**Full-Stack Assignment Documentation**

This repository contains the code and configuration files required to deploy and manage a full-stack application consisting of a Next.js frontend and a Django backend. The repository also includes Dockerization, CI/CD pipeline configuration, and infrastructure as code (IaC) scripts.

**Checking Out the Repository**

$ git clone [github.com/soulpage/fullstack-assignment](https://github.com/soulpage/fullstack-assignment)

## **Installing the Toolchain**

### **On Linux**

You can install the necessary dependencies on a Linux system as follows:

bash

Copy code

$ sudo apt install -y docker docker-compose python3 python3-pip nodejs npm

Next, install Docker Compose if not already installed:

bash

Copy code

$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.3.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose

Ensure the Docker daemon is running:

bash

Copy code

$ sudo systemctl start docker

$ sudo systemctl enable docker

## **Dockerizing the Application**

### **Frontend Dockerfile**

The Dockerfile for the Next.js frontend application is located in the frontend directory. Here's a breakdown of the file:

# Stage 1: Build the app

FROM node:18-alpine AS builder

# Set the working directory

WORKDIR /app

# Copy the package.json and package-lock.json (if available)

COPY package\*.json ./

# Install dependencies

RUN npm install

# Copy the rest of the application code

COPY . .

# Build the Next.js application

RUN npm run build

# Stage 2: Serve the app with a lightweight web server

FROM node:18-alpine AS runner

# Set the working directory

WORKDIR /app

# Install only production dependencies

COPY --from=builder /app/package\*.json ./

RUN npm install --production

# Copy the built Next.js application

COPY --from=builder /app/.next ./.next

COPY --from=builder /app/public ./public

# Expose the port the app runs on

EXPOSE 3000

# Start the Next.js application

CMD ["npm", "start"]

### **Backend Dockerfile**

The Dockerfile for the Django backend application is located in the backend directory. Here’s the breakdown:

# Use an official Python runtime as a parent image

FROM python:3.11-slim

# Set the working directory in the container

WORKDIR /app

# Set environment variables

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Install dependencies

RUN pip install --upgrade pip

COPY dependencies.txt /app/

RUN pip install -r dependencies.txt

# Copy the Django project code into the container

COPY . /app/

# Generate a SECRET\_KEY and set it as an environment variable

RUN SECRET\_KEY=$(python -c 'from django.core.management.utils import get\_random\_secret\_key; print(get\_random\_secret\_key())') && \

    echo "DJANGO\_SECRET\_KEY=$SECRET\_KEY" >> /app/.env

# Automatically retrieve frontend and backend URLs from Docker Compose environment variables

ARG FRONTEND\_URL

ARG BACKEND\_URL

RUN echo "FRONTEND\_URL=$FRONTEND\_URL" >> /app/.env && \

    echo "BACKEND\_URL=$BACKEND\_URL" >> /app/.env

# Run database migrations (optional)

# RUN python manage.py migrate

# Collect static files (for production)

RUN python manage.py collectstatic --noinput

# Expose the port the app runs on

EXPOSE 8000

# Start the Django server using Gunicorn

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "backend.wsgi:application"]

### **Docker Compose File**

The docker-compose.yml file is used to manage multi-container deployments. It can be found at the root of the repository:

version: '3.8'

services:

  frontend:

    build: ./frontend

    ports:

      - "3000:3000"

    environment:

      - NODE\_ENV=production

    networks:

      - app-network

  backend:

    build:

      context: ./backend

      dockerfile: Dockerfile

      args:

        FRONTEND\_URL: "http://127.0.0.1:4000"

        BACKEND\_URL: "http://127.0.0.1:8000"

    ports:

      - "8000:8000"

    environment:

      - DJANGO\_ENV=production

    networks:

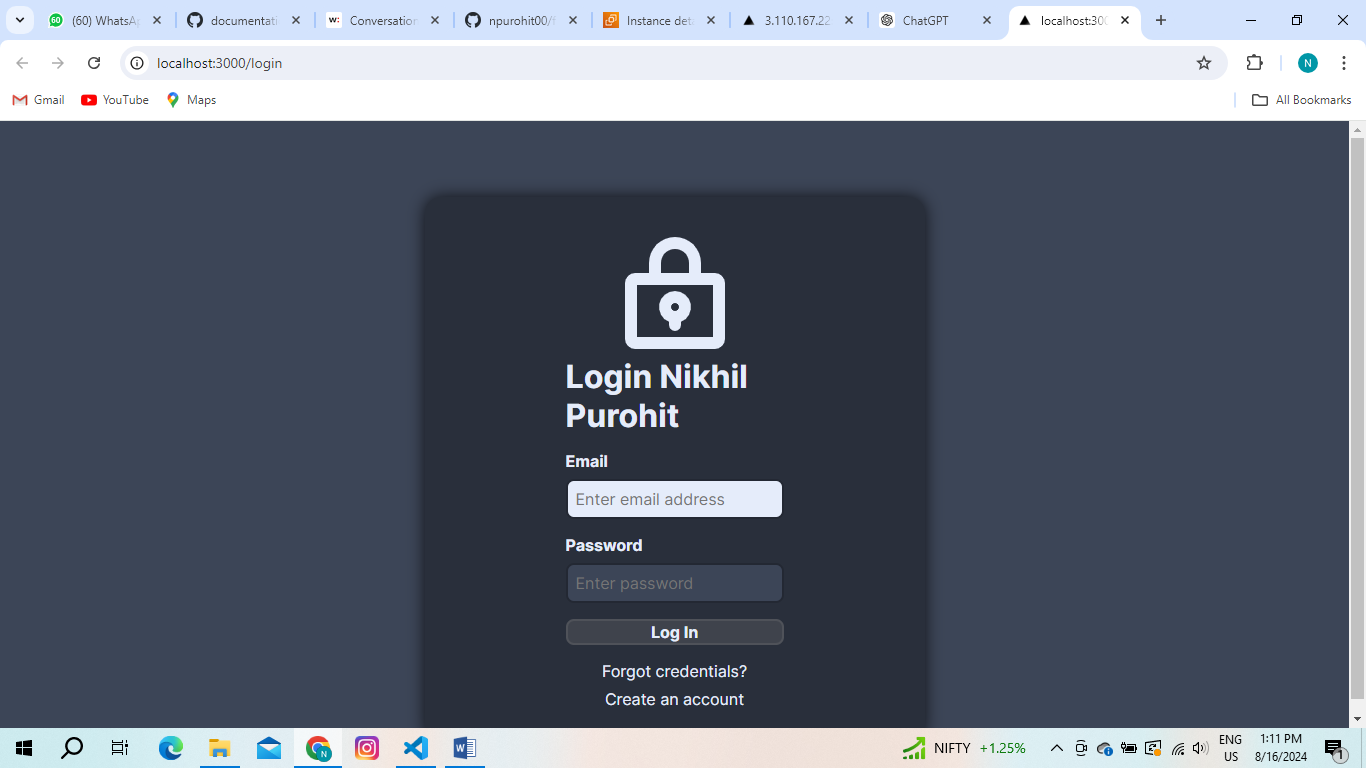
      - app-network

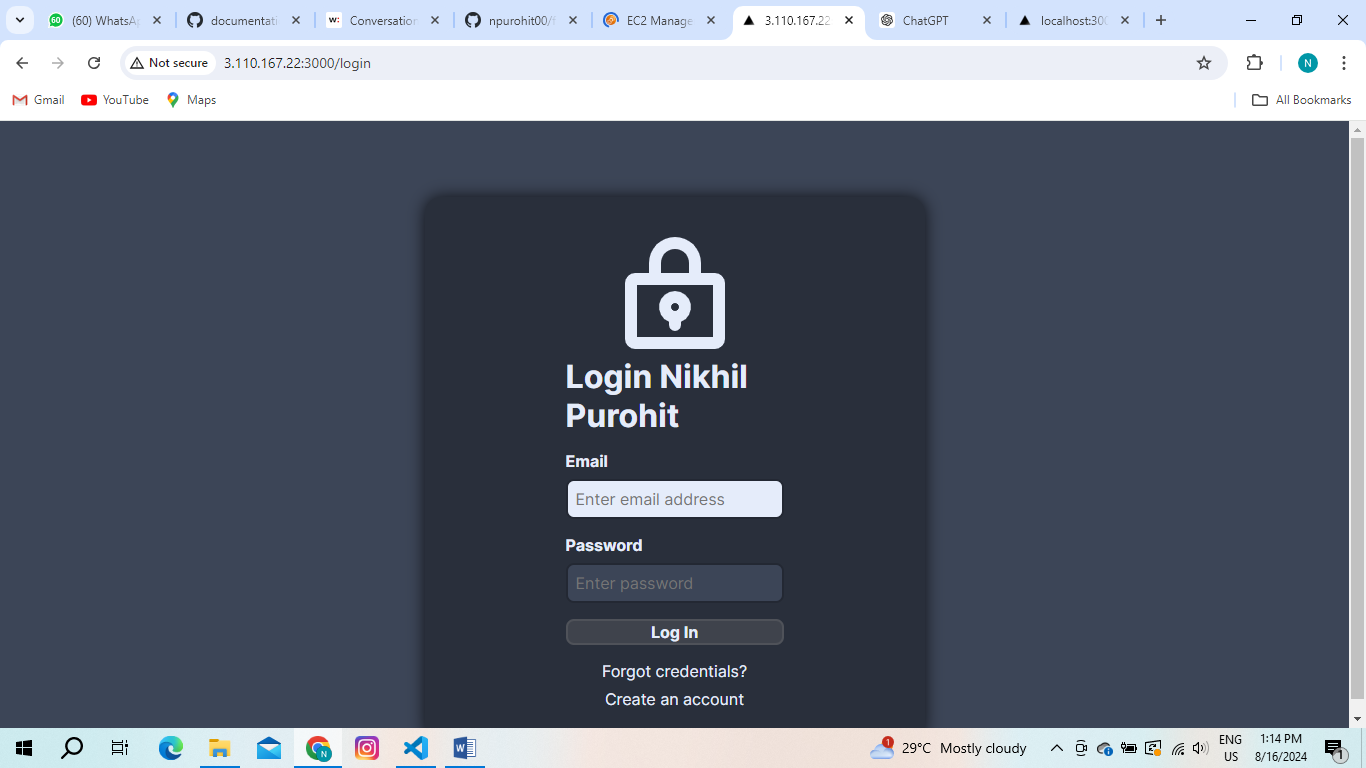
networks:

  app-network:

Command for running the Docker container  
  
sudo docker-compose up –build

Initially I have run the docker container in the local and then next step is to push the code to the aws EC2 for the deployment of the project .



Next step I have created a EC2 and clone the code in the aws EC2 and run the code using the command sudo docker-compose up –build.  
  
  
  
  
  
  
  
**Next step is to make the project CI/CD pipeline using tool GitHub Actions**

**CI/CD Pipeline Using GitHub Actions**

**Overview**

This pipeline will automatically build, test, and deploy your full-stack application (Next.js frontend and Django backend) to an AWS EC2 instance whenever you push code to a specific branch.

**Setting Up GitHub Secrets**

Before configuring the GitHub Actions workflow, you need to set up some secrets in your GitHub repository:

1. **EC2\_SSH\_KEY**: The SSH private key to connect to your EC2 instance.
2. **EC2\_USER**: The username for the EC2 instance (typically ec2-user for Amazon Linux).
3. **EC2\_HOST**: The public IP address or hostname of your EC2 instance.

To add these secrets:

* Go to your GitHub repository.
* Navigate to **Settings** > **Secrets and variables** > **Actions**.
* Click on **New repository secret** and add the above secrets.

**GitHub Actions Workflow (deploy.yml)**

Create a file named deploy.yml in the .github/workflows directory of your repository. This file will define the steps of your CI/CD pipeline.

name: CI/CD Pipeline

on:

  push:

    branches:

      - nikhil  # Trigger the workflow on pushes to the 'nikhil' branch

jobs:

  deploy:

    runs-on: ubuntu-latest

    steps:

    - name: Checkout code

      uses: actions/checkout@v3

    - name: Set up SSH

      uses: webfactory/ssh-agent@v0.5.4

      with:

        ssh-private-key: ${{ secrets.EC2\_SSH\_KEY }}

    - name: Copy Docker Compose file to EC2

      run: scp -o StrictHostKeyChecking=no -r docker-compose.yml ${{ secrets.EC2\_USER }}@${{ secrets.EC2\_HOST }}:~/fullstack/

    - name: Copy backend files to EC2

      run: scp -o StrictHostKeyChecking=no -r backend ${{ secrets.EC2\_USER }}@${{ secrets.EC2\_HOST }}:~/fullstack/

    - name: Copy frontend files to EC2

      run: scp -o StrictHostKeyChecking=no -r frontend ${{ secrets.EC2\_USER }}@${{ secrets.EC2\_HOST }}:~/fullstack/

    - name: Deploy on EC2

      run: |

        ssh -o StrictHostKeyChecking=no ${{ secrets.EC2\_USER }}@${{ secrets.EC2\_HOST }} << EOF

          cd ~/fullstack

          docker-compose down

          docker-compose up -d --build

        EOF

"After adding all the required files for the GitHub Actions, I pushed the code again with some changes to the login UI and noticed the code updates."