Django Application Deployment Guide
(Ubuntu 22.04)

Table of Contents

1. Introduction

This guide provides step-by-step instructions for deploying a Django application using Docker, Dockerhub, Gunicorn, Nginx, and implementing GitHub CI/CD workflows.

2. Prerequisites

Before you begin, ensure you have the following installed:

2.1 Install Python

```
$ python3
```

If it is not installed, please run the following commands

```
$ sudo apt update

$ sudo apt install python3
```

2.2 Install Docker

```
$ sudo apt update

$ sudo apt install apt-transport-https ca-certificates curl software-
properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt update

$ apt-cache policy docker-ce

$ sudo apt install docker-ce
```

3. Local Development

3.1 Download the app

```
$ git clone https://github.com/codelover1110/user_management_backend.git
```

3.2 Run by Django Comamnd

```
$ cd /user_management_backend

$ python manage.py runserver
```

3.3 Run by Docker

3.3.1 Add Dockerfile

Add the Dockerfile to the root directory and it should look like this.

```
# Use an official Python runtime as a parent image

FROM python:3.10.12


# Set environment variables

ENV PYTHONUNBUFFERED 1

ENV DJANGO_SETTINGS_MODULE profile_project.settings


# Set the working directory in the container

WORKDIR /app


# Install any needed packages specified in requirements.txt

COPY requirements.txt /app/

RUN pip install -r requirements.txt


# Copy the current directory contents into the container at /app/

COPY . /app/
```

```
# Run python command
RUN python manage.py makemigrations
RUN python manage.py migrate

# Collect static files
RUN python manage.py collectstatic --noinput

# Install Nginx
RUN apt-get update && apt-get install -y nginx

# Copy your custom Nginx configuration file
COPY nginx.conf /etc/nginx/sites-available/default

# Remove the default Nginx configuration
RUN rm /etc/nginx/sites-enabled/default

# Create the target directory and a symbolic link to the Nginx configuration file
RUN mkdir -p /etc/nginx/sites-enabled/ && ln -s /etc/nginx/sites-available/default
/etc/nginx/sites-enabled/

# Expose port 80 for Nginx
EXPOSE 80

# Start both Gunicorn and Nginx
CMD ["sh", "-c", "gunicorn --bind 0.0.0.0:8000 --workers 4 profile_project.wsgi:application &
nginx -g 'daemon off;'"]
```

3.3.2 Add nginx.conf

Add the nginx.conf to the root directory and it should look like this.

```
# Define upstream block to point to the internal IP address of Gunicorn container
upstream django {
    server 172.17.0.2:8000;  # Update with the actual internal IP address of your Gunicorn container
}

# Main Nginx server block
server {
    listen 80;

    # Use your server's IP address or domain name
    server_name 45.32.161.172;  # Update with your actual server IP address or domain name

    # Location block for handling requests to your Django application
    location / {
        proxy_pass http://django;
    }

    # Location block for serving static files
    location /static/ {
        alias /app/static/;
        try_files $uri $uri/ =404;
    }

    # Location block for serving media files
```

```
    location /media/ {
        alias /app/media/;
        try_files $uri $uri/ =404;
    }

    # Additional configurations can be included here if needed
    # include /path/to/your/nginx/includes/*.conf;
}
```

3.3.3 Run

With the Dockerfile, you can go ahead and build the image with the following command:

```
$ docker build -t <dockerhub-username>/<repository-name>
```

For example,

```
$ docker build -t codelover1110/profile_project_image
```

Now we will go ahead and run the created image.

```
$ docker run -dp 8000:8000 codelover1110/profile_project_image
```

Now, we can see the app is running

http://localhost:8000

4. Sever Deployment

4.1 Setup Docker

It is same as 2.2

4.2 GitHub Actions Workflow

On Github you need to go to the repository settings, under secrets -> actions, and add the secrets

4.2.1 GitHub Repository Secrets for Docker:

Add the necessary secrets (e.g., CI_REGISTRY_USER, CI_REGISTRY_PASSWORD, CI_REGISTRY) in your GitHub repository settings. This ensures secure storage of sensitive information.

CI_REGISTRY_IMAGE: Dockerhub repository name (i.e: codelover1110/profile_project_image)

CI_REGISTRY_PASSWORD: Dockerhub account password (i.e: codelover1110_pass)

CI_REGISTRY_USER: Dockerhub account username (i.e: codelover1110)

4.2.2 .github/workflows/main.yml

Create a new GitHub Actions workflow file in the .github/workflows directory, for example, main.yml.

```
name: CI/CD Pipeline


on:
  push:
    branches:
      - main


jobs:
  build-and-deploy:
    runs-on: ubuntu-latest


    steps:
      - name: Checkout Code
        uses: actions/checkout@v2


      - name: Build Docker Image
        run: sudo docker build -t ${{ secrets.CI_REGISTRY_IMAGE }} .
```

```
    - name: Login to Docker Hub
      run: sudo docker login -u ${{ secrets.CI_REGISTRY_USER }} -p ${{
secrets.CI_REGISTRY_PASSWORD }}

    - name: Push Docker Image
      run: sudo docker push ${{ secrets.CI_REGISTRY_IMAGE }}

    - name: Deploy to VPS
      uses: appleboy/ssh-action@master
      with:
        host: ${{ secrets.VPS_HOST }}
        username: ${{ secrets.VPS_USERNAME }}
        key: ${{ secrets.VPS_SSH_KEY }}
        script: |
          set -e  # Exit immediately if a command exits with a non-zero status

          # Pull the latest Docker image
          sudo docker pull ${{ secrets.CI_REGISTRY_IMAGE }}

          # Stop and remove the existing container, ignoring errors if it doesn't exist
          sudo docker stop profile_project_backend || true
          sudo docker rm profile_project_backend || true

          # Run the new Docker container
          sudo docker run -d -p 80:80 --name profile_project_backend ${{
secrets.CI_REGISTRY_IMAGE }}
```

4.2.3 GitHub Repository Secrets for VPS:

Add additional secrets for your VPS connection, such as VPS_HOST, VPS_USERNAME, and VPS_SSH_KEY.

VPS_HOST: Server ip address (i.e: 45.32.161.172)

VPS_USERNAME: Server username (i.e: root)

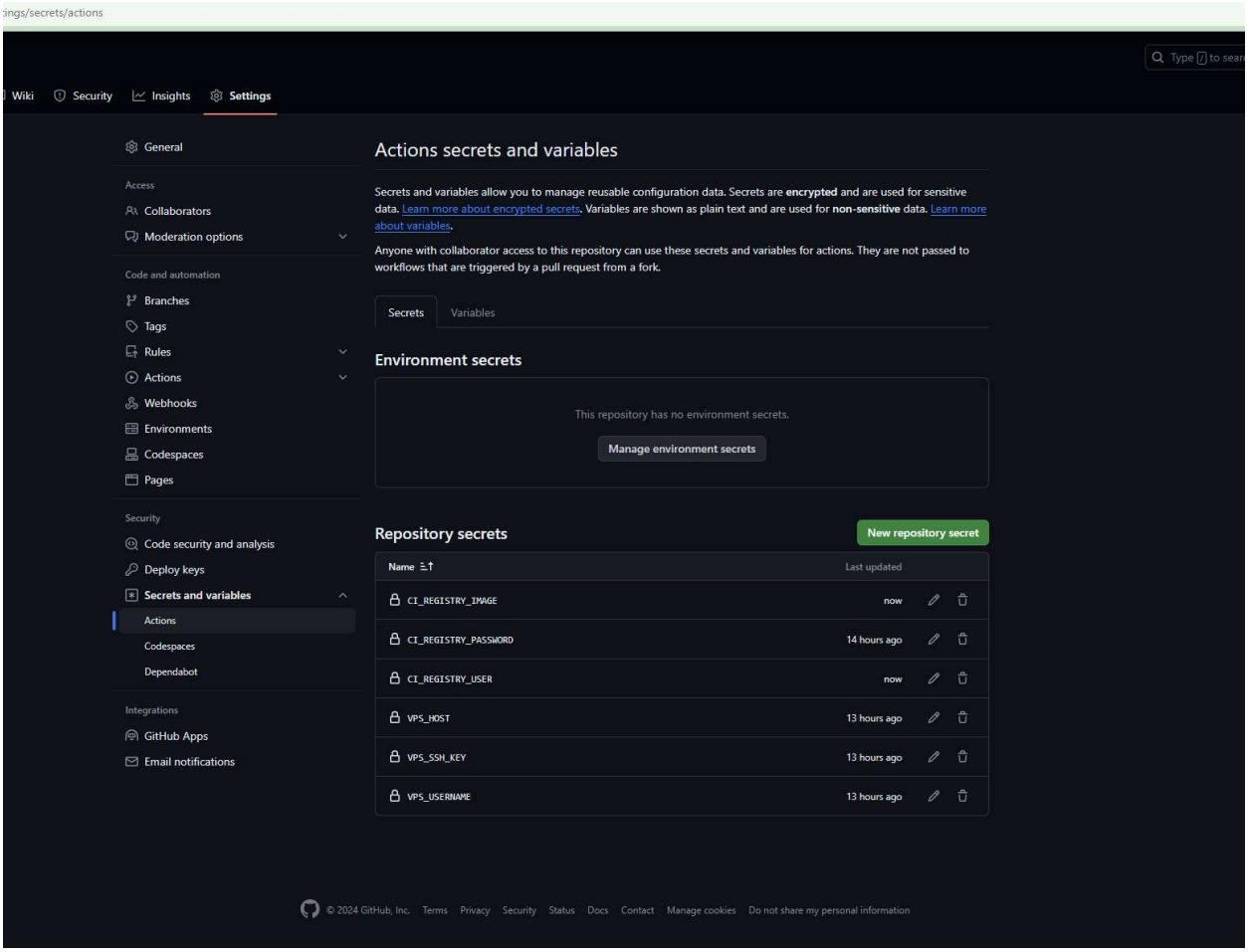
VPS_SSH_KEY:

Generate an SSH key pair on your CI/CD server, and add the public key to the ~/.ssh/authorized_keys file on your VPS. This allows your CI/CD server to connect to your VPS via SSH without a password.
To set the VPS_SSH_KEY as a secret in your GitHub repository, you'll need to follow these steps:

- Generate SSH Key Pair:
  On your local machine, generate an SSH key pair if you don't have one:
  ```
  ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
  ```
  Follow the prompts to generate the keys.

- Add Public Key to VPS:
  Copy the contents of the public key (~/.ssh/id_rsa.pub by default) and add it to the ~/.ssh/authorized_keys file on your VPS. This allows the GitHub Actions workflow to authenticate with your VPS using this key.

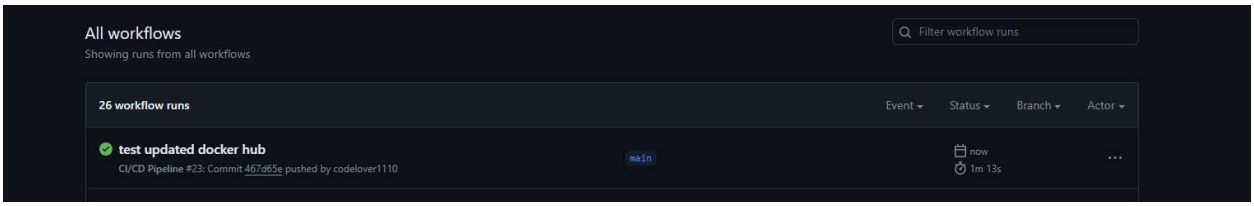
- Create GitHub Repository Secret:
  - Go to your GitHub repository.
  - Click on "Settings" in the repository menu.
  - In the left sidebar, click on "Secrets."
  - Click on "New repository secret."
  - Name the secret VPS_SSH_KEY and paste the contents of your private key (~/.ssh/id_rsa by default).



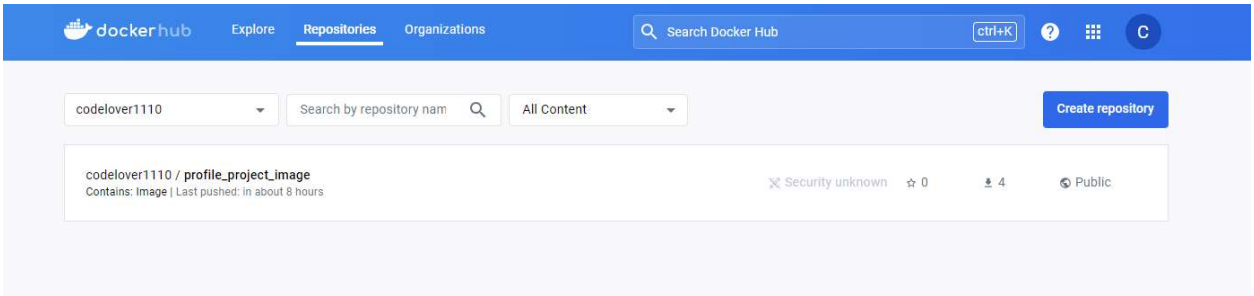
4.3 Push to Git

```
$ git add .

$ git commit –m “updated xxx”

$ git push
```

You can see all workflows in the Github Actions



And you can see Dockerhub repository is updated.



Now we can see the app is running on the server.

For example,

http://45.32.161.172/admin

username: test_user

password: test_pass

To create a superuser on the server, you can run this command

```
$ docker exec -it profile_project_backend python manage.py createsuperuser
```

Then, follow the prompts to set super user.

5. Conclusion

Congratulations! Your Django application is now deployed using Docker, Dockerhub, Gunicorn, Nginx, and GitHub CI/CD workflows.