

DevOps Intern Assignment for Soulpage IT Solutions Pvt. Ltd.

[Github-link](#)

Part 1: Deployment and Configuration

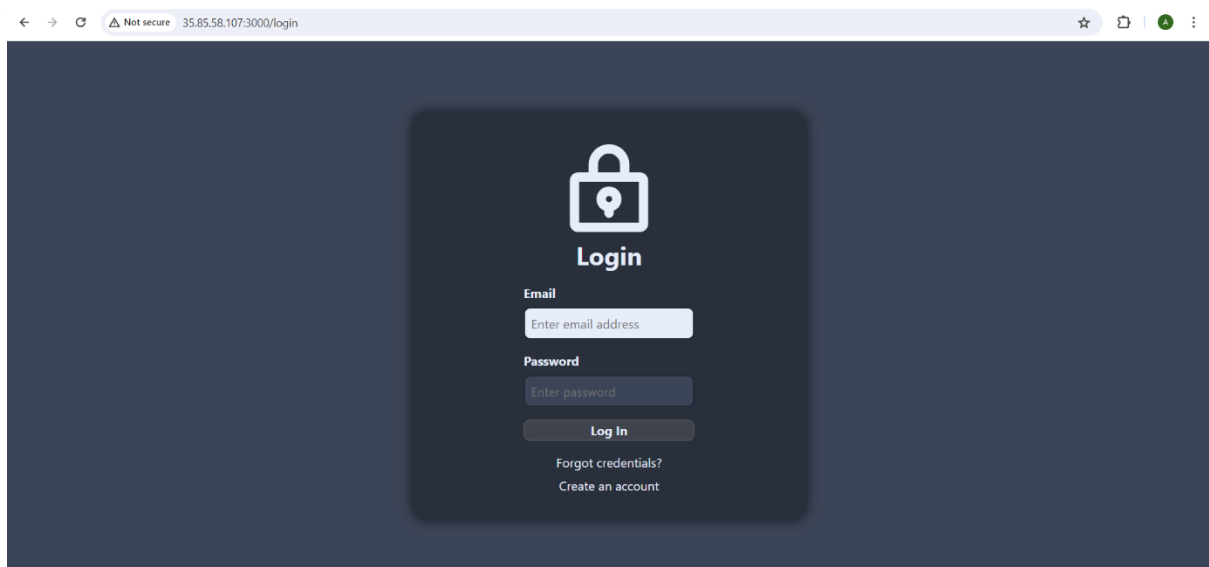
Next.js Deployment: Deployed the Next.js application port number 3000 ,Using the given below command

1. `git clone <repository_url>`
2. `npm install`
3. `npm run build`
4. `npm start`

then updated security group port 3000, anyone can access.

the application is accessible via a web browser using the public-IP

`http:// 35.85.58.107:3000/`



Django Deployment:

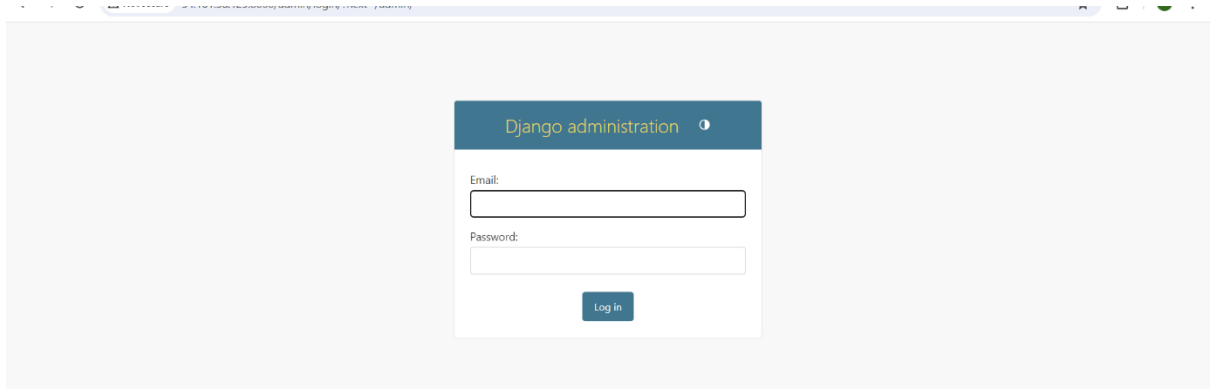
Deployed the Next.js application port number 8000 ,Using the given below command

then updated security group port 8000, anyone can access.

the application is accessible via a web browser using the public-IP

`http:// 35.85.58.107:8000/admin/`

1. `git clone <repository_url>`
2. `python3 -m venv venv source venv/bin/activate`
3. `pip install -r requirements.txt`
4. `pip install gunicorn`
5. `python manage.py migrate`
6. `python manage.py migrate`

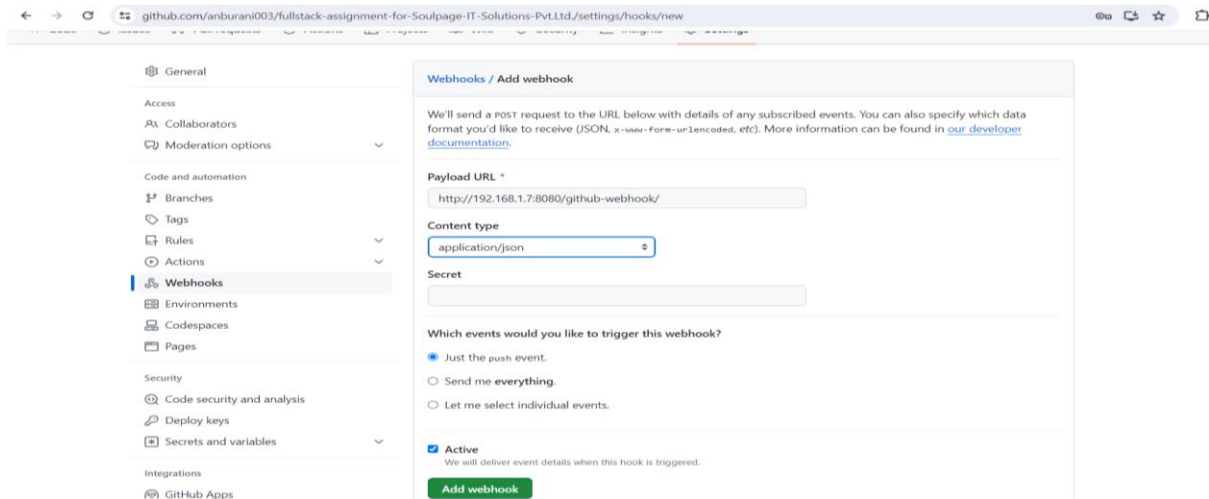


Part 2: Tasks CI/CD Pipeline:

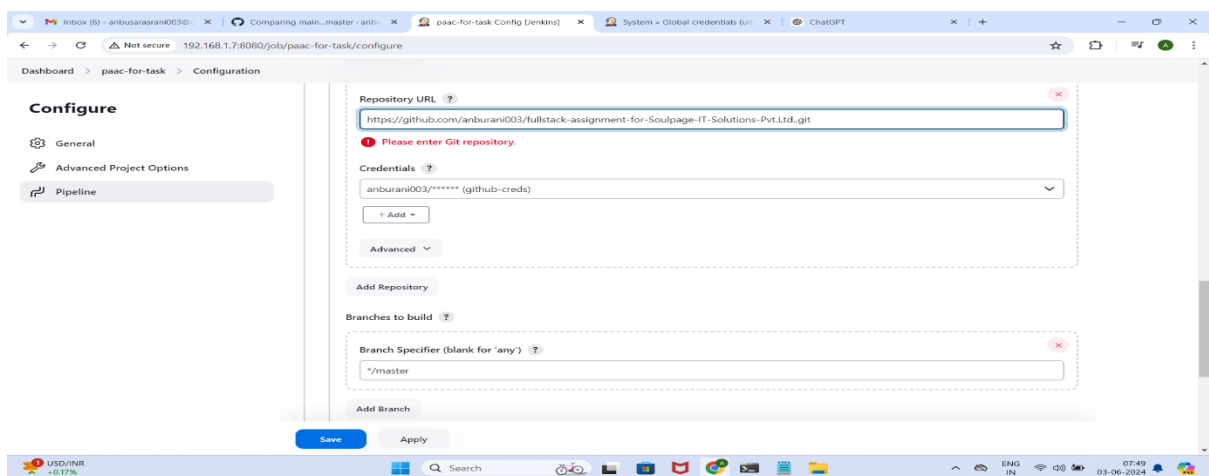
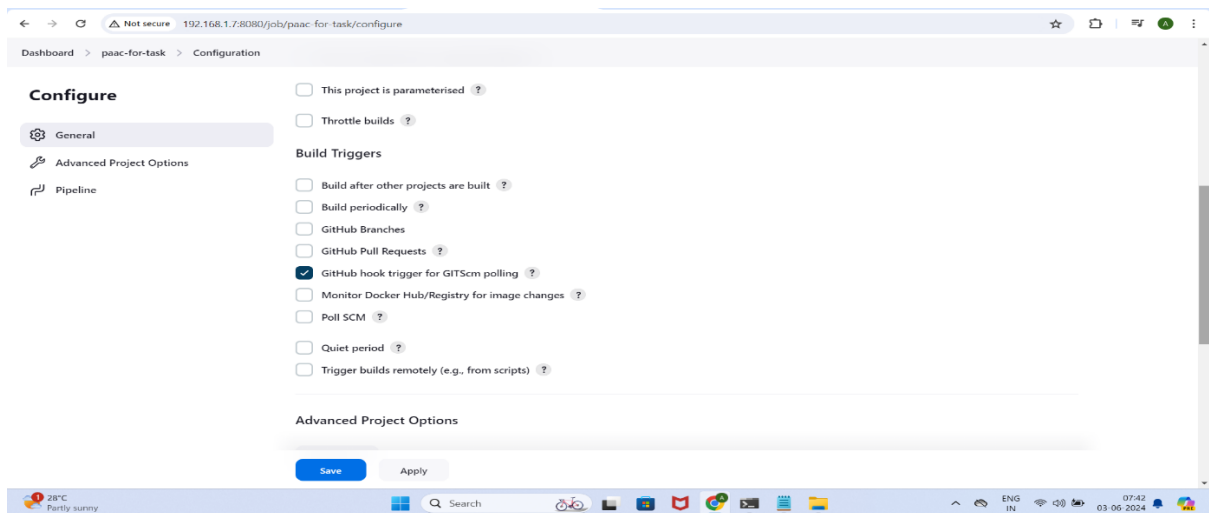
Configure the pipeline to automatically build, and deploy the applications on every push and changes to the repository. Using the tool Jenkins, GitHub and Docker hub

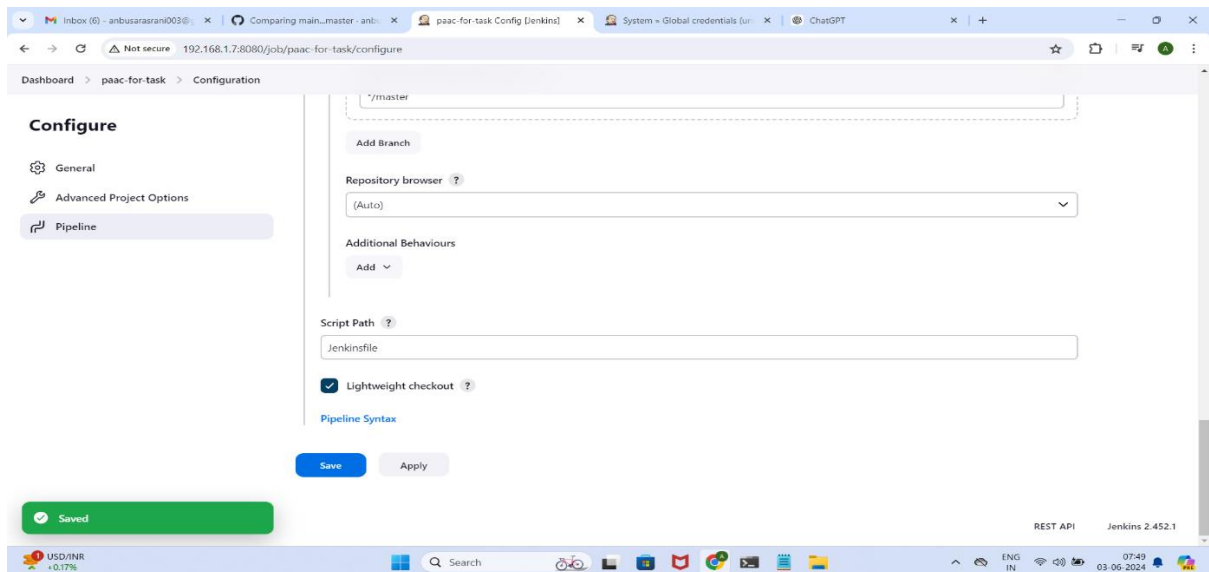
1. Pushed the all Dockerfile for frontend and backend ,docker-compose.yml and other required files to GitHub.
2. Created and configured the Jenkins for CI/CD.
3. Integrated Jenkins & GitHub for continuous integration.
4. Enabled CI/CD pipeline in Jenkins for automation process for every code push and change to GitHub
5. Automated process to build Docker image and push it to Docker hub.

Creating webhook (created the webhook within project repo)



Setting the CI/CD pipeline process for automated docker images build and deploy to docker hub.





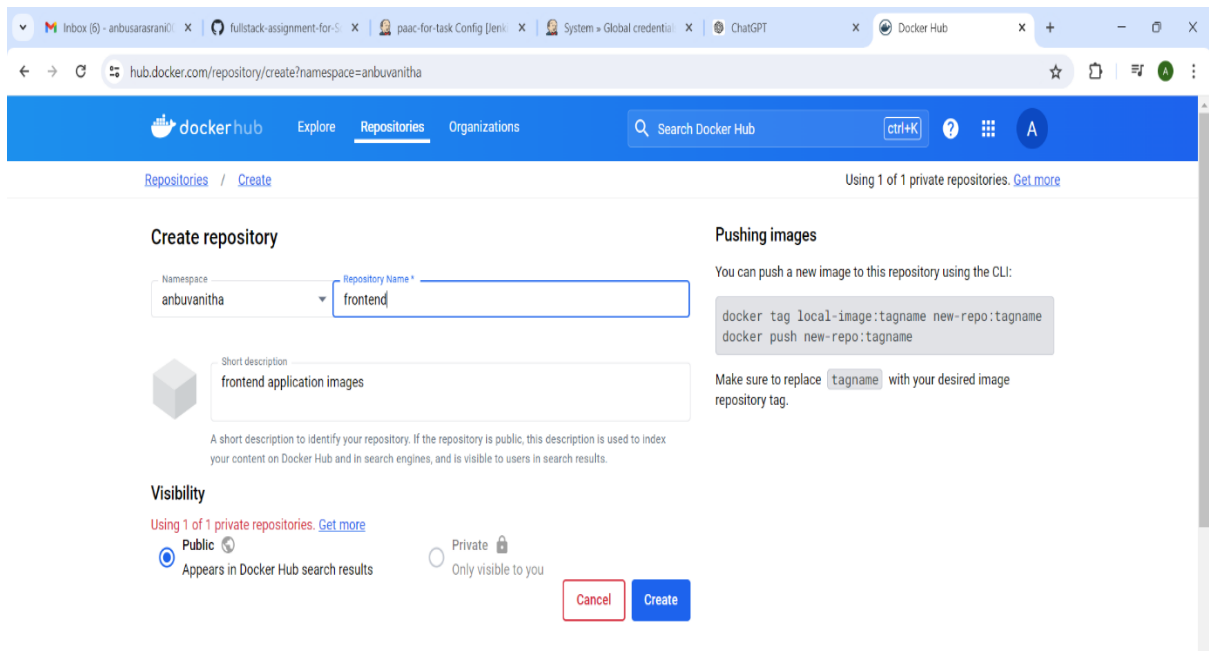
Creating the repository in Docker hub (created the repo name same as building Docker images name.)

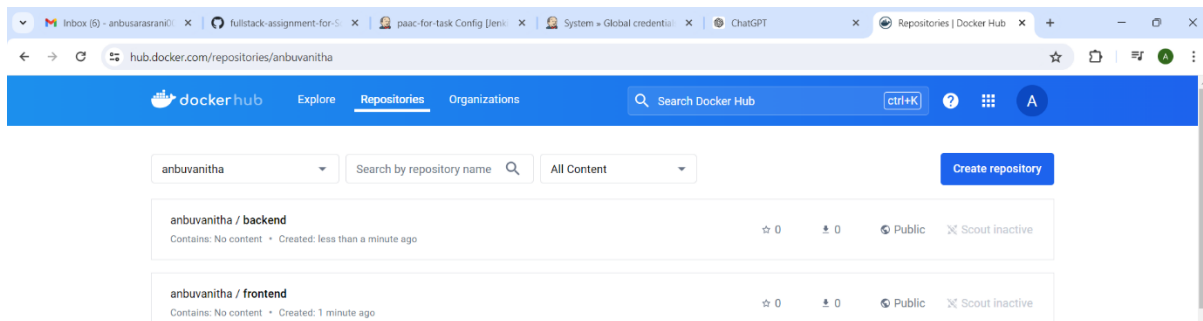
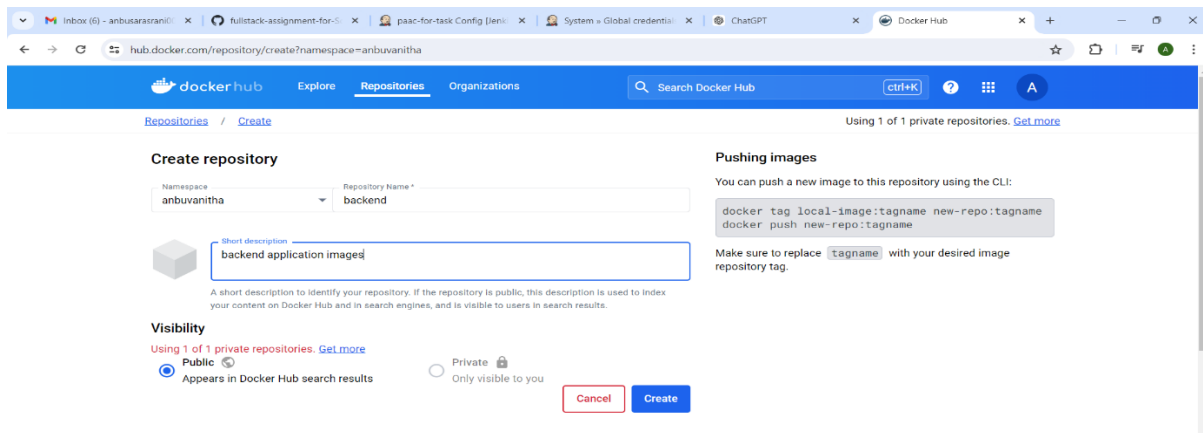
```
docker build -t anbuvanitha/frontend:v1 ./frontend
```

```
docker build -t anbuvanitha/backend:v1 ./backend
```

```
docker push anbuvanitha/frontend:v1
```

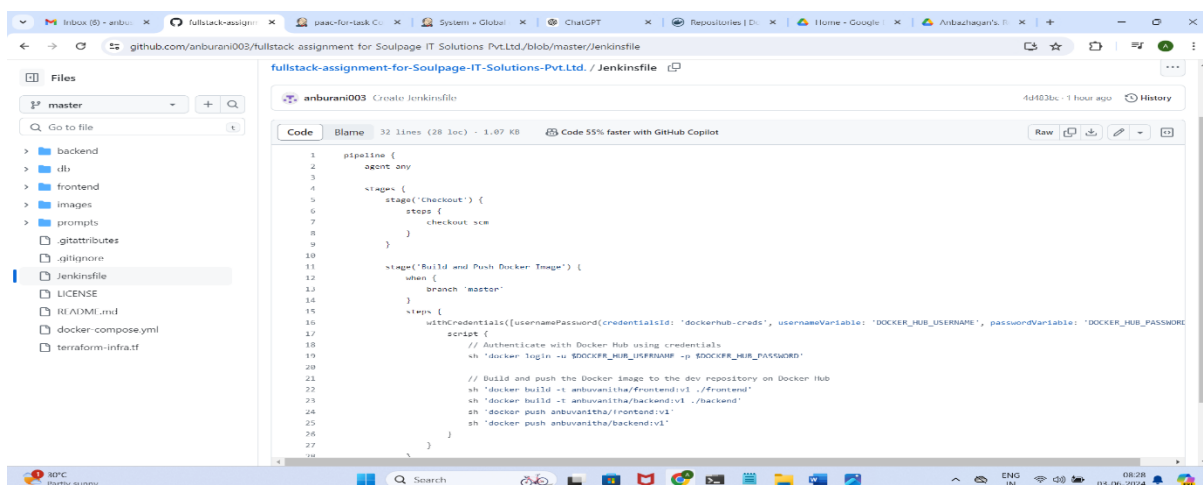
```
docker push anbuvanitha/backend:v1
```





Jenkinsfile (contain the Jenkinsfile in project repository -

<https://github.com/anburani003/fullstack-assignment-for-Soulpage-IT-Solutions-Pvt.Ltd..git>)



Then , run the CI/CD pipeline ,push the codes to the Github repo and make changes in the repo automated build the images and deploy images to Docker hub .

Part 3: Infrastructure as Code (IaC) and Dockerization

Write the terraform template for to automate the provisioning of infrastructure of both Next.js and Django applications. File name main.tf

```
GNU nano 7.2 main.tf
# Define provider (e.g., AWS)
provider "aws" {
  region = "us-west-2"
}

# Define resources (e.g., EC2 instance)
resource "aws_instance" "app_server" {
  ami           = "ami-0cf2b4e024cdb6960"
  instance_type = "t2.micro"
  tags = [
    {
      Name = "app_server"
    }
  ]
}

# Use user data to install dependencies and set up environment
user_data = <<EOF
#!/bin/bash
sudo apt-get update
sudo apt-get install -y nodejs npm python3 python3-pip

# Clone repositories
git clone https://github.com/your_nextjs_app.git /home/ubuntu/nextjs_app
git clone https://github.com/your_django_app.git /home/ubuntu/django_app

# Install Next.js dependencies
EOF

git clone https://github.com/your_nextjs_app.git /home/ubuntu/nextjs_app
git clone https://github.com/your_django_app.git /home/ubuntu/django_app

# Install Next.js dependencies
cd /home/ubuntu/nextjs_app
npm install

# Install Django dependencies
cd /home/ubuntu/django_app
pip3 install -r requirements.txt

# Run Django migrations
python3 manage.py migrate

# Start Next.js and Django servers
nohup npm --prefix /home/ubuntu/nextjs_app run dev &
nohup python3 /home/ubuntu/django_app/manage.py runserver 0.0.0.0:8000 &
EOF

# Define outputs
output "app_server_ip" {
  value = aws_instance.app_server.public_ip
}
```

i-080e273e4c4ac8e06 (terraform)
PublicIPs: 52.36.53.76 PrivateIPs: 172.31.20.56

```
GNU nano 7.2 main.tf
git clone https://github.com/your_nextjs_app.git /home/ubuntu/nextjs_app
git clone https://github.com/your_django_app.git /home/ubuntu/django_app

# Install Next.js dependencies
cd /home/ubuntu/nextjs_app
npm install

# Install Django dependencies
cd /home/ubuntu/django_app
pip3 install -r requirements.txt

# Run Django migrations
python3 manage.py migrate

# Start Next.js and Django servers
nohup npm --prefix /home/ubuntu/nextjs_app run dev &
nohup python3 /home/ubuntu/django_app/manage.py runserver 0.0.0.0:8000 &
EOF

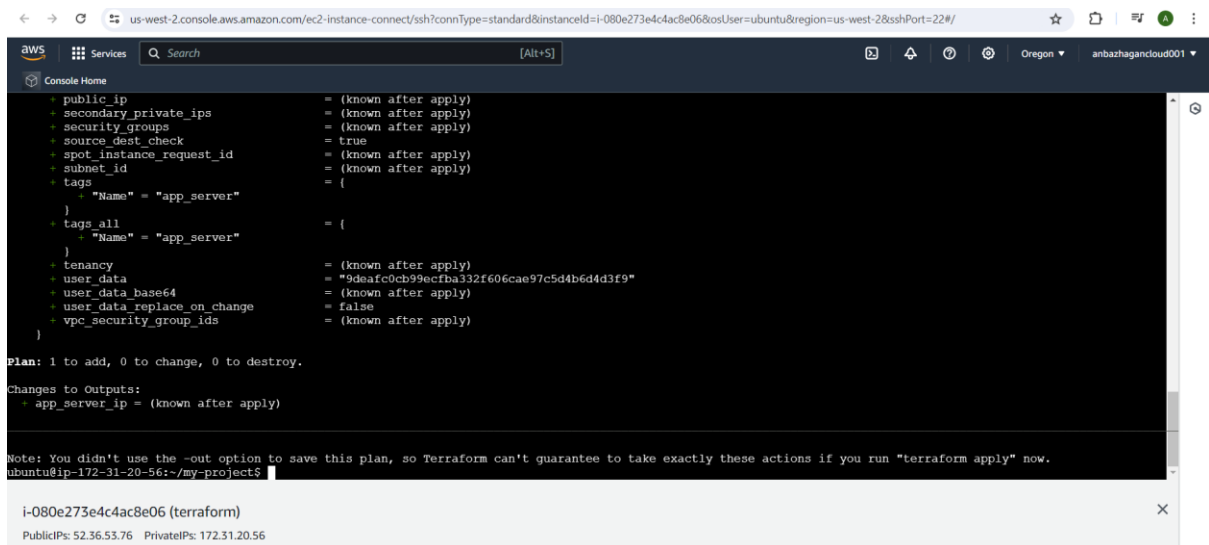
# Define outputs
output "app_server_ip" {
  value = aws_instance.app_server.public_ip
}
```

i-080e273e4c4ac8e06 (terraform)
PublicIPs: 52.36.53.76 PrivateIPs: 172.31.20.56

Then used the commands:

1. Terraform init
2. Terraform validate
3. Terraform fmt
4. Terraform plan
5. Terraform apply

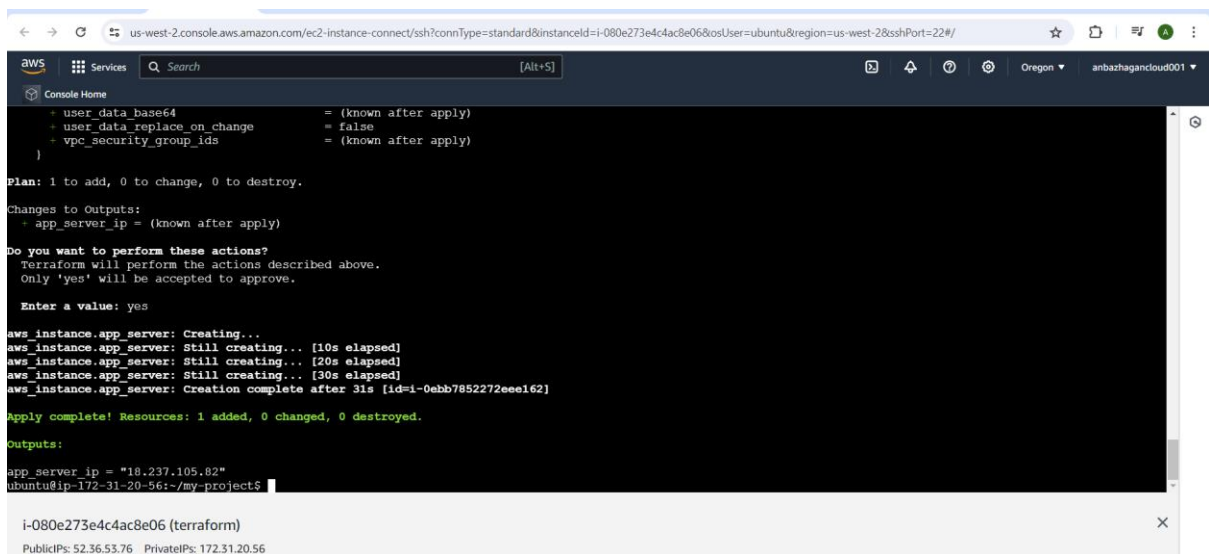
After run the Terraform plan command



The screenshot shows the AWS Management Console interface. The main content area displays the output of a Terraform plan command. It lists various attributes of an EC2 instance, such as public ip, secondary private ips, security groups, source_dest_check, spot_instance_request_id, subnet_id, tags, tags_all, tenancy, user_data, user_data_base64, user_data_replace_on_change, and vpc_security_group_ids. Each attribute is followed by its value or a note like "(known after apply)". Below the plan, it states "Plan: 1 to add, 0 to change, 0 to destroy." and "Changes to Outputs: app_server_ip = (known after apply)". A note at the bottom says: "Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run 'terraform apply' now." The terminal prompt is "ubuntu@ip-172-31-20-56:~/my-project\$". At the bottom, a notification bar shows the instance ID "i-080e273e4c4ac8e06 (terraform)" and its public and private IP addresses.

```
us-west-2.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-080e273e4c4ac8e06&osUser=ubuntu&region=us-west-2&sshPort=22#/  
[Alt+S]  
Oregon ▾ anbazhagancoud001 ▾  
Console Home  
+ public ip = (known after apply)  
+ secondary_private_ips = (known after apply)  
+ security_groups = (known after apply)  
+ source_dest_check = true  
+ spot_instance_request_id = (known after apply)  
+ subnet_id = (known after apply)  
+ tags = {  
  + "Name" = "app_server"  
}  
+ tags_all = {  
  + "Name" = "app_server"  
}  
+ tenancy = (known after apply)  
+ user_data = "9daafc0cb99ecfba332f606cae97c5d4b6d4d3f9"  
+ user_data_base64 = (known after apply)  
+ user_data_replace_on_change = false  
+ vpc_security_group_ids = (known after apply)  
}  
Plan: 1 to add, 0 to change, 0 to destroy.  
Changes to Outputs:  
+ app_server_ip = (known after apply)  
  
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.  
ubuntu@ip-172-31-20-56:~/my-project$  
  
i-080e273e4c4ac8e06 (terraform)  
PublicIPs: 52.36.53.76 PrivateIPs: 172.31.20.56
```

After run the Terraform apply command



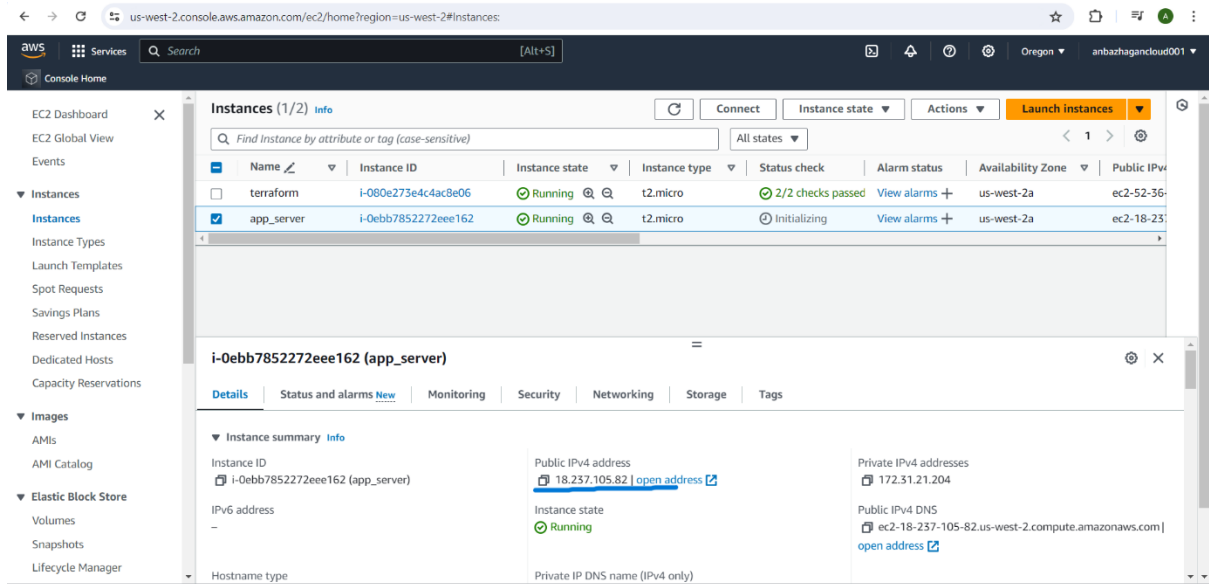
The screenshot shows the AWS Management Console interface. The main content area displays the output of a Terraform apply command. It lists various attributes of an EC2 instance, such as user_data_base64, user_data_replace_on_change, and vpc_security_group_ids. Each attribute is followed by its value or a note like "(known after apply)". Below the plan, it states "Plan: 1 to add, 0 to change, 0 to destroy." and "Changes to Outputs: app_server_ip = (known after apply)". It then asks "Do you want to perform these actions?" and "Terraform will perform the actions described above. Only 'yes' will be accepted to approve." The user enters "yes". The output shows the instance creation progress: "aws_instance.app_server: Creating...", "aws_instance.app_server: Still creating... [10s elapsed]", "aws_instance.app_server: Still creating... [20s elapsed]", "aws_instance.app_server: Still creating... [30s elapsed]", and "aws_instance.app_server: Creation complete after 31s [id=i-0ebb7852272eeei62]". It then states "Apply complete! Resources: 1 added, 0 changed, 0 destroyed." and "Outputs: app_server_ip = '18.237.105.82'". The terminal prompt is "ubuntu@ip-172-31-20-56:~/my-project\$". At the bottom, a notification bar shows the instance ID "i-080e273e4c4ac8e06 (terraform)" and its public and private IP addresses.

```
us-west-2.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-080e273e4c4ac8e06&osUser=ubuntu&region=us-west-2&sshPort=22#/  
[Alt+S]  
Oregon ▾ anbazhagancoud001 ▾  
Console Home  
+ user_data_base64 = (known after apply)  
+ user_data_replace_on_change = false  
+ vpc_security_group_ids = (known after apply)  
}  
Plan: 1 to add, 0 to change, 0 to destroy.  
Changes to Outputs:  
+ app_server_ip = (known after apply)  
  
Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.  
  
Enter a value: yes  
  
aws_instance.app_server: Creating...  
aws_instance.app_server: Still creating... [10s elapsed]  
aws_instance.app_server: Still creating... [20s elapsed]  
aws_instance.app_server: Still creating... [30s elapsed]  
aws_instance.app_server: Creation complete after 31s [id=i-0ebb7852272eeei62]  
  
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.  
Outputs:  
app_server_ip = "18.237.105.82"  
ubuntu@ip-172-31-20-56:~/my-project$  
  
i-080e273e4c4ac8e06 (terraform)  
PublicIPs: 52.36.53.76 PrivateIPs: 172.31.20.56
```

After provisioning of infrastructure for both Next.js and Django application instance

Instance name - app_server

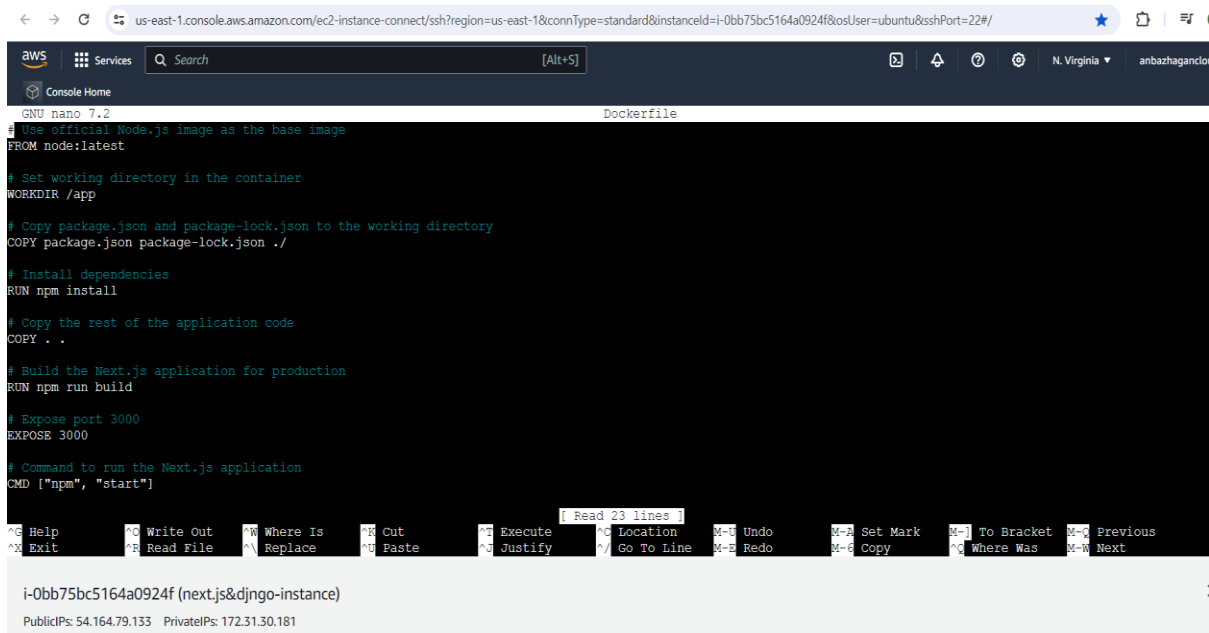
Public-IP - 18.237.105.82



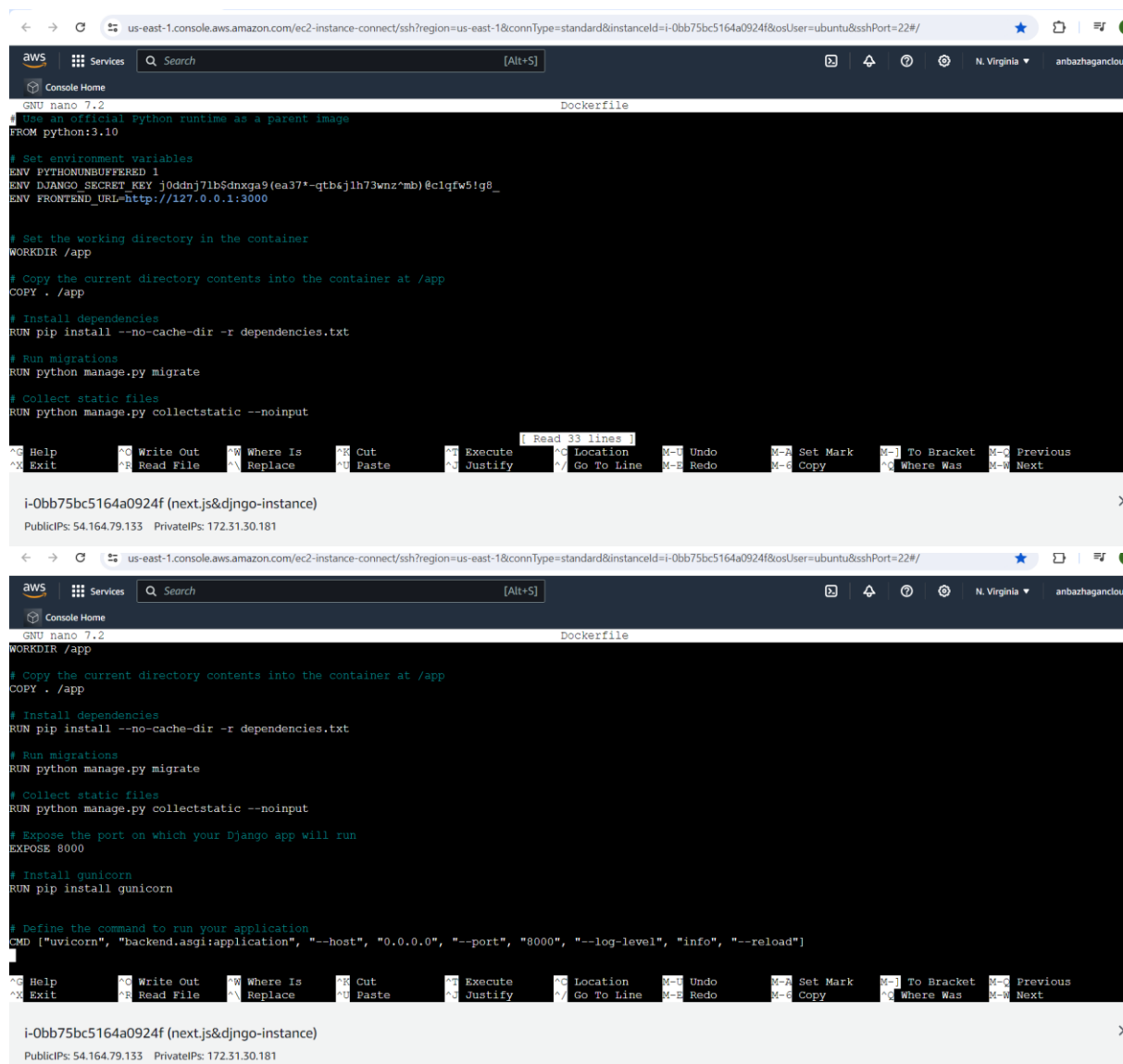
Dockerization:

Create Dockerfiles for both the Next.js and Django applications.

Frontend/ Dockerfile



Backend /Dockerfile



The screenshot displays the AWS Management Console interface for an EC2 instance. The terminal window shows the content of a Dockerfile being edited with nano. The Dockerfile includes instructions to use a Python 3.10 runtime, set environment variables for Django, copy the application code, install dependencies, run migrations, collect static files, and expose port 8000. The instance details at the bottom show the instance ID i-0bb75bc5164a0924f and its public/private IP addresses.

```
GNU nano 7.2 Dockerfile
# Use an official Python runtime as a parent image
FROM python:3.10

# Set environment variables
ENV PYTHONUNBUFFERED 1
ENV DJANGO_SECRET_KEY j0ddnj7lb6dnxga9(ea37*-qtb6j1h73wnz*mb)@clqfw5!g8_
ENV FRONTEND_URL=http://127.0.0.1:3000

# Set the working directory in the container
WORKDIR /app

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

# Install dependencies
RUN pip install --no-cache-dir -r dependencies.txt

# Run migrations
RUN python manage.py migrate

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

# Help
^G Write Out
^X Exit      ^R Read File
^W Where Is  ^C Cut
^_ Replace   ^V Paste
^B Execute   ^J Justify
^_ Location  ^L Undo
^_ Go To Line ^E Redo
^M Set Mark  ^I To Bracket
^_ Where Was ^N Previous
^_ Next

i-0bb75bc5164a0924f (next.js&django-instance)
PublicIPs: 54.164.79.133  PrivateIPs: 172.31.30.181
```

Docker-Compose.yml file



The screenshot displays the AWS Management Console interface for an EC2 instance. The terminal window shows the content of a docker-compose.yml file being edited with nano. The file defines two services: 'backend' and 'frontend'. The 'backend' service is built from the current directory context and exposes port 8000. The 'frontend' service is also built from the current directory context and exposes port 3000. The instance details at the bottom show the instance ID i-0bb75bc5164a0924f and its public/private IP addresses.

```
GNU nano 7.2 docker-compose.yml
version: '3.8'

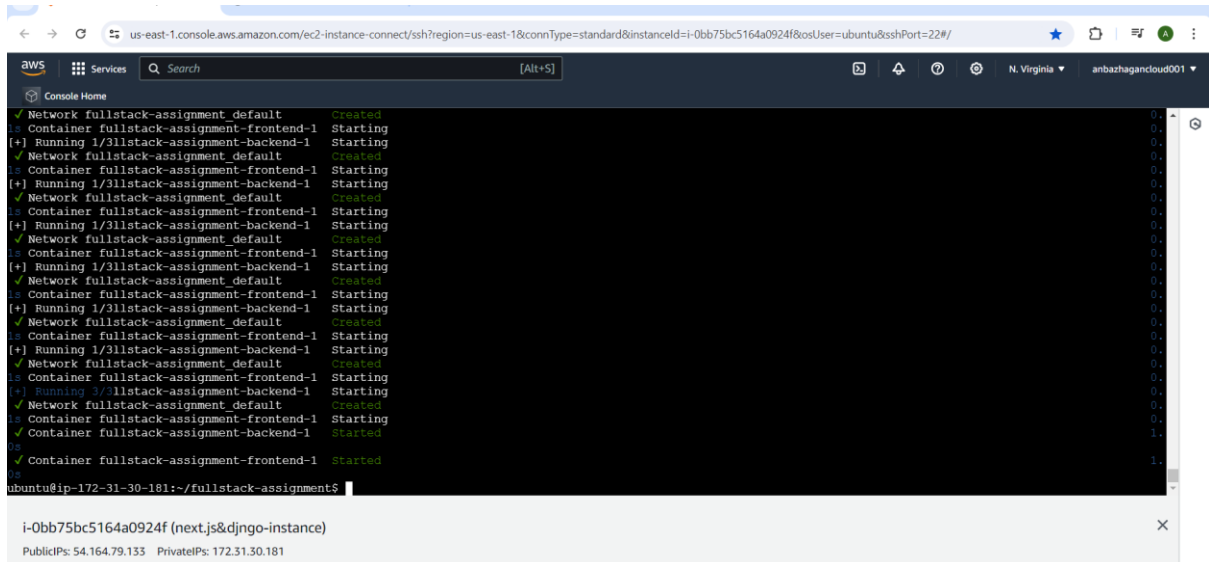
services:
  backend:
    build:
      context: ./backend
    ports:
      - "8000:8000"

  frontend:
    build:
      context: ./frontend
    ports:
      - "3000:3000"

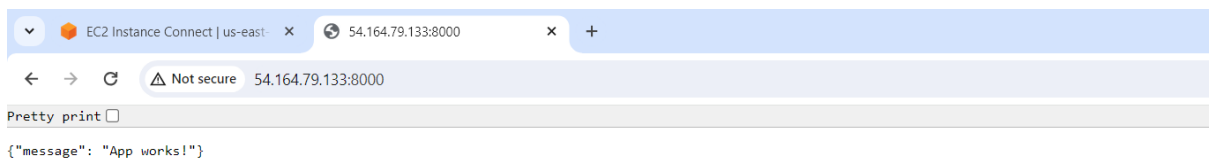
# Help
^G Write Out
^X Exit      ^R Read File
^W Where Is  ^C Cut
^_ Replace   ^V Paste
^B Execute   ^J Justify
^_ Location  ^L Undo
^_ Go To Line ^E Redo
^M Set Mark  ^I To Bracket
^_ Where Was ^N Previous
^_ Next

i-0bb75bc5164a0924f (next.js&django-instance)
PublicIPs: 54.164.79.133  PrivateIPs: 172.31.30.181
```

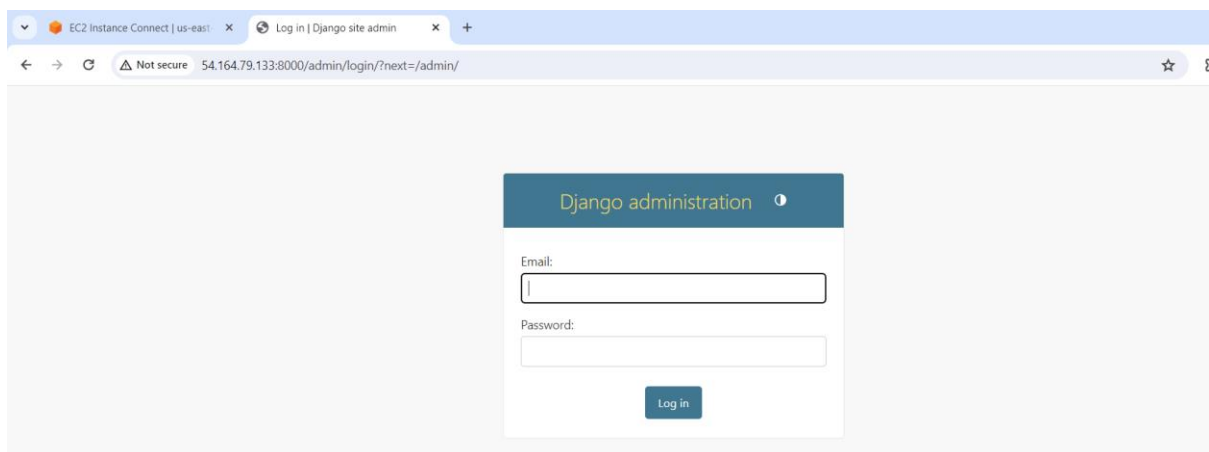
After run the **docker-compose up -d** to build and run the both container applications .



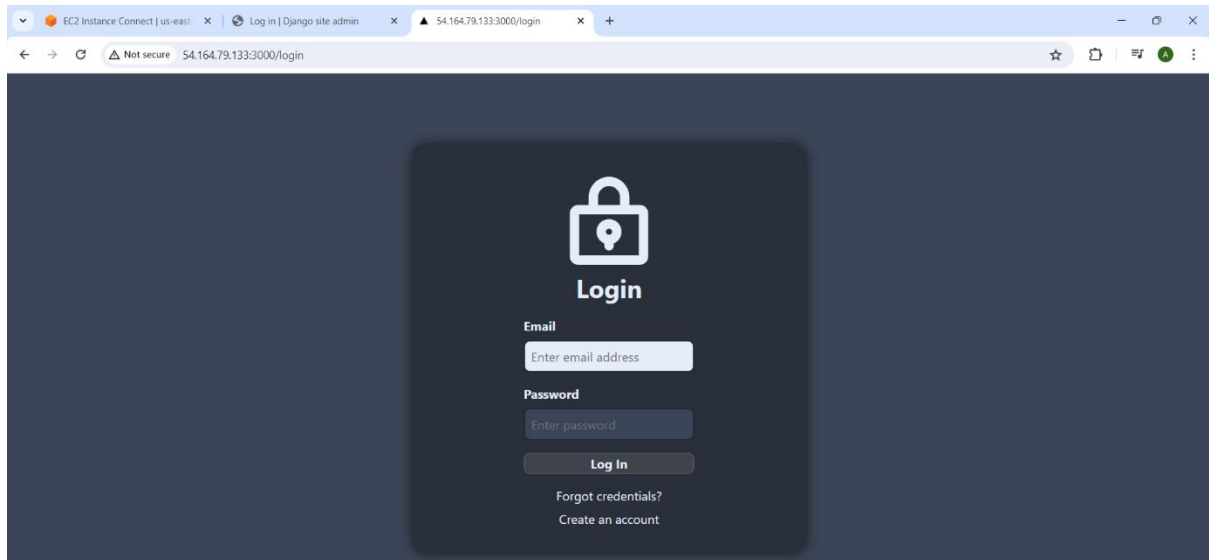
Backend application access with browser using **https://54.164.79.133:8000**



Backend application access with browser using **https://54.164.79.133:8000/admin**

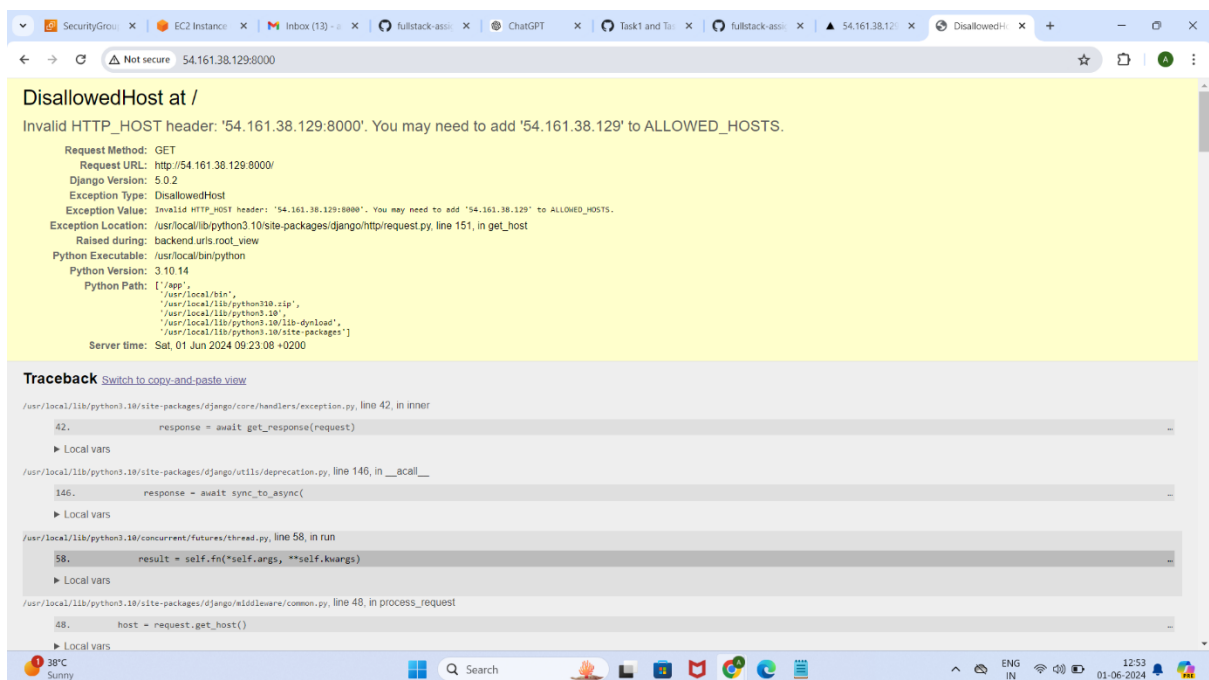


Backend application access with browser using **https://54.164.79.133:3000**



Problem-solving and troubleshooting

After build and deploying the application for Next.js and a Django Python. then access Brower with IP address **http://54.161.38.129:8000/**, facing this issue

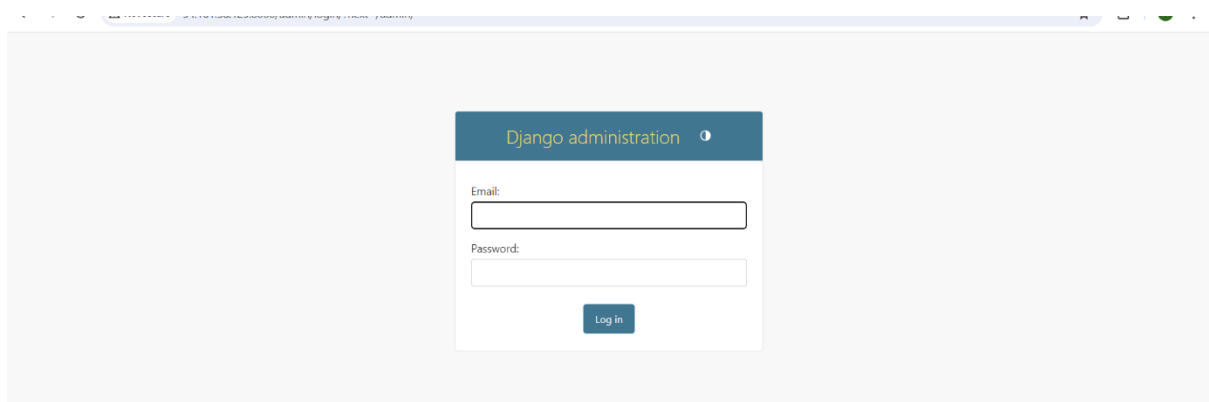


Then update the with in the directory [backend/backend/settings.py](#) file given below scrip

```
ALLOWED_HOSTS = [  
    '127.0.0.1',  
    'localhost',  
    '54.161.38.129',  
]
```

Then rebuild the images then access application through Brower with IP address

<http://54.161.38.129:8000/admin/>



Thank you ..!