ssh/sshd_config`:

  ```
  sudo nano /etc/ssh/sshd_config
  ```

  Set:

```
PermitRootLogin no
```

- `PasswordAuthentication no  # If using SSH keys`

  Then restart SSH:

  ```
  sudo systemctl restart sshd
  ```

- **Change the SSH port (optional but recommended):**

  Modify `/etc/ssh/sshd_config`:

  ```
  Port 2222  # Example non-default port
  ```

  Update firewall rules (`ufw`/`firewalld`/`iptables`) and restart SSH.

4. Set Up a Firewall

- **UFW (Ubuntu/Debian):**

```
sudo ufw allow 2222/tcp  # If using custom SSH port
sudo ufw enable
```

- **Firewalld (RHEL/CentOS):**

```
sudo firewall-cmd --permanent --add-port=2222/tcp
sudo firewall-cmd --reload
```

5. Install Essential Tools

Install commonly needed utilities:

```
sudo apt install -y curl wget git htop tmux net-tools  # Debian/Ubuntu
sudo dnf install -y curl wget git htop tmux net-tools  # RHEL/Fedora
```

6. Set Up Automatic Updates

- **Debian/Ubuntu (Unattended Upgrades):**

```
sudo apt install unattended-upgrades
sudo dpkg-reconfigure unattended-upgrades
```

- **RHEL/CentOS (DNF Automatic):**

```
sudo dnf install dnf-automatic
sudo systemctl enable --now dnf-automatic.timer
```

7. Configure Time Synchronization

Ensure the system clock is accurate:

```
sudo timedatectl set-timezone Your/Timezone  # e.g., America/New_York
sudo systemctl enable --now systemd-timesyncd  # or `chronyd` on RHEL
```

8. Monitor System Resources

- Check running processes and resource usage:

  ```
  htop
  ```

- Check disk space:

  ```
  df -h
  ```

9. Disable Unnecessary Services

List running services:

```
sudo systemctl list-units --type=service
```

Disable unused ones (e.g., `bluetooth`, `cups`):

```
sudo systemctl disable --now servicename
```

10. Set Up Backups (Optional but Recommended)

Configure automated backups for critical data (e.g., using `rsync`, `borg`, or cloud storage).

11. (Optional) Install and Configure Fail2Ban

Protect against brute-force attacks:

```
sudo apt install fail2ban  # Debian/Ubuntu

sudo dnf install fail2ban  # RHEL/Fedora

sudo systemctl enable --now fail2ban
```

12. Log Out and Test Access

- Exit the session and verify you can log back in with your new user and SSH key (if configured).
- Test `sudo` privileges:

  ```
  sudo whoami
  ```
  (Should return `root`.)

Summary Checklist

✅ Update system packages

✅ Create a non-root user with `sudo`

✅ Secure SSH (disable root login, use keys, change port)

✅ Enable firewall

✅ Install essential tools

✅ Configure automatic updates

✅ Sync timezone and clock

✅ Monitor resources

✅ Disable unnecessary services

✅ Set up backups (optional)

✅ Install Fail2Ban (optional)

Creating A Python Environment

Step 1: Navigate to `/opt/`

The `/opt/` directory is commonly used for installing third-party software.

```
cd /opt/
```

Step 2: Install Python 3 Virtual Environment (venv)

If `python3-venv` is not installed (common on minimal Linux installations), install it first:

Debian/Ubuntu

```
sudo apt update && sudo apt install python3-venv -y
```

RHEL/CentOS/Fedora

```
sudo dnf install python3-virtualenv -y  # or `yum` for older versions
```

Step 3: Create a Python Virtual Environment

Create a virtual environment named `venv` inside `/opt/`:

```
python3 -m venv /opt/venv
```

This will generate a self-contained Python environment in `/opt/venv/`.

Step 4: Activate the Virtual Environment

Navigate into the `venv` directory and activate the environment.

```
cd /opt/venv
source bin/activate  # Activates the virtual environment
```

Your terminal prompt should now show `(venv)`, indicating the virtual environment is active.

Step 5: Verify Python Version Inside venv

Check the Python version inside the virtual environment:

```
/opt/venv/bin/python --version
```

Example output:

```
Python 3.12.6
```

Step 6: Install Django Inside the Virtual Environment

Use `pip` (included in the virtual environment) to install Django:

```
/opt/venv/bin/python -m pip install django
```

Verify the installation:

```
/opt/venv/bin/python -m django --version
```

Example output:

```
5.2.0
```

Step 7: Create a New Django Project

Use `django-admin` to create a project (replace `projectname` with your desired name):

```
/opt/venv/bin/django-admin startproject projectname
```

This creates a new directory `/opt/venv/projectname/` with the Django project structure.

Step 8: Navigate to the Django Project

Move into the project directory:

```
cd projectname
```

The directory structure should look like:

```
projectname/
├── manage.py
└── projectname/
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── asgi.py/wsgi.py
```

Step 9: Run the Django Development Server

Start the development server to verify the setup:

```
/opt/venv/bin/python manage.py runserver 0.0.0.0:8000
```

- Access the server at `http://<your-server-ip>:8000` in a web browser.
- You should see the Django welcome page.

Summary of Commands

Step	Command	Description
1	`cd /opt/`	Move to `/opt/` directory
2	`sudo apt install python3-venv`	Install Python virtual environment

3	`python3 -m venv /opt/venv`	Create a virtual environment
4	`source bin/activate`	Activate the virtual environment
5	`/opt/venv/bin/python --version`	Check Python version
6	`/opt/venv/bin/python -m pip install django`	Install Django
7	`/opt/venv/bin/django-admin startproject projectname`	Create a Django project
8	`cd projectname`	Navigate into the project
9	`/opt/venv/bin/python manage.py runserver 0.0.0.0:8000`	Start Django server

Initialize a Git Repository & Configure SSH Keys

First of all, install git into the Virtual Machine by running
Apt install git -y

Create a GitHub repository and push the project to GitHub.
Secure the repository with a GitHub deploy key from the settings tab.

1. Inside the VM, run the command:
   ssh-keygen
2. Name the key generated
3. Access the key and copy the encrypted key data to GitHub. Run this command to see the key:
   cat ~/ .ssh/myproject-id_rsa.pub

Since the project is using a unique key, allow the key to have pull requests and not write requests

Best Practices

1. Avoid using secrets in the code or pushing them to the GitHub repository.
2. Use environment variables and keep them in a .env file

Installing & Setting Up PostgreSQL Database

First search for a package name in the Linux server

```
apt-cache search postgre | grep ^postgres
```

Now install postgres with:

```
apt install postgresql -y
```

Check if postgres is running and is active by:

```
systemctl status postgresql
```

Also check if postgresql is listening on localhost TCP by:

```
netstat -ntlp
```

To connect as the default admin to the postgresql database that has been installed:

```
sudo -u postgres psql
```

At this point you will be inside postgresql interactive shell where you can run the following initial commands:

\l = checks the tables in the database
\du = checks the users in the database

Run these command to create a **database**,a **user** and **grant privileges** to the user that you have created:

```
CREATE DATABASE my-database;
CREATE USER me-user WITH ENCRYPTED PASSWORD '1234567';
```

(Replace my-database with the name you want for your database and me-USER with the preferred username for your database)

```
GRANT ALL PRIVILEGES ON DATABASE my-database TO me-user;
```

Configure Django to use PostgreSQL

In the settings.py file where we have the database settings, update the engine to use the PostgreSQL settings and credentials the specific database you want to connect with.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'yourdbname',
        'USER': 'youruser',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',  # or other service URL
        'PORT': '5432',
    }
```

Also ensure that the **psycopg2-binary** package is installed before running migrations.
To run migrations use the following command:

/opt/venv/bin/python manage.py migrate

To check whether the connection was successful run the following command:

/opt/venv/bin/python manage.py dbshell

Install and Test Gunicorn WSGI Server

Install gunicorn using the following command:

/opt/venv/bin/python -m pip install gunicorn

To confirm that it has been installed correctly:

/opt/venv/bin/gunicorn --version

Invoke the gunicorn from the project root

/opt/venv/bin/gunicorn -b 127.0.0.1:8000 myproject.wsgi

In another terminal, install curl to enable us to run the http request to test if gunicorn is running appropriately

Apt install curl -y

Run the following command:

Curl http://127.0.0.1:8000/

At this point you should be able to see the default landing page for django.

Set up Systemd Service for Gunicorn

A systemd service file makes sure that it starts gunicorn when the server starts and if it ever crashes it will automatically restart it.
The service file lives in the system folder and this is how to navigate to it

cd /etc/systemd/system

While in the directory use this command to see the existing files:

```
ls -la
```

Ensure you are in the **etc/systemd/system** directory, create and edit a new file. Use the command below:

```
vim gunicorn.service
```

Add the following lines to the file:

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target

[Service]
Type=simple
User=your_username
WorkingDirectory=/opt/myproject
ExecStart=/opt/venv/bin/gunicorn -b 127.0.01:8000
myproject.wsgi
Restart=on-failure


[Install]
WantedBy=multi-user.target
```

Save the file and exit
Create the system user as described in the file above by running the command below:

```
Useradd --system --no-create-home --shell=/sbin/ nologin your_username
```

This user that is created has **no home directory, no shell and no ability** to log in privileges as a security measure for the application.
The user is strictly for running the particular gunicorn service that was created above.
Now that the user is created & our systemd file is ready, we can start the service by running

```
systemctl start gunicorn
```

Also run:

```
systemctl enable gunicorn
```

This will tell the server(linux) to start every time the server is rebooted.
Ensure that the system is running by running

```
systemctl status gunicorn
```

In order to check the logs you can run the following command:

```
journalctl -u gunicorn
```

You can use net-tools by running

```
netstat -ntlp
```

This checks how ports are and the components listening ports.

Install & Test NGINX

To install Nginx run this command in the terminal:

```
apt install nginx -y
```

To check whether Nginx has been installed properly and is active run this command:

```
systemctl status nginx
```

You can also run:

```
netstat -ntlp
```

Nginx will be configured to SSL & bind to port 443.
In most cases you will find Nginx has created a webroot directory:

```
/var/www/html
```

Nginx also creates a configuration directory

```
/etc/nginx
```

Among the files within the Nginx configuration are:

1. **nginx.conf** - this is the main config file
2. conf.d - everything in this directory gets included, where you can include all the custom config files. It is empty by default
3. sites-available - contains all the possible site configurations. Place all the possible site configurations into the sites available directory.
4. sites-enabled - contains the sites you want active. Place all the sites you want active in this directory.
   In order to disab;e a site, delete the link in the sites-enabled and the original link/config remains in the sites-available just that it is not used.

Now run this command to see the ports that Nginx is listening to:

```
netstat -ntlp
```

To get the public address of the virtual machine use this command:

```
Ip addr
```

Install dns utils by running this command:

```
apt install dnsutils -y
```

To check the domain status run this command:

```
nslookup yourdomain.com
```

Configuration of Static files & Media directories in Django

Static files include the stylesheets, images & media. These are served directly from the file system.

First ensure that you have configured the directory for the staticfiles in the settings.py file
In the settings.py file include the following configurations:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    BASE_DIR / 'static',
]
STATIC_ROOT = BASE_DIR / 'staticfiles'

MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'uploads'
```

Run the collect static command to gather all static files from the app to put them in the right directory for deployment.

```
cd /opt/myproject
/opt/venv/bin/python manage.py collectstatic
```

Create an uploads directory by running this command:

```
mkdir uploads
```

For the media & upload directories, we need to grant the user that we created in the systemd service the correct privileges.
To see the privileges & owner of the rights run this command:

```
ls -la --color
```

To add the user/give permission to the user running the gunicorn service, run the command:

```
chown -R root:username uploads
```

To see the privileges & owner of the rights run this command once more:

```
ls -la --color
```

The group changes but does not include write permissions. To modify the group permissions on the uploads directory run the following command:

```
Chmod -R 775 uploads
```

Configure A Virtual Host in Nginx

NGINX Virtual Host Configuration for Django Explained

It involves setting up the reverse proxy & static files handling in Nginx

So far we have installed Nginx and it was set to the default Nginx landing page, which was being served.

Navigate to the Nginx root directory that contains the config files and folders:

```
cd /etc/nginx/
```

If we list the directories you will see the directories we had discussed earlier: sites-available, sites-enabled, conf.d etc. run this command to list these directories:

```
Ls -ls --color
```

Open up the nginx.conf file to have a look at the default settings but do not modify it. Run this command:

```
vim nginx.conf
```

Move or navigate to the sites-available directory by running this command:

```
cd /etc/nginx/sites-available/
```

Create a specific config file for your project by running this command:

```
vim myproject.conf
```

This configuration file sets up NGINX (the web server) to serve your Django application. Let me break it down in simple terms:

```
# /etc/nginx/sites-available/eportfolio.conf

# Virtual host config for my project

#Redirect non-www to www.

#server {

    #listen 0.0.0.0:8000;

    # listen [::]:80;

    #server_name myproject.com;

    #return 301 http://www.createwitheric.com$request_uri;

    #}


# Basic Server Configuration

server {

    # Listen on IPv4 and IPv6 interfaces on port 80 (HTTP)
```

```
    listen 0.0.0.0:80;
    listen [::]:80;
    # Domain name for this virtual host
    server_name createwitheric.com;

    ## SSL Certificate Validation
    # Required for Let's Encrypt certificate issuance/renewal
    location /.well-known/acme-challenge {
        alias /opt/myproject/static/.well-known/acme-challenge/;
    }

    ## Static Files Configuration
    location /static/ {
        alias /opt/myproject/static/;
    }
   location /media/ {
        alias /opt/myproject/uploads/;
    }
    # robots.txt - Search engine instructions
    location /robots.txt {
        alias /opt/eportfolio/static/robots.txt;
    }
    # Favicon - Browser tab icon
    location /favicon.ico {
        alias /opt/eportfolio/static/favicon.ico;
    }
    ## Application Server Configuration
    # Proxy all other requests to Django/WSGI server - The reverse Proxy
```

```
    location / {

        proxy_pass http://127.0.0.1:8000;

        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    }


    ## Logging Configuration

    access_log /var/log/nginx/eportfolio-access.log;

    error_log /var/log/nginx/eportfolio-error.log;

}
```

Basic Server Setup

```
server {
    listen 0.0.0.0:80;
    listen [::]:80;
    server_name createwitheric.com;
```

- `server` block defines a virtual host configuration
- `listen` directives tell NGINX to listen on:
    - Port 80 (HTTP) on all available IPv4 addresses (`0.0.0.0`)
    - Port 80 on all available IPv6 addresses (`[::]`)
- `server_name` specifies this configuration applies to requests for "createwitheric.com"

SSL Certificate Validation

```
#Webroot for SSL

    location /.well-known/acme-challenge {

        alias /opt/myproject/static/.well-known/acme-challenge/;

    }
```

This NGINX configuration block is used for **SSL certificate validation** (typically with Let's Encrypt/Certbot). Here's a breakdown:

Purpose

- Allows **Let's Encrypt** (or other ACME-based CA) to verify domain ownership by checking a special file in this directory.
- Required for **HTTP-01 challenges** when obtaining/renewing SSL certificates.

How It Works

1. **When you request an SSL certificate** (e.g., via Certbot):
    - Let's Encrypt sends a unique token (e.g., `abc123`).
    - Certbot saves a validation file at: `/opt/myproject/static/.well-known/acme-challenge/abc123`
2. **Let's Encrypt checks** if this file is publicly accessible at: `http://yourdomain.com/.well-known/acme-challenge/abc123`
3. If accessible → Domain ownership confirmed → SSL certificate issued.

Serving Static Files

```
#robots.txt
location /robots.txt {
    alias /opt/eportfolio/static/robots.txt;
}

#favicon
location /favicon.ico {
    alias /opt/eportfolio/static/favicon.ico;
}

#static files
location /static/ {
    alias /opt/eportfolio/static/;
}
location /media/ {
    alias /opt/eportfolio/uploads/;
}
```

These sections handle different types of static files:

1. **robots.txt**: Tells search engines which pages to crawl or ignore
    - When someone visits `createwitheric.com/robots.txt`, NGINX serves the file from `/opt/eportfolio/static/robots.txt`
2. **favicon.ico**: The small icon displayed in browser tabs
    - Serves the favicon from `/opt/eportfolio/static/favicon.ico`
3. **Static files** (CSS, JavaScript, images):
    - Any URL starting with `/static/` (like `createwitheric.com/static/style.css`) will look for files in `/opt/eportfolio/static/`
4. **Media files** (user-uploaded content):
    - URLs starting with `/media/` will look for files in `/opt/eportfolio/uploads/`

Passing Requests to Django

```
#WSGI SERVER
location / {
    proxy_pass http://127.0.0.1:8000;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
```

- This handles all other requests (those not matching the static file patterns above)
- `proxy_pass` sends these requests to your Django application running on the same server (`127.0.0.1`) on port 8000
- `proxy_set_header` passes the visitor's real IP address to Django (otherwise Django would only see NGINX's IP)

Logging

```
access_log /var/log/nginx/eportfolio-access.log;
error_log /var/log/nginx/eportfolio-error.log;
```

- `access_log` records all requests to your site
- `error_log` records any problems NGINX encounters
- These help with troubleshooting and monitoring traffic

Key Concepts to note:

1. **Static vs Dynamic Content**:
    - Static: Files that don't change (images, CSS, JS)
    - Dynamic: Content generated by Django (pages with database content)
2. **Why NGINX?**:
    - NGINX is faster at serving static files than Django
    - It acts as a "middleman" between visitors and your Django app
    - Provides security benefits by shielding Django directly from the internet
3. **Location Blocks**:
    - Define how different URL patterns should be handled
    - NGINX checks these in order (though this configuration doesn't have conflicts)
4. **Proxy Pass**:
    - Forwards requests it can't handle itself to your Django application

This configuration is a common setup for Django deployments, where NGINX handles static files and proxies other requests to Django (typically through Gunicorn or uWSGI running on port 8000).

Ensure that you save this specific config file and exit the vim editor.

Create A Symbolic Link(sym link)

Run the command below to create a sym link:

```
ln -s /etc/nginx/sites-{available,enabled}/eportfolio.conf
```

This command is creates a **symbolic link** (shortcut) between two directories to enable your Nginx configuration. Let me break it down in simple terms:

Command Explanation:

```
ln -s /etc/nginx/sites-{available,enabled}/eportfolio.conf
```

1. `ln -s`
    - `ln` = "link" command

    - `-s` = creates a *symbolic* link (like a Windows shortcut) rather than a hard link
2. `/etc/nginx/sites-{available,enabled}/eportfolio.conf`

   This is actually expanded by the shell to:

3. `/etc/nginx/sites-available/eportfolio.conf`
   `/etc/nginx/sites-enabled/eportfolio.conf`
    - First path: Source file (`sites-available`)
    - Second path: Link location (`sites-enabled`)

What This Does:

- Creates a shortcut in `sites-enabled` that points to your config file in `sites-available`
- Nginx only reads configurations from `sites-enabled` when running
- This keeps disabled sites in `sites-available` (organized storage) while only enabling specific ones

Why This Structure?

- **sites-available**: Storage for all possible configurations
- **sites-enabled**: Only contains links to active configurations
- Makes it easy to disable sites without deleting files:

  ```
  unlink /etc/nginx/sites-enabled/eportfolio.conf  # Disable
  ```

- ```
  ln -s /etc/nginx/sites-{available,enabled}/eportfolio.conf

  This method is called brace expansion # Re-enable
  ```

To check whether the sym link has been created list the sites-enabled directory by running this command:

```
ls -ls /etc/nginx/sites-enabled
```

Full Command (More Readable Version):

Many admins prefer writing it explicitly:

```
ln -s /etc/nginx/sites-available/eportfolio.conf /etc/nginx/sites-enabled/
```

After running this, you should:

1. Test the Nginx configuration:
```

```
sudo nginx -t
```

2. Reload Nginx if the test passes:

```
systemctl reload nginx
```

```
Or
```

```
systemctl restart nginx
```

Add SSL Encryption to Nginx Virtual Host

Sources:

1. Lets Encrypt
2. OpenSSL

Optional:To generate a self sign SSL Certificate run this command which we will use in the dev site(staging) :

> openssl req -newkey rsa:2048 -nodes keyout privkey.pem -x509 -days 36500 -out certficiate.pem

Use certbot to get a certificate from Lets Encrypt by running the command:

STEPS: Let's Encrypt (For Production)

1. Install Certbot

```
apt install certbot
```

2. Use the dry run approach - if you did not include the location block for the ssl certificate

```
certbot certonly --dry-run
```

You will be prompted on a number of preferred options. Choose the following sequentially as follows:

- Place files in webroot directory.(webroot)
- Enter the domain you want to attach the certificates to.
- Submit the email you wish to use for the domain
- Input the webroot for your domain by submitting the directory which will be: /opt/myproject/static

3. Use the certbot approach without the dry run approach if you included the location block for the ssl certificate.

```
certbot certonly
```

You will be prompted on a number of preferred options. Follow the preferred options above ☝

As long as the configuration in Nginx is okay on the root directory you can confirm certificates by running this command:

```
certbot certificate
```

Reload Nginx to pick up the configurations:

```
systemctl reload nginx
```

4. Configure NGINX for HTTPS

Modify and add another server block in

`/etc/nginx/sites-available/myporject.conf`

```
# HTTP to HTTPS redirect
server {
    listen 80;
    listen [::]:80;
    server_name createwitheric.com;


     ## SSL Certificate Validation

    # Required for Let's Encrypt certificate issuance/renewal

    location /.well-known/acme-challenge {

        alias /opt/myproject/static/.well-known/acme-challenge/;

    }

    return 301 https://myproject.com$host$request_uri;

}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name myproject.com;

    ssl_certificate /etc/letsencrypt/live/createwitheric.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/createwitheric.com/privkey.pem;

    # Include remaining configuration (static files, proxy_pass, etc.)
    # ...
}
```

Save the config file and exit

5. Test & Reload NGINX

```
sudo nginx -t && sudo systemctl restart nginx
```

Certificate is saved at: /etc/letsencrypt/live/createwitheric.com/fullchain.pem
Key is saved at: /etc/letsencrypt/live/createwitheric.com/privkey.pem

Full Working Example

```
# Redirect HTTP to HTTPS (Non-WWW)
server {
    listen 80;
    listen [::]:80;
    server_name yourdomain.com;

    # Certbot challenge directory
    location /.well-known/acme-challenge {
        alias /var/www/yourdomain/.well-known/acme-challenge/;
    }

    return 301 https://yourdomain.com$request_uri;
}

# Main HTTPS Configuration
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name yourdomain.com www.yourdomain.com;

    # SSL Certificates (Let's Encrypt)
    ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;

    # SSL Security Settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
```

```
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;
    add_header Strict-Transport-Security "max-age=63072000; includeSubDomains;
preload";

    # Static Files
    location /static/ {
        alias /var/www/yourdomain/static/;
        expires 30d;
        access_log off;
    }

    # Media Files
    location /media/ {
        alias /var/www/yourdomain/uploads/;
        expires 30d;
        access_log off;
    }

    # Reverse Proxy (Django/Flask/Node.js)
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # Logging
    access_log /var/log/nginx/yourdomain-access.log;
    error_log /var/log/nginx/yourdomain-error.log;
}
```

Common Errors & Fixes

A. SSL Certificate Issues

Error	Cause	Solution

`Certbot fails with 404 on /.well-known/acme-challenge`	Wrong `alias` path or permissions	Ensure: ✔ Path exists (`mkdir -p`) ✔ NGINX has read access (`chown -R www-data:www-data`)

`SSL_ERROR_NO_CYPHER_OVERLAP`	Missing `ssl_protocols` or weak ciphers	Use:
		`ssl_protocols TLSv1.2 TLSv1.3;`
		`ssl_ciphers "EECDH+AESGCM...";`

`NET::ERR_CERT_COMMON_NAME_INVALID`	Wrong domain in certificate	Regenerate cert: `sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com`

B. Redirection Problems

Error	Cause	Solution
`Redirects to HTTP instead of HTTPS`	Missing `return 301 https://`	Ensure the HTTP block has: `return 301 https://$host$request_uri;`
`WWW not redirecting to non-WWW`	No separate `server` block for `www`	Add: `nginx   server {   listen 80;   server_name www.yourdomain.com;  `

		`return 301 https://yourdomain.com$request_uri;   }`
`Too many redirects`	Misconfigured `proxy_set_header`	Add: `proxy_set_header X-Forwarded-Proto $scheme;`

C. DNS & Connectivity Issues

Error	Cause	Solution
`DNS_PROBE_FINISHED_NXDOMAIN`	DNS records missing	Add `A` record: `yourdomain.com → SERVER_IP`
`ERR_CONNECTION_REFUSED`	Firewall blocking ports	Run: `sudo ufw allow 'Nginx Full'`
`SSL Handshake Failed`	Wrong certificate path	Verify: `ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;`

4. Best Practices

Always Test NGINX Config

```
sudo nginx -t && sudo systemctl restart nginx
```

Automate SSL Renewal

```
sudo crontab -e
```

Add:

```
0 12 * * * /usr/bin/certbot renew --quiet
```

Use HSTS for Security

```
add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";
```

Check SSL Configuration

Test your site at:

🔗 SSL Labs Test

5. Conclusion

By following this guide:

✔ You avoid **common SSL pitfalls**

✔ Ensure **secure HTTPS enforcement**

✔ Fix **DNS and redirection issues**

Final Command Checklist

```
# 1. Test & restart NGINX
sudo nginx -t && sudo systemctl restart nginx

# 2. Verify SSL
curl -I https://yourdomain.com
```