End-to-End DevOps Project Documentation

Sri Vignesh K V

Introduction

This document provides a structured, end-to-end walkthrough of the DevOps project. It is organized to mirror a real delivery document: prerequisites, environment setup, containerization with Docker, CI/CD with Jenkins, image registry integration, deployment on AWS EC2, and observability with Prometheus and Grafana.

Prerequisites

- Git & GitHub account with repository access
- Docker Desktop / Docker Engine and Docker Hub account
- JDK 11+ (if Jenkins master is local) and Jenkins server access with required plugins (Pipeline, Docker, Git)
- AWS account with EC2 permissions; key pair for SSH
- Security Group allowing HTTP (80) and required ports for Prometheus/Grafana
- Prometheus and Grafana binaries/containers
- Basic Linux CLI utilities (curl, systemctl or docker compose)

Project Components

Dockerfile & Docker Build

Dockerfile :

```
FROM node:20
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 80
CMD ["npm", "start"]
```

This creates a Node.js app container:

- **FROM node:20** → Base image (Node.js v20).
- **WORKDIR /app** → Sets working directory inside container.
- **COPY . .** → Copies all file in this project files into container.
- **RUN npm install** → Installs dependencies.
- **EXPOSE 80** → Opens port 80 for HTTP traffic.
- **CMD ["npm", "start"]** → Default command to run app.

build.sh :

```
#!/bin/bash
docker build -t devops-build:latest .
```

This script builds the Docker image with the tag `devops-build:latest`.

docker-compose.yml :

```
version: '3'
services:
  app:
    build: .
    ports:
      - "80:80"
```

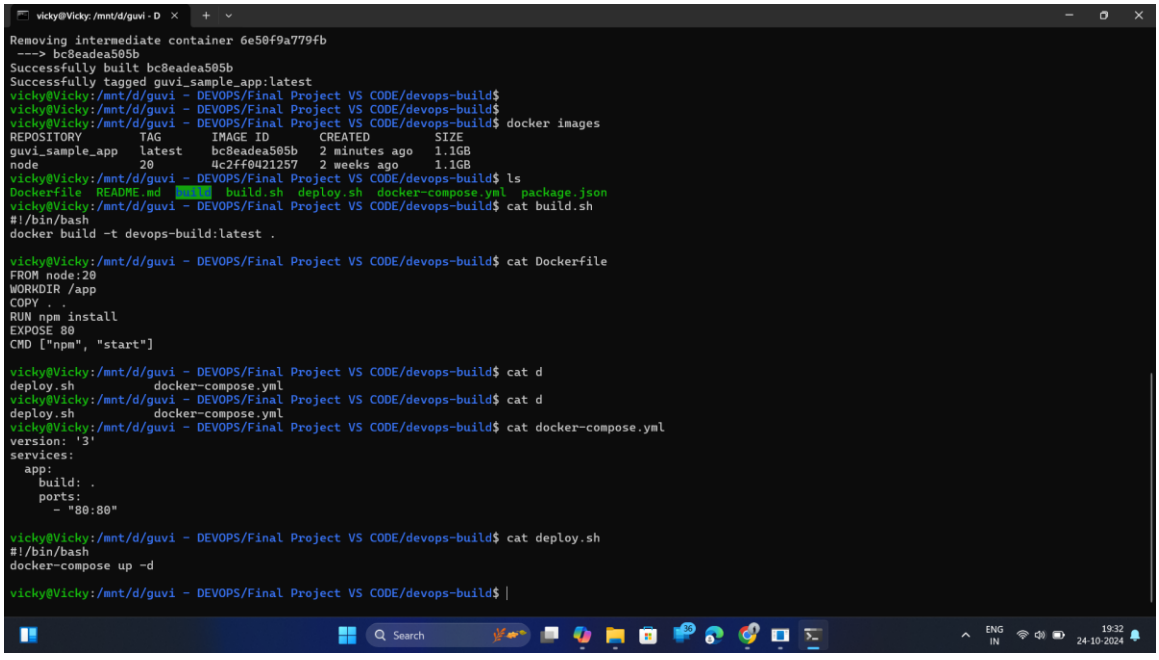
This uses Docker Compose to:

- Build image from Dockerfile in current directory (.).
- Run service named app.
- Map container port 80 → host port 80.

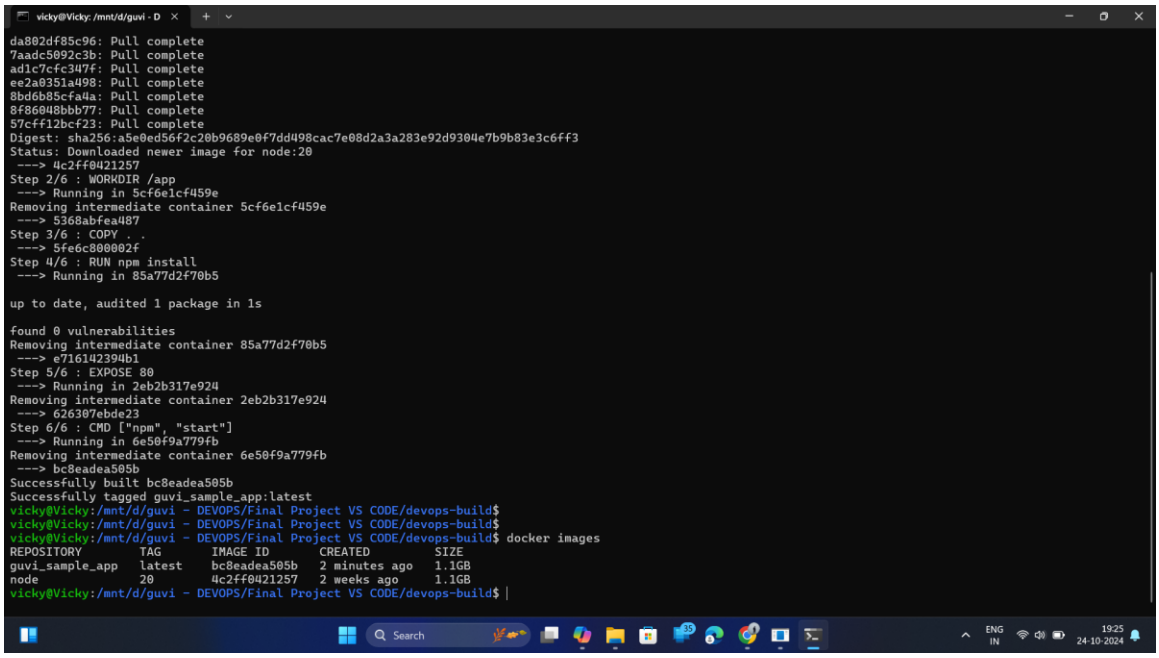
deploy.sh :

```
#!/bin/bash
docker-compose up -d
```

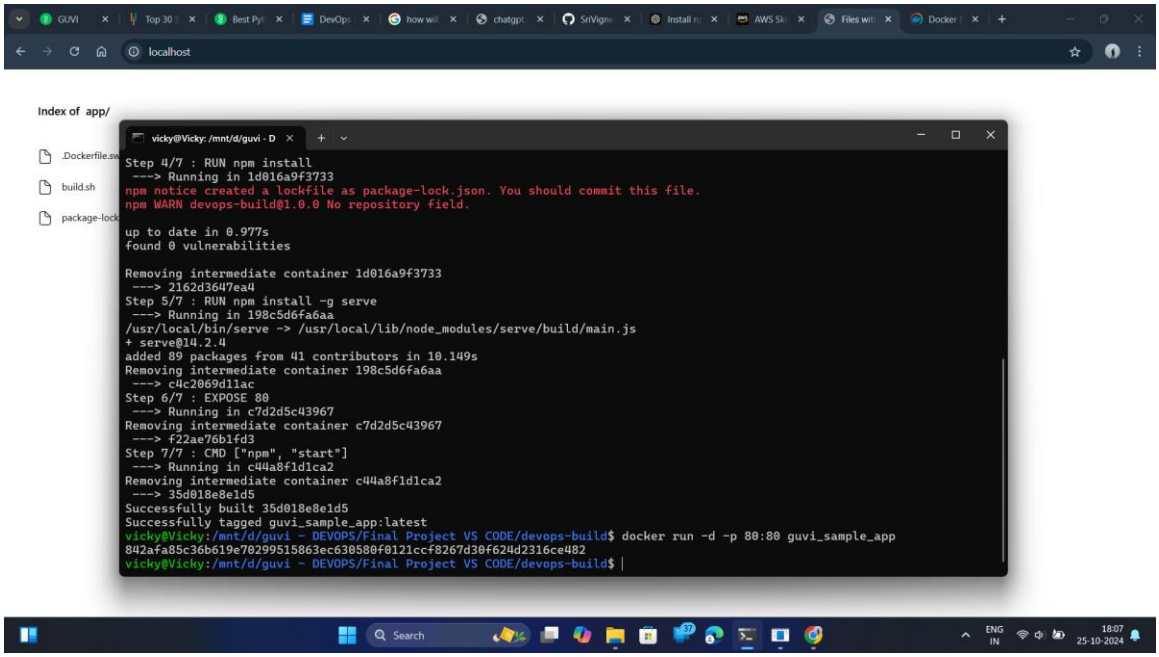
This script builds the Docker image with the tag `devops-build:latest`.



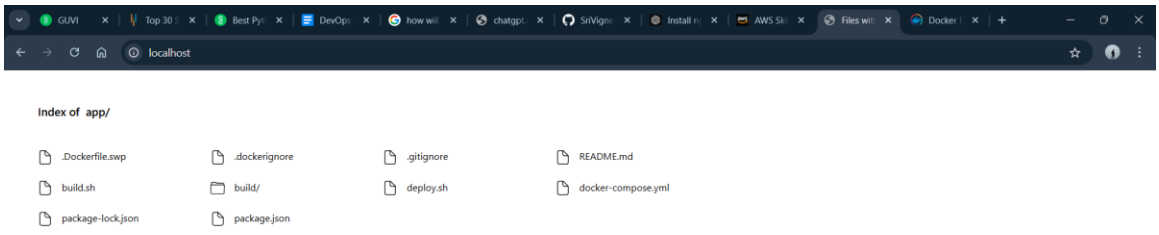
Shows the Dockerfile and project directory layout used for the build.



Lists local Docker images after build to verify the IMAGE ID and TAG.

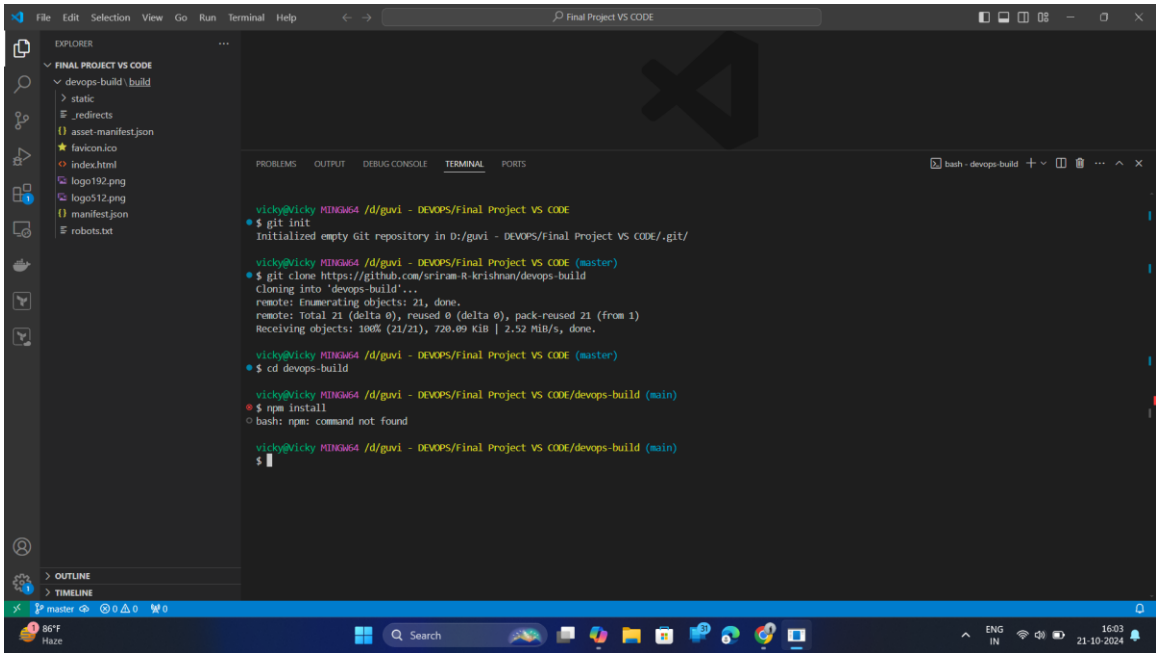


Confirms a successful Docker build, including build steps and final image output.

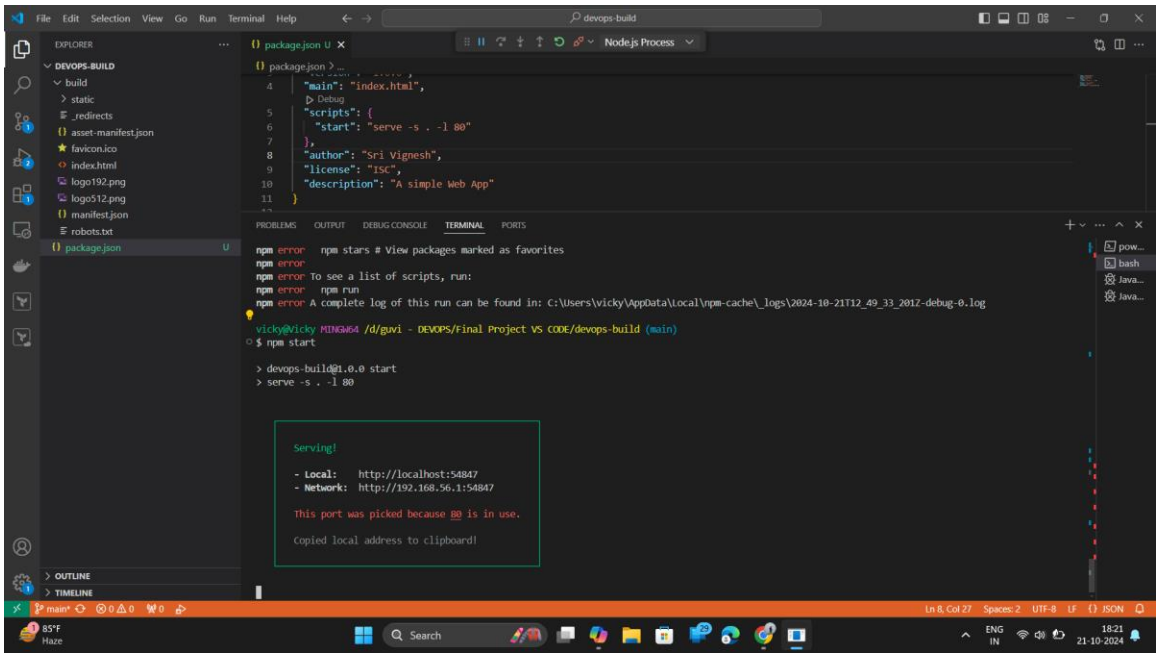


Demonstrates the container running locally; verify exposed port and health.

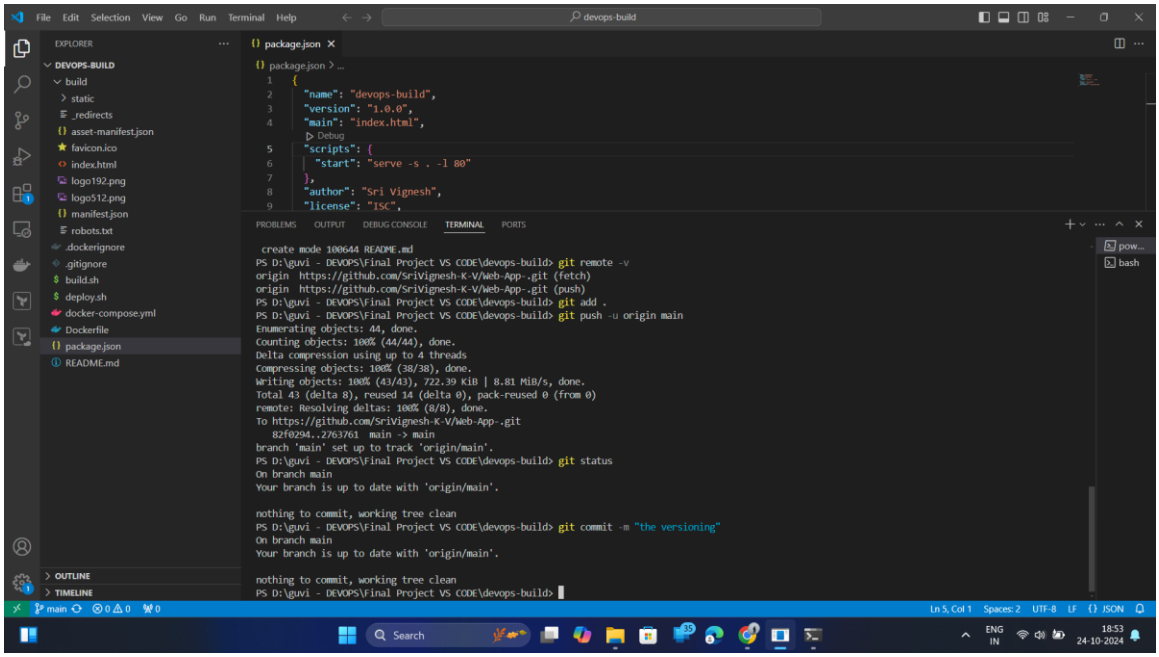
Source Control & Git



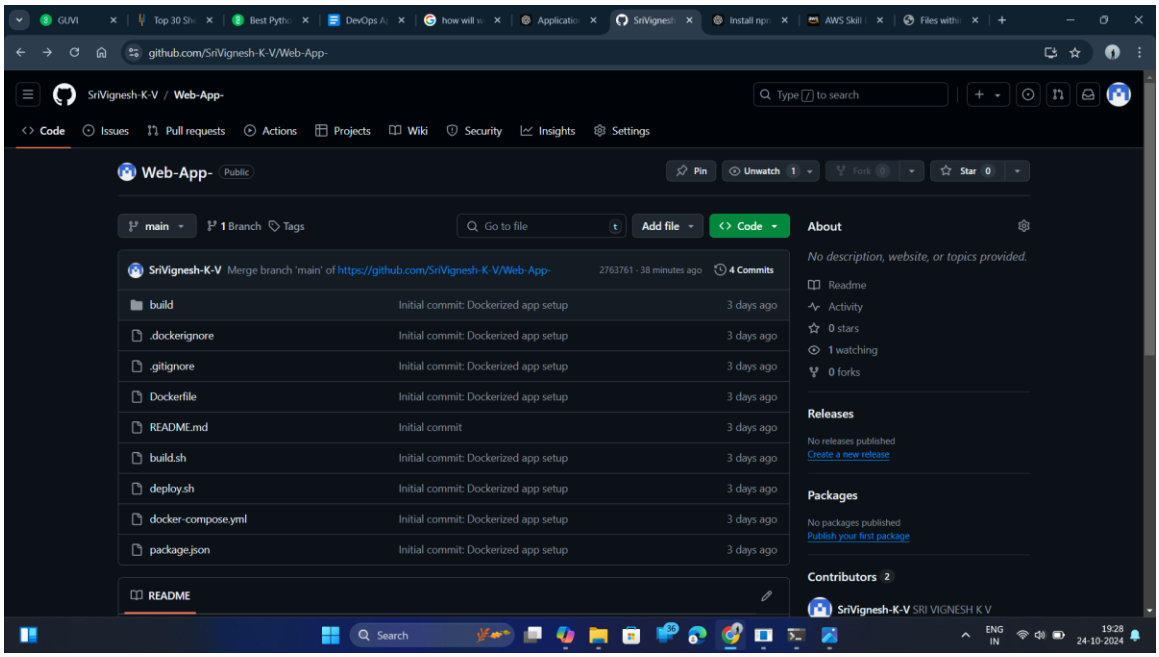
Repository cloned locally to begin development and containerization.



Application starts successfully on the development machine.



Git push completed; source changes are available remotely.



Repository state visible in GitHub after push.

Jenkins CI/CD Pipeline

```
pipeline {
    agent any

    environment {
        GIT_REPO = 'https://SriVignesh-K-
V:%40V%21gnesh098@github.com/SriVignesh-K-V/App.git'
        DOCKER_IMAGE = 'srivigneshkv/guvi_sample_app:latest'
        DOCKER_HUB_CREDENTIALS = 'docker-hub-credentials' // ID of the
Docker Hub credentials in Jenkins
    }

    stages {
        stage('Clone Repository') {
            steps {
                git branch: 'main', url: "${env.GIT_REPO}"
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    dockerImage = docker.build("${env.DOCKER_IMAGE}")
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    withDockerRegistry([credentialsId:
"${env.DOCKER_HUB_CREDENTIALS}", url: 'https://index.docker.io/v1/']) {
                        dockerImage.push()
                    }
                }
            }
        }
    }

    post {
        always {
            cleanWs()
        }
    }
}
```

Explanation :

- **Agent Declaration**

```
agent any
```

The pipeline can run on any available Jenkins agent/node.

- **Environment Variables**

```
environment {
    GIT_REPO = 'https://...@github.com/SriVignesh-K-V/App.git'
    DOCKER_IMAGE = 'srivigneshkv/guvi_sample_app:latest'
    DOCKER_HUB_CREDENTIALS = 'docker-hub-credentials'
}
```

- **GIT_REPO** → GitHub repository with your application code.
- **DOCKER_IMAGE** → Image name/tag that will be built and pushed.
- **DOCKER_HUB_CREDENTIALS** → Credential ID configured in Jenkins for Docker Hub login.

- **Stage 1: Clone Repository**

```
stage('Clone Repository') {
    steps {
        git branch: 'main', url: "${env.GIT_REPO}"
    }
}
```

Clones your app repo from GitHub into the Jenkins workspace.

- **Stage 2: Build Docker Image**

```
stage('Build Docker Image') {
    steps {
        script {
            dockerImage = docker.build("${env.DOCKER_IMAGE}")
        }
    }
}
```

- **Builds a Docker image** using the `Dockerfile` in the repo.
- Tags it with the value from `DOCKER_IMAGE`.

- **Stage 3: Push Docker Image**

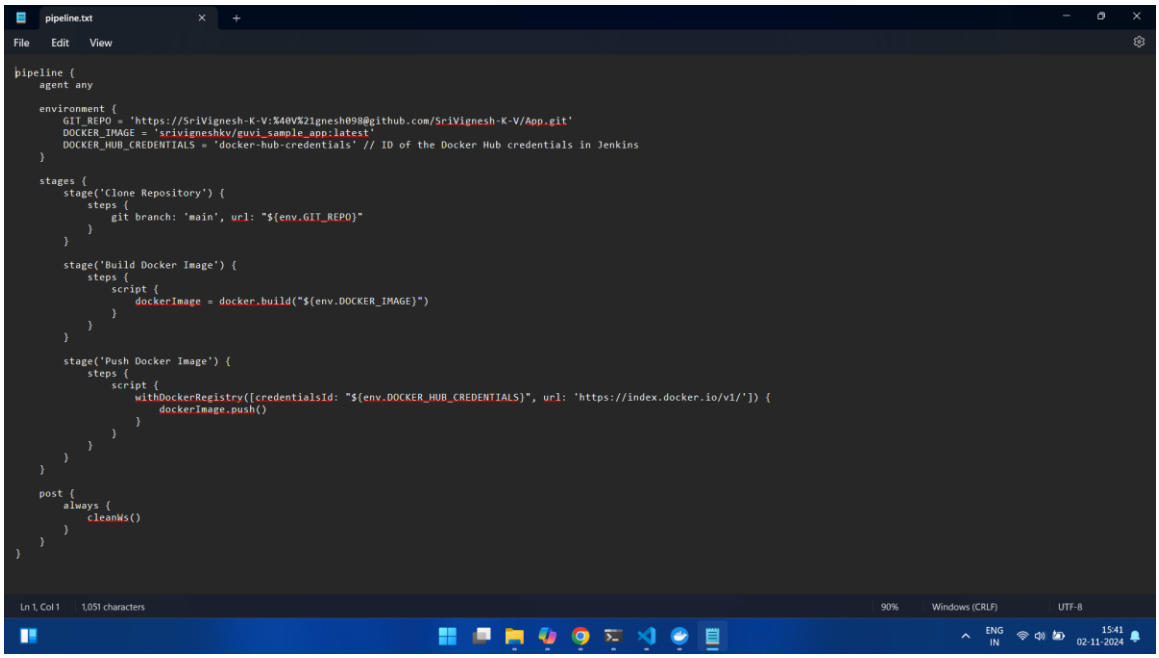
```
stage('Push Docker Image') {
    steps {
        script {
            withDockerRegistry([credentialsId:
"${env.DOCKER_HUB_CREDENTIALS}", url: 'https://index.docker.io/v1/']) {
                dockerImage.push()
            }
        }
    }
}
```

- Logs in to Docker Hub using Jenkins credentials.
- **Pushes** the newly built image to your Docker Hub repo.

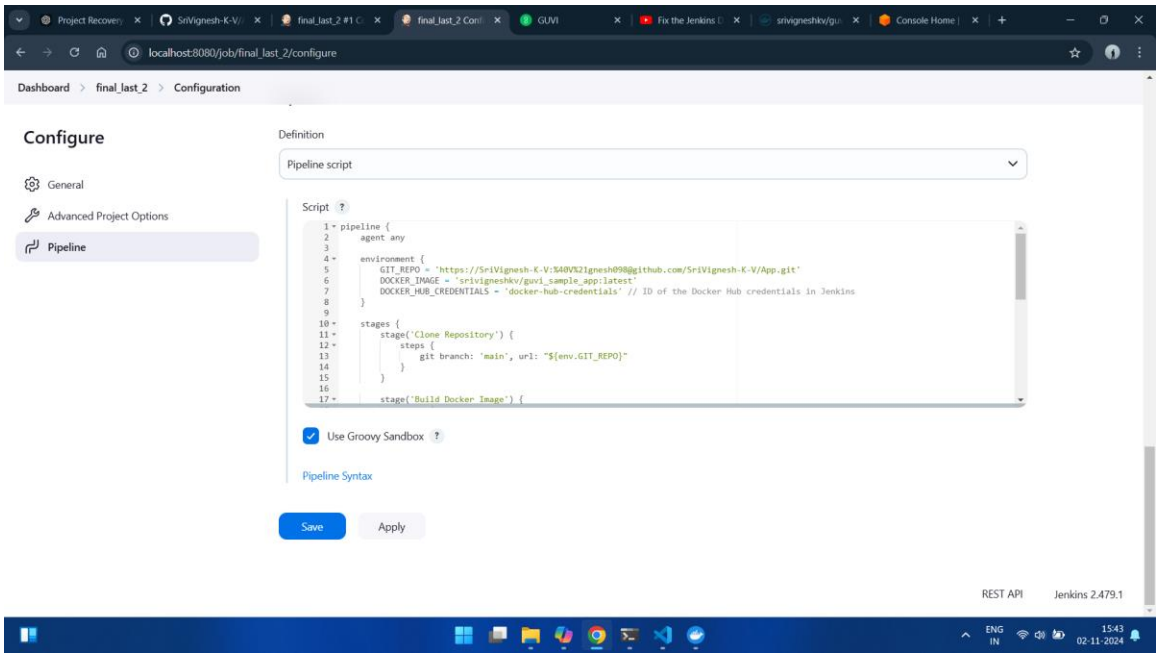
- **Post Section**

```
post {
    always {
        cleanWs()
    }
}
```

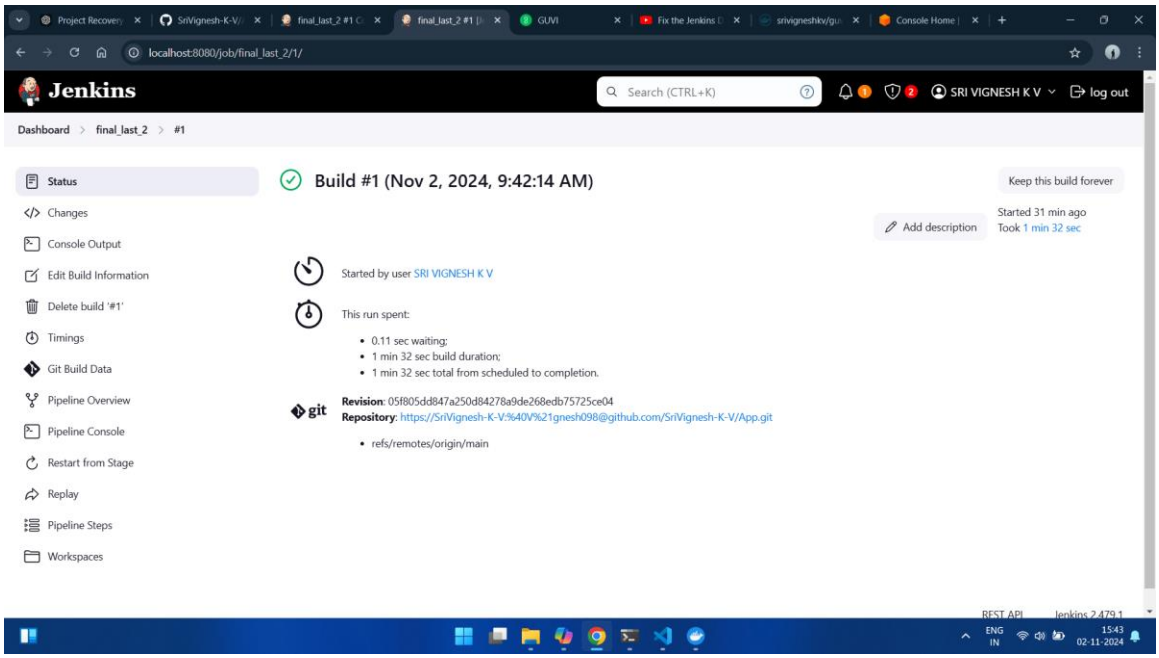
- After the pipeline completes (**success/failure**), it cleans the Jenkins workspace to free up space.



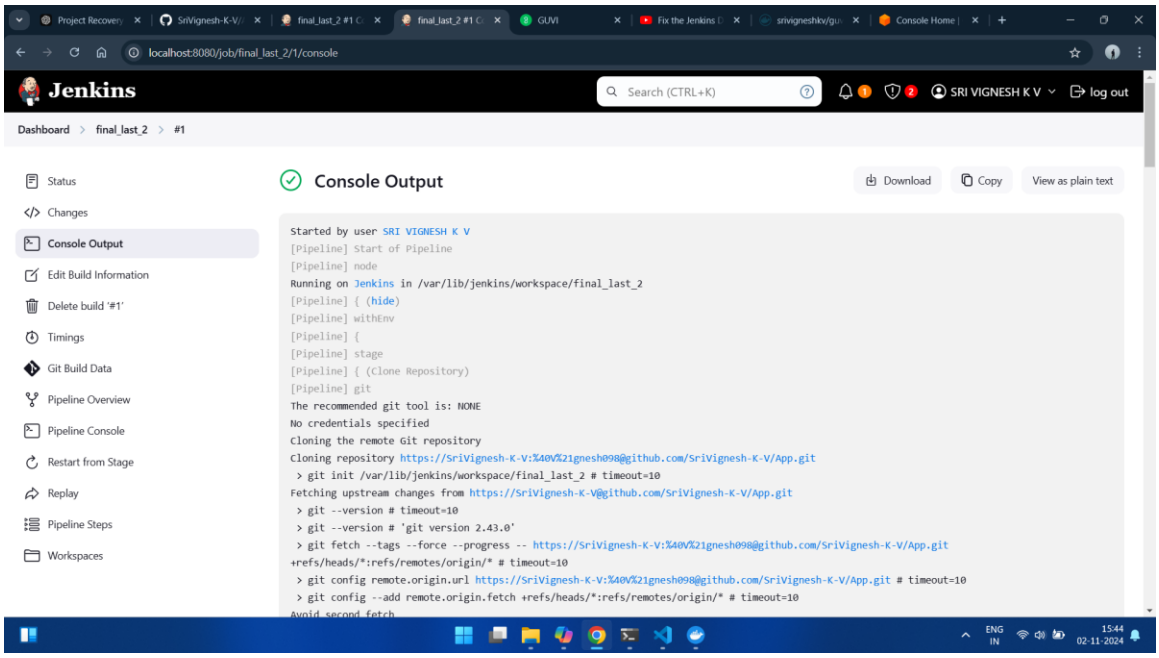
Groovy pipeline script for quick visual reference.



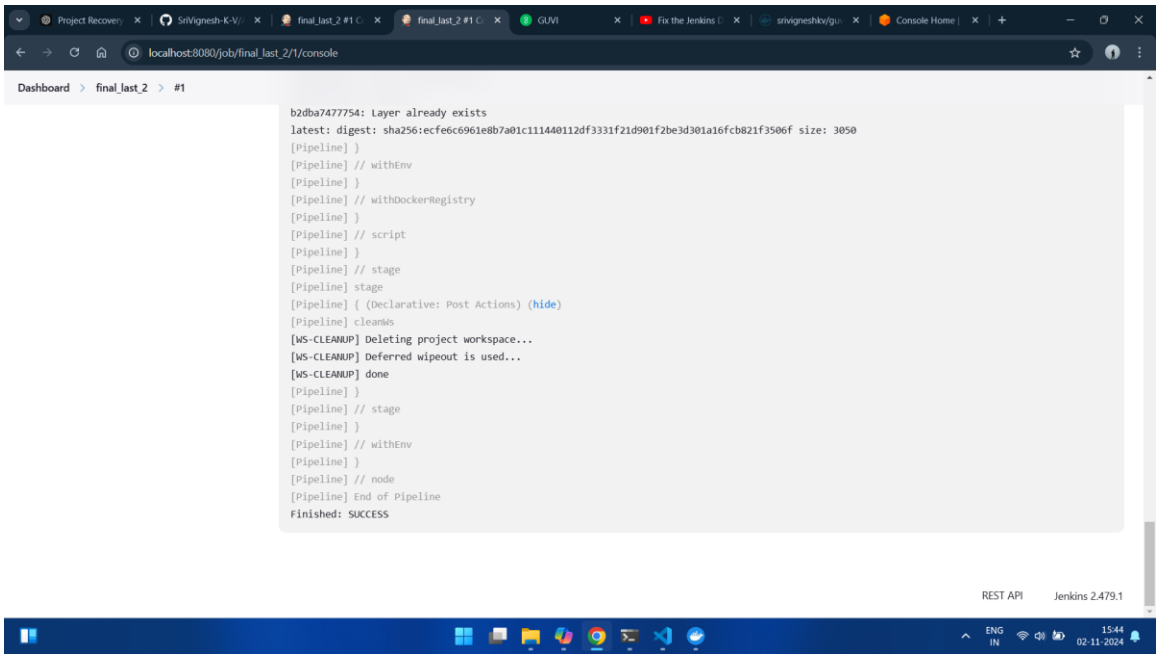
Jenkins UI overview where the pipeline job is configured.



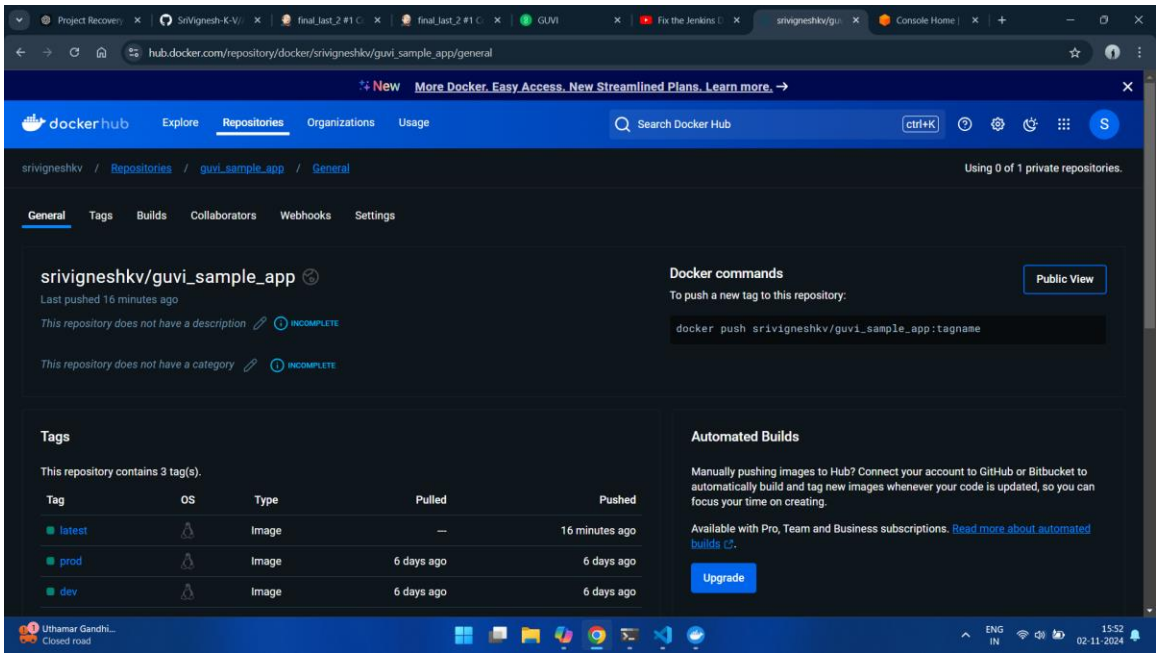
A sample build run executing stages.



Console output (part 1) highlighting checkout/build steps.



Console output (part 2) highlighting push/deploy steps.

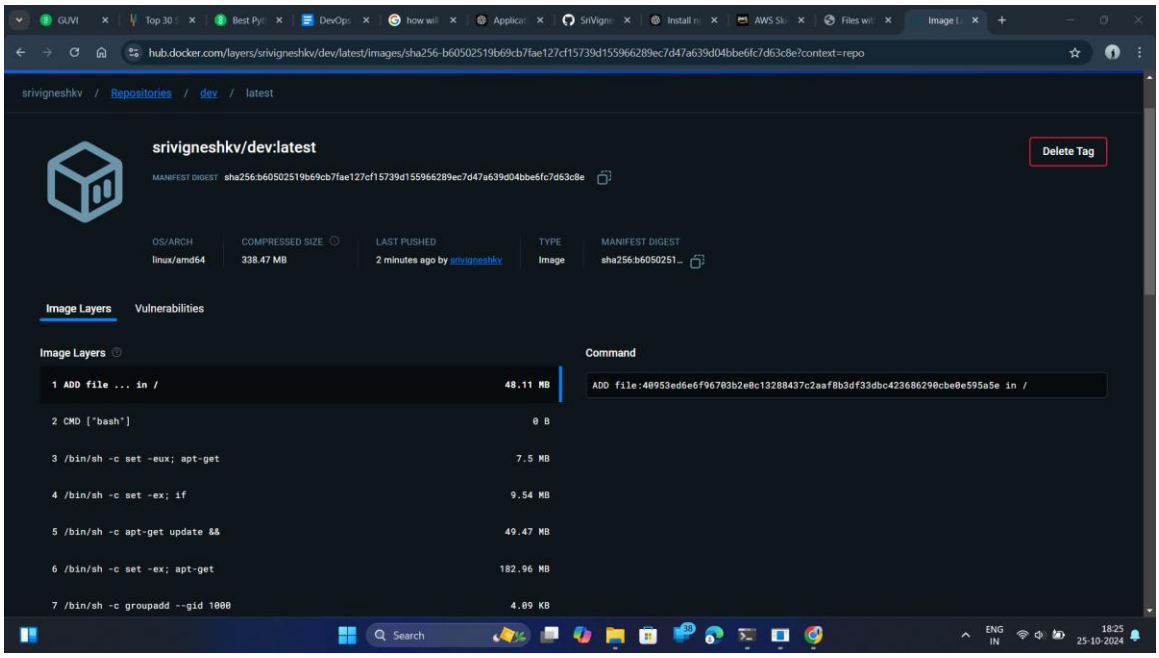


Confirms Docker image successfully pushed by Jenkins with the latest tag.

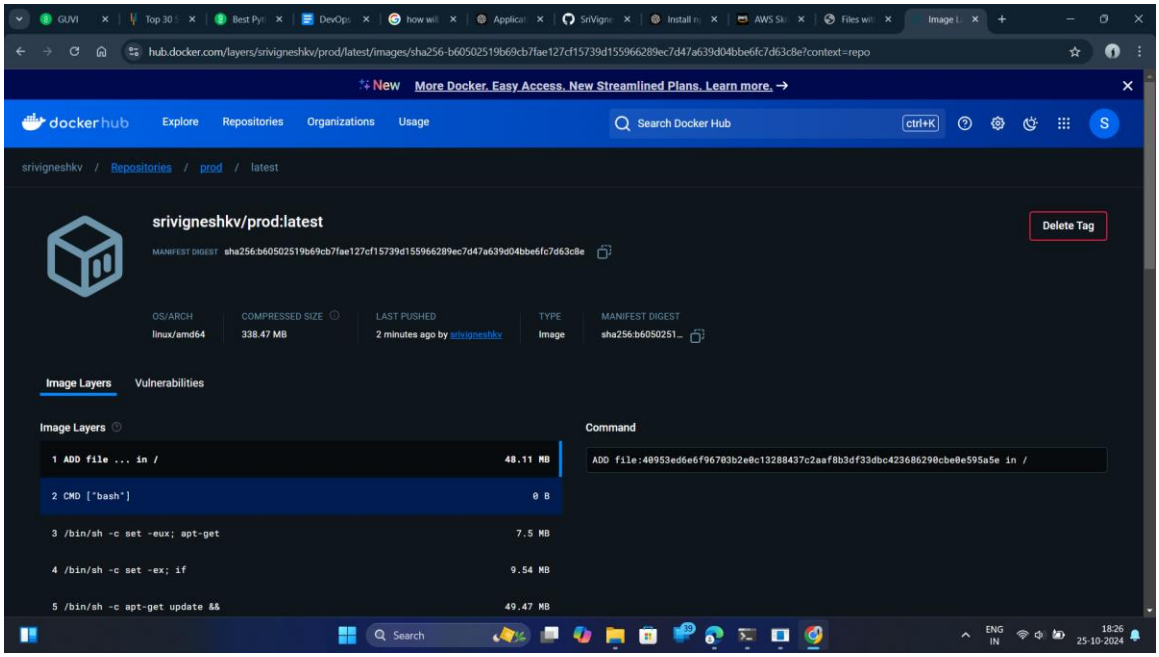
Docker Hub Integration

```
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker tag guvi_sample_app:latest srivigneshkv/dev:latest
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker tag guvi_sample_app:latest srivigneshkv/prod:latest
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker login -u srivigneshkv
Password:
Login Succeeded
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker push srivigneshkv/dev:latest
The push refers to repository [docker.io/srivigneshkv/dev]
3eb1e38d8f9d: Pushed
fa5fcbd59ad0: Pushed
db80cc17b276: Pushed
df8fba5658a9: Pushed
0d5f5a015e5d: Mounted from library/node
3c777d951de2: Mounted from library/node
f8a91dd5fc84: Mounted from library/node
cb81227abde5: Mounted from library/node
e01a454893a9: Mounted from library/node
c45660adde37: Mounted from library/node
fe0fb3ab4a0f: Mounted from library/node
f1186e5061f2: Mounted from library/node
b2dba7477754: Mounted from library/node
latest: digest: sha256:b60502519b69cb7fae127cf15739d155966289ec7d47a639d04bbe6fc7d63c8e size: 3050
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker push srivigneshkv/prod:latest
The push refers to repository [docker.io/srivigneshkv/prod]
3eb1e38d8f9d: Mounted from srivigneshkv/dev
fa5fcbd59ad0: Mounted from srivigneshkv/dev
db80cc17b276: Mounted from srivigneshkv/dev
df8fba5658a9: Mounted from srivigneshkv/dev
0d5f5a015e5d: Mounted from srivigneshkv/dev
3c777d951de2: Mounted from srivigneshkv/dev
f8a91dd5fc84: Mounted from srivigneshkv/dev
cb81227abde5: Mounted from srivigneshkv/dev
e01a454893a9: Mounted from srivigneshkv/dev
c45660adde37: Mounted from srivigneshkv/dev
fe0fb3ab4a0f: Mounted from srivigneshkv/dev
f1186e5061f2: Mounted from srivigneshkv/dev
b2dba7477754: Mounted from srivigneshkv/dev
latest: digest: sha256:b60502519b69cb7fae127cf15739d155966289ec7d47a639d04bbe6fc7d63c8e size: 3050
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$
```

Docker push command execution to the remote registry.

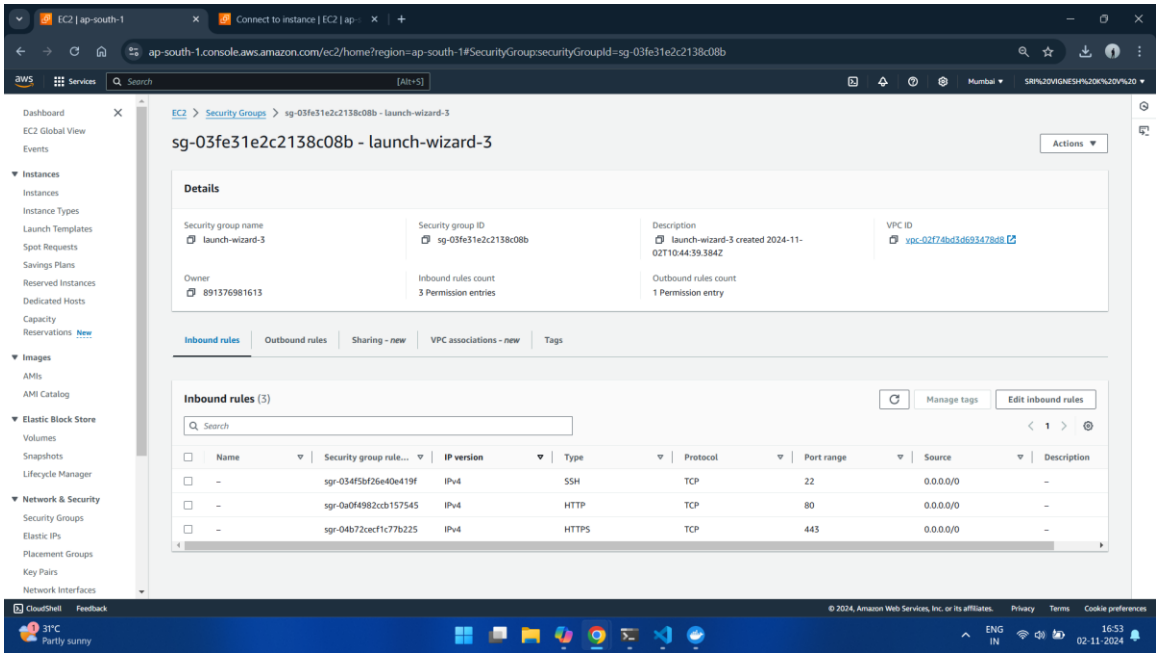


Docker Hub repository for the 'dev' tag.

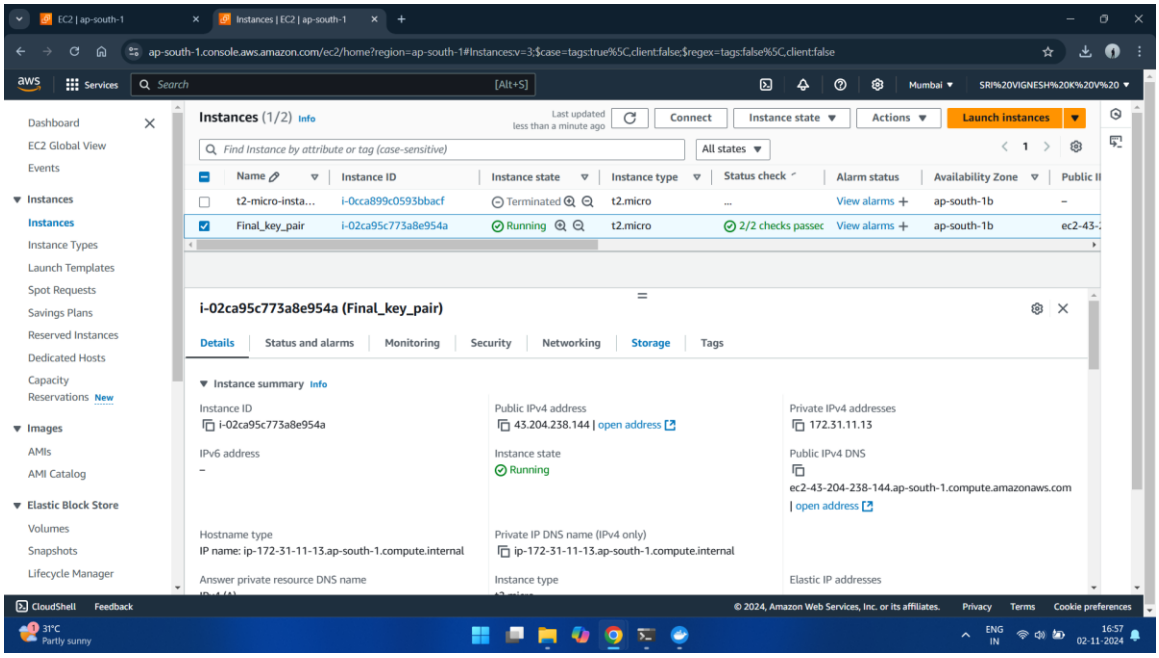


Docker Hub repository for the 'prod' tag, demonstrating environment tagging strategy.

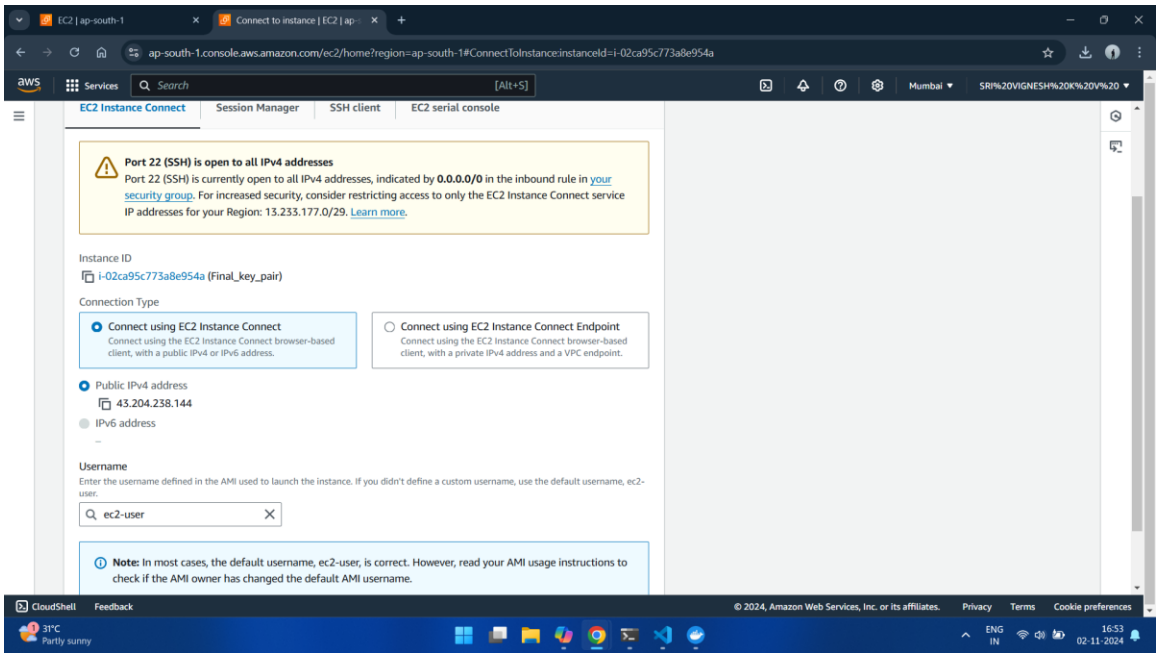
AWS EC2 Setup & Access



Inbound Security Group rules—ensure port 80 (HTTP) and required monitoring ports are open.



EC2 instance details (AMI, instance type, public IP).

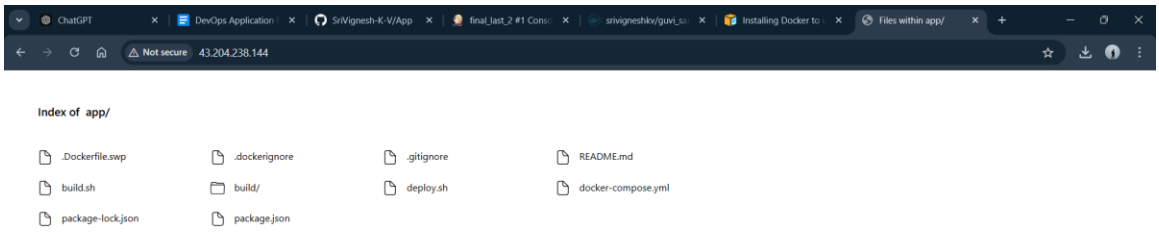


Connecting to the EC2 instance via the AWS Console/SSH.

Deploying & Running the Container

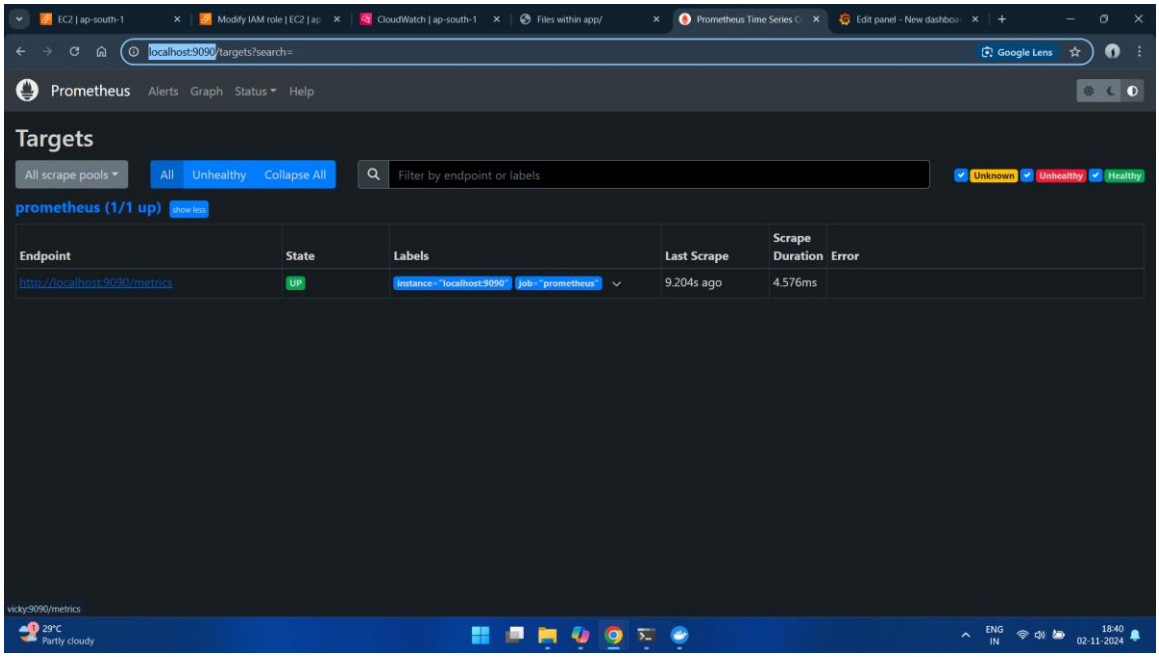
```
Docker version 25.0.5, build 5dc9bcc
[ec2-user@ip-172-31-11-13 ~]$ groups
docker adm wheel systemd-journal ec2-user
[ec2-user@ip-172-31-11-13 ~]$ docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to
create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using
 SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: srivigneshkv
Password:
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[ec2-user@ip-172-31-11-13 ~]$ docker pull srivigneshkv/guvi_sample_app:latest
latest: Pulling from srivigneshkv/guvi_sample_app
2ff1d7c41c74: Pull complete
b253aeafeaa7: Pull complete
3d2201bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df589451: Pull complete
6f51ee005dea: Pull complete
5f32ed3c3f27: Pull complete
0c8cc2f24a4d: Pull complete
0d27a8e86132: Pull complete
d3acd812b6ba: Pull complete
bf6723a76312: Pull complete
0d023b5afae5: Pull complete
84079e2b3c79: Pull complete
Digest: sha256:ecfe6c6961e8b7a01c111440112df3331f21d901f2be3d301a16fcb821f3506f
Status: Downloaded newer image for srivigneshkv/guvi_sample_app:latest
docker.io/srivigneshkv/guvi_sample_app:latest
[ec2-user@ip-172-31-11-13 ~]$ docker run -d -p 80:80 --name guvi_sample_app srivigneshkv/guvi_sample_app:latest
e678453275a539127810ea7ca74073e01ab3ee41fb4497991519cb25ab349e32
[ec2-user@ip-172-31-11-13 ~]$ docker ps -a
CONTAINER ID   IMAGE                              COMMAND                  CREATED          STATUS          PORTS                                     NAMES
e678453275a5   srivigneshkv/guvi_sample_app:latest   "docker-entrypoint.s…"   10 seconds ago   Up 9 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   guvi_sampl
e_app
[ec2-user@ip-172-31-11-13 ~]$
```

Container is up and running on the target host.

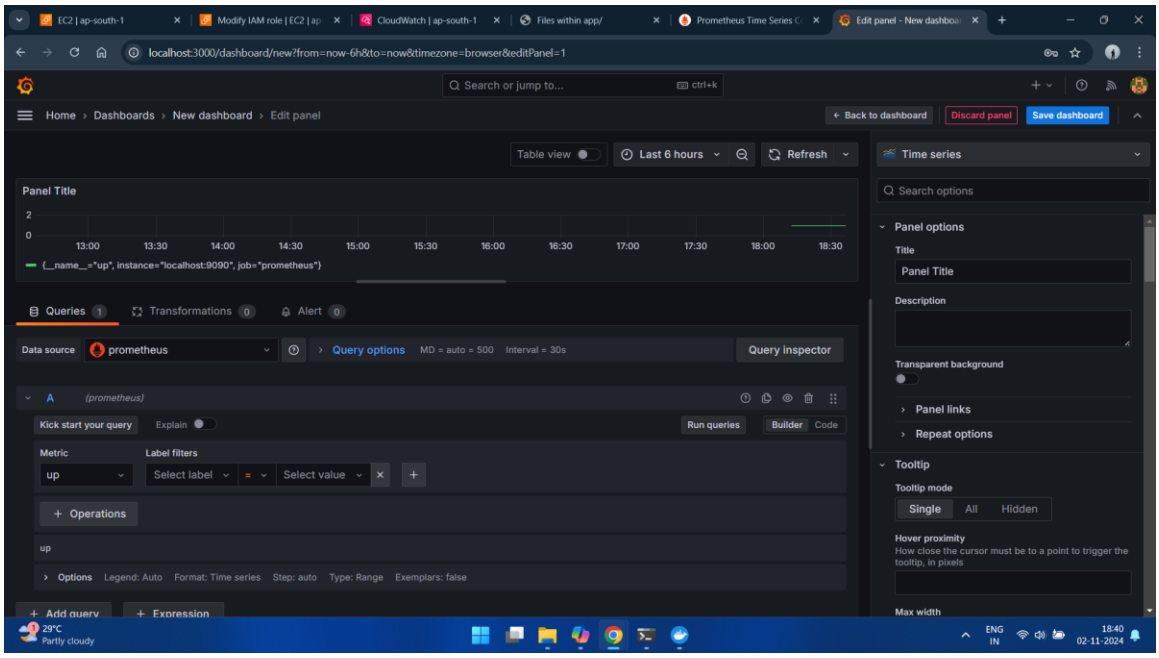


Web application is accessible on port 80 from a browser—validate service reachability.

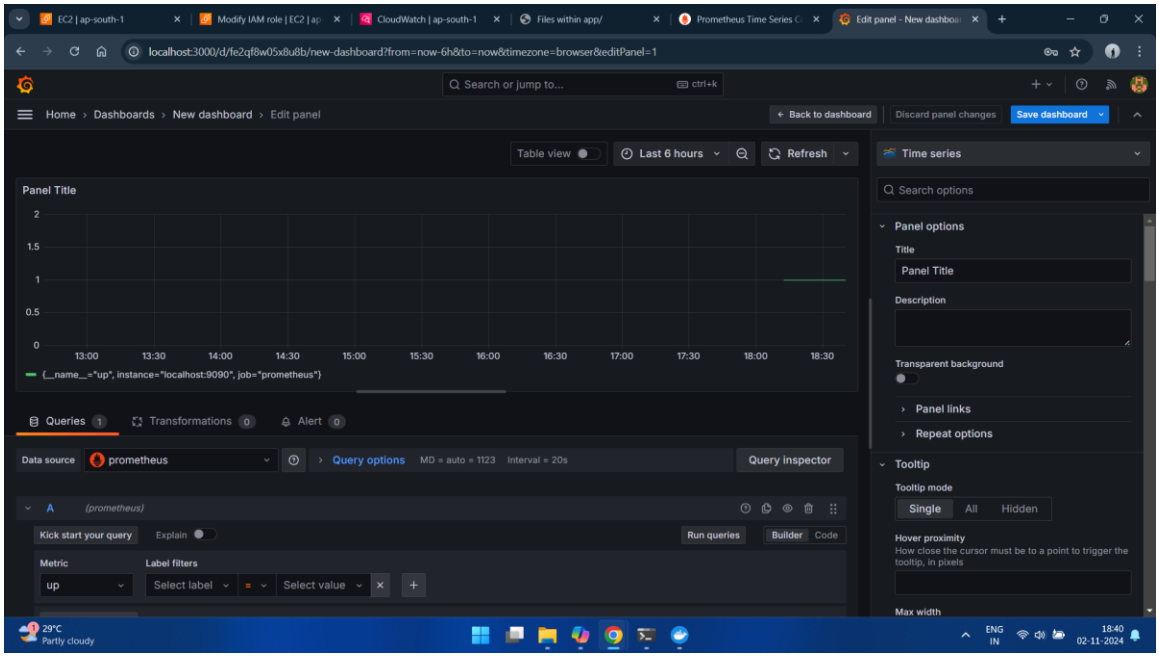
Observability: Prometheus & Grafana



Prometheus target/metrics page—validates metrics scraping.

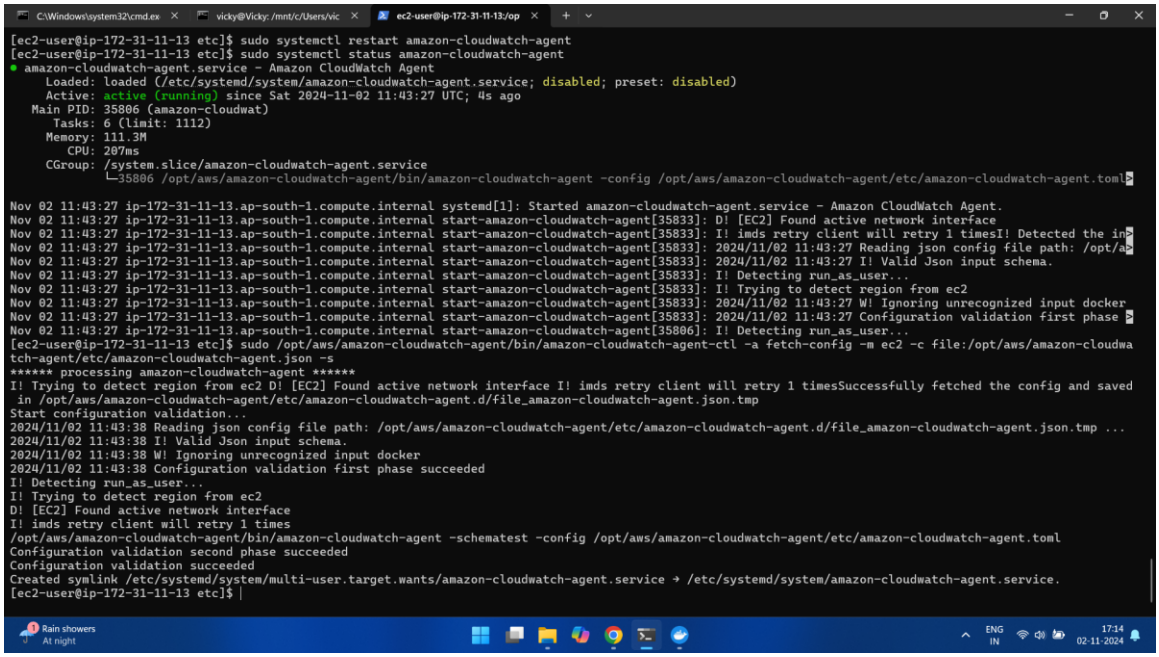


Grafana service starting/up—confirm UI availability.



Grafana dashboard visualizing application/infrastructure metrics.

Optional Attempt: CloudWatch



Attempted CloudWatch integration—document the goal, steps tried, and what failed for transparency.

Conclusion

I have implemented a practical CI/CD pipeline that builds a Dockerized application, pushes the image to Docker Hub, deploys/runs it on EC2, and instruments it with Prometheus/Grafana.