

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"
```

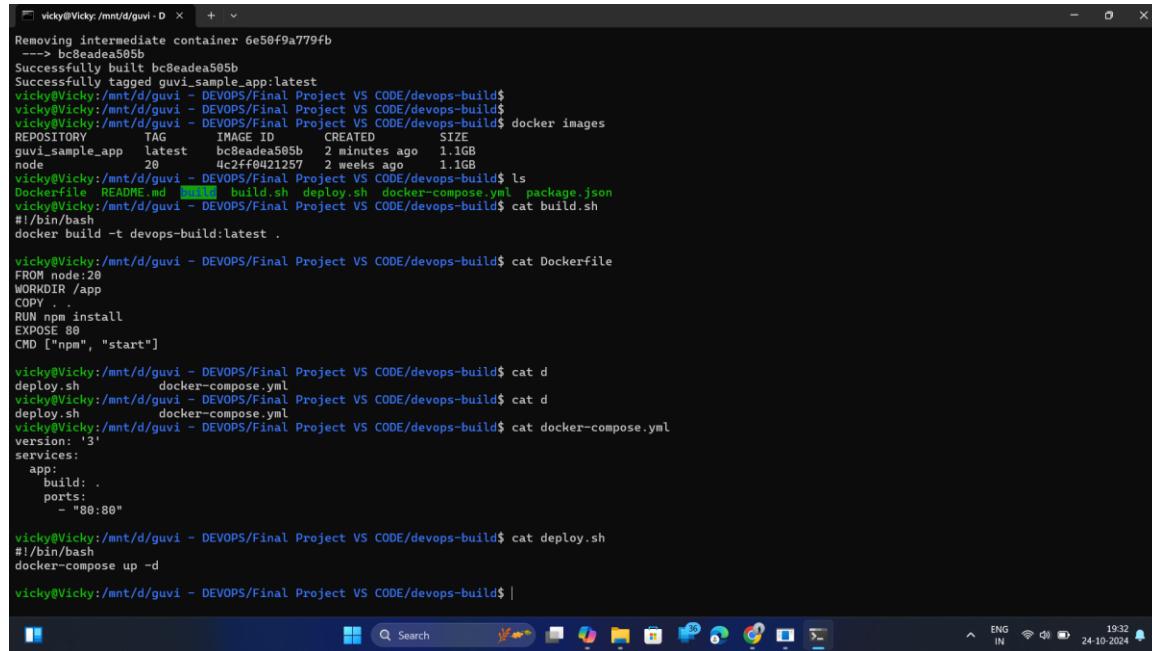
This uses Docker Compose to:

- 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.



The screenshot shows a Windows terminal window with a black background and white text. It displays the command-line steps taken to build a Docker image. The commands shown include removing intermediate containers, building the Docker image, tagging it as 'guvi_sample_app:latest', listing Docker images, and finally building the Dockerfile. The terminal also shows the contents of the Dockerfile, which specifies a base image of 'node:20', a working directory of '/app', copying the current directory to '/app', running 'npm install', exposing port 80, and setting the command to 'npm start'. The Dockerfile also includes a 'version' field set to '3'. The services section of the Dockerfile defines a service named 'app' that uses the built image, maps port 80 to 80, and runs it in the background. The terminal also shows the deployment script 'deploy.sh' being run, which uses 'docker-compose up -d' to start the service.

```
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker rm -f 6e50f9a779fb
Removing intermediate container 6e50f9a779fb
--> bc8eadea595b
Successfully built bc8eadea595b
Successfully tagged guvi_sample_app:latest
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker images
REPOSITORY          TAG           IMAGE ID        CREATED         SIZE
guvi sample_app    latest        bc8eadea595b   2 minutes ago  1.1GB
node               20            4c2ff0421257   2 weeks ago   1.1GB
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ ls
Dockerfile README.md build.sh deploy.sh docker-compose.yml package.json
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat build.sh
#!/bin/bash
docker build -t devops-build:latest .

vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat Dockerfile
FROM node:20
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 80
CMD ["npm", "start"]

vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat d
deploy.sh      docker-compose.yml
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat d
deploy.sh      docker-compose.yml
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat docker-compose.yml
version: '3'
services:
  app:
    build: .
    ports:
      - "80:80"

vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ cat deploy.sh
#!/bin/bash
docker-compose up -d

vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ |
```

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

```

vicky@Vicky: /mnt/d/guvi - D + 
da802df85c96: Pull complete
7aadcf592c3b: Pull complete
adicfcfc347ff: Pull complete
ee2a0351a498: Pull complete
8bd6885cfada: Pull complete
8f86048bb7f7: Pull complete
57cff12bcf23: Pull complete
Digest: sha256:a8e0ed56f2c20b9689e0f7dd498cac7e08d2a3a283e92d9304e7b9b83e3c6ff3
Status: Image is up to date for node:20
Status: Image is newer than local for node:20
-> 4c2ff80f21257
Step 2/6 : WORKDIR /app
--> Running in 5cf6elcf459e
Removing intermediate container 5cf6elcf459e
-> 5368abfex487
Step 3/6 : COPY .
--> 5fe6c880002f
Step 4/6 : RUN npm install
--> Running in 85a77d2f70b5
up to date, audited 1 package in 1s

found 0 vulnerabilities
Removing intermediate container 85a77d2f70b5
-> e716142394b1
Step 5/6 : EXPOSE 80
--> Running in 2eb2b317e924
Removing intermediate container 2eb2b317e924
-> 626307ebde23
Step 6/6 : CMD ["npm", "start"]
--> Running in 6e50f9a779fb
Removing intermediate container 6e50f9a779fb
-> bc8adea505b
Successfully built bc8adea505b
Successfully tagged guvi_sample_app:latest
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ 
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ 
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
guvi_sample_app    latest   bc8adea505b  2 minutes ago  1.1GB
node               20      4c2ff0421257  2 weeks ago   1.1GB
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ 

```

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

```

Index of app/
vicky@Vicky: /mnt/d/guvi - D + 
.Dockerfile
build.sh
package-lock.json

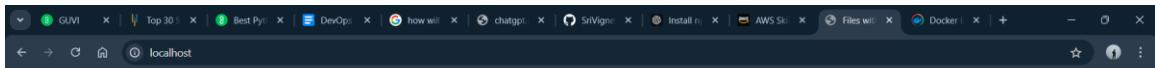
Step 4/7 : RUN npm install
--> Running in 1d016a9f3733
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN devops-build@1.0.0 No repository field.

up to date in 0.977s
found 0 vulnerabilities

Removing intermediate container 1d016a9f3733
-> 2162d3547ea4
Step 5/7 : RUN npm install -- serve
--> Running in 198c5d6fa6aa
/usr/local/bin/serve -> /usr/local/lib/node_modules/serve/build/main.js
+ serve@14.2.4
added 89 packages from 41 contributors in 10.149s
Removing intermediate container 198c5d6fa6aa
-> c4c2069d11ac
Step 6/7 : EXPOSE 80
--> Running in c7d2d5c43967
Removing intermediate container c7d2d5c43967
-> f22ae76b1fd3
Step 7/7 : CMD ["npm", "start"]
--> Running in c44a8f1d1ca2
Removing intermediate container c44a8f1d1ca2
-> 35d018e8e1d5
Successfully built 35d018e8e1d5
Successfully tagged guvi_sample_app:latest
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ docker run -d -p 80:80 guvi_sample_app
842afa85c36b519e70299515863ec630580f0121ccf8267d3bf24d2316ce482
vicky@Vicky:/mnt/d/guvi - DEVOPS/Final Project VS CODE/devops-build$ 

```

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



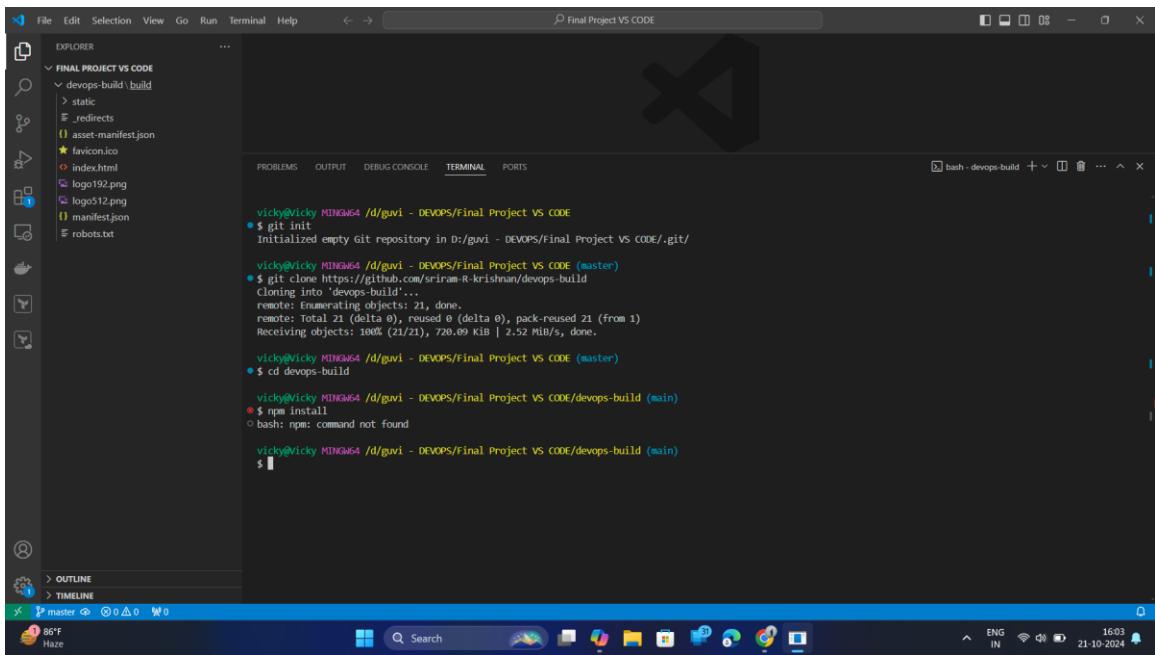
Index of app/

.Dockerfile.swp .dockerignore .gitignore README.md
build.sh build/ deploy.sh docker-compose.yml
package-lock.json package.json



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

Source Control & Git



Repository cloned locally to begin development and containerization.

The screenshot shows a Windows desktop environment with the Visual Studio Code application open. The title bar reads "File Edit Selection View Go Run Terminal Help devops-build".

The Explorer sidebar on the left lists files and folders under "DEVOPS-BUILD", including "package.json", "build", "static", ".redirections", "asset-manifest.json", "favicon.ico", "index.html", "logo192.png", "logo512.png", "manifest.json", and "robots.txt".

The main editor area displays the content of "package.json". The "scripts" section includes a "start" script set to "serve -s . -l 80". Other fields in the JSON object include "author", "license", and "description".

The bottom terminal window shows the command "npm start" being run, which triggers the "devops-build@1.0.0 start" script and "serve -s . -l 80". The output indicates that port 80 is in use, so it has chosen port 54847. It also copies the local address to the clipboard.

A preview window titled "Serving!" shows the local address "http://localhost:54847".

The status bar at the bottom shows "In 8 Col 27 Spaces 2 UTF-8 ⌂ JSON".

Application starts successfully on the development machine.

The screenshot shows a Windows desktop environment with the Visual Studio Code application open. The title bar reads "File Edit Selection View Go Run Terminal Help". The left sidebar has sections for "EXPLORER", "DEVOPS-BUILD", "package.json", "package.json > ...", "build", "static", ".redirections", "asset-manifest.json", "favicon.ico", "index.html", "logo192.png", "logo512.png", "manifest.json", "robots.txt", ".dockerignore", ".gitignore", ".buildah", "deploy.sh", "docker-compose.yml", "Dockerfile", "package.json", and "README.md". The main area shows a terminal window titled "TERMINAL" with the following content:

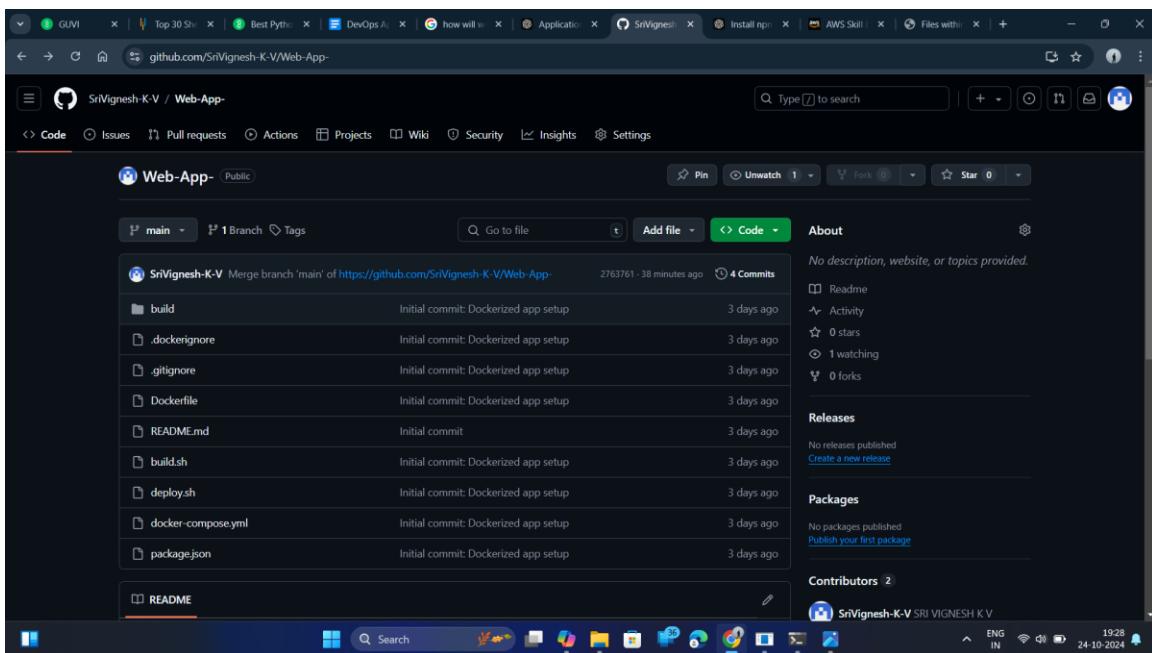
```
create mode 100644 README.ad
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build> git remote -v
origin https://github.com/SriVignesh-K-VWeb-App.git (fetch)
origin https://github.com/SriVignesh-K-VWeb-App.git (push)
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build> git add .
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build> git push -u origin main
Enumerating objects: 44, done.
Counting objects: 100% (44/44), done.
Delta compression using up to 4 threads
Compressing objects: 100% (38/38), done.
Writing objects: 100% (43/43) 723,39 KB | 8.81 MiB/s, done.
Total 43 (delta 33), reused 0 (delta 0), pack-reused 0 (from 0)
remote: everything is up-to-date.
To https://github.com/SriVignesh-K-VWeb-App.git
 82f0294..2763761 main -> main
branch 'main' set up to track 'origin/main'.
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build> git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build> git commit -m "the versioning"
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS D:\g\ui\ - DEVOPS\final Project VS CODE\devops-build>
```

The bottom status bar shows "In 5 Col 1 Spaces: 2 UTF-8 LF JSON" and icons for battery, signal, and network.

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**

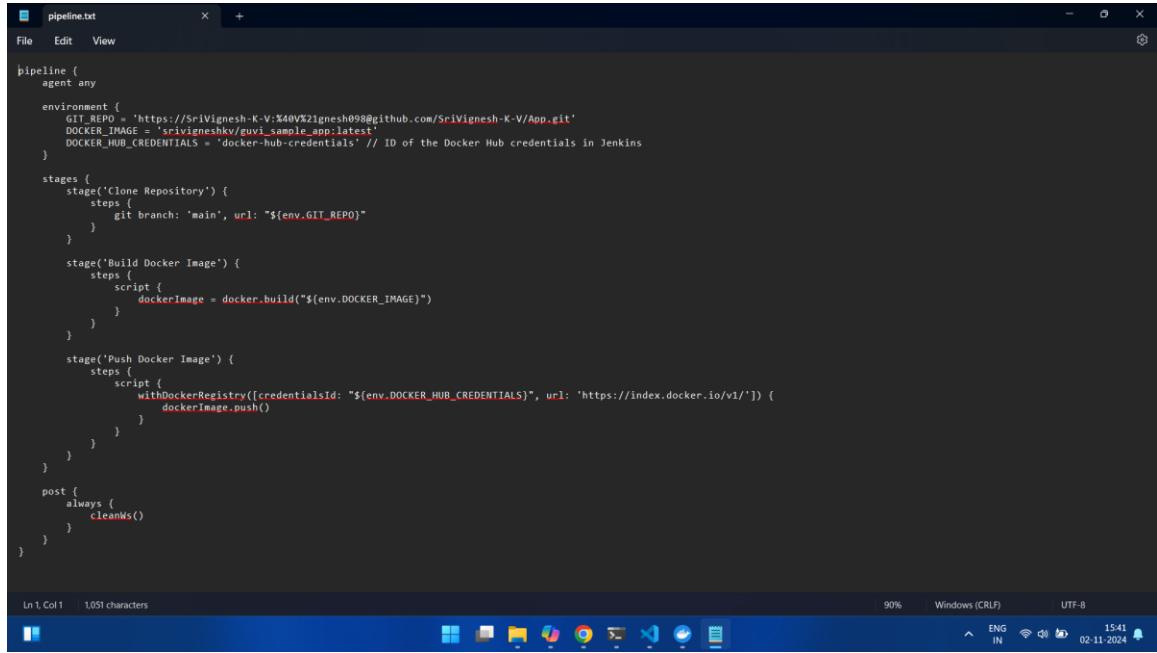
```
stage('Push Docker Image') {
    steps {
        script {
            withDockerRegistry([credentialsId:
"${env.DOCKER_HUB_CREDENTIALS}", url: 'https://index.docker.io/v1/']) {
                dockerImage.push()
            }
        }
    }
}
```

- 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.



```
pipeline {
    agent any

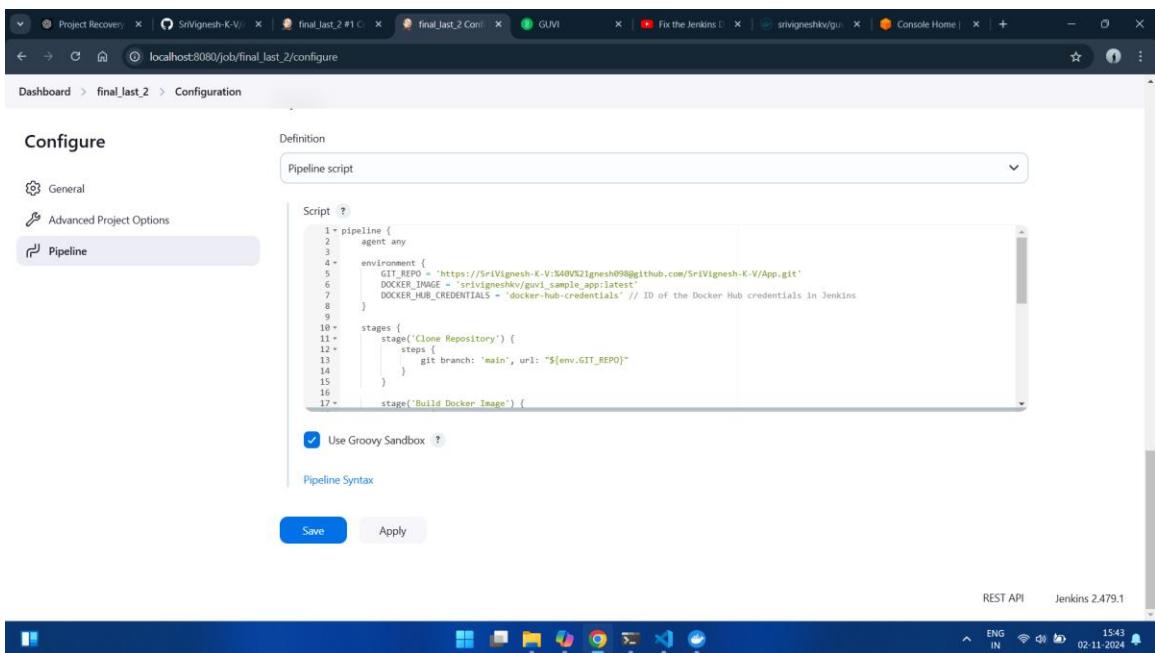
    environment {
        GIT_REPO = 'https://SriVignesh-K-V:%40V%21gnesh098@github.com/SriVignesh-K-V/App.git'
        DOCKER_IMAGE = 'srivigeshkv/uvui_sample_app:latest'
        DOCKER_HUB_CREDENTIALS = 'docker-hub-credentials' // ID of the Docker Hub credentials in Jenkins
    }

    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()
                    }
                }
            }
        }
    }

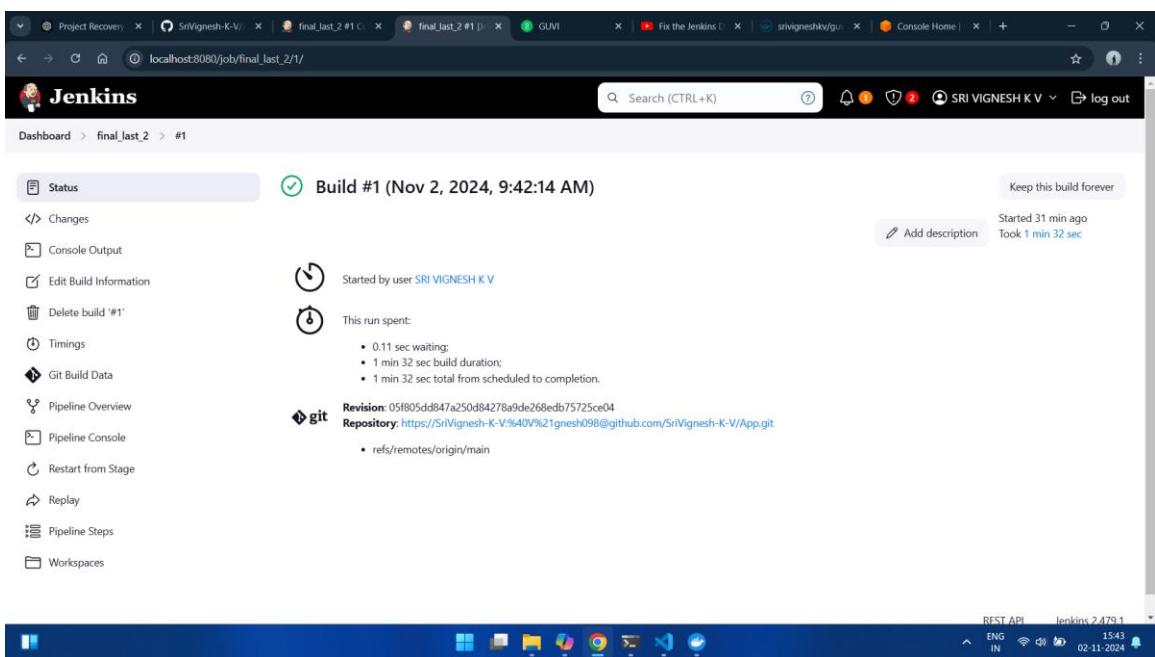
    post {
        always {
            cleanWs()
        }
    }
}
```

The screenshot shows the Jenkins pipeline script 'pipeline.txt' open in a code editor. The script is a Groovy pipeline definition. It starts with an 'environment' block setting variables for the GitHub repository URL, Docker image name, and Docker Hub credentials. It then defines three stages: 'Clone Repository', 'Build Docker Image', and 'Push Docker Image'. Each stage has a single step: cloning the repository, building the Docker image, and pushing it to Docker Hub respectively. Finally, a 'post' block is defined, which always runs the 'cleanWs()' command to clean the Jenkins workspace.

Groovy pipeline script for quick visual reference.



Jenkins UI overview where the pipeline job is configured.



A sample build run executing stages.

The screenshot shows the Jenkins console output for a pipeline job named 'final_last_2'. The output highlights the git checkout and build steps. The pipeline starts by cloning a repository from GitHub:

```

Started by user SRI VIGNESH K V
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/final_last_2
[Pipeline] { (hide)
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Clone Repository)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://SriVignesh-K-V:%40%21gnesh098@github.com/SriVignesh-K-V/App.git
> git init /var/lib/jenkins/workspace/final_last_2 # timeout=10
Fetching upstream changes from https://SriVignesh-K-V@github.com/SriVignesh-K-V/App.git
> git --version # timeout=10
> git --version # git version 2.43.0'
> git fetch --tags --force --progress -- https://SriVignesh-K-V:%40%21gnesh098@github.com/SriVignesh-K-V/App.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://SriVignesh-K-V:%40%21gnesh098@github.com/SriVignesh-K-V/App.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch.

```

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

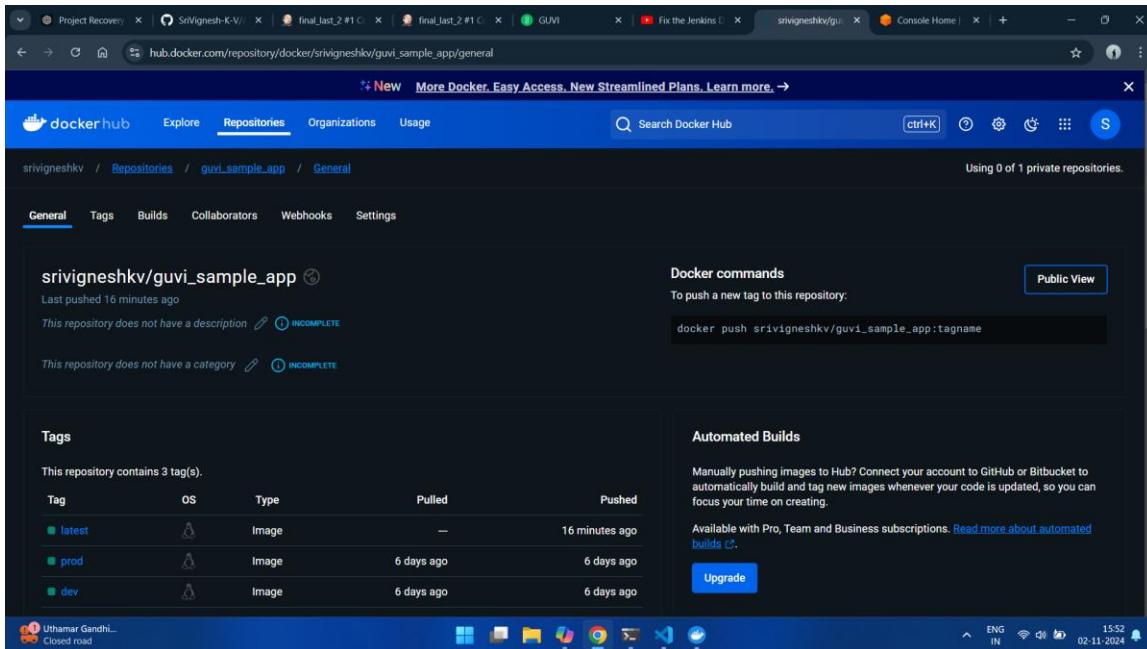
The screenshot shows the Jenkins console output for the same pipeline job. The output highlights the push and deploy steps. It shows the workspace being cleaned up and the pipeline ending successfully:

```

b2dba7477754: Layer already exists
lastest: digest: sha256:ecfe6c6961e8b7a01c111440112df3331f21d901f2be3d301a16fc821f3506f size: 3050
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // withDockerRegistry
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions) (hide)
[Pipeline] cleanses
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] deferred wipeout is used...
[WS-CLEANUP] done
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

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

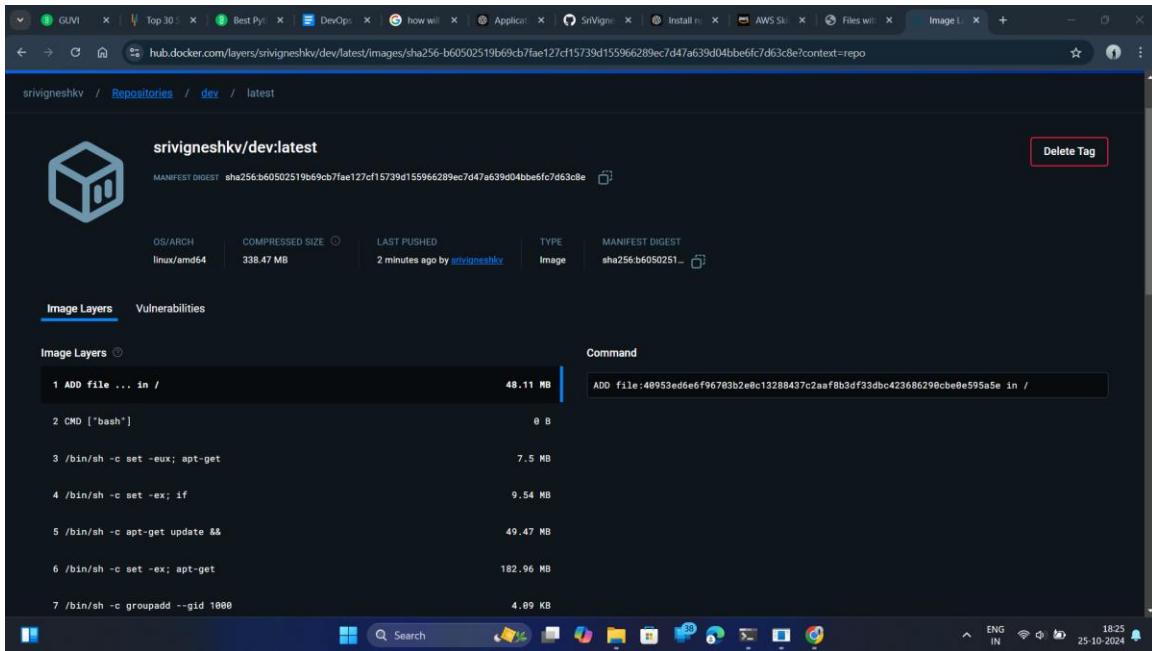


Confirms Docker image successfully pushed by Jenkins with the latest tag.

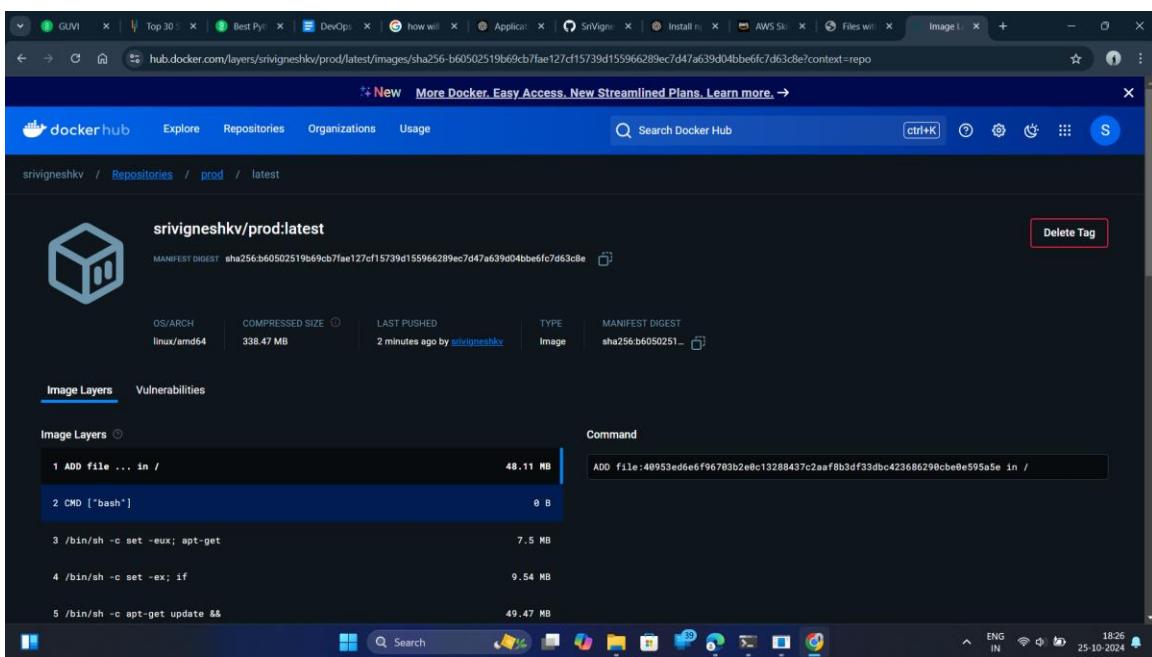
Docker Hub Integration

```
vicky@Vicky:/mnt/d/guvi - D ~ + ~
vicky@Vicky:/mnt/d/guvi - DEVP05/Final Project VS CODE/devops-build$ docker tag guvi_sample_app:latest srivigneshkv/dev:latest
vicky@Vicky:/mnt/d/guvi - DEVP05/Final Project VS CODE/devops-build$ docker tag guvi_sample_app:latest srivigneshkv/prod:latest
vicky@Vicky:/mnt/d/guvi - DEVP05/Final Project VS CODE/devops-build$ docker login -u srivigneshkv
Password:
Login Succeeded
vicky@Vicky:/mnt/d/guvi - DEVP05/Final Project VS CODE/devops-build$ docker push srivigneshkv/dev:latest
The push refers to repository [docker.io/srivigneshkv/dev]
3eb1e38d8f9d: Pushed
fa5fcbd59ad0: Pushed
db80c17b276: Pushed
df8fba5a658a9: Pushed
0d5f5aa15e5d: Mounted from library/node
3c777d951de2: Mounted from library/node
f8a91dd5fca4: Mounted from library/node
cb8127abde5: Mounted from library/node
e01au54893a9: Mounted from library/node
c45660adde37: Mounted from library/node
fe0fb3ab4a0f: Mounted from library/node
f1186e5861f2: Mounted from library/node
b2dba74777754: Mounted from library/node
latest: digest: sha256:b60502519b69cb7fae127cf15739d155966289ec7d47a639d04bbe6fc7d63c8e size: 3050
vicky@Vicky:/mnt/d/guvi - DEVP05/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
db80c17b276: Mounted from srivigneshkv/dev
df8fba5a658a9: Mounted from srivigneshkv/dev
0d5f5aa15e5d: Mounted from srivigneshkv/dev
3c777d951de2: Mounted from srivigneshkv/dev
f8a91dd5fca4: Mounted from srivigneshkv/dev
cb8127abde5: Mounted from srivigneshkv/dev
e01au54893a9: Mounted from srivigneshkv/dev
c45660adde37: Mounted from srivigneshkv/dev
fe0fb3ab4a0f: Mounted from srivigneshkv/dev
f1186e5861f2: Mounted from srivigneshkv/dev
b2dba74777754: Mounted from srivigneshkv/dev
latest: digest: sha256:b60502519b69cb7fae127cf15739d155966289ec7d47a639d04bbe6fc7d63c8e size: 3050
vicky@Vicky:/mnt/d/guvi - DEVP05/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

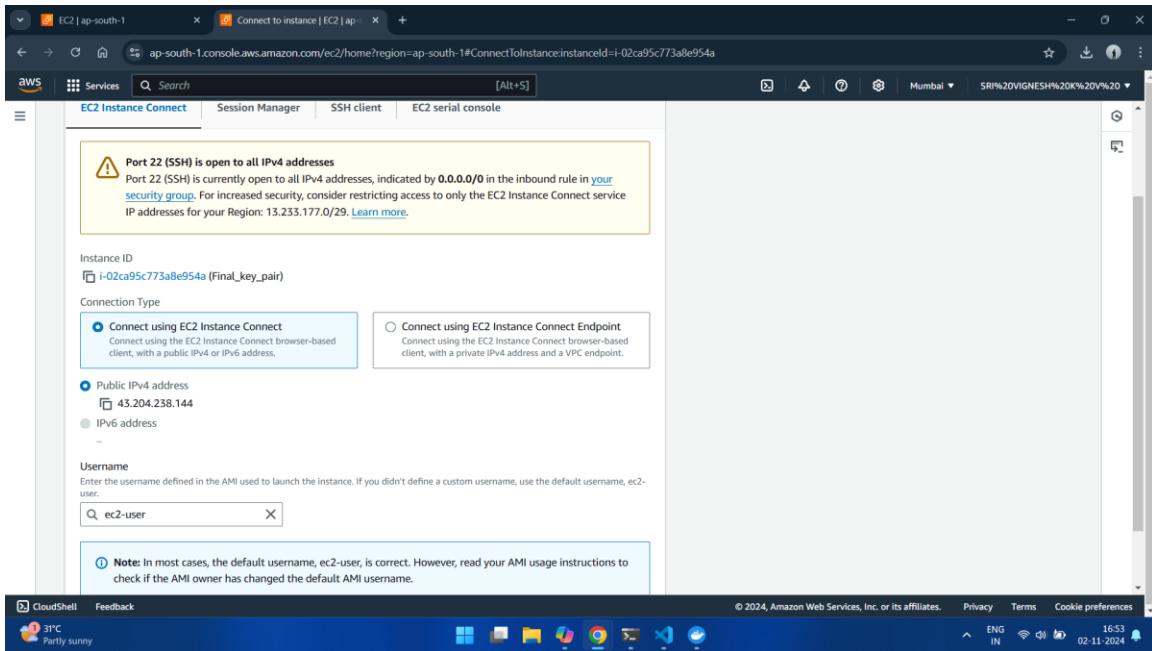
The screenshot shows the AWS Security Groups console. The security group 'sg-03fe31e2c2138c08b - launch-wizard-3' is selected. The 'Details' section shows the security group ID (sg-03fe31e2c2138c08b), creation date (2024-11-02T10:44:39.384Z), owner (891376981613), inbound rules count (3), and outbound rules count (1). The 'Inbound rules' tab is selected, displaying three rules:

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sgr-034f5bf26e40e419f	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sgr-0a04982cb157545	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-04b72cccf1c77b225	IPv4	HTTPS	TCP	443	0.0.0.0/0	-

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

The screenshot shows the AWS Instances console. An instance named 'Final_key_pair' (ID i-02ca95c773a8e954a) is selected. The 'Instances (1/2) info' table shows the instance state (Running), instance type (t2.micro), and availability zone (ap-south-1b). The 'i-02ca95c773a8e954a (Final_key_pair)' details page shows the instance summary, including the instance ID (i-02ca95c773a8e954a), public IPv4 address (43.204.238.144), private IP DNS name (ip-172-31-11-13.ap-south-1.compute.internal), and elastic IP addresses (172.31.11.13).

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



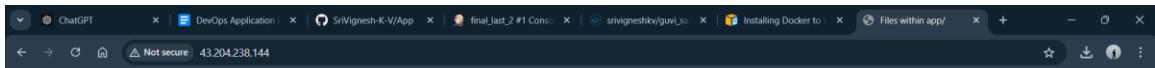
Connecting to the EC2 instance via the AWS Console/SSH.

Deploying & Running the Container

```
C:\Windows\system32\cmd.exe x vicky@Vicky: /mnt/c/Users/vic x ec2-user@ip-172-31-11-13~ x + 
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
b253aeafea9a: Pull complete
3d281bd995c: Pull complete
1de76e268b10: Pull complete
d9a8df5f89451: Pull complete
6f51ee083dea: Pull complete
5f3ze83c3f27: Pull complete
0c8cc2f24a1d: Pull complete
0d218e80132: Pull complete
d3acd12b6fb: Pull complete
bf4f723a76312: Pull complete
0d023b5afae5: Pull complete
84079fe2b3c79: Pull complete
Digest: sha256:ccf46c6961e8b7a01c1114408112df3331f21d901f2be3d301a16fc821f3506f
Status: Image is up to date 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
e67845275a5393127810ea7ca74073e0lab5ee4fb4497991519cb25ab349e32
[ec2-user@ip-172-31-11-13 ~]$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e67845275a5 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_sample_app
[ec2-user@ip-172-31-11-13 ~]$
```

Container is up and running on the target host.



Index of app/

.Dockerfile.swp .dockerignore .gitignore README.md
build.sh build/ deploy.sh docker-compose.yml
package-lock.json package.json



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

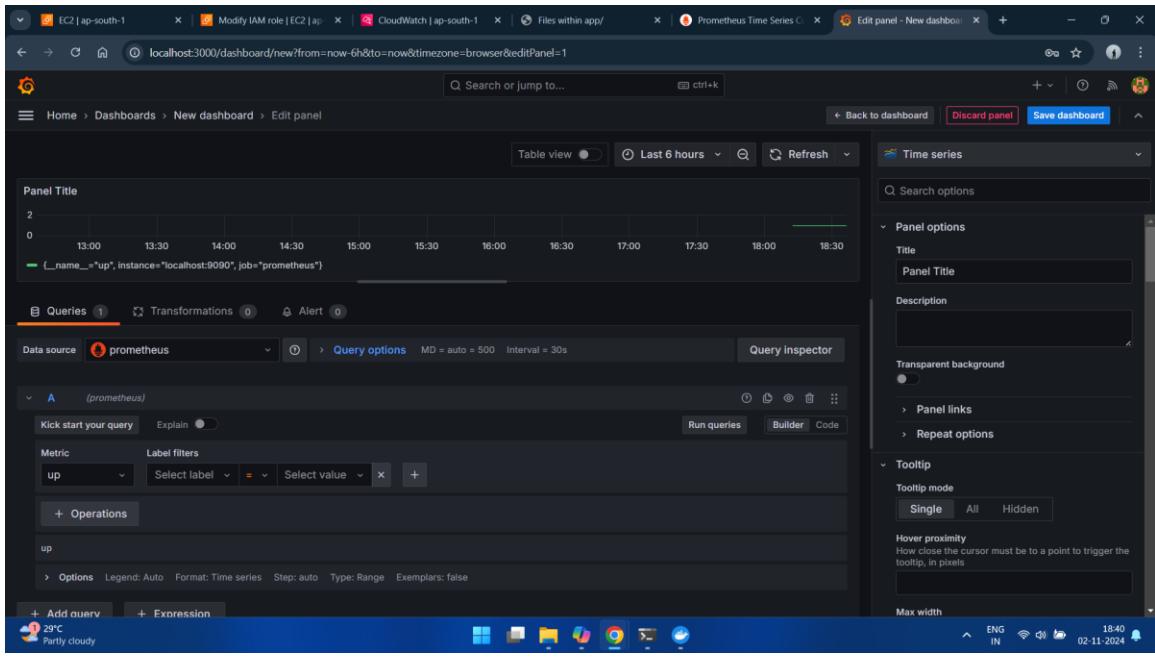
Observability: Prometheus & Grafana

The screenshot shows the Prometheus Targets page. The URL in the address bar is `localhost:9090/targets?search=`. The page title is "Targets". There is a search bar labeled "Filter by endpoint or labels" and checkboxes for "Unknown", "Unhealthy", and "Healthy". A table lists one target:

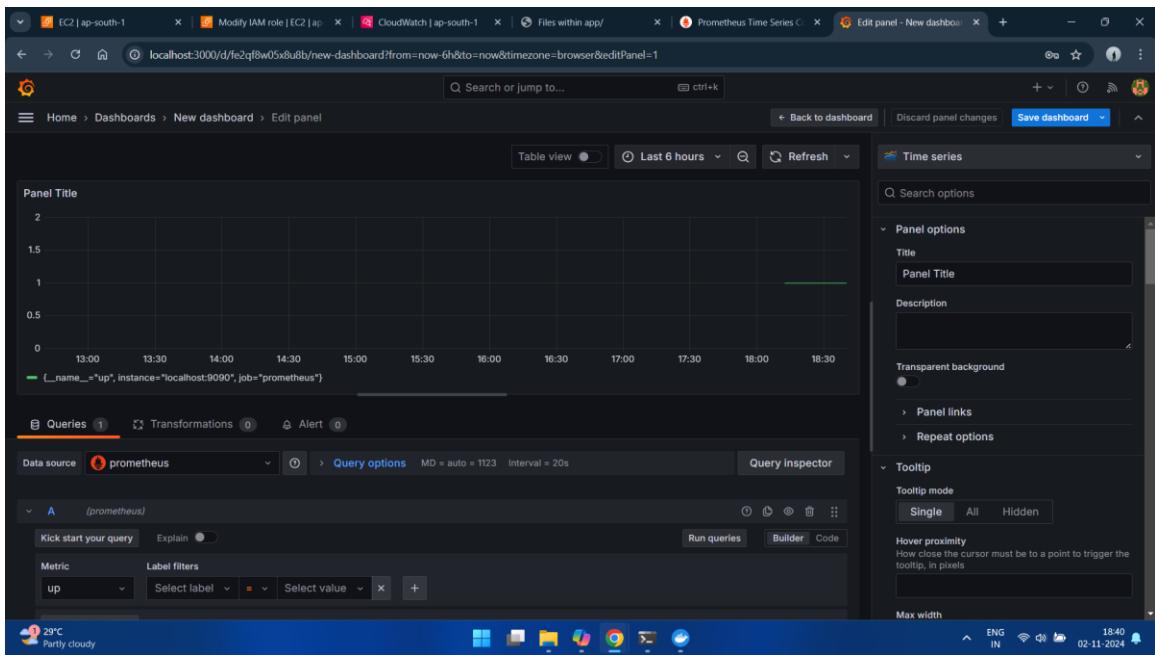
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance:"localhost:9090" job:"prometheus"	9.204s ago	4.576ms	

At the bottom of the screen, the taskbar shows a browser window titled "vicky:9090/metrics" and other system icons.

Prometheus target/metrics page—validates metrics scraping.



Grafana service starting/up—confirm UI availability.



Grafana dashboard visualizing application/infrastructure metrics.

Optional Attempt: CloudWatch

```
C:\Windows\system32\cmd.exe x vicky@Vicky: /mnt/c/Users/vic x ec2-user@ip-172-31-11-13:op + - o x [ec2-user@ip-172-31-11-13 etc]$ sudo systemctl restart amazon-cloudwatch-agent [ec2-user@ip-172-31-11-13 etc]$ sudo systemctl status amazon-cloudwatch-agent ● amazon-cloudwatch-agent.service - Amazon CloudWatch Agent Loaded: loaded (/etc/systemd/system/amazon-cloudwatch-agent.service; disabled; preset: disabled) Active: active (running) since Sat 2024-11-02 11:43:27 UTC; 4s ago Main PID: 35806 (amazon-cloudwat) Tasks: 6 (limit: 1112) Memory: 111.3M CPU: 287ms CGroup: /system.slice/amazon-cloudwatch-agent.service └─35806 /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -config /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal systemd[1]: Started amazon-cloudwatch-agent.service - Amazon CloudWatch Agent. Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: D! [EC2] Found active network interface start-amazon-cloudwatch-agent[35833]: I! imds retry client will retry 1 times! Detected the in... Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: 2024/11/02 11:43:27 Reading json config file path: /opt/aw... Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: 2024/11/02 11:43:27 I! Valid Json input schema. Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: I! Detecting run_as_user... Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: I! Trying to detect region from ec2 start-amazon-cloudwatch-agent[35833]: I! Trying to detect region from ec2 Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35833]: 2024/11/02 11:43:27 W! Ignoring unrecognized input docker start-amazon-cloudwatch-agent[35833]: 2024/11/02 11:43:27 Configuration validation first phase p... Nov 02 11:43:27 ip-172-31-11-13.ap-south-1.compute.internal start-amazon-cloudwatch-agent[35806]: I! Ignoring unrecognized input docker start-amazon-cloudwatch-agent[35833]: 2024/11/02 11:43:27 Configuration validation first phase p... [ec2-user@ip-172-31-11-13 etc]$ sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwa... tch-agent/etc/amazon-cloudwatch-agent.json -s ***** processing amazon-cloudwatch-agent ***** I! Trying to detect region from ec2 D! [EC2] Found active network interface I! imds retry client will retry 1 timesSuccessfully fetched the config and saved in /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json.tmp Start configuration validation... 2024/11/02 11:43:38 Reading json config file path: /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.d/file_amazon-cloudwatch-agent.json.tmp ... 2024/11/02 11:43:38 I! Valid Json input schema. 2024/11/02 11:43:38 W! Ignoring unrecognized input docker 2024/11/02 11:43:38 Configuration validation first phase succeeded I! Detecting run_as_user... I! Trying to detect region from ec2 D! [EC2] Found active network interface I! imds retry client will retry 1 times /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent -schematest -config /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.toml Configuration validation second phase succeeded Configuration validation succeeded Created symlink /etc/systemd/system/multi-user.target.wants/amazon-cloudwatch-agent.service → /etc/systemd/system/amazon-cloudwatch-agent.service. [ec2-user@ip-172-31-11-13 etc]$ |
```

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.