StudyBud — Deployment Documentation (Group C)

Project: StudyBud
Group: Group C
Date: 13 Dec 2025
Live Application URL:
➡️ http://98.92.250.22:9000

---

1. Project Overview

This document describes the complete CI/CD deployment process for the StudyBud application. The deployment uses Jenkins, Docker, and AWS EC2 to automate building, packaging, and releasing the application.
The system supports persistent user data, allowing users to:

- Sign up
- Log in after redeployment
- Create posts and TGroups without losing data

---

2. Architecture Summary

- Source Control: GitHub
- CI/CD Tool: Jenkins
- Containerization: Docker (multi-stage build)
- Server: AWS EC2 (Ubuntu 24.04 LTS)
- Backend Framework: Django
- Database: SQLite (persistent)
- Branch Used: GroupC

500MB to ~200MB.

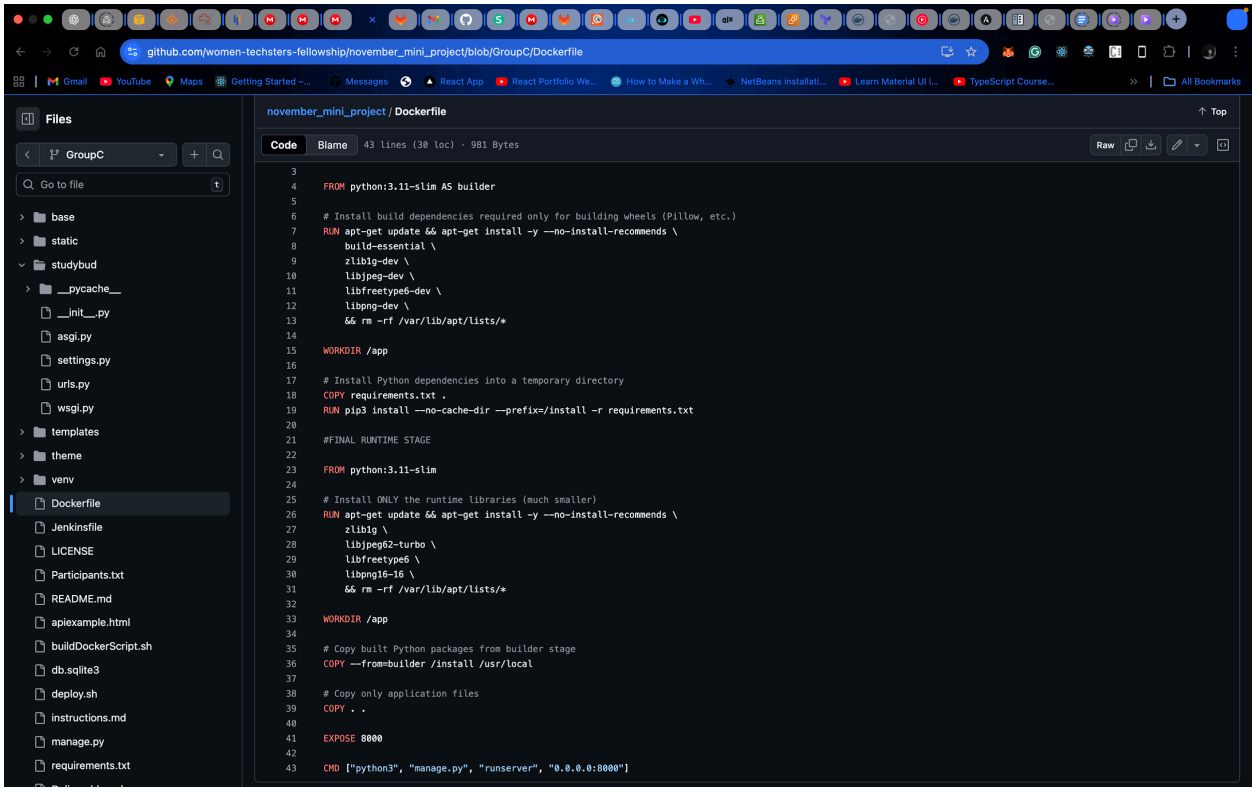
3. Deployment Artifacts (Screenshots)

The following deployment components were implemented and verified:

- Dockerfile
- Jenkinsfile
- buildDockerScript.sh
- deploy.sh
- Django settings.py (ALLOWED_HOSTS)
- Jenkins successful build UI
- Live application running on EC2

4. Docker Configuration

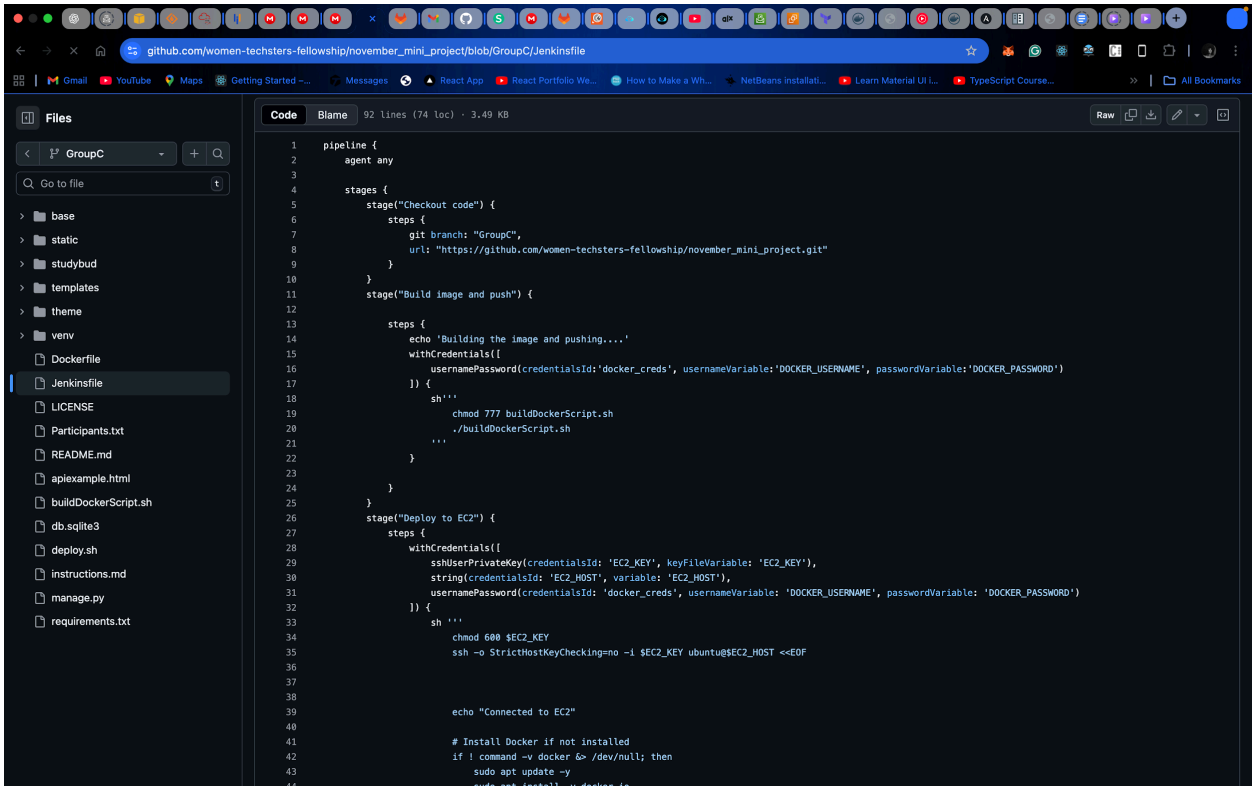
The Dockerfile defines how the StudyBud application is containerized.
We implemented a multi-stage Docker build to reduce the image size from ~

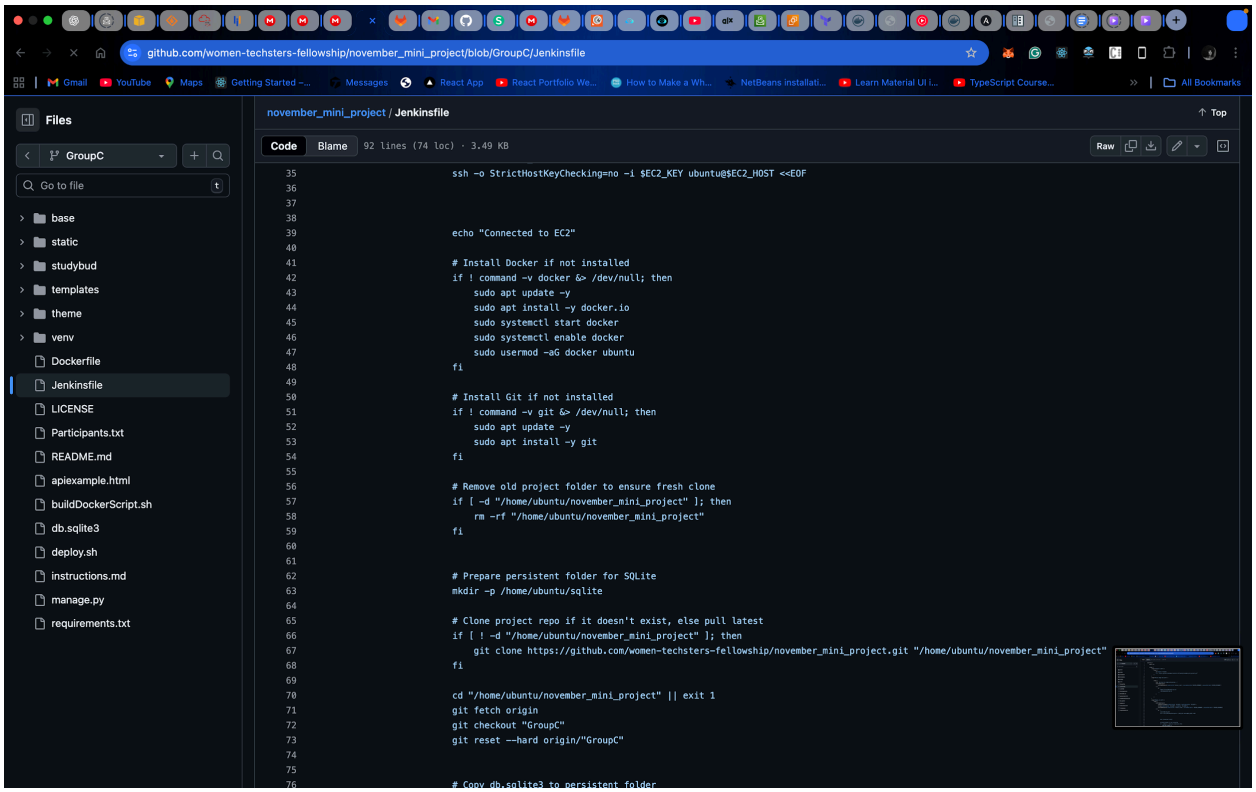


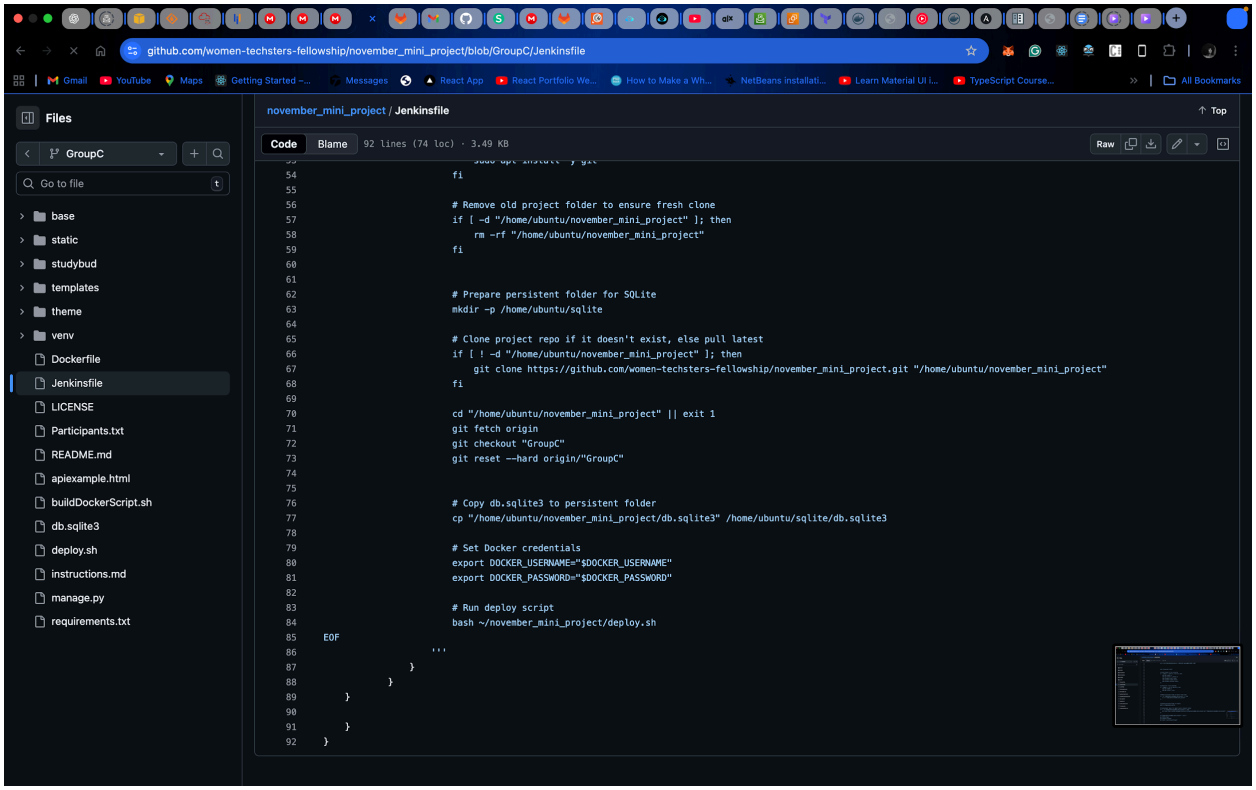
5. Jenkins Pipeline Setup

The Jenkinsfile defines the CI/CD pipeline stages:

- Source code checkout
- Docker image build (multi-stage)
- Docker Hub push
- EC2 deployment


github.com/women-techsters-fellowship/november_mini_project/blob/GroupC/Jenkinsfile
Files
GroupC
Go to file
base
static
studybud
templates
theme
venv
Dockerfile
Jenkinsfile
LICENSE
Participants.txt
README.md
apiexample.html
buildDockerScript.sh
db.sqlite3
deploy.sh
instructions.md
manage.py
requirements.txt
Code
Blame
92 lines (74 loc) · 3.49 KB
Raw
pipeline {
agent any
stages {
stage("Checkout code") {
steps {
git branch: "GroupC",
url: "https://github.com/women-techsters-fellowship/november_mini_project.git"
}
}
stage("Build image and push") {
steps {
echo 'Building the image and pushing....'
withCredentials([
usernamePassword(credentialsId:'docker_creds', usernameVariable:'DOCKER_USERNAME', passwordVariable:'DOCKER_PASSWORD')
]) {
sh'''
chmod 777 buildDockerScript.sh
./buildDockerScript.sh
'''
}
}
}
stage("Deploy to EC2") {
steps {
withCredentials([
sshUserPrivateKey(credentialsId: 'EC2_KEY', keyFileVariable: 'EC2_KEY'),
string(credentialsId: 'EC2_HOST', variable: 'EC2_HOST'),
usernamePassword(credentialsId: 'docker_creds', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')
]) {
sh '''
chmod 600 $EC2_KEY
ssh -o StrictHostKeyChecking=no -i $EC2_KEY ubuntu@$EC2_HOST <<EOF
echo "Connected to EC2"
# Install Docker if not installed
if ! command -v docker &> /dev/null; then
sudo apt update -y


github.com/women-techsters-fellowship/november_mini_project/blob/GroupC/Jenkinsfile
Files
GroupC
Go to file
base
static
studybud
templates
theme
venv
Dockerfile
Jenkinsfile
LICENSE
Participants.txt
README.md
apiexample.html
buildDockerScript.sh
db.sqlite3
deploy.sh
instructions.md
manage.py
requirements.txt
november_mini_project / Jenkinsfile
Top
Code
Blame
92 lines (74 loc) · 3.49 KB
Raw
ssh -o StrictHostKeyChecking=no -i $EC2_KEY ubuntu@$EC2_HOST <<EOF
echo "Connected to EC2"
# Install Docker if not installed
if ! command -v docker &> /dev/null; then
sudo apt update -y
sudo apt install -y docker.io
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker ubuntu
fi
# Install Git if not installed
if ! command -v git &> /dev/null; then
sudo apt update -y
sudo apt install -y git
fi
# Remove old project folder to ensure fresh clone
if [-d "/home/ubuntu/november_mini_project"]; then
rm -rf "/home/ubuntu/november_mini_project"
fi
Prepare persistent folder for SQLite
mkdir -p /home/ubuntu/sqlite
# Clone project repo if it doesn't exist, else pull latest
if [! -d "/home/ubuntu/november_mini_project"]; then
git clone https://github.com/women-techsters-fellowship/november_mini_project.git "/home/ubuntu/november_mini_project"
fi
cd "/home/ubuntu/november_mini_project" || exit 1
git fetch origin
git checkout "GroupC"
git reset --hard origin/"GroupC"
Copy db.sqlite3 to persistent folder

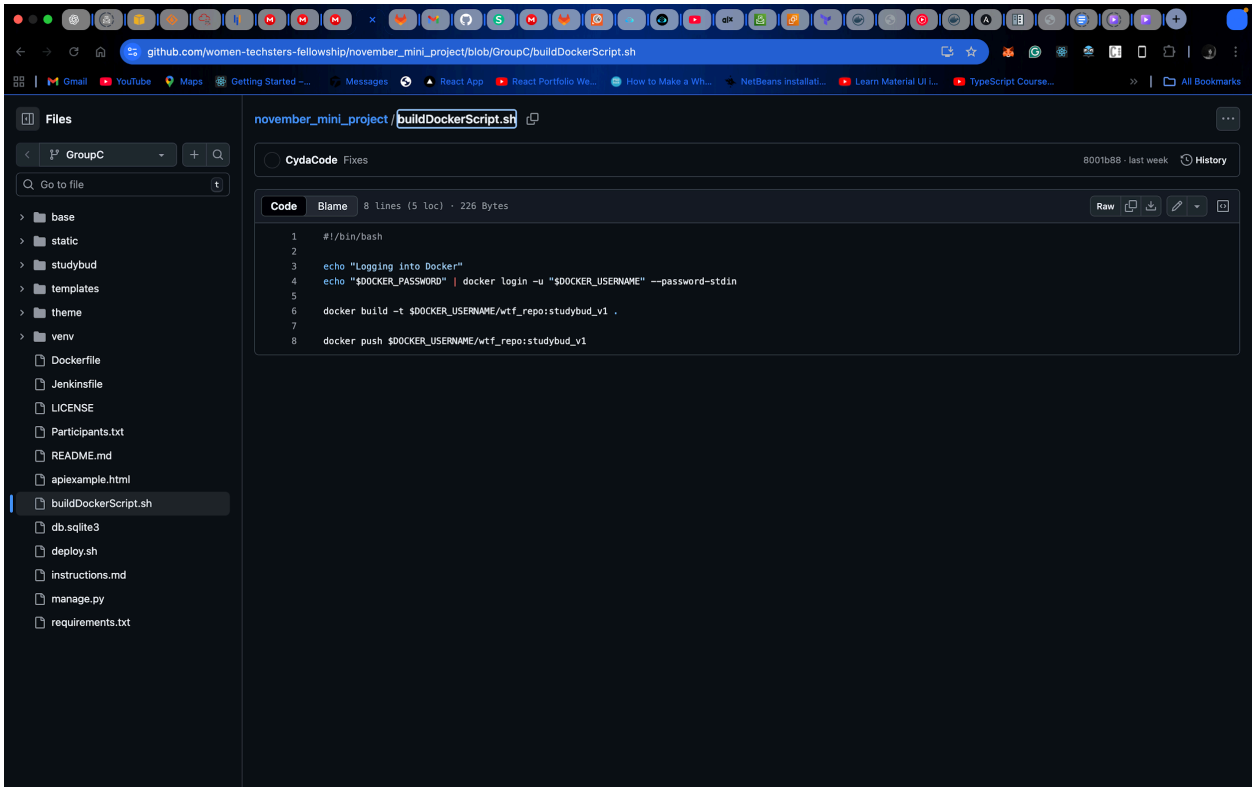


---

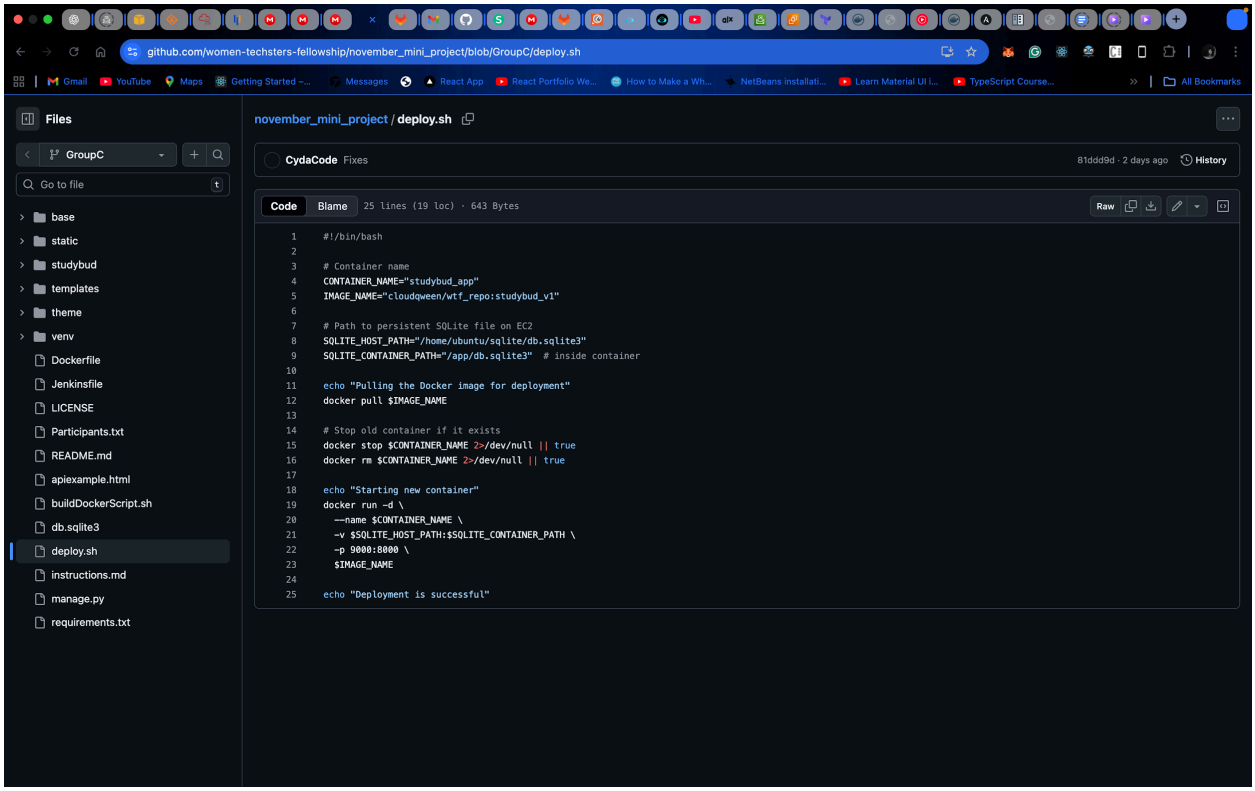
6. Build & Deployment Scripts

Custom scripts were used to automate Docker image building and EC2 deployment:

- buildDockerScript.sh

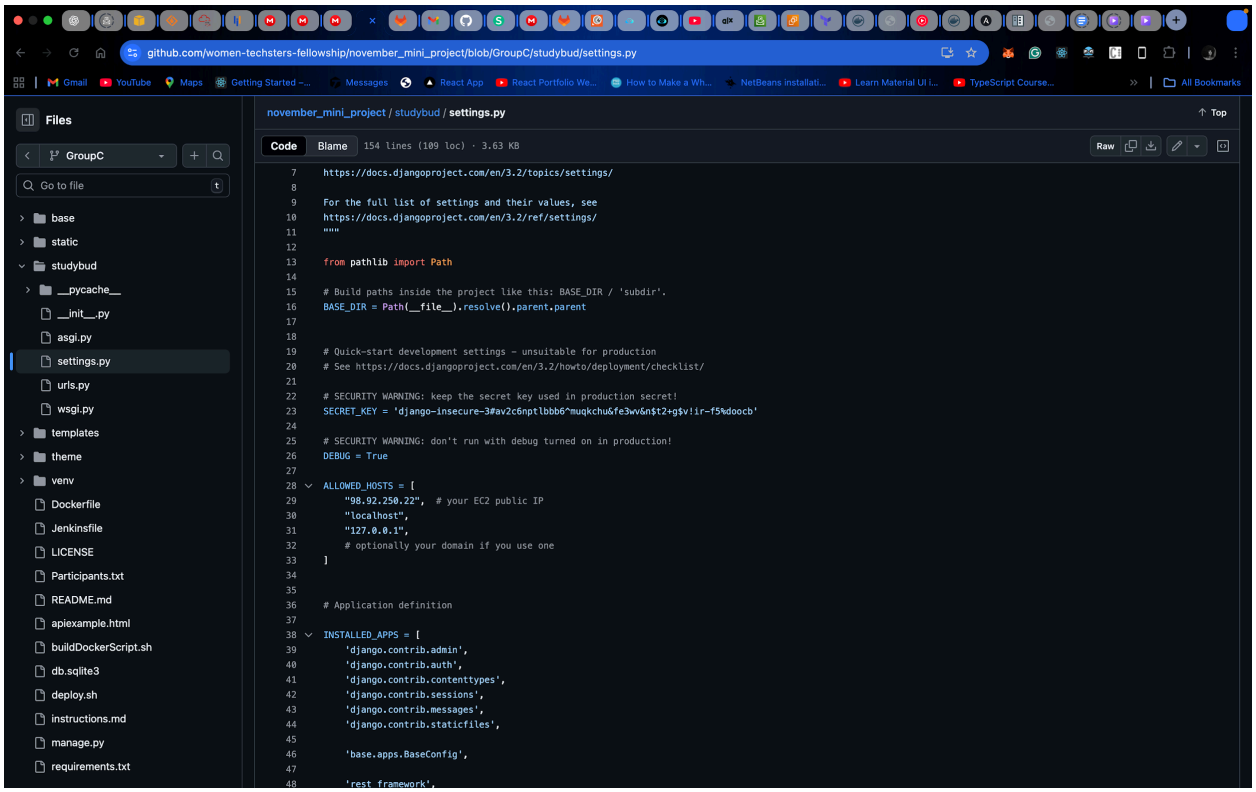


- deploy.sh



7. Django Configuration

To allow external access, the EC2 public IP was added to ALLOWED_HOSTS in settings.py.



---

8. Database Persistence (SQLite)

SQLite was configured with persistent storage on the EC2 host, so data is retained across container restarts and redeployments.
Impact:

- Users can sign up and log in successfully
- User accounts persist across deployments
- Users can create posts and TGroups
- Created content remains after container restarts

---

9. Jenkins Build Verification

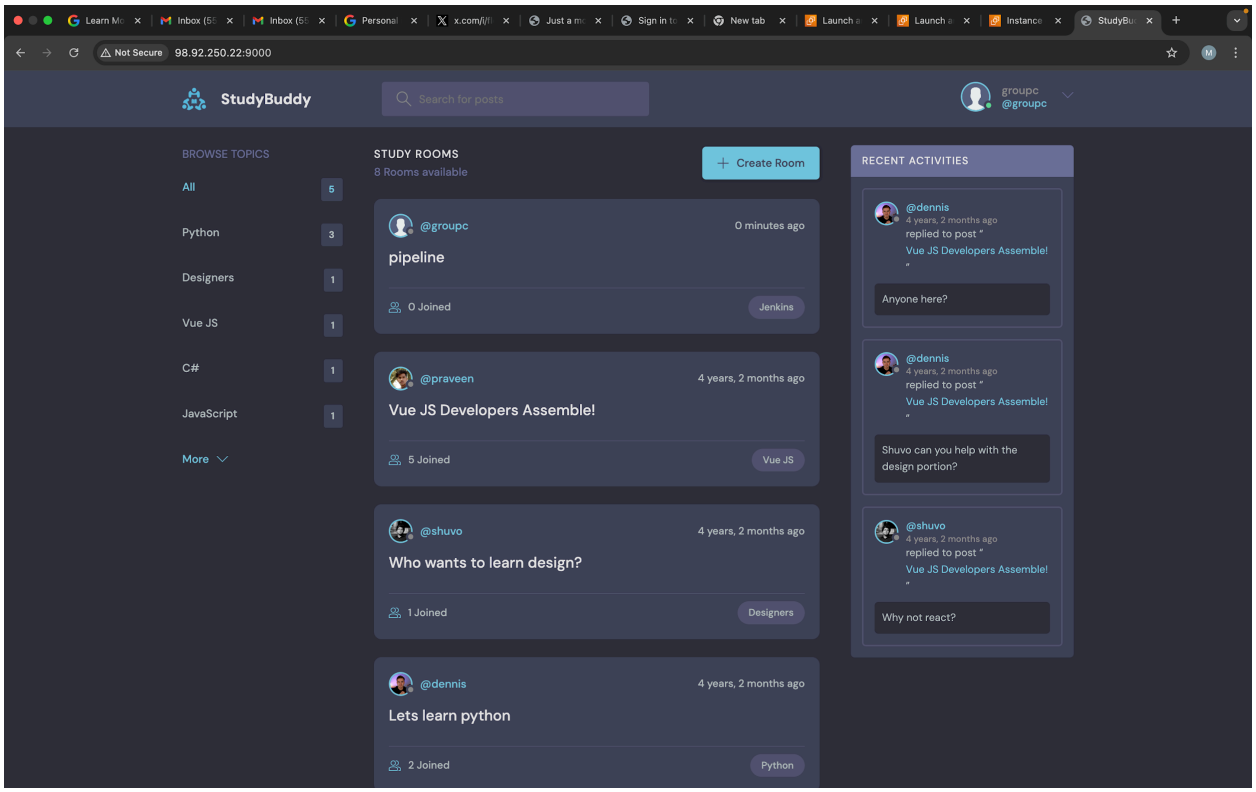
The Jenkins pipeline completed successfully with all stages passing.

---

10. Live Application Verification & Database Persistence (SQLite)

SQLite was configured with persistent storage on the EC2 host, so data is retained across container restarts and redeployments.
Impact:

- Users can sign up and log in successfully
- User accounts persist across deployments
- Users can create posts and Groups
- Created content remains after container restarts
- Application running on EC2
- Signup page
- Group creation



---

11. Errors Encountered & Resolutions

Issue	Resolution

Jenkins script parsing errors	Deployment logic moved to scripts
Old code running on EC2	Clean redeployments enforced
Port mismatch	Docker ports aligned with Django
ALLOWED_HOSTS error	EC2 IP added
Data loss on restart	SQLite persistence implemented

---

12. Conclusion

The StudyBud application was successfully deployed using a multi-stage Docker build in a Jenkins CI/CD pipeline, hosted on AWS EC2, and configured with persistent SQLite storage. Users can interact with the application, create accounts, posts, and Groups, and data persists across redeployments.