Deploying a Django Application on AWS

Prerequisites

1. AWS Account : You need an AWS account.

2. Django Application : Your Django application should be ready for deployment.

Table- Initial Setup and Configuration

1. **AWS Setup**:
    - Launch an EC2 instance with appropriate configurations.
    - Ensure security groups are configured to allow necessary traffic (SSH, HTTP, HTTPS).
2. **Server Preparation**:
    - Update and upgrade the server.
    - Install necessary packages.
3. **Database Setup**:
    - Use PostgreSQL for the database.
    - Create a database and user.
4. **Storage Setup**:
    - Use AWS S3 for storing documents and images.
    - Configure Django to use S3 for media files.

Step 2: Install and Configure Dependencies

1. **Django Project Setup**:
    - Clone your Django project repository.
    - Set up a virtual environment and install dependencies.
2. **Django Settings Configuration**:
    - Configure Django settings for production, including database, static files, and media files.
    - Set up JWT, Djoser, Langchain, OpenAI, and Hugging Face settings.

Step 3: Continuous Integration/Continuous Deployment (CI/CD)

1. **CI/CD with GitHub Actions**:
    - Create GitHub Actions workflows to automate testing and deployment.
    - Use Docker to containerize the Django application.
    - Deploy the application to the EC2 instance using GitHub Actions.

*Step 1: Set Up an EC2 Instance

1. Launch an EC2 Instance :

   - Go to the AWS Management Console.

   - Navigate to the EC2 Dashboard.

   - Click on "Launch Instance".

- Choose an Amazon Machine Image (AMI) (e.g., Ubuntu 20.04 LTS).
- Choose an instance type (e.g., t2.micro for free tier).
- Configure instance details, storage, and tags as needed.
- Configure the security group to allow SSH (port 22), HTTP (port 80), and HTTPS (port 443).
- Review and launch the instance.
- Download the key pair (.pem file) for SSH access.

2. SSH into the Instance :

```sh
chmod 400 your-key-pair.pem
ssh -i your-key-pair.pem ubuntu@your-ec2-public-ip
```

Deploying a Django Application on AWS

*Step 2: Set Up the Server

1. Update and Upgrade the Server :

```sh
sudo apt update
sudo apt upgrade -y
```

2. Install Necessary Packages :

```sh
sudo apt install python3-pip python3-dev libpq-dev nginx curl -y
```

3. Install and Configure PostgreSQL (Optional) :

If you are using PostgreSQL, install it and create a database and user.

```sh
sudo apt install postgresql postgresql-contrib -y
sudo -u postgres psql
CREATE DATABASE myproject;
CREATE USER myprojectuser WITH PASSWORD 'password';
ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
ALTER ROLE myprojectuser SET timezone TO 'UTC';
GRANT ALL PRIVILEGES ON DATABASE myproject TO myprojectuser;
\q
```

Deploying a Django Application on AWS

*Step 3: Deploy the Django Application

1. Clone Your Django Application :

```sh
sudo apt install git -y
git clone https://github.com/your-repo/our-django-app.git
cd our-django-app
```

2. Set Up a Virtual Environment :

```sh
sudo apt install python3-venv -y
python3 -m venv myprojectenv
source myprojectenv/bin/activate
```

3. Install Python Dependencies :

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

4. Configure Django Settings :

   Update your Django settings for production (e.g., `ALLOWED_HOSTS`, database settings, static files settings).

```python
# settings.py
ALLOWED_HOSTS = ['your-ec2-public-ip', 'your-domain.com']

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'myproject',
        'USER': 'myprojectuser',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '',
    }
}

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

5. Collect Static Files :

```sh
python manage.py collectstatic
```

6. Run Migrations :

```sh

python manage.py migrate

```

*Step 4: Set Up Gunicorn

1. Install Gunicorn :

```sh

pip install gunicorn

```

2. Create a Systemd Service File for Gunicorn :

```sh

sudo nano /etc/systemd/system/gunicorn.service

```

Add the following content to the file:
```
[Unit]
Description=gunicorn daemon

After=network.target

[Service]
User=ubuntu

Group=www-data

WorkingDirectory=/home/ubuntu/your-django-app

ExecStart=/home/ubuntu/your-django-app/myprojectenv/bin/gunicorn  --workers 3 --bind
unix:/home/ubuntu/our-django-app/myproject.sock  myproject.wsgi:application

[Install]
WantedBy=multi-user.target

```

3. Start and Enable Gunicorn :

   ```sh

   sudo systemctl start gunicorn

   sudo systemctl enable gunicorn

   ```

*Step 5: Configure Nginx

1. Create an Nginx Configuration File :

   ```sh

   sudo nano /etc/nginx/sites-available/your-django-app

   ```

Add the following content:

```
server {
    listen 80;
    server_name your-ec2-public-ip your-domain.com;

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/ubuntu/your-django-app/myproject.sock;
    }

    location /static/ {
        alias /home/ubuntu/your-django-app/static/;
    }

    location /media/ {
        alias /home/ubuntu/your-django-app/media/;
    }
}
```

2. Enable the Nginx Configuration :

```sh

sudo ln -s /etc/nginx/sites-available/your-django-app /etc/nginx/sites-enabled

sudo nginx -t

sudo systemctl restart nginx

```

*Step 6: CI/CD with GitHub Actions

Create GitHub Actions Workflow: Create a .github/workflows/deploy.yml file in your repository.

```
name: Django CI/CD

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:latest
        env:
          POSTGRES_DB: myproject
          POSTGRES_USER: myprojectuser
          POSTGRES_PASSWORD: password
        ports:
          - 5432:5432

    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.8'

      - name: Install dependencies
        run: |
          python -m venv venv
          source venv/bin/activate
          pip install -r requirements.txt

      - name: Run tests
        run: |
          source venv/bin/activate
          python manage.py test

  deploy:
    runs-on: ubuntu-latest
    needs: build

    steps:
      - uses: actions/checkout@v2

      - name: Set up SSH
        uses: webfactory/ssh-agent@v0.5.3
        with:
          ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}

      - name: Deploy to EC2
        run: |
          ssh -o StrictHostKeyChecking=no ubuntu@your-ec2-public-ip 'cd /home/ubuntu/your-django-app && git pull origin main && source myprojectenv/bin/activate && pip install -r requirements.txt && python manage.py migrate && sudo systemctl restart gunicorn && sudo systemctl restart nginx'
```

*Step 7: Secure the Server

1. Load Environment Variables in `settings.py` :

```python
import os
from pathlib import Path
from dotenv import load_dotenv


load_dotenv()


BASE_DIR = Path(__file__).resolve().parent.parent


SECRET_KEY = os.getenv('SECRET_KEY')
DEBUG = os.getenv('DEBUG') == 'True'
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.getenv('DB_NAME'),
        'USER': os.getenv('DB_USER'),
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST'),
        'PORT': '',
    }
}
```

Deploying a Django Application on AWS

*Final Steps

1. Reload Gunicorn and Nginx :

```sh
sudo systemctl daemon-reload

sudo systemctl restart gunicorn

sudo systemctl restart nginx
```

- Access Your Application :

Open your browser and navigate to `http://your-ec2-public-ip` or `https://your-domain.com`.

- Secure the Server

Set Up SSL with Let's Encrypt:

```
sudo apt install certbot python3-certbot-nginx -y
sudo certbot --nginx -d your-domain.com
```

By following these steps, you can deploy your Django application on AWS with a production-ready setup. This includes configuring the server, setting up the database, deploying the application with Gunicorn and Nginx, securing it with SSL, and managing environment variables.