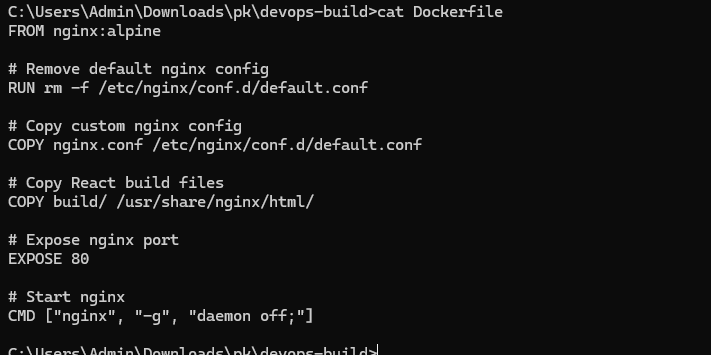
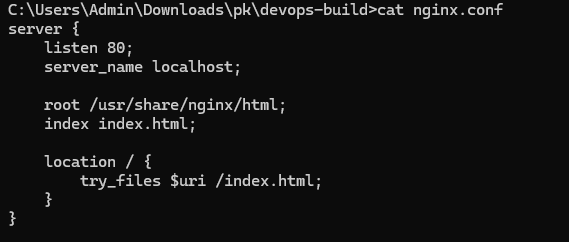
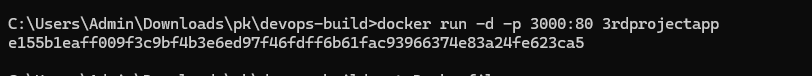
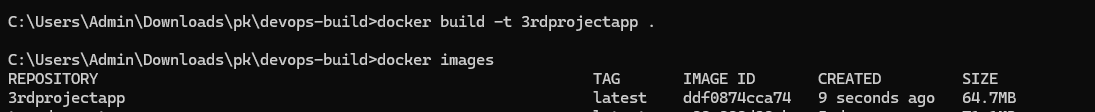
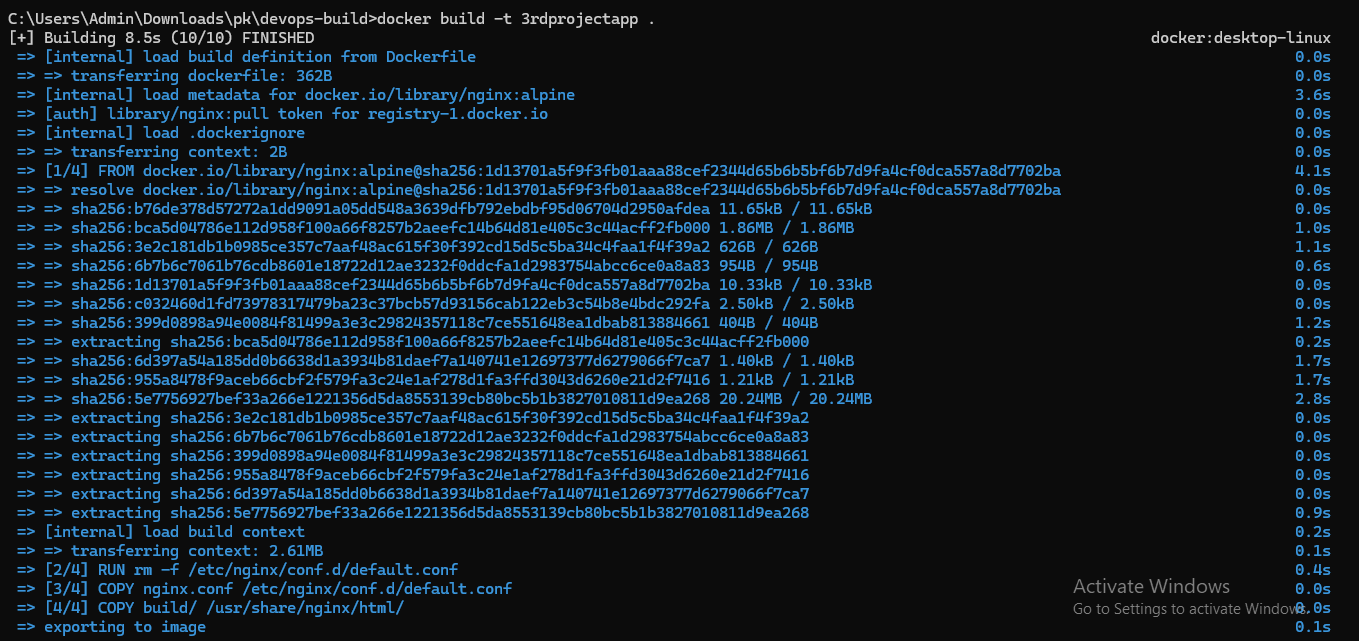
Created Docker file



Configuration file



It builds/pulls the required images and starts containers in detached mode.



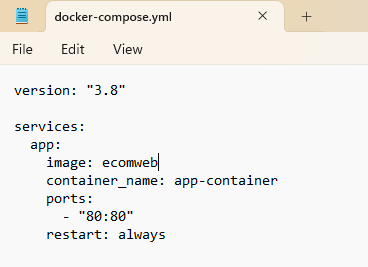
Result: containers are up and the application becomes accessible on the configured port.

• Application Access: Port 3000 allowed from Anywhere (0.0.0.0/0) so anyone with the public IP can access the application.

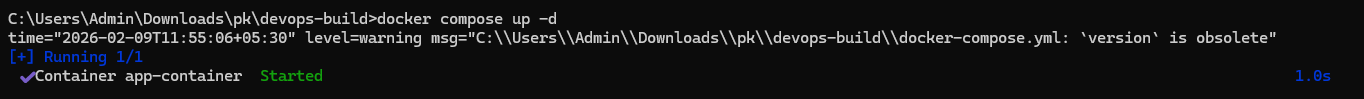
Docker-compose

docker compose up –d

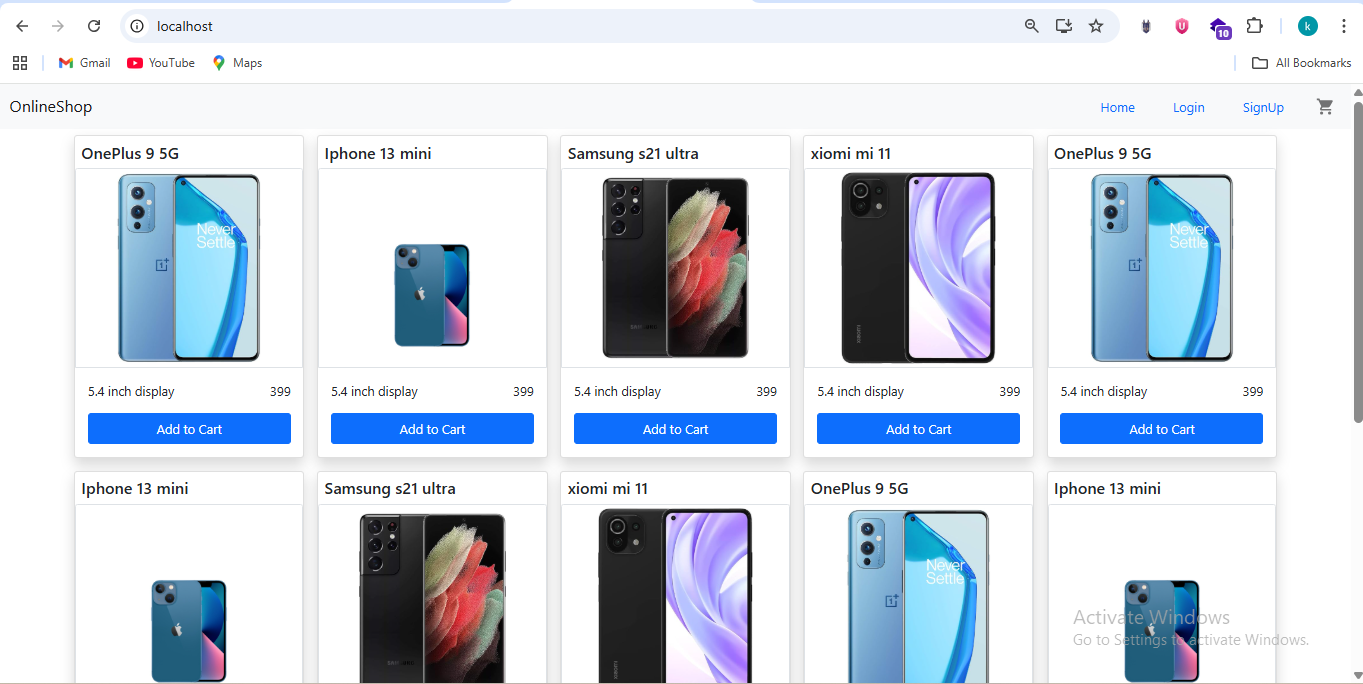
Docker Compose – Start Application



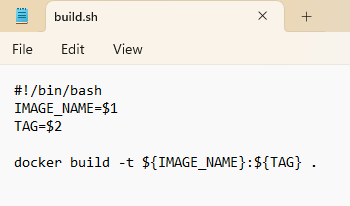
Docker Compose – Start Application

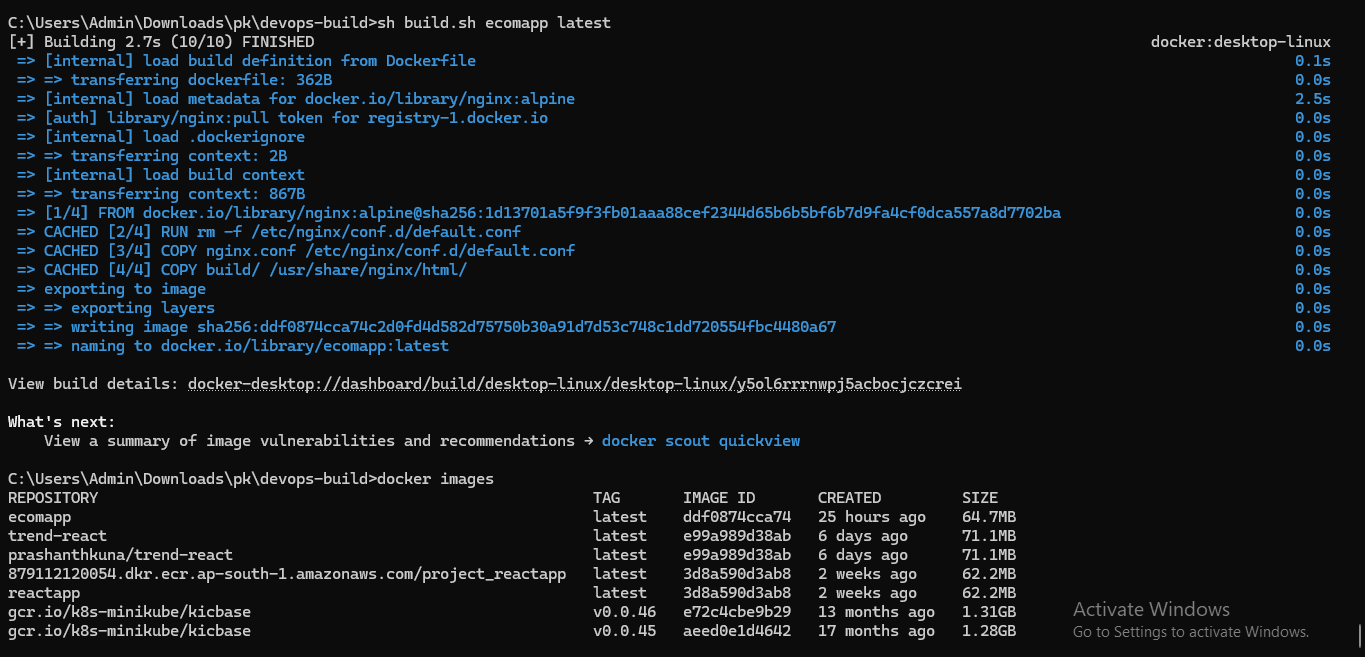


This screenshot shows the application being started using Docker Compose.

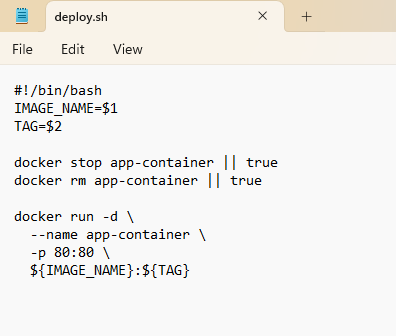
Bash scripting

This screenshot highlights the bash script used to automate deployment steps.

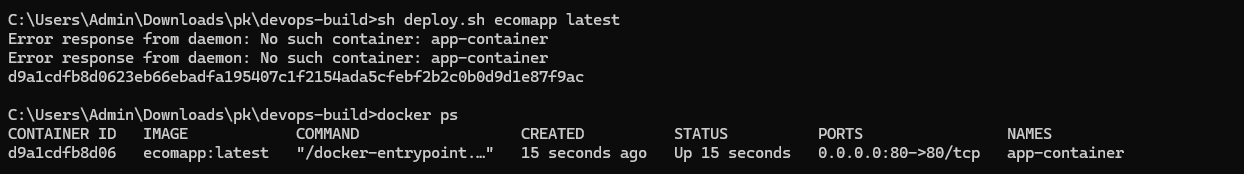




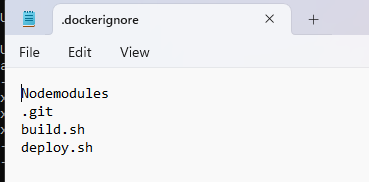
The script pulls the latest image, stops/removes the old container, and starts the updated container.

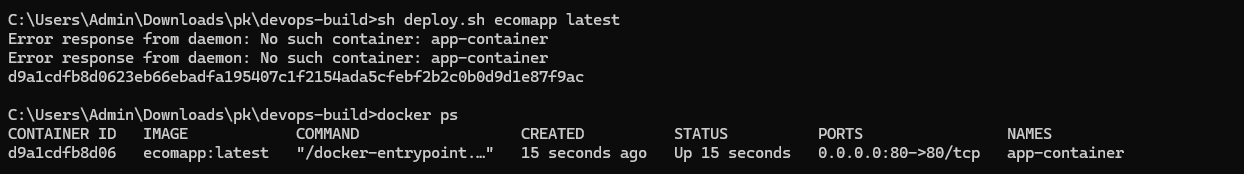


This makes deployments consistent, repeatable, and reduces manual errors.

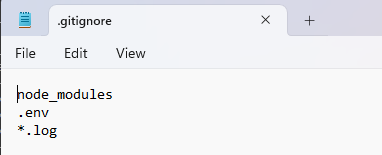


Using Dockerignore, it won’t allow the required files mentioned in the ignore file.

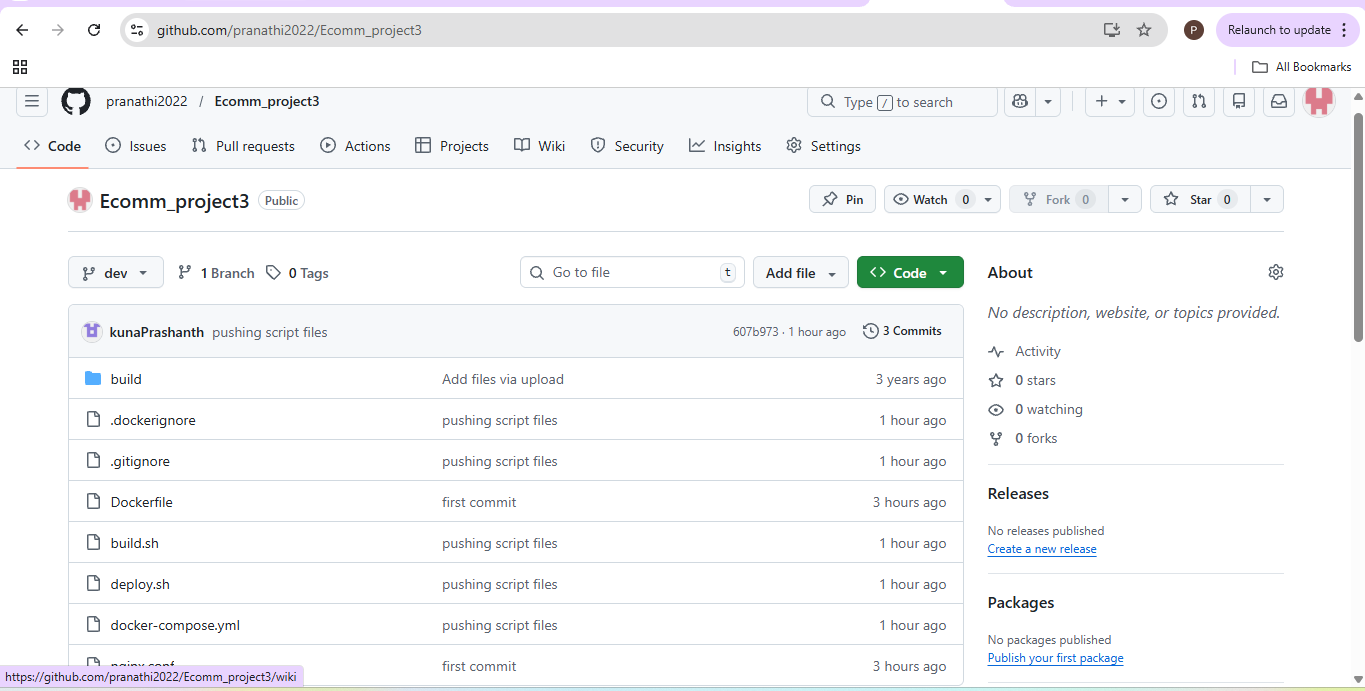
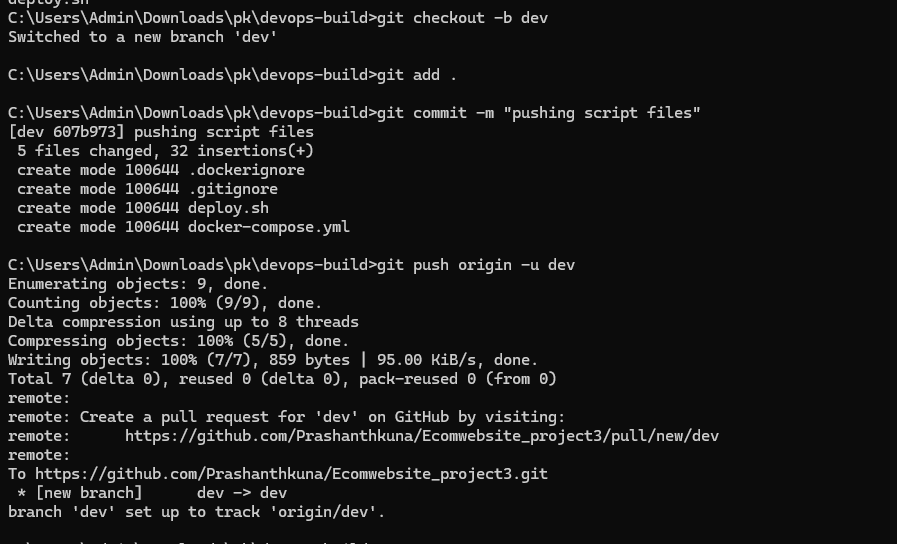




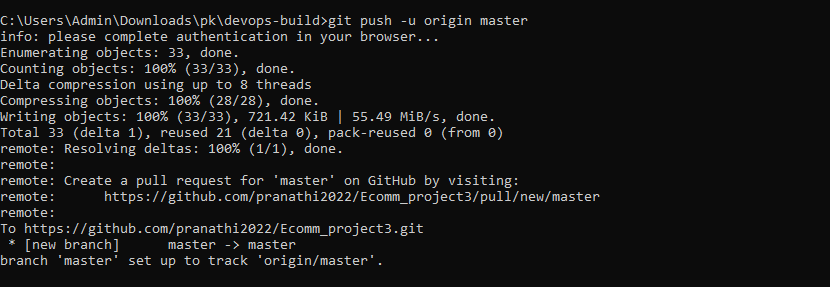
Using gitignore, it won’t allow the required files mentioned in the ignore file.



Creating new branch in git – dev and committing the changes then pushing the required files into github



Creating new branch in git – master and committing the changes then pushing the required files into github

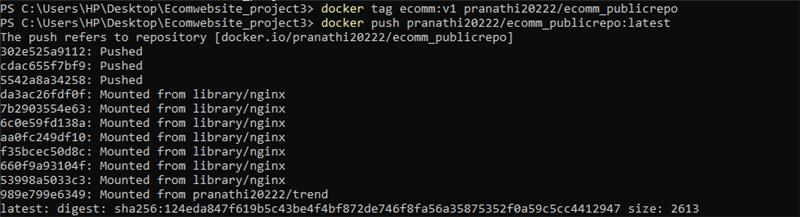
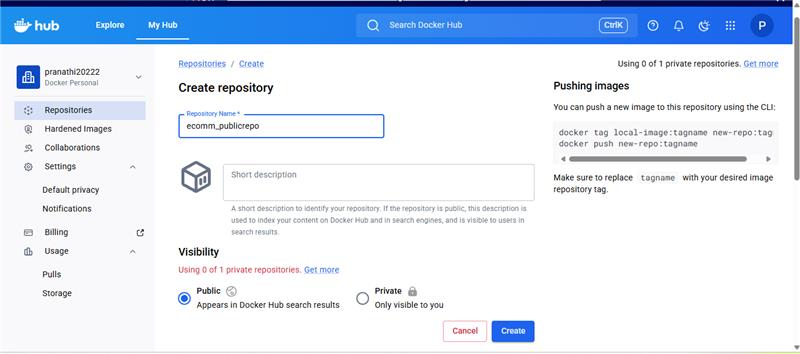


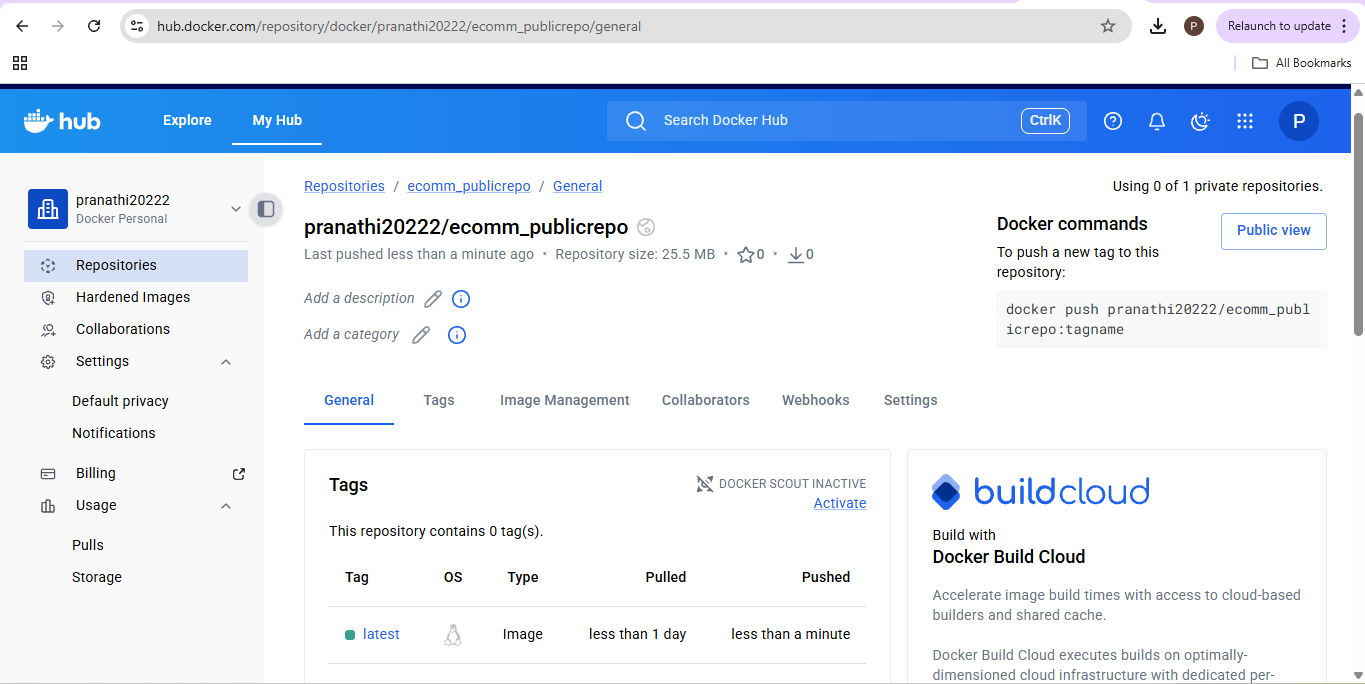
DockerHub repositories created for this project.

DockerHub – Public and Private Repositories

Public Repo – ecomm\_publicrepo

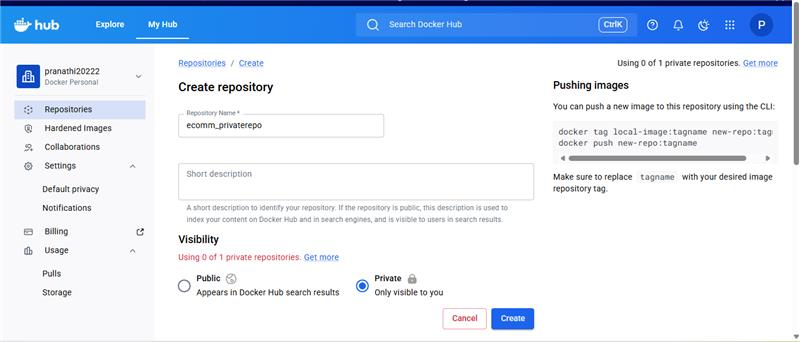
• Public repository: stores dev images for testing and development.

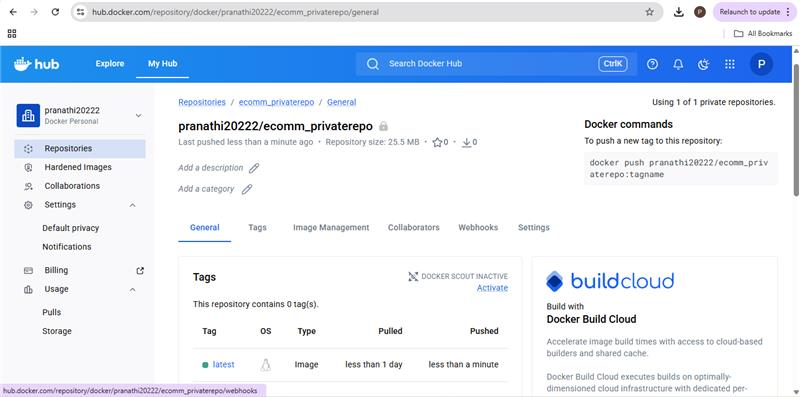




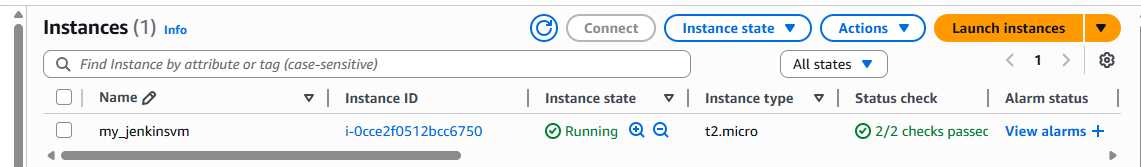
Private repo – ecomm\_privaterepo

• Private repository: stores prod images for production deployments.

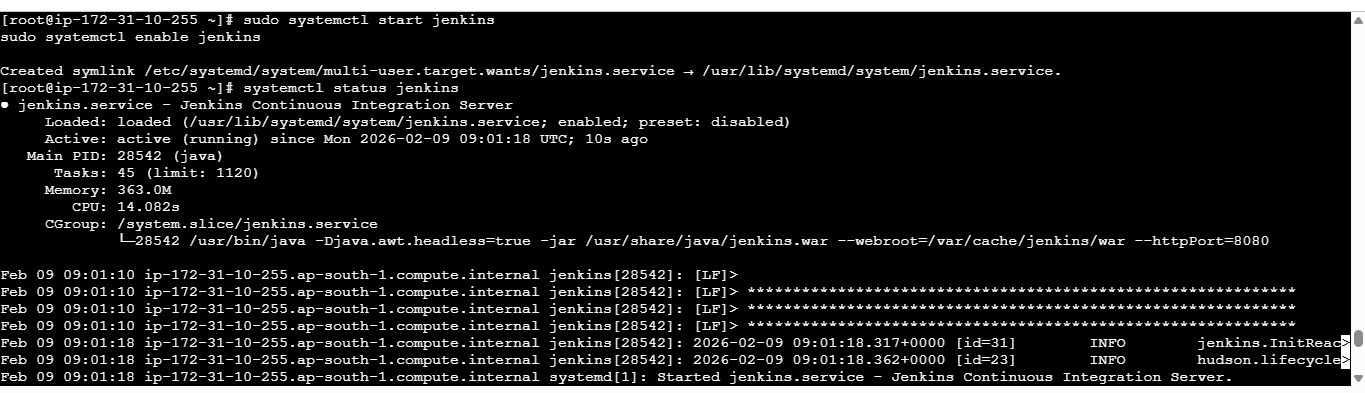




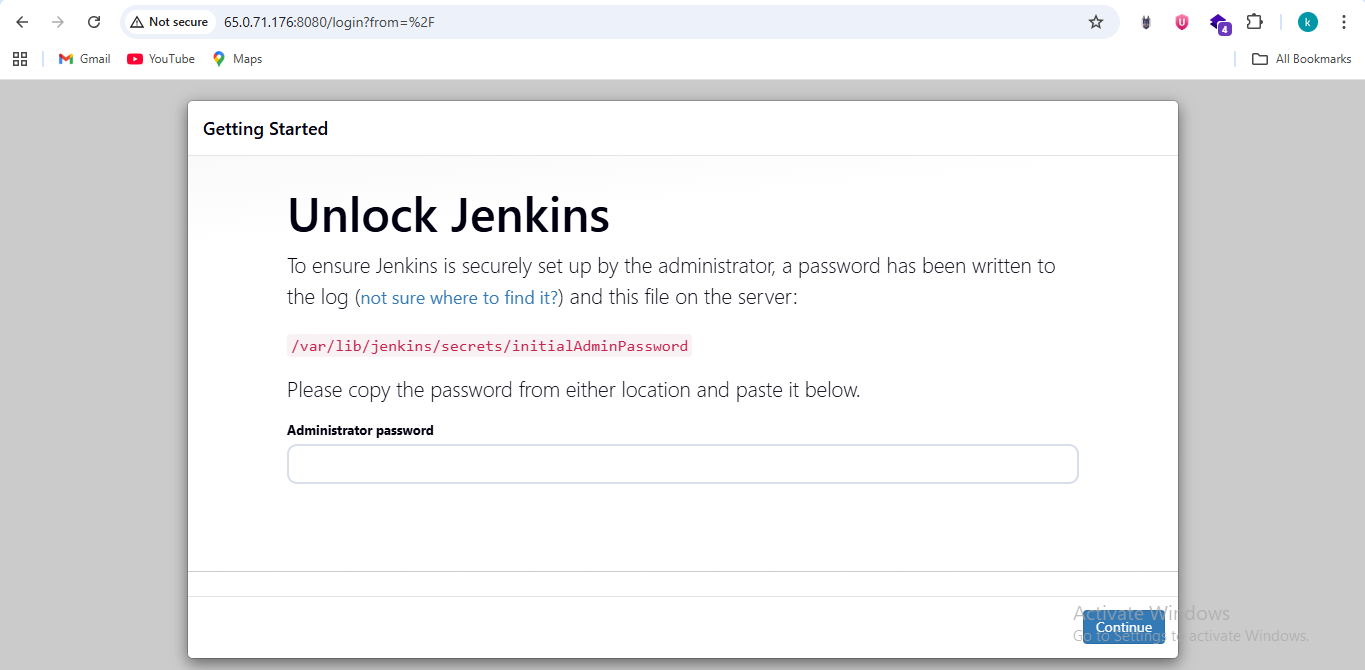
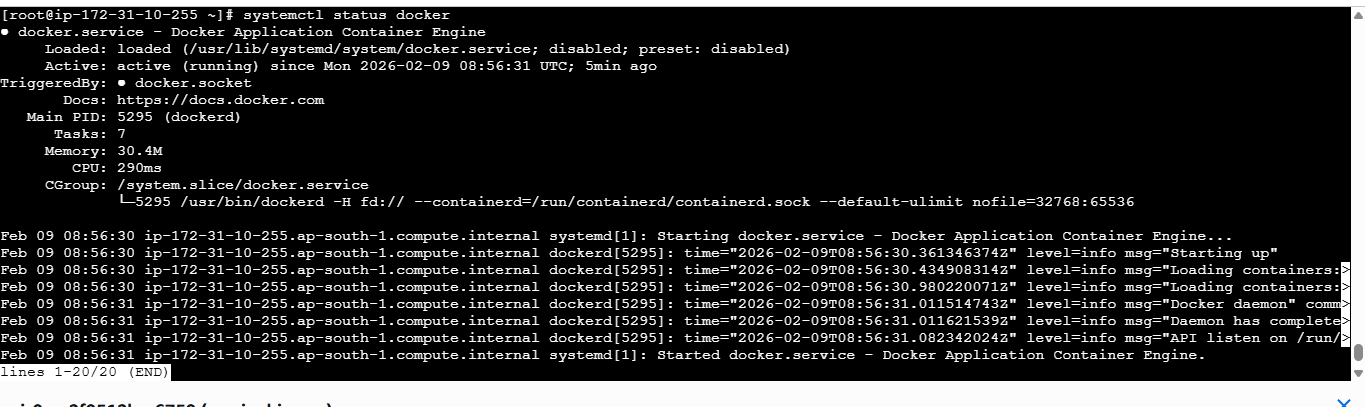
Created EC2 instance and installed Jenkins,Java, docker and Git

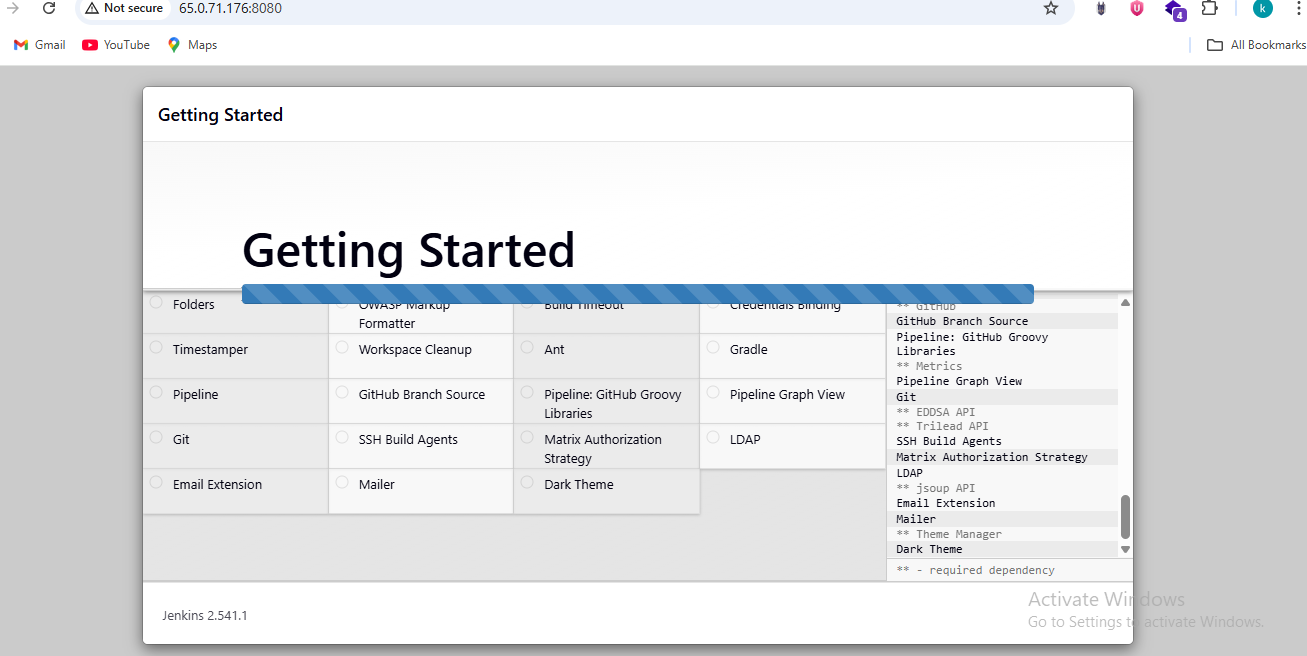
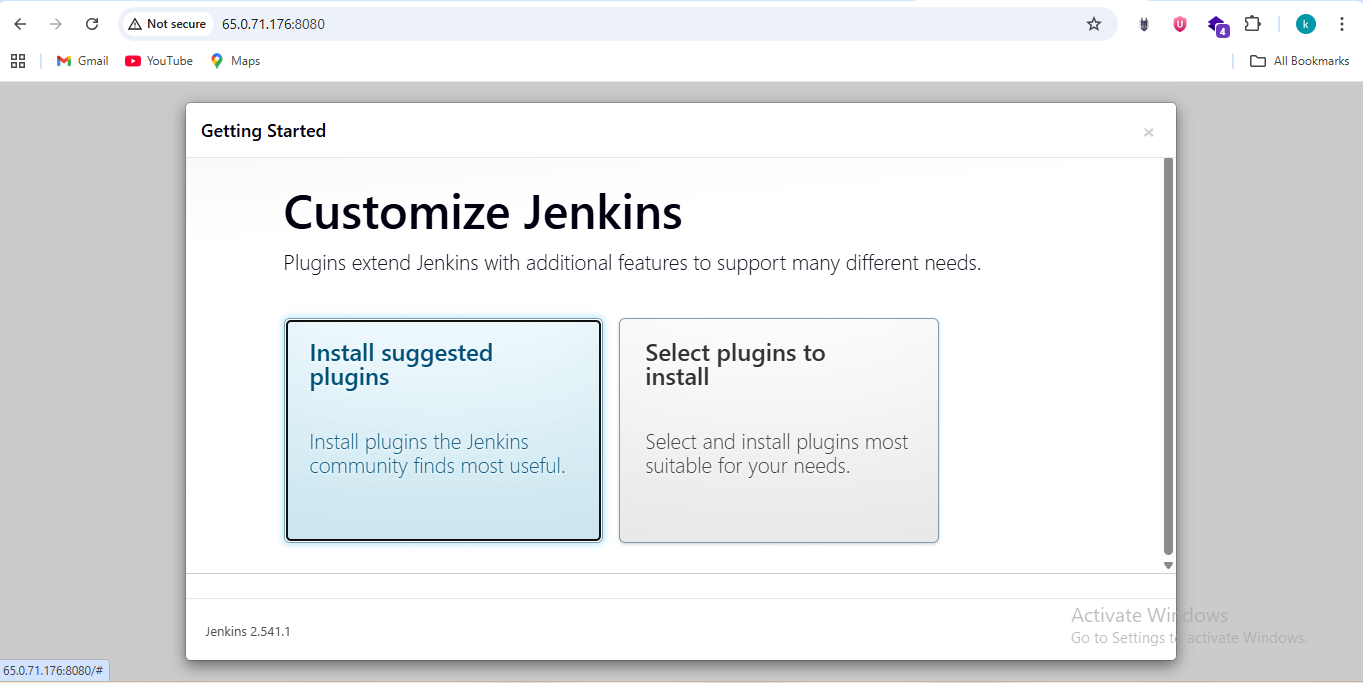


Systemctl status Jenkins



Systemctl status docker

Installing required plugins and getting started



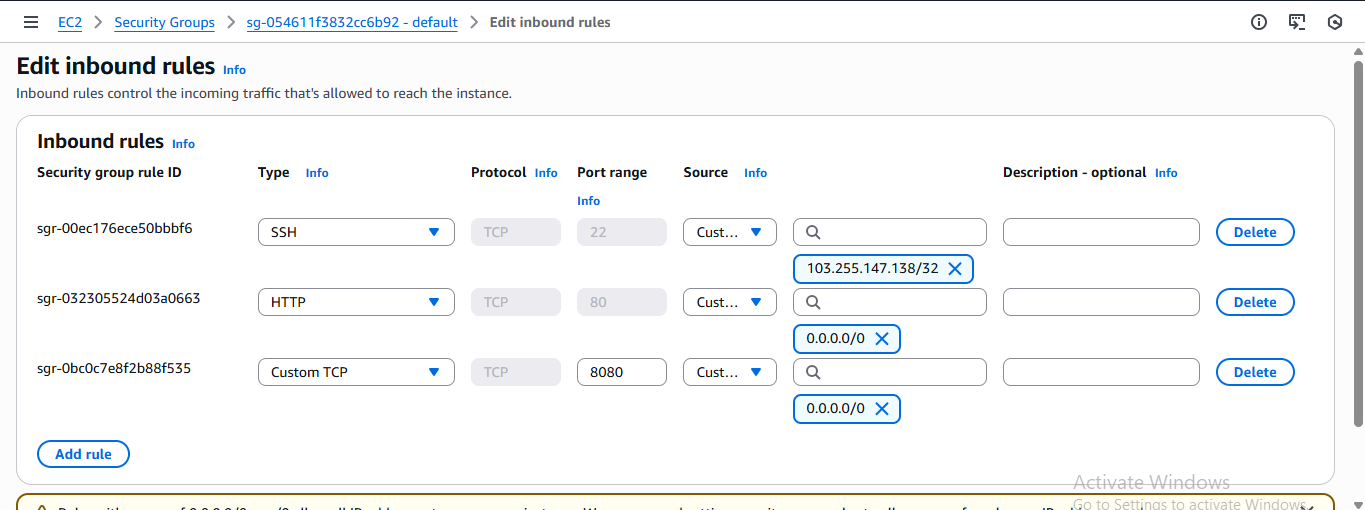
**AWS – Security Group Configuration (SSH + Application Access)**

This screenshot shows the Security Group rules configured for the EC2 instance as per the project requirement.

Rules applied:

• Secure Login: SSH port 22 allowed only from my public IP (/32) to restrict server access.

This ensures the application is publicly reachable while server login remains protected.



Created multibranch pipeline

Jenkins – Multibranch Pipeline Configuration

This screenshot shows the Jenkins Multibranch Pipeline job created for the Git repository.

pipeline {

agent any

environment {

IMAGE\_NAME = "ecom-app"

DEV\_REPO = "pranathi20222/ecomm\_publicrepo"

PROD\_REPO = "pranathi20222/ecomm\_privaterepo"

}

stages {

stage('Checkout') {

steps {

checkout scm

}

}

stage('Build Docker Image') {

steps {

script {

echo "Building Docker image for branch: ${env.BRANCH\_NAME}"

if (env.BRANCH\_NAME == 'dev') {

sh "docker build -t ${IMAGE\_NAME}:dev ."

}

if (env.BRANCH\_NAME == 'master') {

sh "docker build -t ${IMAGE\_NAME}:prod ."

}

}

}

}

stage('Docker Login & Push Image') {

steps {

script {

withCredentials([

usernamePassword(

credentialsId: 'dockerhubcreds',

usernameVariable: 'DOCKER\_USER',

passwordVariable: 'DOCKER\_PASS'

)

]) {

sh '''

echo "$DOCKER\_PASS" | docker login -u "$DOCKER\_USER" --password-stdin

'''

if (env.BRANCH\_NAME == 'dev') {

sh """

docker tag ${IMAGE\_NAME}:dev ${DEV\_REPO}:dev

docker push ${DEV\_REPO}:dev

"""

}

if (env.BRANCH\_NAME == 'master') {

sh """

docker tag ${IMAGE\_NAME}:prod ${PROD\_REPO}:latest

docker push ${PROD\_REPO}:latest

"""

}

}

}

}

}

stage('Deploy to Server') {

when {

branch 'master'

}

steps {

sh """

./deploy.sh ${PROD\_REPO} latest

"""

}

}

}

post {

success {

echo "✅ Pipeline SUCCESS for branch: ${env.BRANCH\_NAME}"

}

failure {

echo "❌ Pipeline FAILED for branch: ${env.BRANCH\_NAME}"

}

}

}

Jenkins automatically detects branches (dev and master) and triggers builds based on commits.

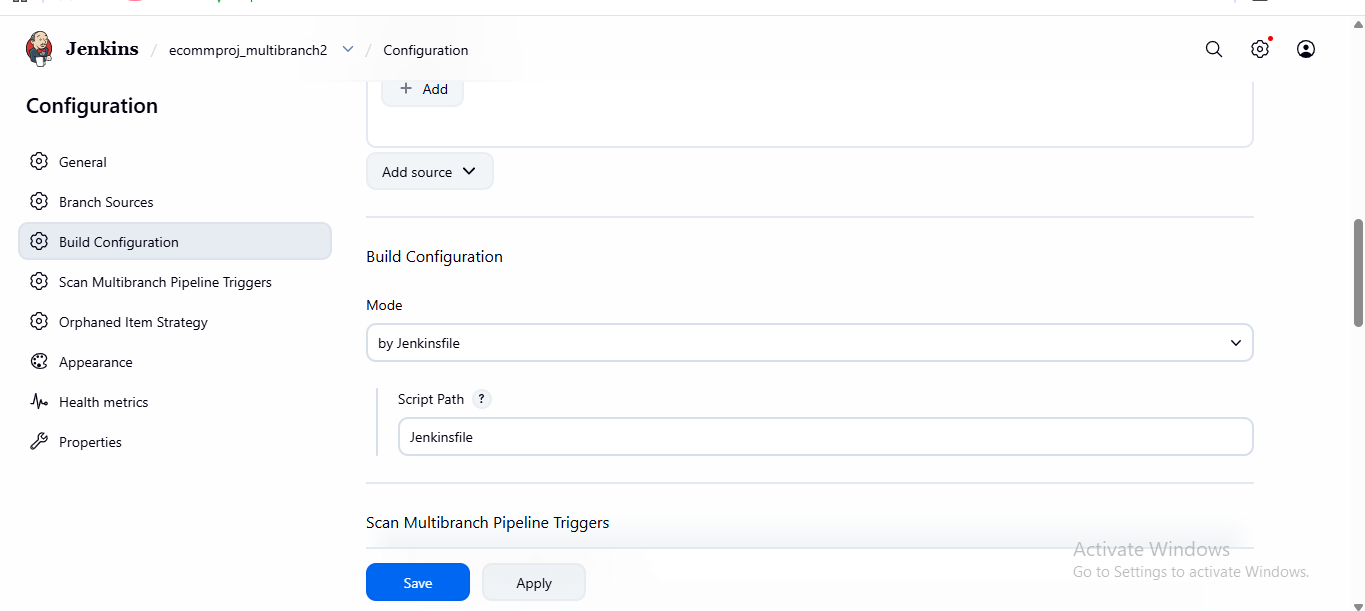
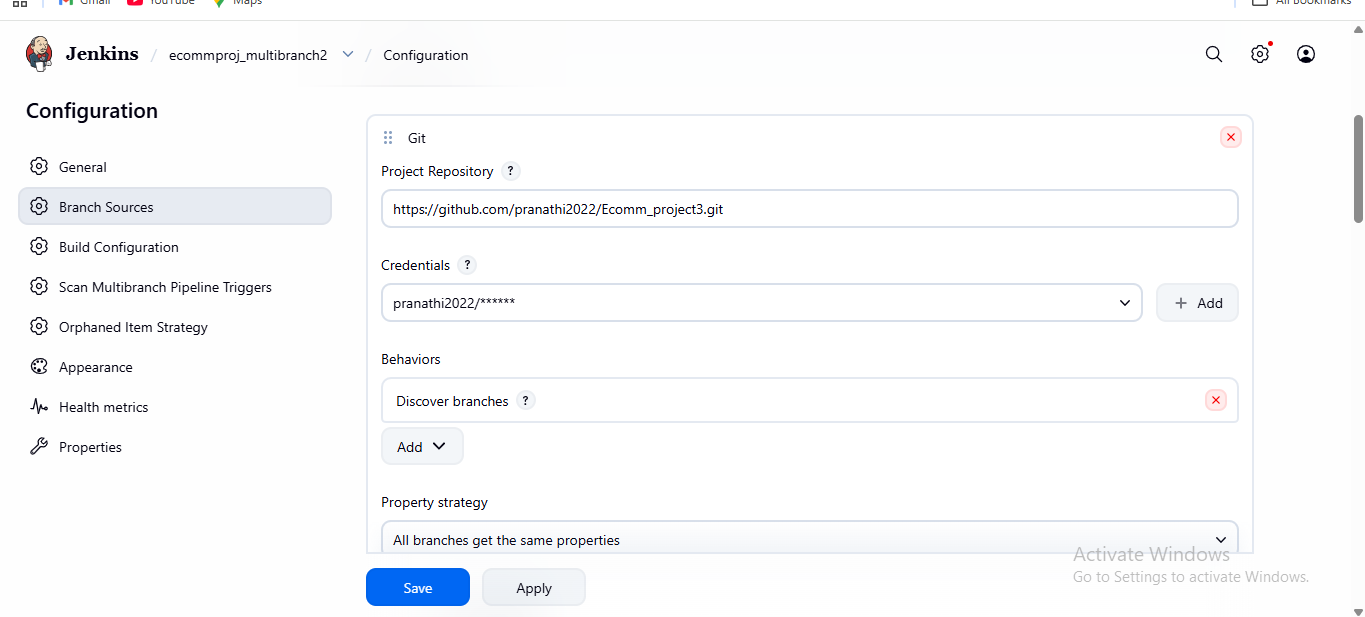
This enables separate CI/CD flows for development and production.

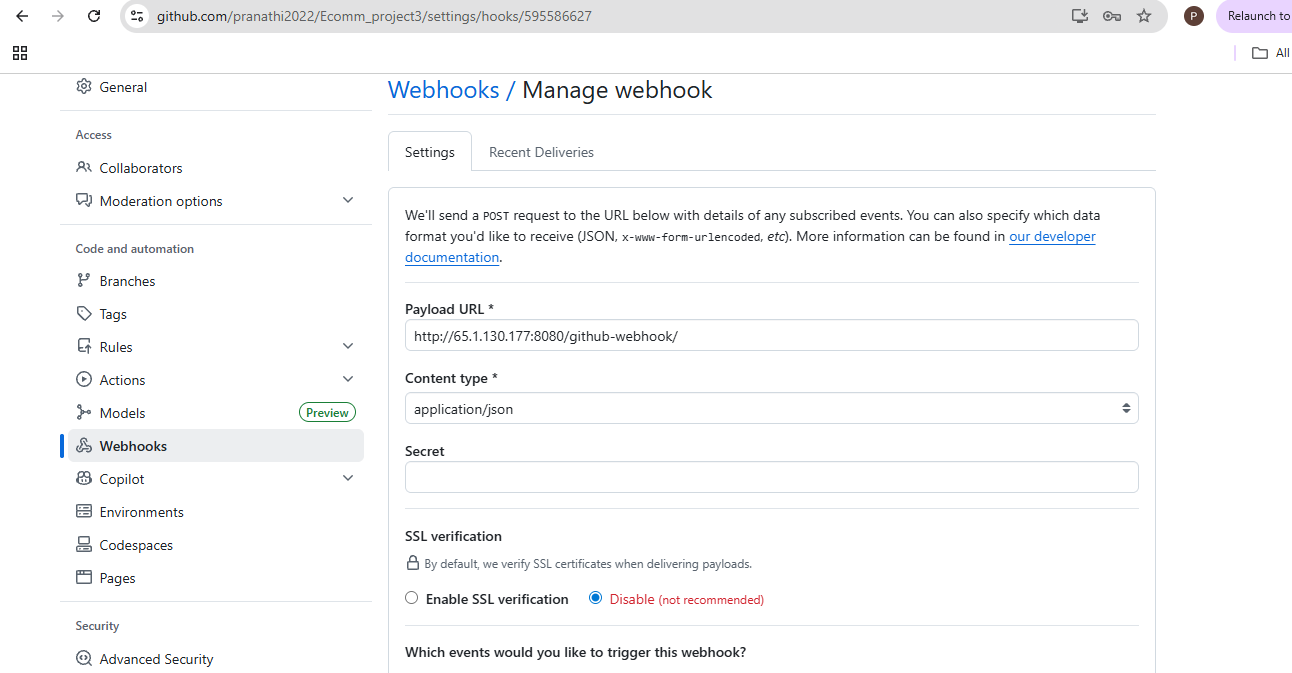
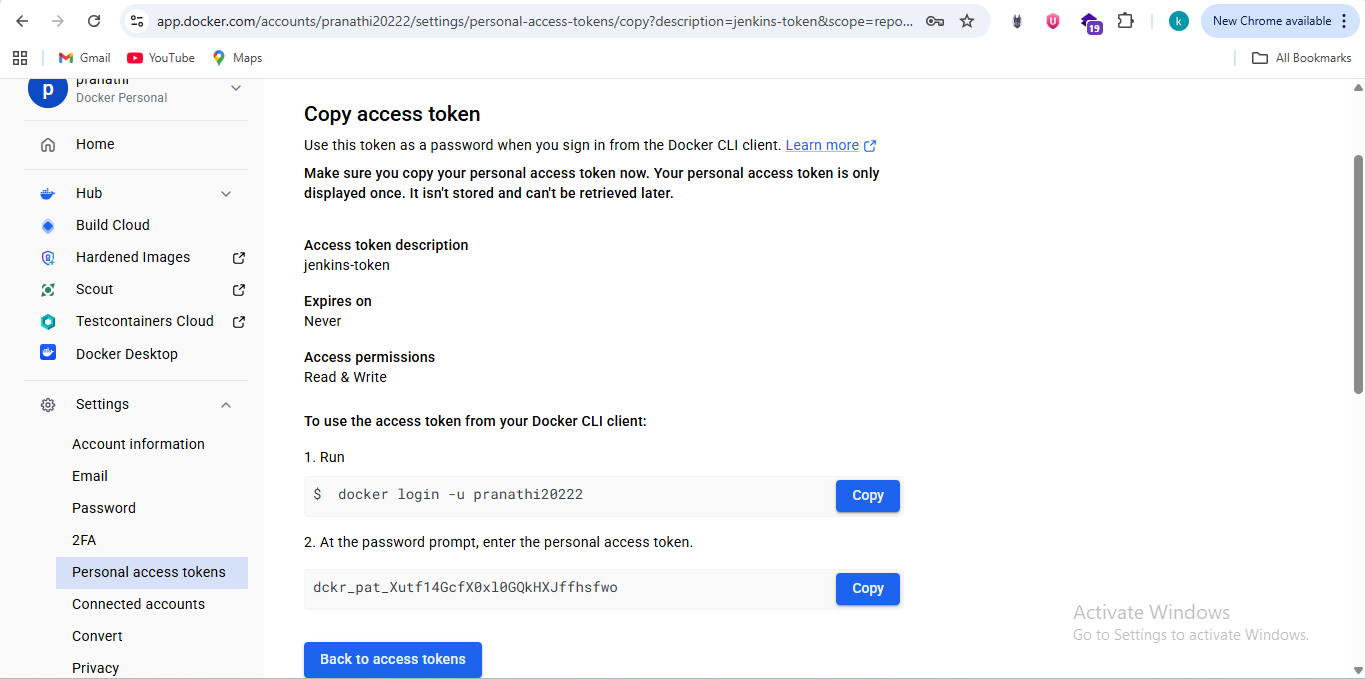
Jenkinsfile – Branch Based Build & Push Logic

The Jenkinsfile stages configured for CI/CD.

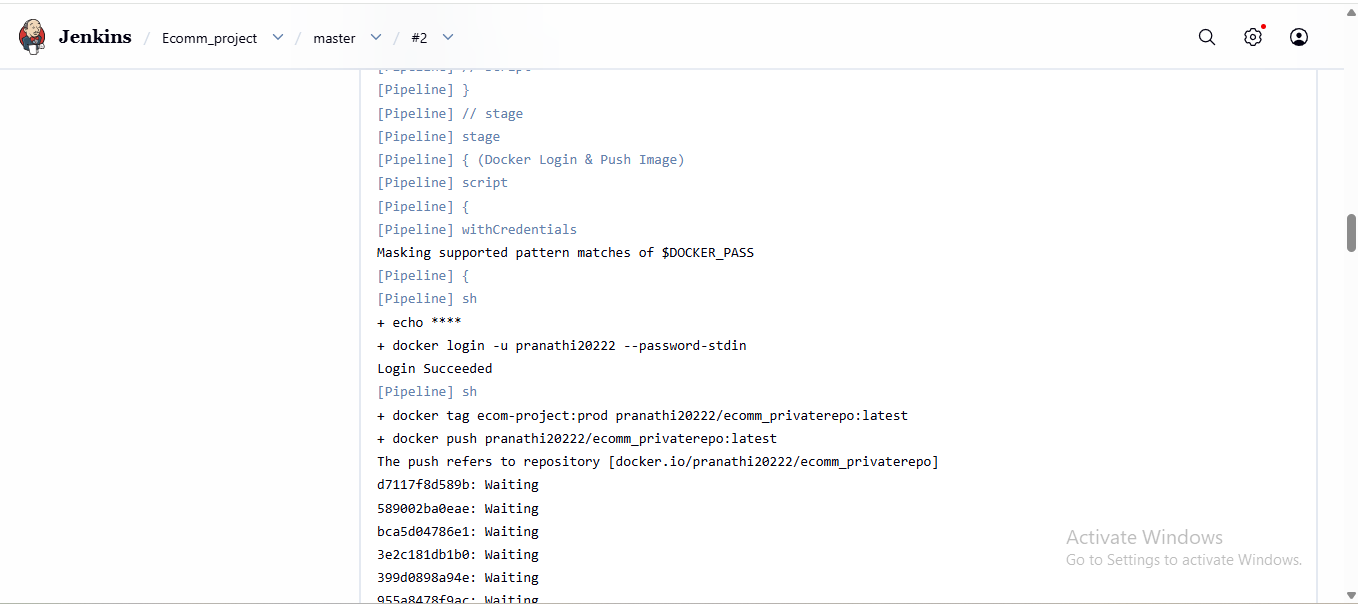
Deployment is triggered only from the master branch.

Images are pushed automatically by Jenkins after successful builds.

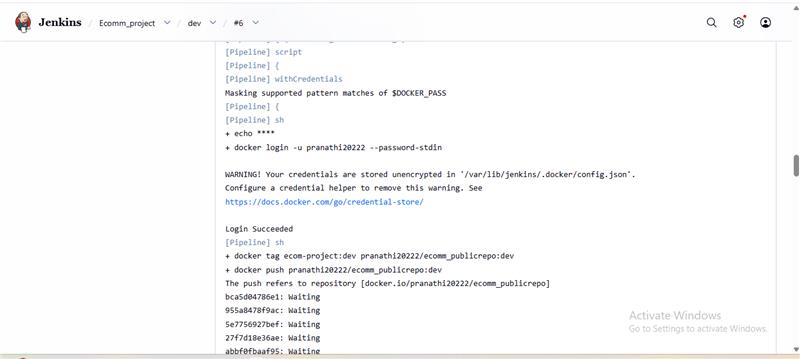




pushing docker image from master repo to the private docker hub repo



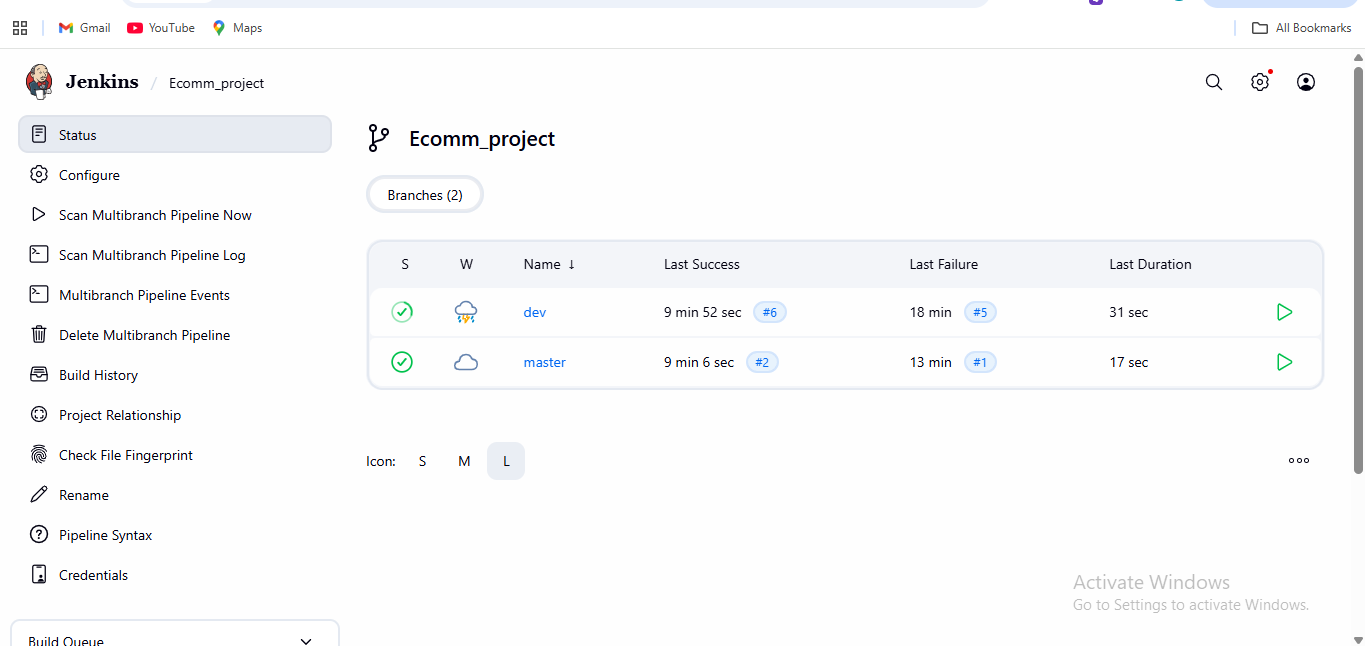
pushing the image from the dev repo to the docker hub public repo

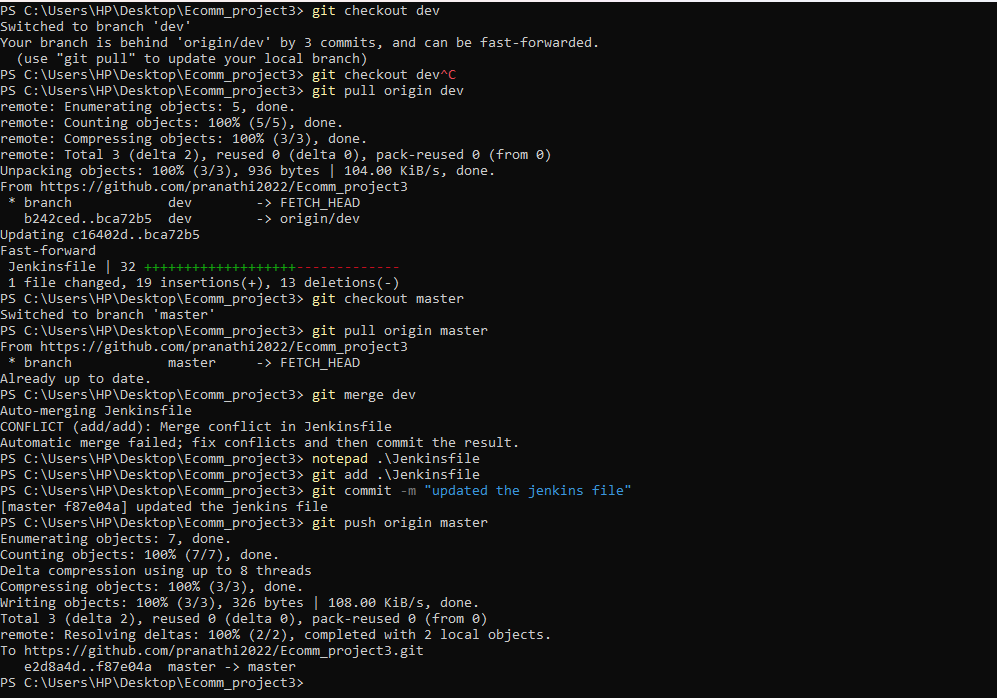


Branch-based logic implemented:

• dev branch → build image with dev tag → push to DockerHub public repo.

• master branch → build image with prod/latest tag → push to DockerHub private repo.





Git Workflow – dev to master Promotion

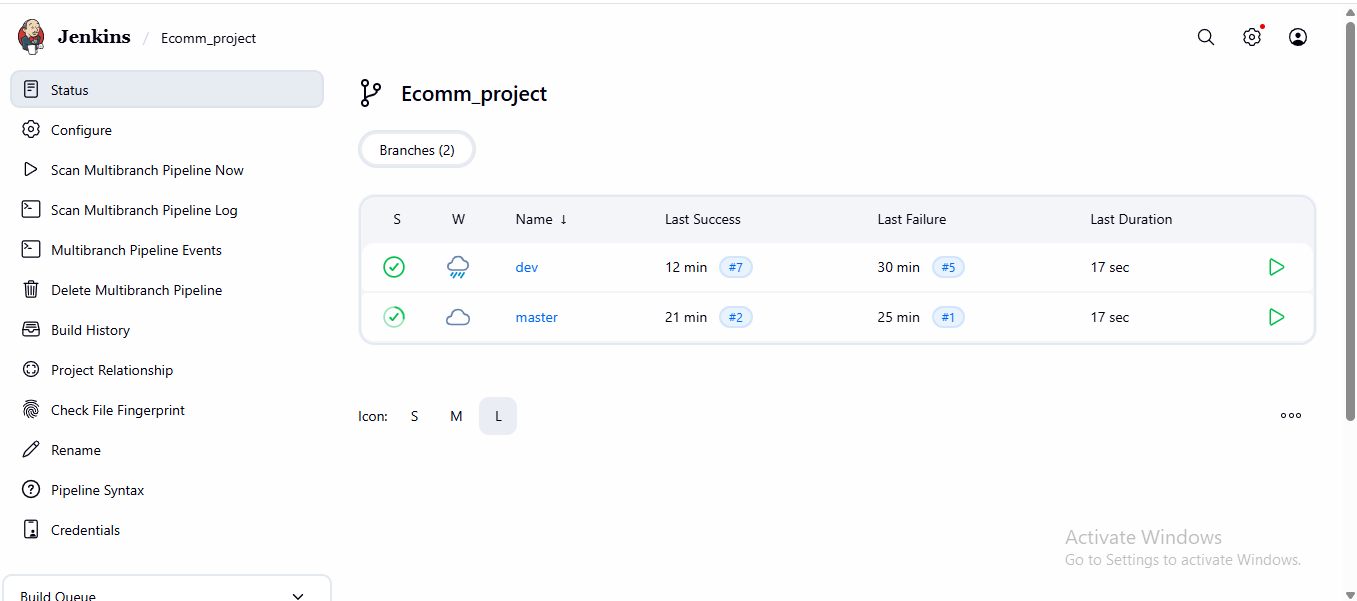
This screenshot confirms the Git workflow followed in the project:

1) Developer commits and pushes code to dev branch.

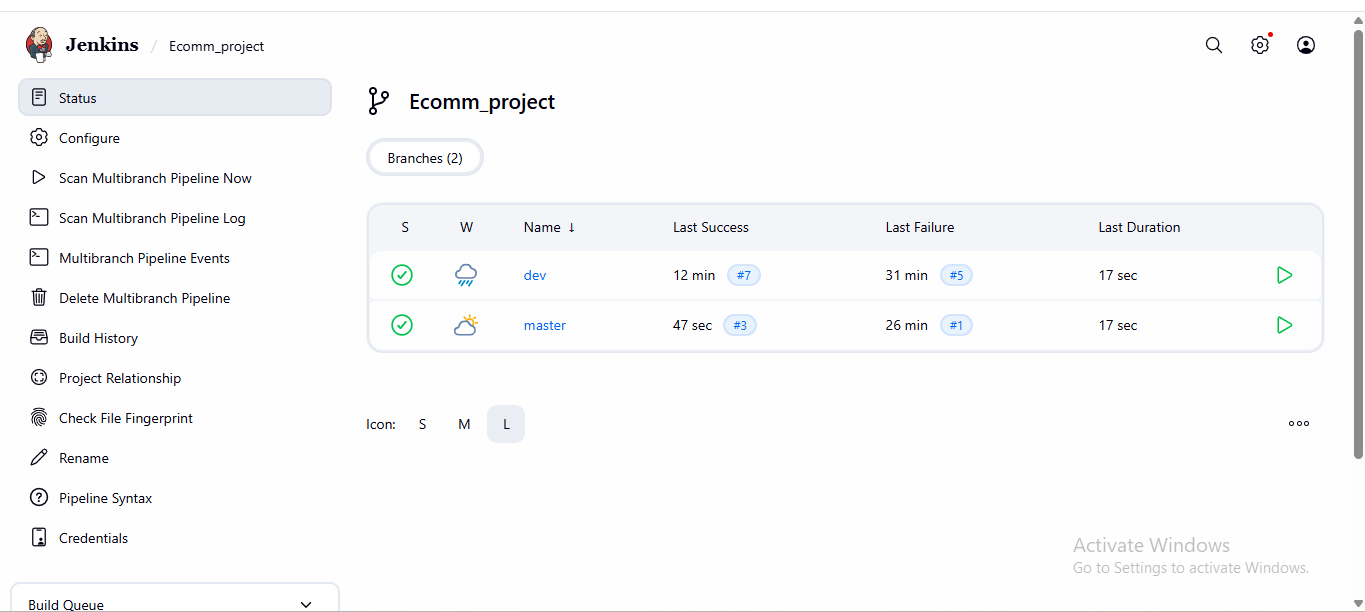
2) Jenkins dev pipeline runs and pushes image to DockerHub public repo.

3) After validation, dev branch is merged into master.

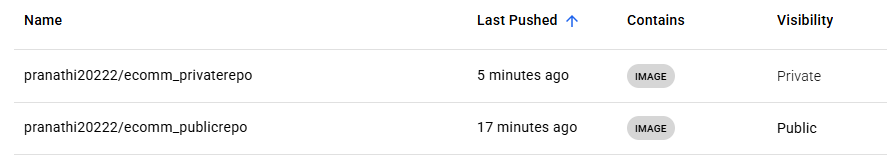
4) Jenkins master pipeline runs and pushes image to DockerHub private repo.

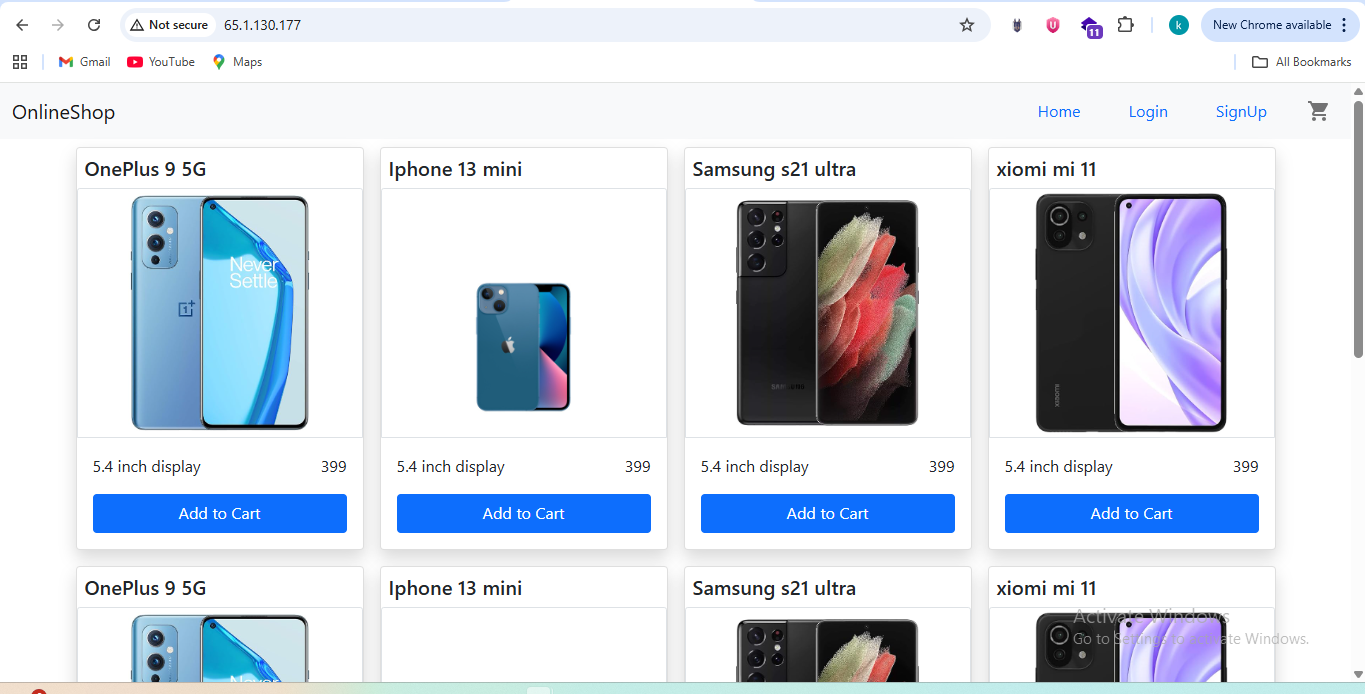


Both pipelines ran successfully.

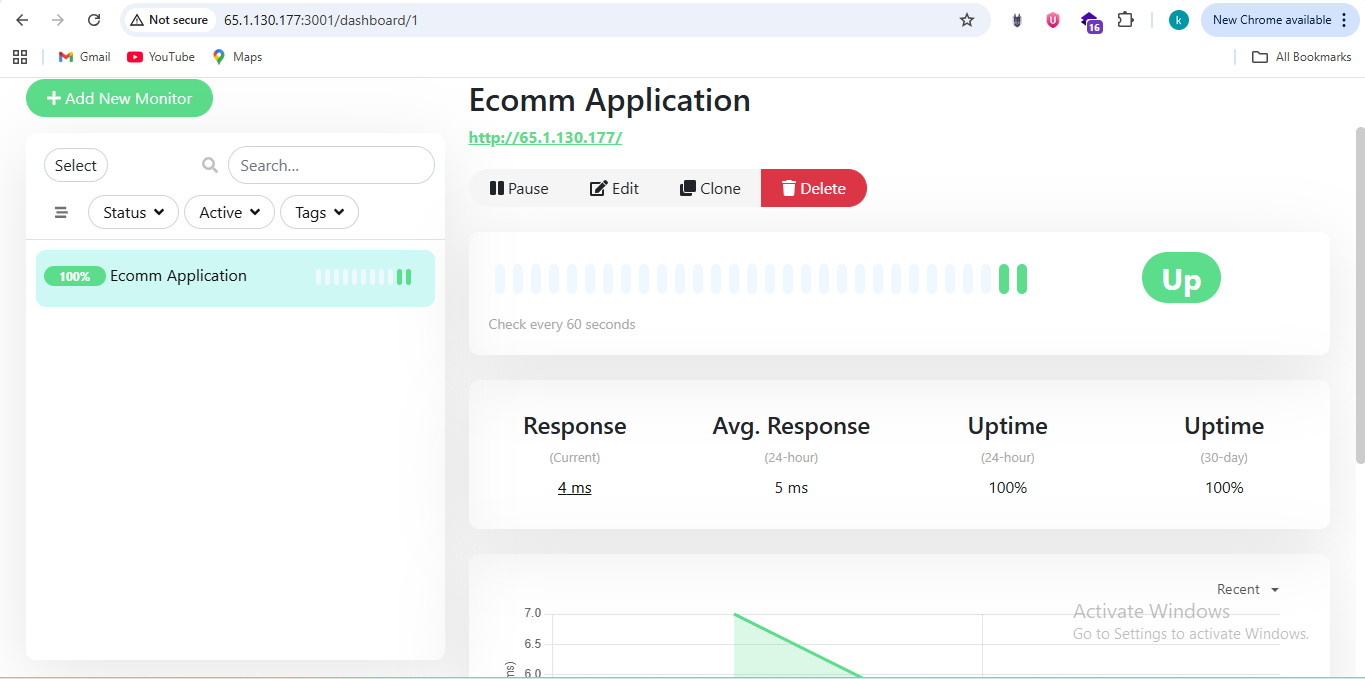
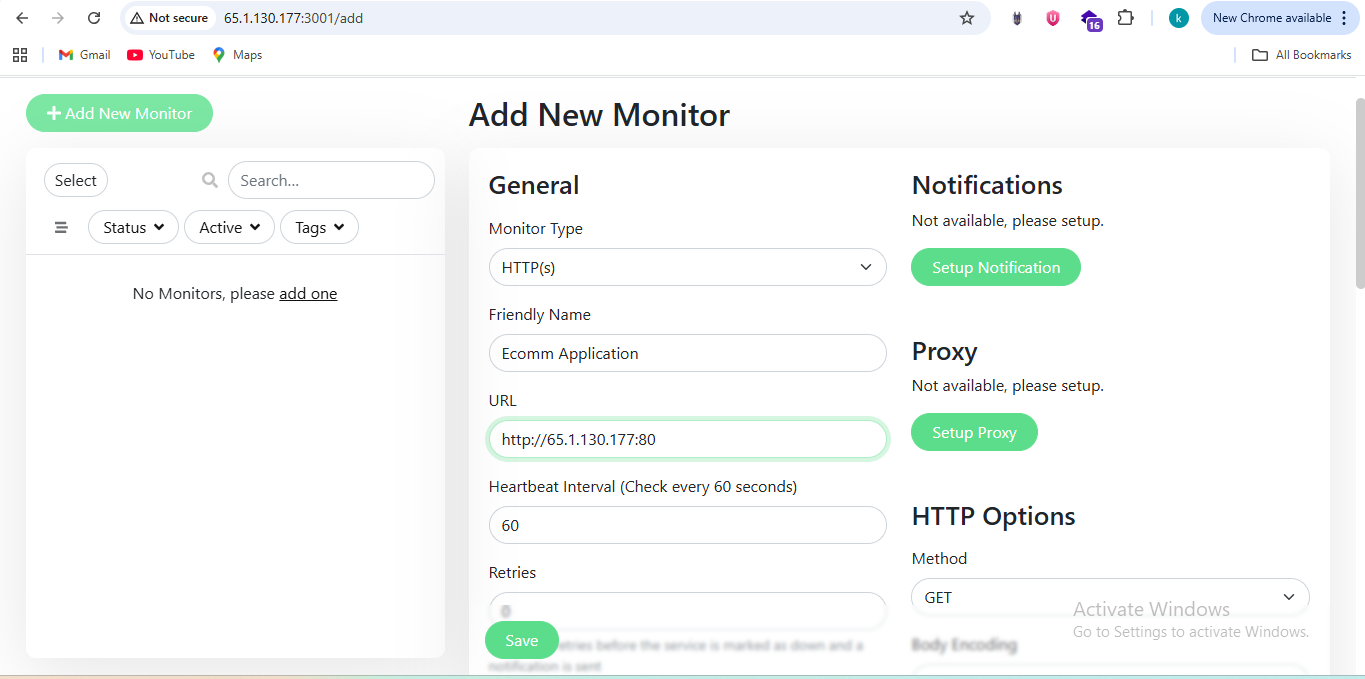
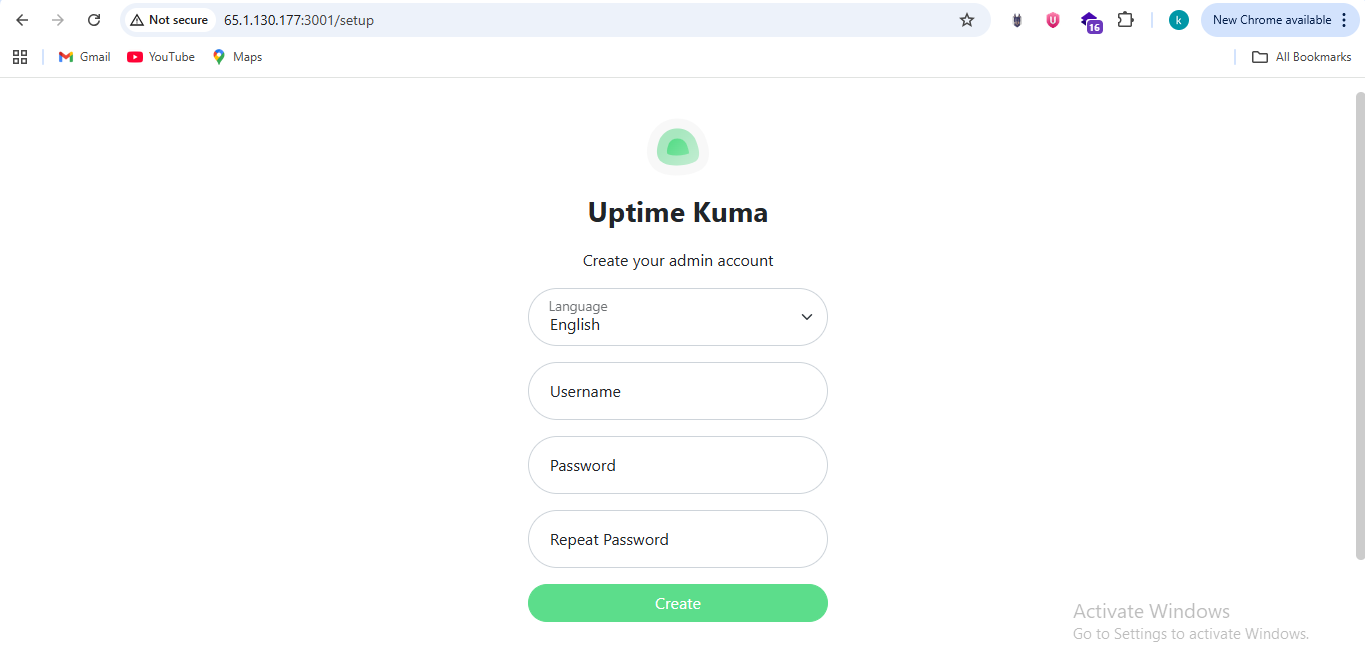
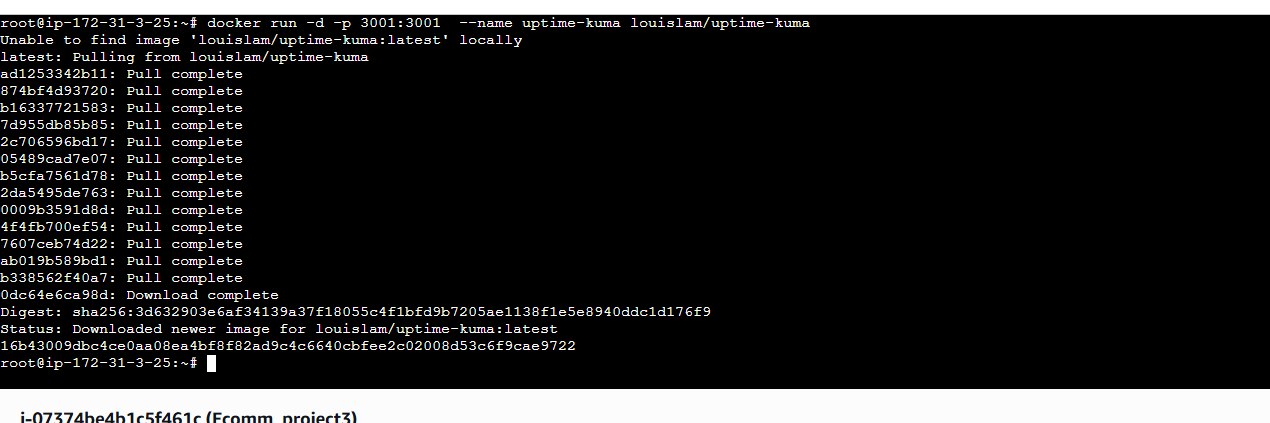


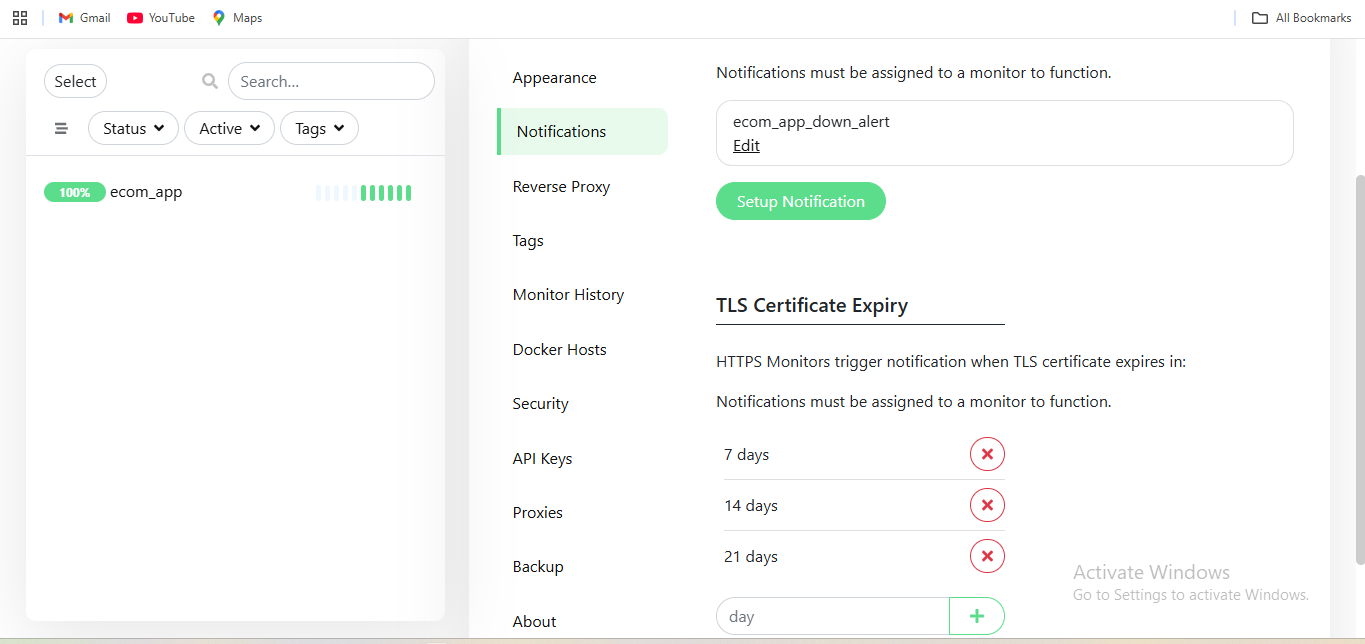
Docker hub repos





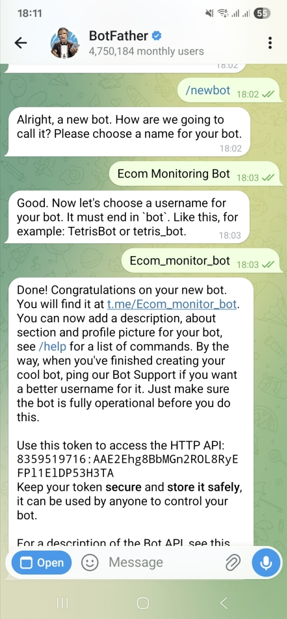
Monitoring using kuma



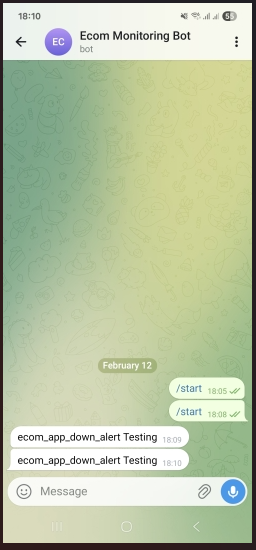


Telegram Bot – Chat ID Validation

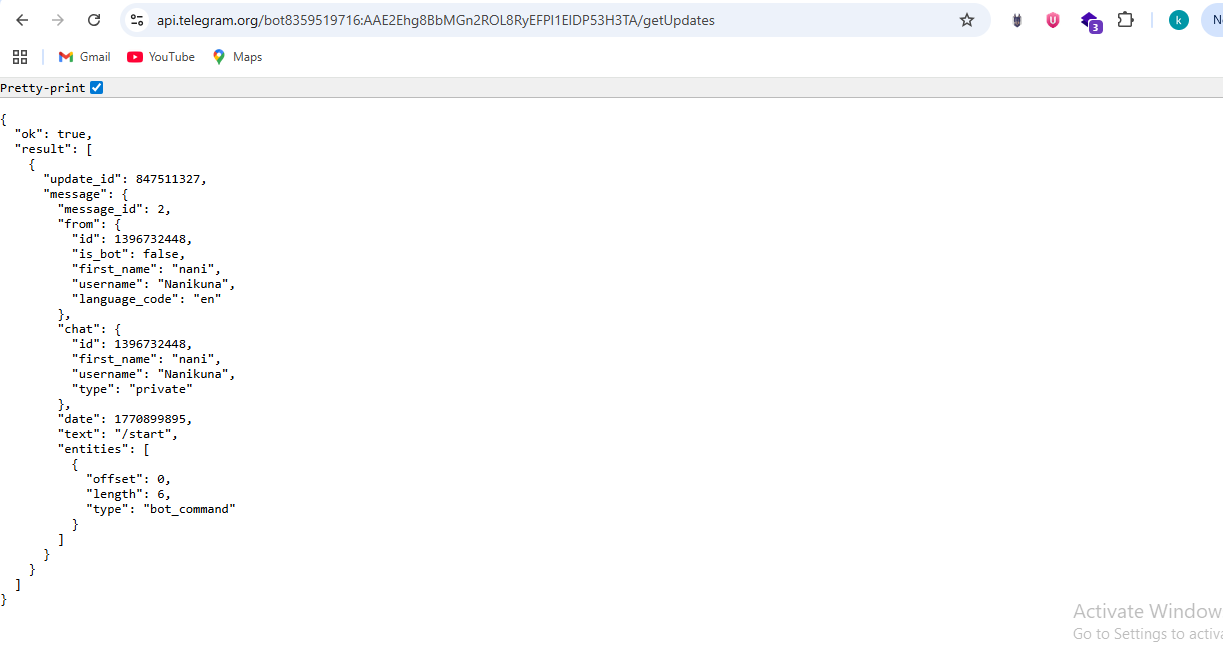
This screenshot shows testing of the Telegram Bot API to fetch updates and identify the chat ID.



8359519716:AAE2Ehg8BbMGn2ROL8RyEFPl1ElDP53H3TA



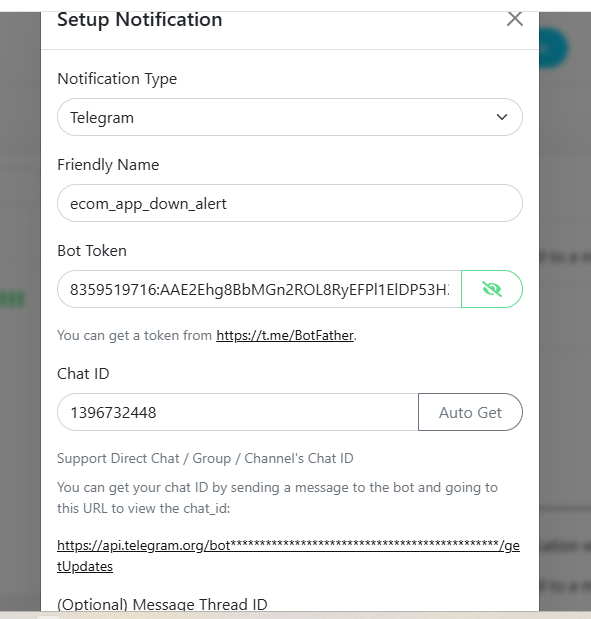
Endpoint - <https://api.telegram.org/bot8359519716:AAE2Ehg8BbMGn2ROL8RyEFPl1ElDP53H3TA/getUpdates>

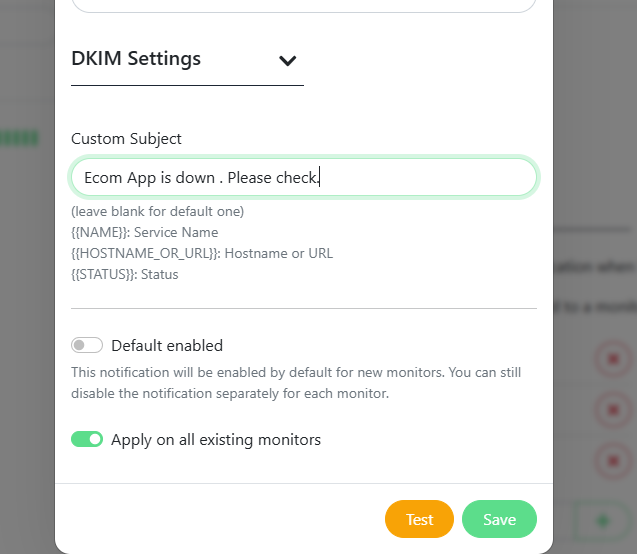


The chat\_id is taken from the response and used for sending automated notifications.

This enables CI/CD and monitoring alerts to be delivered to Telegram.

Uptime Kuma – Downtime and Recovery Test



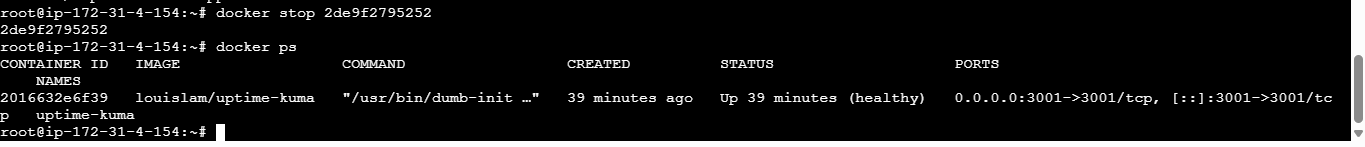


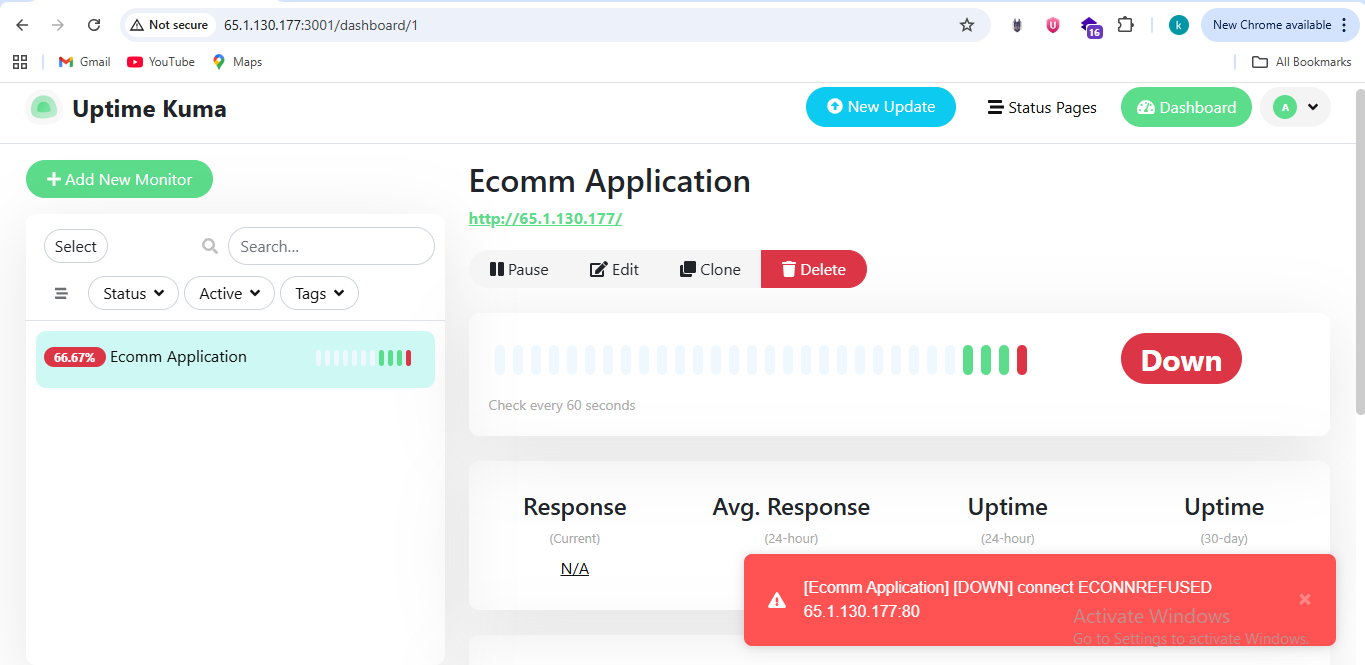
Test performed:

• Container stopped intentionally → Kuma detected downtime (red).

• Container started again → Kuma detected recovery (green).

This validates end-to-end monitoring and alerting for the deployed application.





This screenshot shows Uptime Kuma monitoring the application health.

