

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)

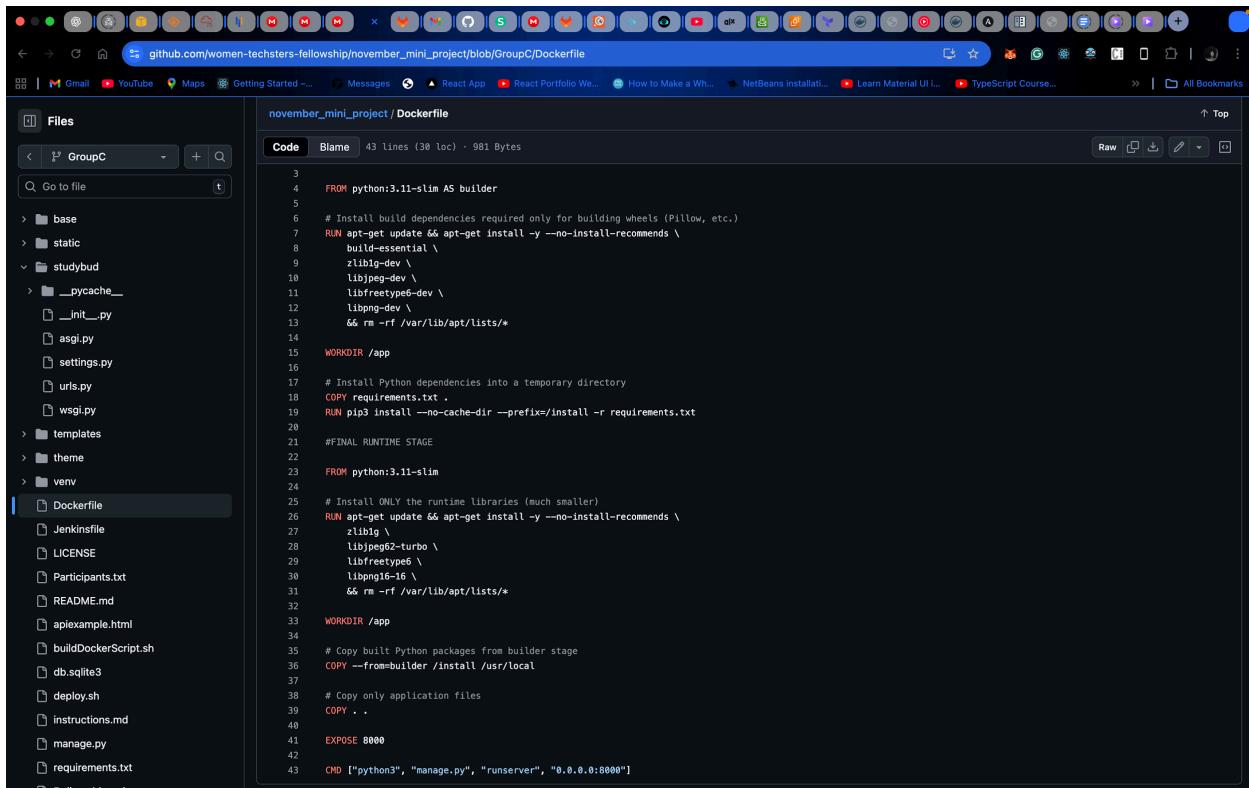
The following deployment components were implemented and verified:

- 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 ~



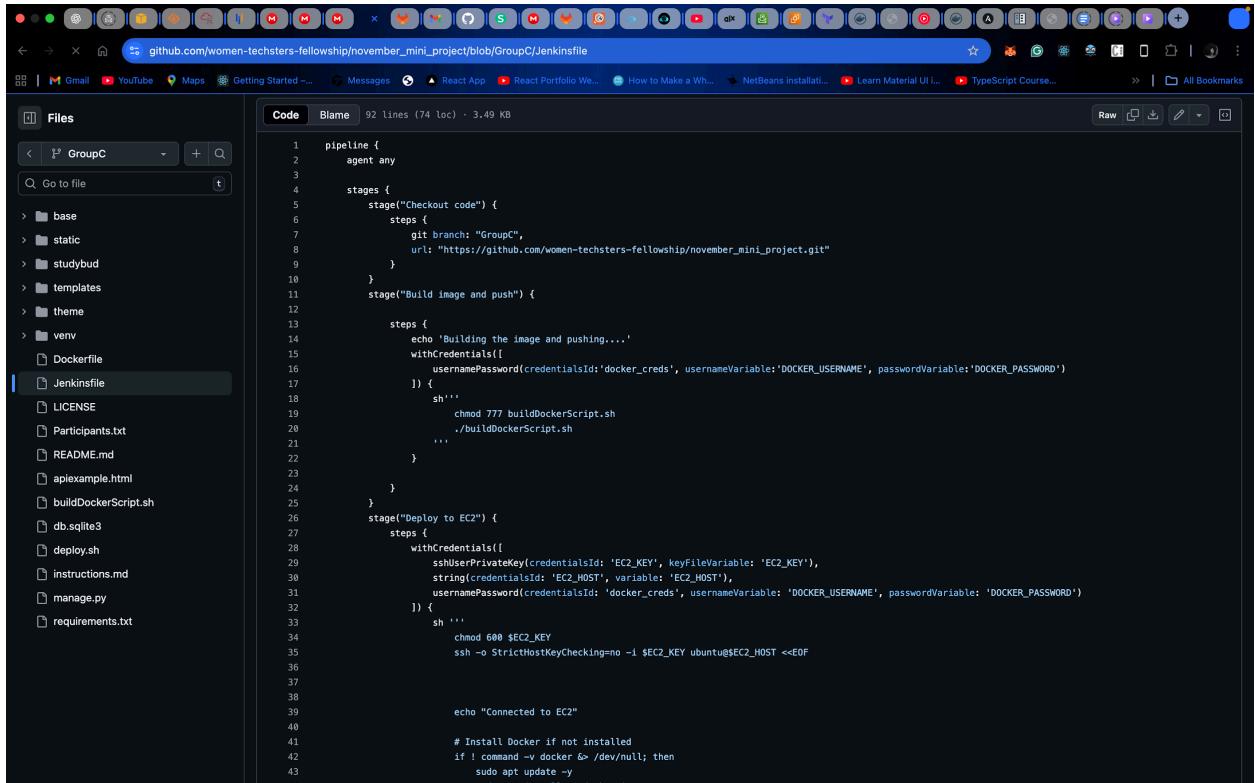
The screenshot shows a GitHub browser window with the URL https://github.com/women-techsters-fellowship/november_mini_project/blob/GroupC/Dockerfile. The left sidebar lists project files including `Dockerfile`, `Jenkinsfile`, `LICENSE`, `Participants.txt`, `README.md`, `apixample.html`, `buildDockerScript.sh`, `db.sqlite3`, `deploy.sh`, `instructions.md`, `manage.py`, and `requirements.txt`. The right pane displays the `Dockerfile` content:

```
FROM python:3.11-slim AS builder
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    zlib1g-dev \
    libjpeg-dev \
    libfreetype6-dev \
    libpng-dev \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /app
# Install Python dependencies into a temporary directory
COPY requirements.txt .
RUN pip3 install --no-cache-dir --prefix=/install -r requirements.txt
#FINAL RUNTIME STAGE
FROM python:3.11-slim
# Install ONLY the runtime libraries (much smaller)
RUN apt-get update && apt-get install -y --no-install-recommends \
    zlib1g \
    libjpeg62-turbo \
    libfreetype6 \
    libpng16 \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /app
# Copy built Python packages from builder stage
COPY --from=builder /install /usr/local
# Copy only application files
COPY . .
EXPOSE 8000
CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```

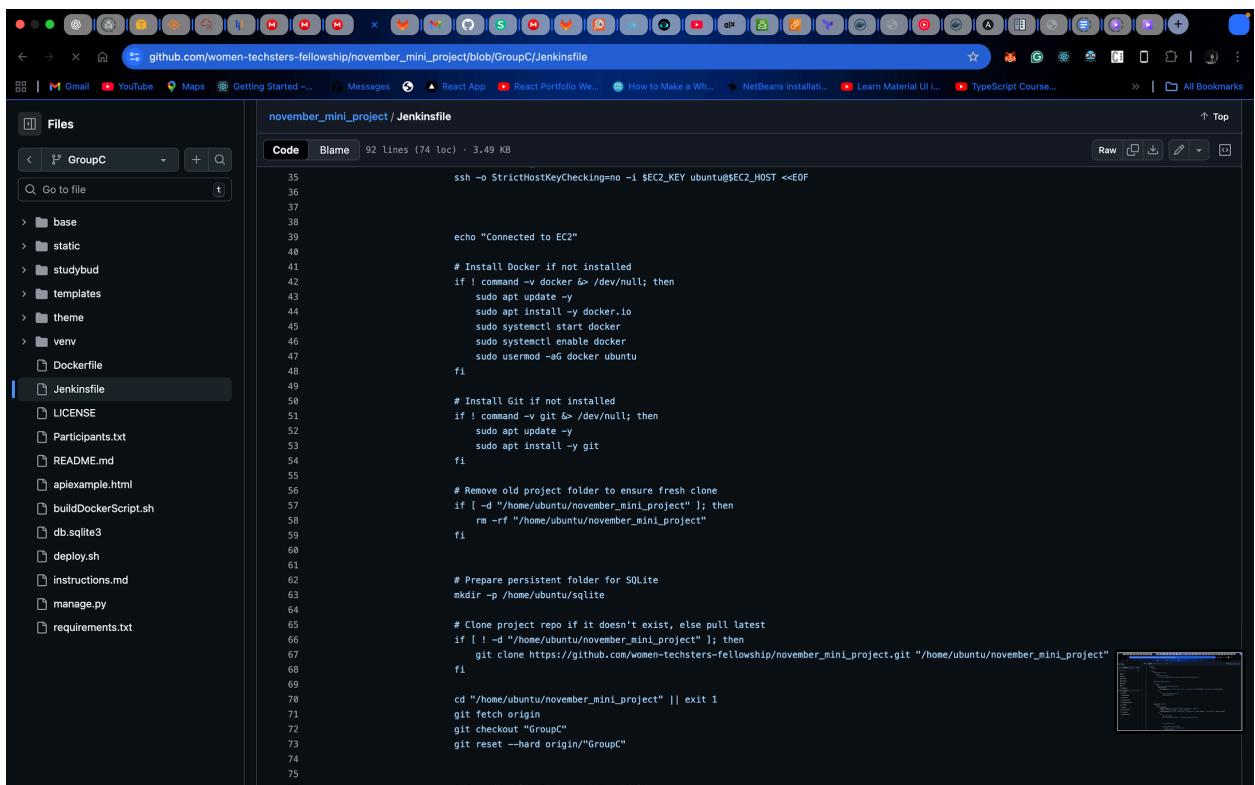
5. Jenkins Pipeline Setup

The Jenkinsfile defines the CI/CD pipeline stages:

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



```
1 pipeline {
2     agent any
3
4     stages {
5         stage("Checkout code") {
6             steps {
7                 git branch: "GroupC",
8                 url: "https://github.com/women-techsters-fellowship/november_mini_project.git"
9             }
10        }
11        stage("Build image and push") {
12
13            steps {
14                echo 'Building the image and pushing....'
15                withCredentials([
16                    usernamePassword(credentialsId:'docker_creds', usernameVariable:'DOCKER_USERNAME', passwordVariable:'DOCKER_PASSWORD')
17                ]) {
18                    sh'''
19                        chmod 777 buildDockerScript.sh
20                        ./buildDockerScript.sh
21                        ...
22                    '''
23                }
24            }
25        }
26        stage("Deploy to EC2") {
27            steps {
28                withCredentials([
29                    sshUserPrivateKey(credentialsId: 'EC2_KEY', keyFileVariable: 'EC2_KEY'),
30                    string(credentialsId: 'EC2_HOST', variable: 'EC2_HOST'),
31                    usernamePassword(credentialsId: 'docker_creds', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')
32                ]) {
33                    sh '''
34                        chmod 600 $EC2_KEY
35                        ssh -o StrictHostKeyChecking=no -i $EC2_KEY ubuntu@$EC2_HOST <>EOF
36
37
38
39                    echo "Connected to EC2"
40
41                    # Install Docker if not installed
42                    if ! command -v docker && /dev/null; then
43                        sudo apt update -y
44                        curl -fsSL https://get.docker.io | sh
45                        sudo systemctl start docker
46                        sudo systemctl enable docker
47                        sudo usermod -aG docker ubuntu
48                    fi
49
50                    # Install Git if not installed
51                    if ! command -v git && /dev/null; then
52                        sudo apt update -y
53                        sudo apt install -y git
54                    fi
55
56                    # Remove old project folder to ensure fresh clone
57                    if [ -d "/home/ubuntu/november_mini_project" ]; then
58                        rm -rf "/home/ubuntu/november_mini_project"
59                    fi
60
61                    # Prepare persistent folder for SQLite
62                    mkdir -p /home/ubuntu/sqlite
63
64
65                    # Clone project repo if it doesn't exist, else pull latest
66                    if [ ! -d "/home/ubuntu/november_mini_project" ]; then
67                        git clone https://github.com/women-techsters-fellowship/november_mini_project.git "/home/ubuntu/november_mini_project"
68                    fi
69
70                    cd "/home/ubuntu/november_mini_project" || exit 1
71                    git fetch origin
72                    git checkout "GroupC"
73                    git reset --hard origin/"GroupC"
74
75
76                    # Copy db.sqlite3 to persistent folder
77                }
78            }
79        }
80    }
81}
```



```
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
```

```
november_mini_project / Jenkinsfile
Code Blame 92 lines (74 loc) · 3.49 KB
54     fi
55
56     # Remove old project folder to ensure fresh clone
57     if [ ! -d "/home/ubuntu/november_mini_project" ]; then
58         rm -rf "/home/ubuntu/november_mini_project"
59     fi
60
61
62     # Prepare persistent folder for SQLite
63     mkdir -p /home/ubuntu/sqlite
64
65     # Clone project repo if it doesn't exist, else pull latest
66     if [ ! -d "/home/ubuntu/november_mini_project" ]; then
67         git clone https://github.com/women-techsters-fellowship/november_mini_project.git "/home/ubuntu/november_mini_project"
68     fi
69
70     cd "/home/ubuntu/november_mini_project" || exit 1
71     git fetch origin
72     git checkout "GroupC"
73     git reset --hard origin/"GroupC"
74
75
76     # Copy db.sqlite3 to persistent folder
77     cp "/home/ubuntu/november_mini_project/db.sqlite3" /home/ubuntu/sqlite/db.sqlite3
78
79     # Set Docker credentials
80     export DOCKER_USERNAME="$DOCKER_USERNAME"
81     export DOCKER_PASSWORD="$DOCKER_PASSWORD"
82
83     # Run deploy script
84     bash ~/november_mini_project/deploy.sh
85 EOF
86     ...
87 }
88 }
89 }
90 }
91 }
```

6. Build & Deployment Scripts

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

- buildDockerScript.sh

The screenshot shows a GitHub browser window with the URL https://github.com/women-techsters-fellowship/november_mini_project/blob/GroupC/buildDockerScript.sh. The left sidebar shows a project structure with files like Dockerfile, Jenkinsfile, LICENSE, Participants.txt, README.md, apixample.html, db.sqlite3, deploy.sh, instructions.md, manage.py, and requirements.txt. The right panel displays the content of the buildDockerScript.sh file:

```
#!/bin/bash
echo "Logging into Docker"
echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
docker build -t $DOCKER_USERNAME/wtf_repo:studybud_v1 .
docker push $DOCKER_USERNAME/wtf_repo:studybud_v1
```

- deploy.sh

The screenshot shows a GitHub browser window with the URL https://github.com/women-techsters-fellowship/november_mini_project/blob/GroupC/deploy.sh. The left sidebar shows the same project structure as the previous screenshot. The right panel displays the content of the deploy.sh file:

```
#!/bin/bash
# Container name
CONTAINER_NAME="studybud_app"
IMAGE_NAME="cloudqueen/wtf_repo:studybud_v1"

# Path to persistent SQLite file on EC2
SQLITE_HOST_PATH="/home/ubuntu/sqlite/db.sqlite3"
SQLITE_CONTAINER_PATH="/app/db.sqlite3" # inside container

echo "Pulling the Docker image for deployment"
docker pull $IMAGE_NAME

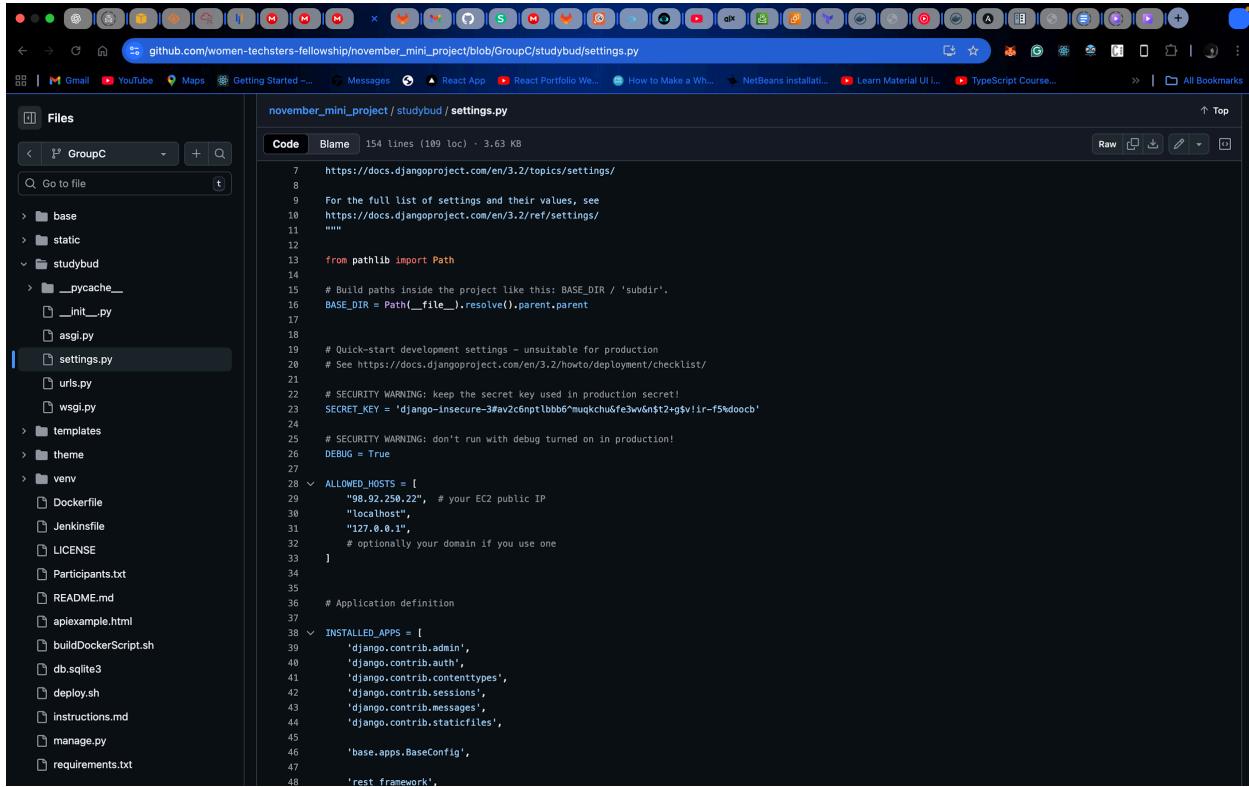
# Stop old container if it exists
docker stop $CONTAINER_NAME >/dev/null || true
docker rm $CONTAINER_NAME >/dev/null || true

echo "Starting new container"
docker run -d \
--name $CONTAINER_NAME \
-v $SQLITE_HOST_PATH:$SQLITE_CONTAINER_PATH \
-p 9000:8000 \
$IMAGE_NAME

echo "Deployment is successful"
```

7. Django Configuration

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



```
https://docs.djangoproject.com/en/3.2/topics/settings/
For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-3fav2c6nptlbbb6^mugkchuf3wv6n$tz+gv!ir-f5%doocb'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

# ALLOWED_HOSTS = [
#     "98.92.250.22", # your EC2 public IP
#     "localhost",
#     "127.0.0.1",
#     # optionally your domain if you use one
# ]
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'base.apps.BaseConfig',
    'rest_framework',
]
```

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

The screenshot shows the StudyBuddy application interface. On the left, there's a sidebar titled "BROWSE TOPICS" with categories like All, Python, Designers, Vue JS, C#, and JavaScript, each with a count of 5, 3, 1, 1, 1, and 1 respectively. In the center, there's a "STUDY ROOMS" section with 8 rooms available. One room, "pipeline" by @groupc, has 0 joined users and was created 0 minutes ago. Another room, "Vue JS Developers Assemble!" by @praveen, has 5 joined users and was created 4 years, 2 months ago. A third room, "Who wants to learn design?" by @shuvo, has 1 joined user and was created 4 years, 2 months ago. A fourth room, "Lets learn python" by @dennis, has 2 joined users and was created 4 years, 2 months ago. On the right, there's a "RECENT ACTIVITIES" sidebar showing interactions from users @dennis and @shuvo. The interface has a dark theme with light-colored cards for each study room.

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.