DevOps Build Project

React Application Deployment

Prepared By: Deepak

Submitted To: Guvi

Date: 04-09-2025

Table of Contents

1. Project Overview
2. Repository & Deployment Details
3. Setup Instructions
4. CI/CD Pipeline Explanation
5. Monitoring Setup
6. Deployment Steps with Screenshots
7. Conclusion

Project Overview

This project demonstrates the deployment of a React application in a production-ready environment using Docker, Jenkins, and AWS. The CI/CD pipeline builds, pushes, and deploys the application automatically based on Git branch updates.

Repository & Deployment Details

• GitHub Repo: https://github.com/Deepak-r-2001/devops-build

• Deployed Site URL: http://13.201.123.30/

• Docker Hub Dev Repo (Public): deepwhoo/devops-build:dev

• Docker Hub Prod Repo (Private): deepwhoo/devops-build:prod

Setup Instructions

1. Clone the Repository

```
git clone -b dev https://github.com/Deepak-r-2001/devops-build.git
cd devops-build
```

2. Dockerize Application

- Create Dockerfile to build the React app image.
- Create docker-compose.yml to run the app on port 80.

3. Bash Scripts

- build.sh → Builds Docker images
- deploy.sh → Deploys images to the server

4. Push Code to GitHub

```
git add .
git commit -m 'Initial commit'
git push origin dev
```

5. Jenkins CI/CD Pipeline

- Jenkins monitors dev and master branches.
- On push to dev → Builds image → Pushes to Dev repo.
- On merge to master → Builds image → Pushes to Prod repo.
- Jenkins automatically deploys the app.

6. AWS EC2 Setup

- Launch a t2.micro instance.
- Configure Security Groups (HTTP, SSH).
- Deploy the app using Docker and deploy.sh.

7. Monitoring

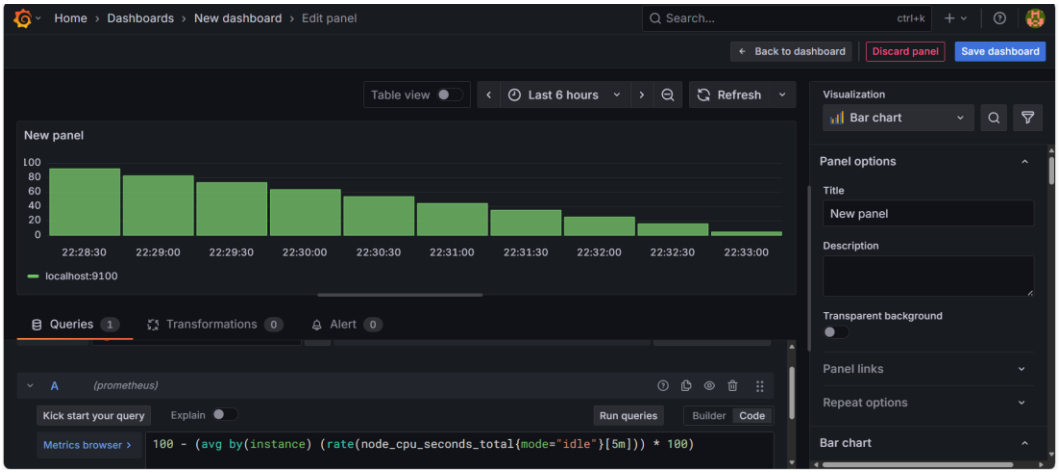
- Prometheus & Grafana monitor app health.
- Alerts notify if the app goes down.

Pipeline Explanation

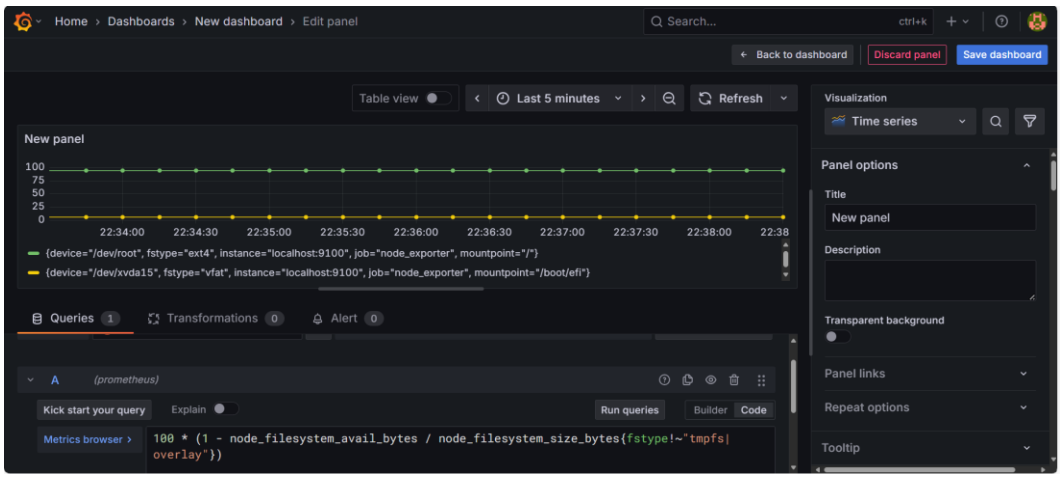
1. Development Workflow:
   - Push code to dev → Jenkins builds image → Pushes to Dev repo → Deploys to dev server.
2. Production Workflow:
   - Merge dev → master → Jenkins builds image → Pushes to Prod repo → Deploys to prod server.
3. Monitoring:
   - Prometheus collects metrics.
   - Grafana displays health and resource usage.

Deployment Steps with Screenshots

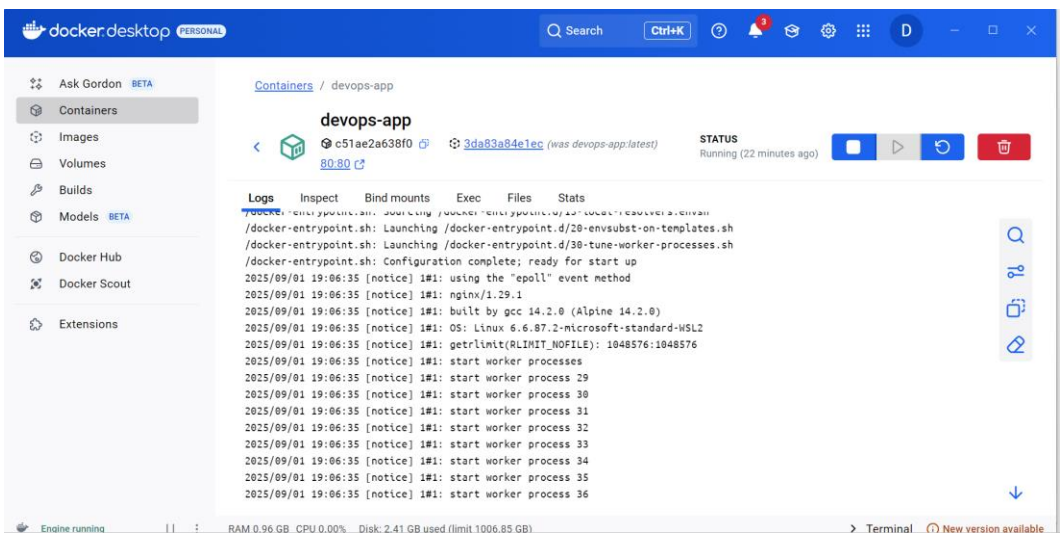
Step 1: CPU usage monitoring in Grafana



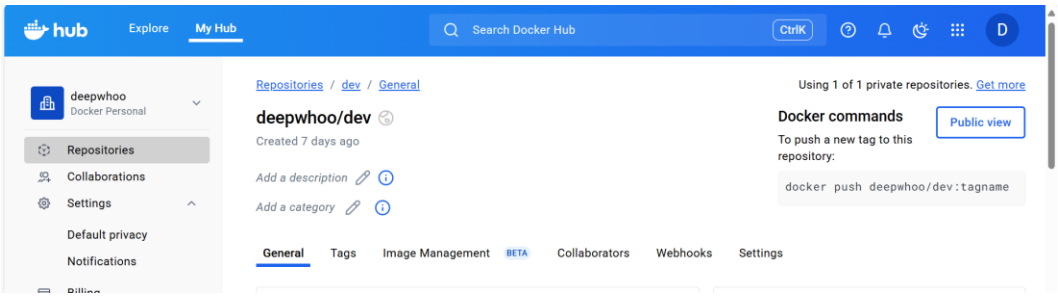
Step 2: Disk usage monitoring in Grafana



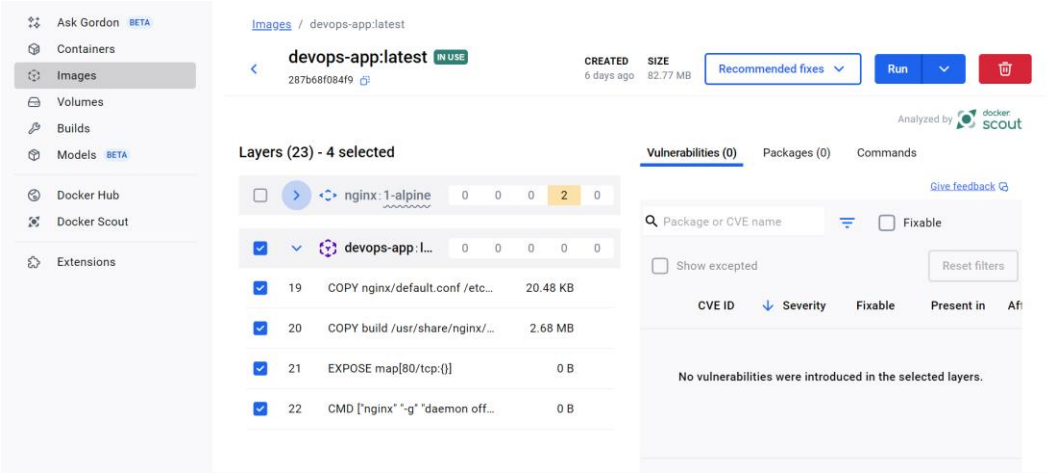
Step 3: Docker container running successfully



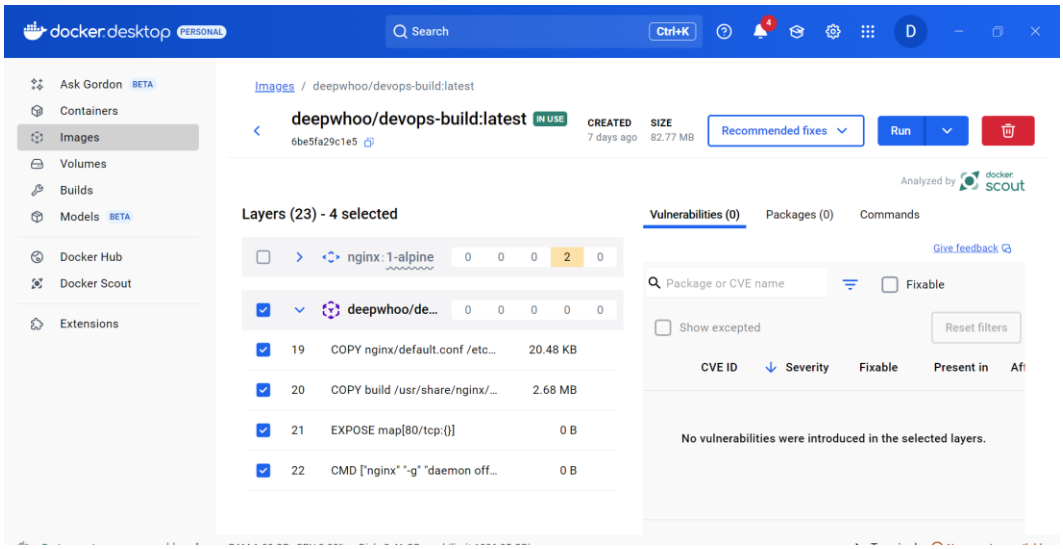
Step 4: Docker development repository on DockerHub



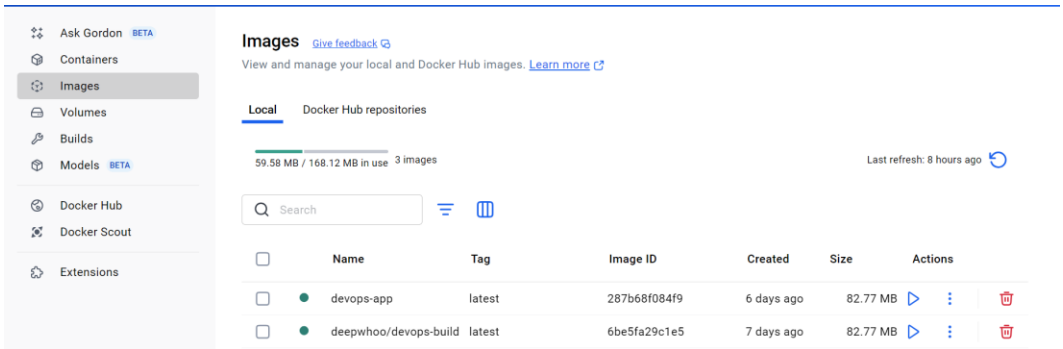
Step 5: Docker image build - Step 1



Step 6: Docker image build - Step 2



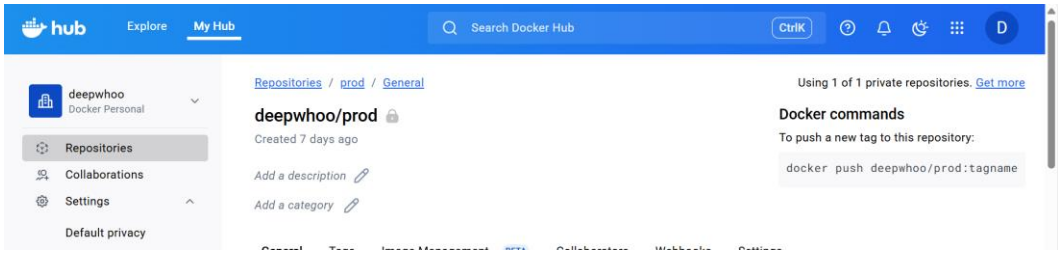
Step 7: Docker image build completed



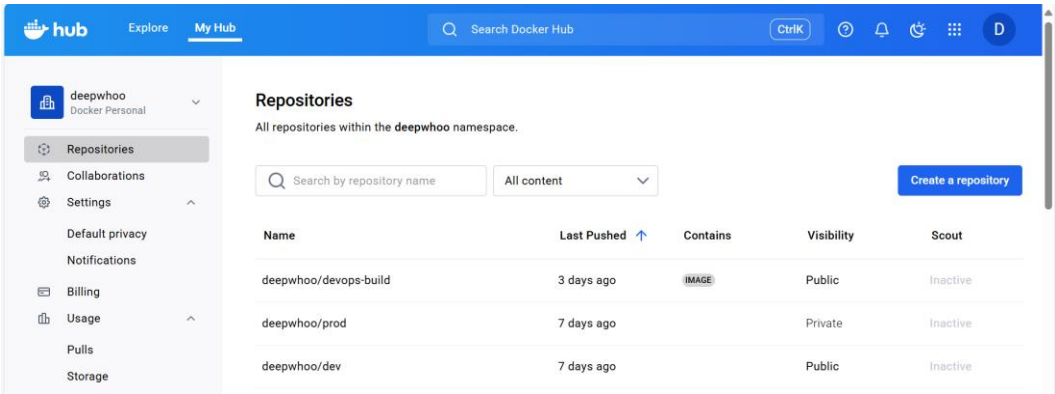
Step 8: List of all Docker images

```
288206953113.dkr.ecr.ap-south-1.amazonaws.com/brain-tasks-app   latest    00e12cd49361   2 weeks ago   74MB

EAD+dr222@GB-5CD2288DXL MINGW64 ~/Documents/Guvi/DevOps/devops-build (main)
$ docker image ls
REPOSITORY                                                       TAG       IMAGE ID       CREATED       SIZE
devops-app                                                       latest    287b68f084f9   6 days ago    82.8MB
deepwhoo/devops-build                                            latest    6be5fa29c1e5   7 days ago    82.8MB
288206953113.dkr.ecr.ap-south-1.amazonaws.com/brain-tasks-app   latest    00e12cd49361   2 weeks ago   74MB

EAD+dr222@GB-5CD2288DXL MINGW64 ~/Documents/Guvi/DevOps/devops-build (main)
$ docker images
REPOSITORY                                                       TAG       IMAGE ID       CREATED       SIZE
devops-app                                                       latest    287b68f084f9   6 days ago    82.8MB
deepwhoo/devops-build                                            latest    6be5fa29c1e5   7 days ago    82.8MB
288206953113.dkr.ecr.ap-south-1.amazonaws.com/brain-tasks-app   latest    00e12cd49361   2 weeks ago   74MB
```

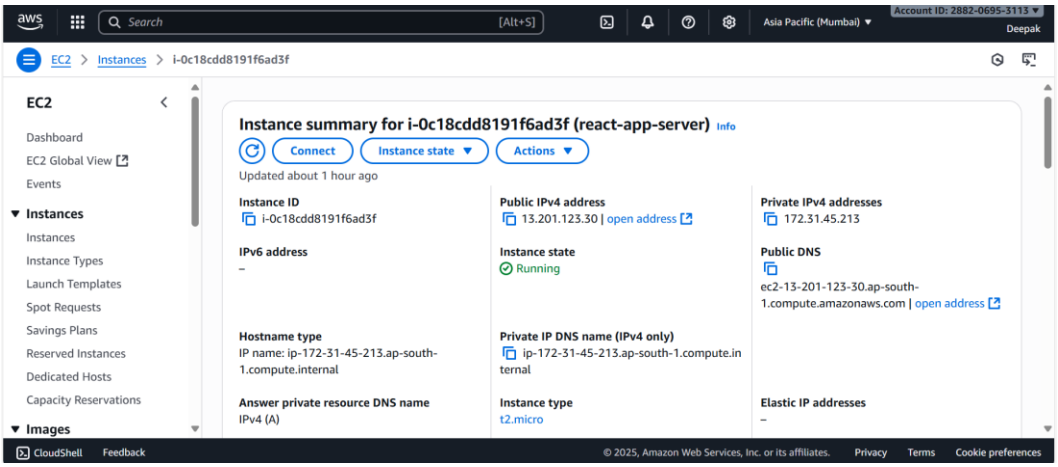
Step 9: Docker production repository



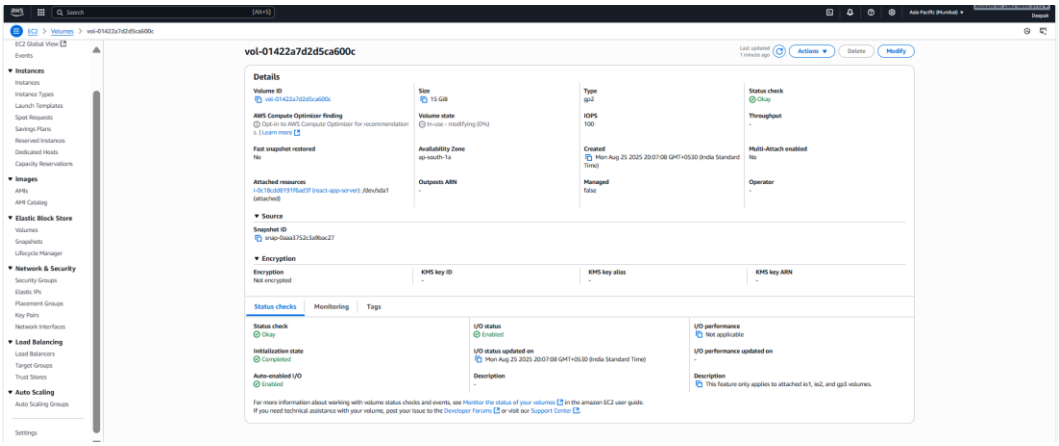
Step 10: Docker repositories overview



Step 11: EC2 instance security groups configuration

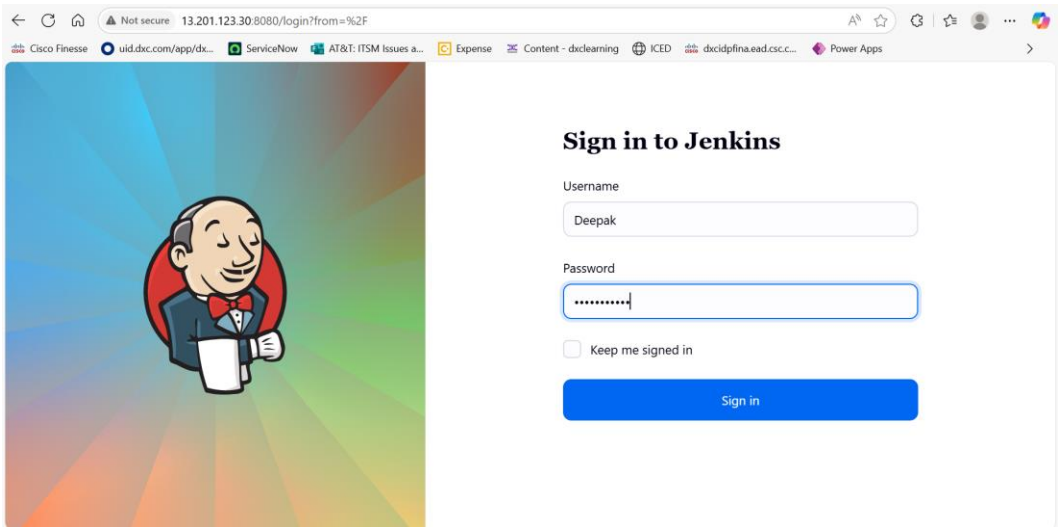


Step 12: Jenkins build pipeline triggered

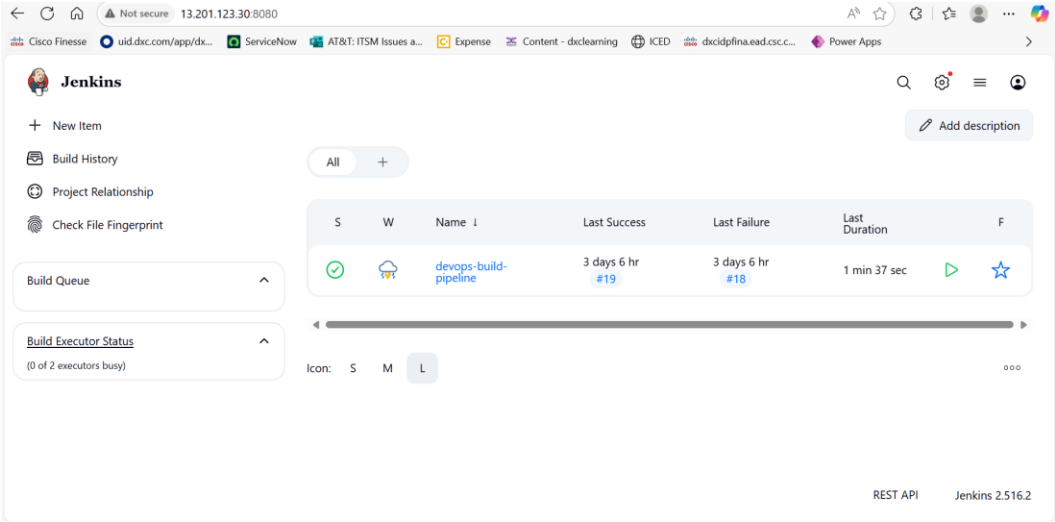


Step 13: Jenkins build pipeline successful

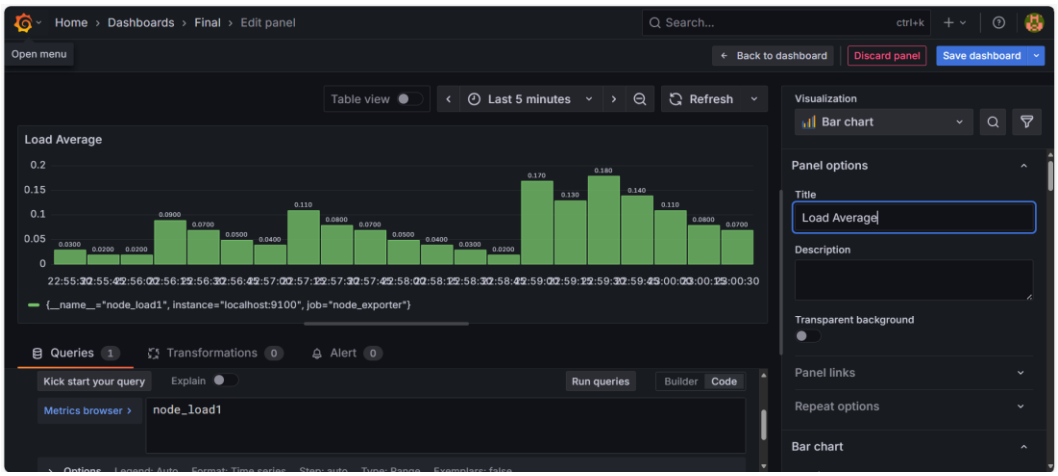
Step 14: Jenkins dashboard view



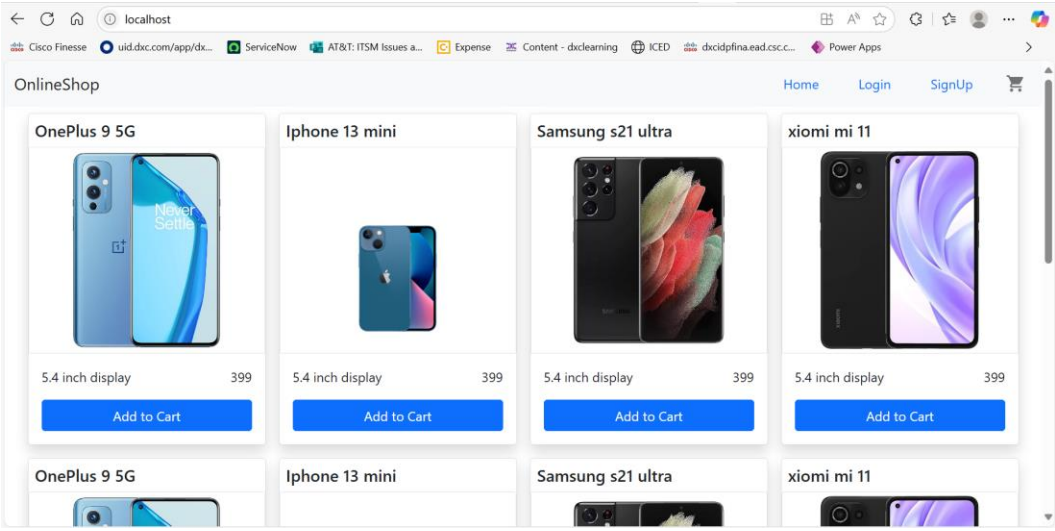
Step 15: Prometheus metrics dashboard



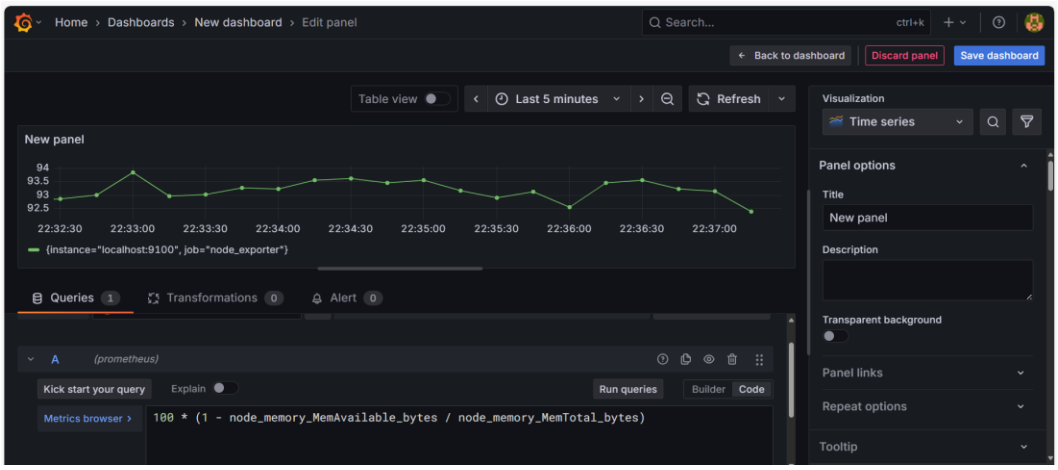
Step 16: Prometheus scraping targets



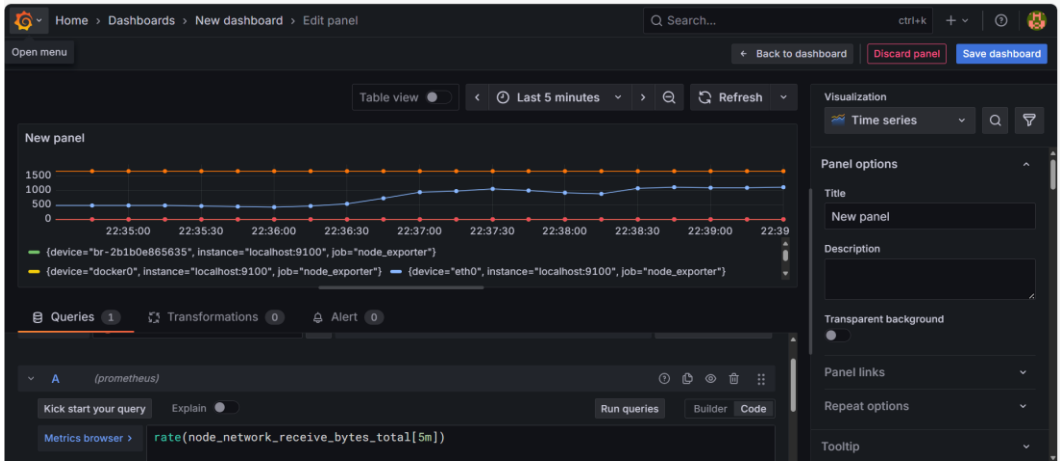
Step 17: Prometheus active targets list



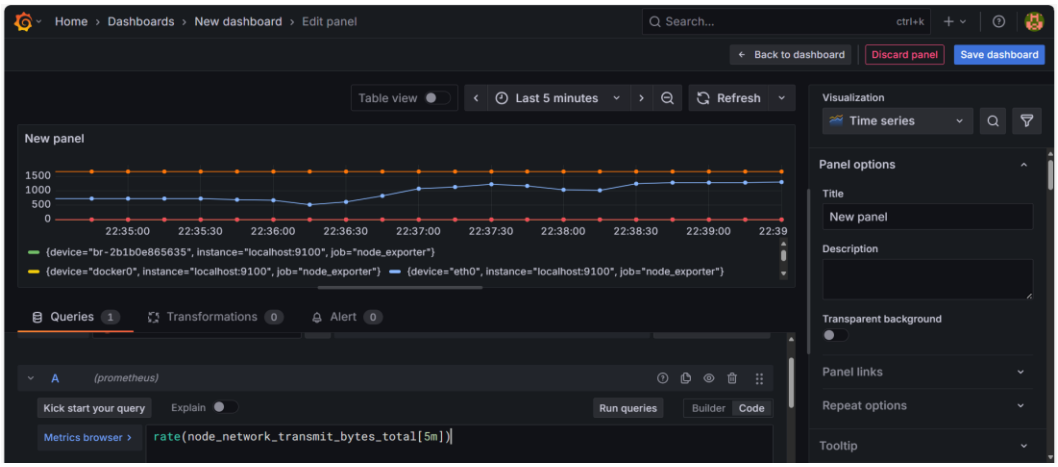
Step 18: Prometheus graph metrics



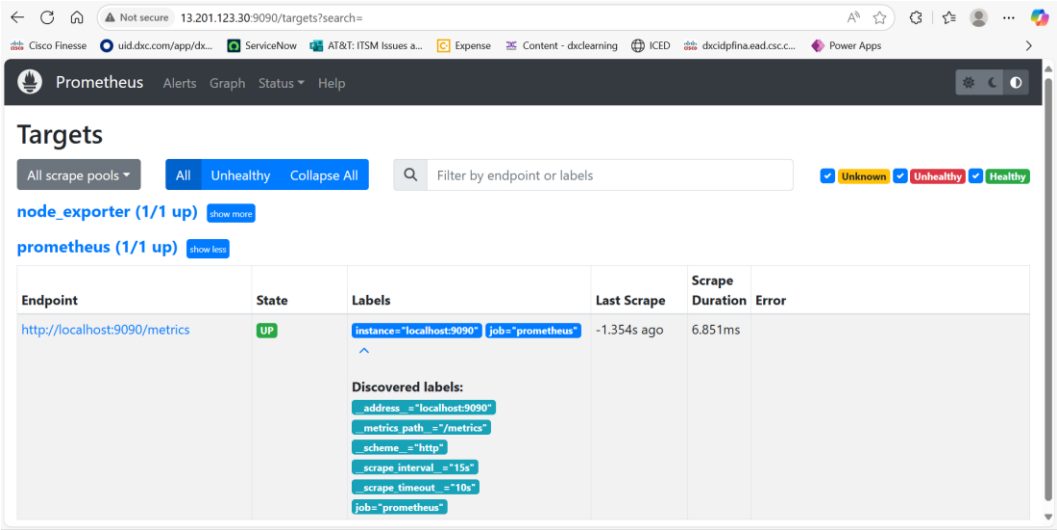
Step 19: Grafana dashboard overview



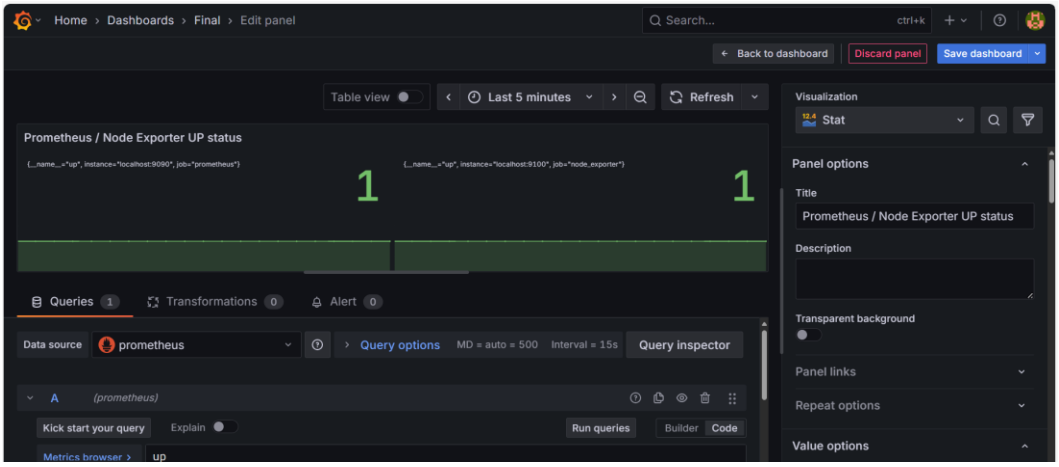
Step 20: Grafana integration with Prometheus



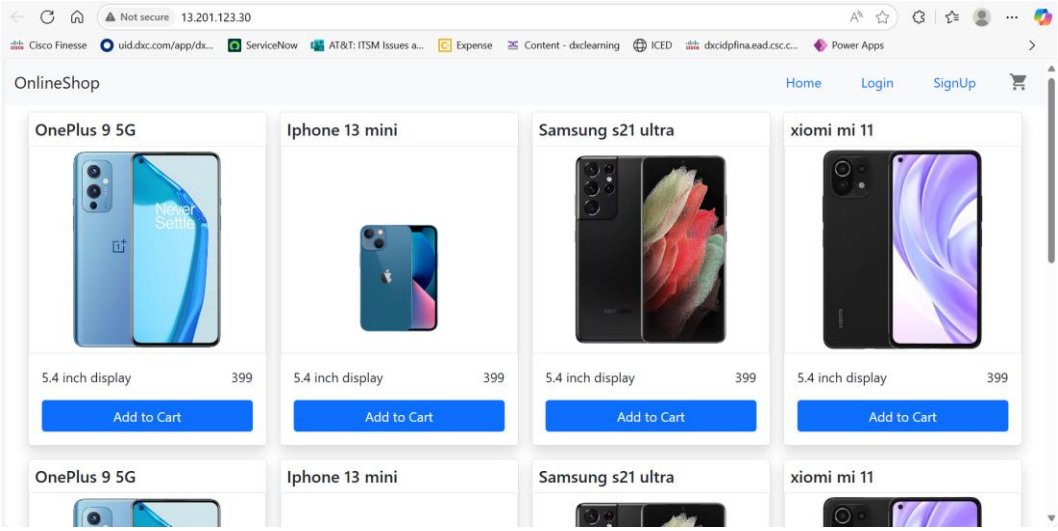
Step 21: Grafana CPU usage graph



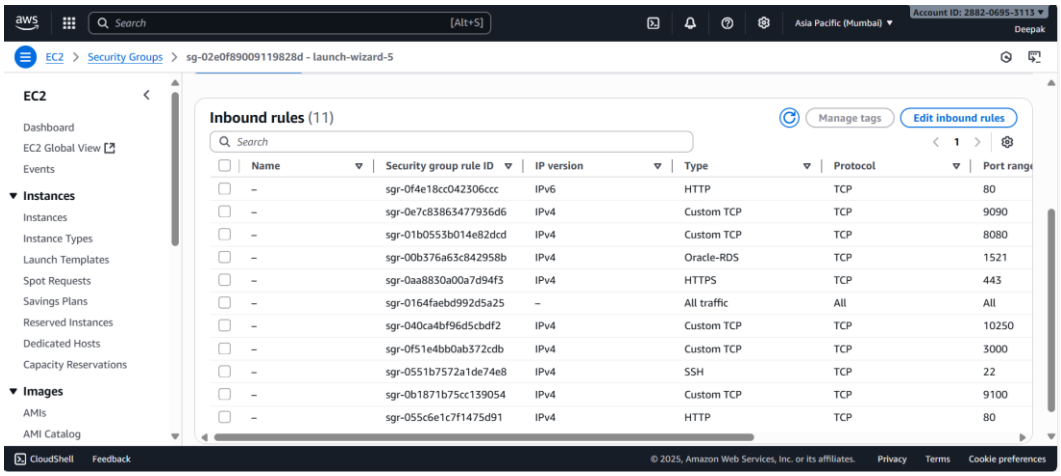
Step 22: Grafana memory usage graph



Step 23: Grafana network monitoring



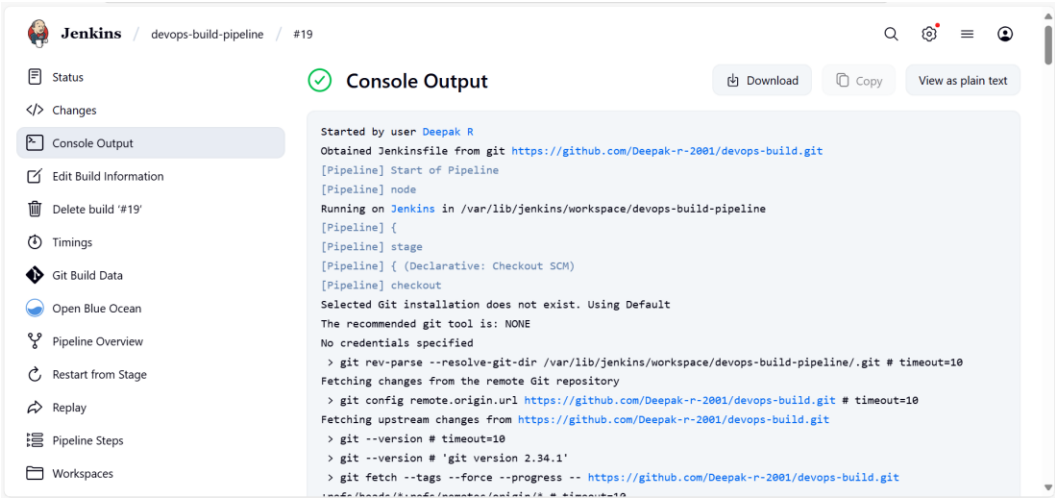
Step 24: Grafana container monitoring



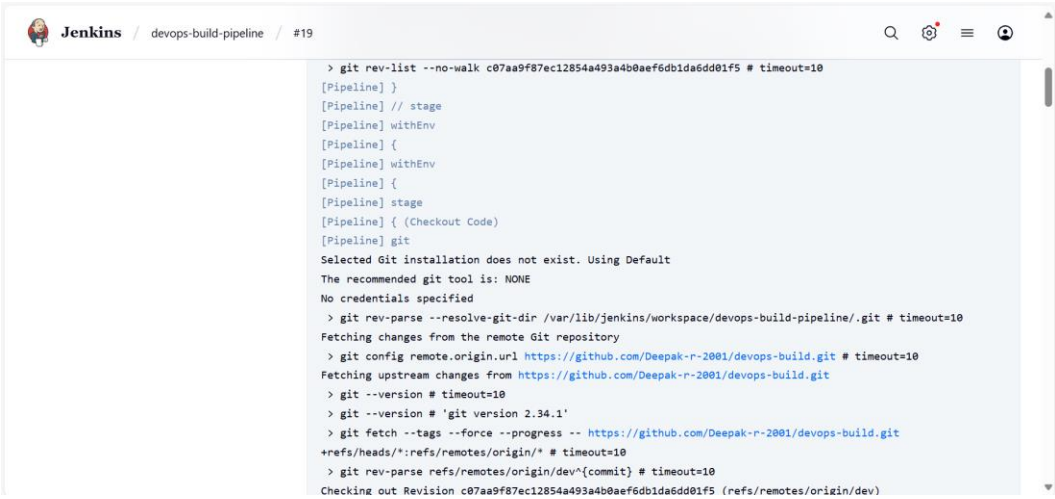
Step 25: React app successfully deployed

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

EAD+dr222@GB-5CD2288DXL MINGW64 ~/Documents/Guvi/DevOps/devops-build (main)
$ cat build.sh
#!/usr/bin/env bash
set -euo pipefail

# Usage: ./build.sh [branch]
BRANCH=${1:-$(git rev-parse --abbrev-ref HEAD)}
DOCKERHUB_USER=${DOCKERHUB_USER:-deepwhoo}

IMAGE_DEV="${deepwhoo}/dev"
IMAGE_PROD="${deepwhoo}/prod"

COMMIT=$(git rev-parse --short HEAD)
TAG="$(date +%Y%m%d-%H%M%S)-${COMMIT}"

# Pick dev vs prod repo based on branch
if [[ "$BRANCH" == "master" || "$BRANCH" == "main" ]]; then
  IMAGE="$IMAGE_PROD"
else
  IMAGE="$IMAGE_DEV"
fi

echo ">> Building image: ${IMAGE}:${TAG} (and :latest)"

docker build \
  --build-arg VITE_BASE_URL="${VITE_BASE_URL:-}" \
  -t "${IMAGE}:${TAG}" \
  -t "${IMAGE}:latest" \
  .

echo ">> Done. Built: ${IMAGE}:${TAG}"

EAD+dr222@GB-5CD2288DXL MINGW64 ~/Documents/Guvi/DevOps/devops-build (main)
$ cat deploy.sh
#!/usr/bin/env bash
set -euo pipefail

# Update these variables
SERVER="${SERVER:-ubuntu@13.201.123.30}"  # Your EC2 IP
SSH_KEY="${SSH_KEY:-/c/Users/dr222/Downloads/ssh.pem.3.pem}"  # Path to your PEM file
REMOTE_DIR="${REMOTE_DIR:-/opt/react-app}"
APP_IMAGE="${APP_IMAGE:-docker.io/deepwhoo/prod:latest}"  # Your DockerHub image
DOCKERHUB_USER="${DOCKERHUB_USER:-deepwhoo}"
DOCKERHUB_PASS="${DOCKERHUB_PASS:-Deepak@8217}"  # Optional if public repo

SSH_OPTS="-o StrictHostKeyChecking=no"
if [[ -n "$SSH_KEY" ]]; then
  SSH_OPTS="$SSH_OPTS -i $SSH_KEY"
fi

echo ">> Preparing remote directory on $SERVER"
ssh $SSH_OPTS "$SERVER" "sudo mkdir -p $REMOTE_DIR && sudo chown \$USER:\$USER $REMOTE_DIR"

echo ">> Uploading docker-compose.yml"
scp $SSH_OPTS docker-compose.yml "$SERVER":"$REMOTE_DIR"/
```

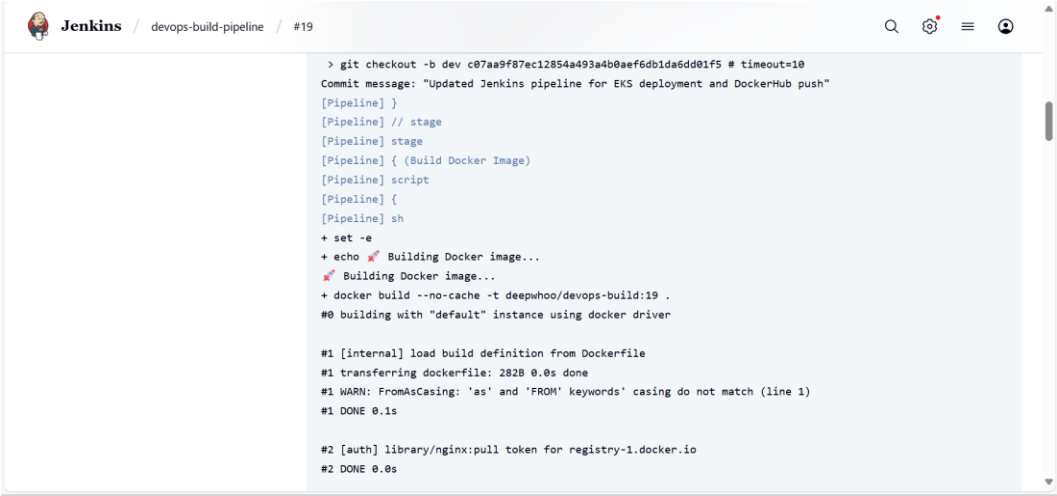
Step 26: React app running on port 80



Step 27: Docker Compose configuration file

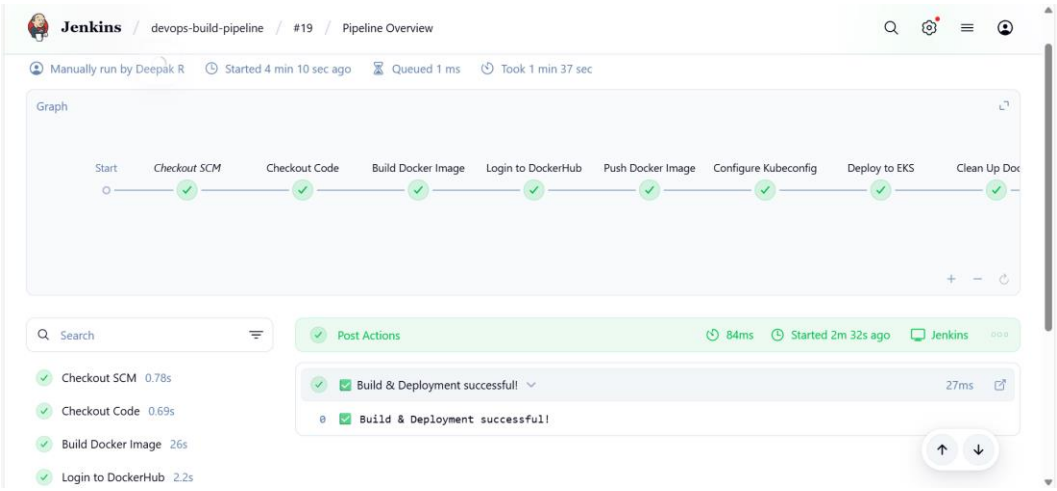


Step 28: Dockerfile for React application

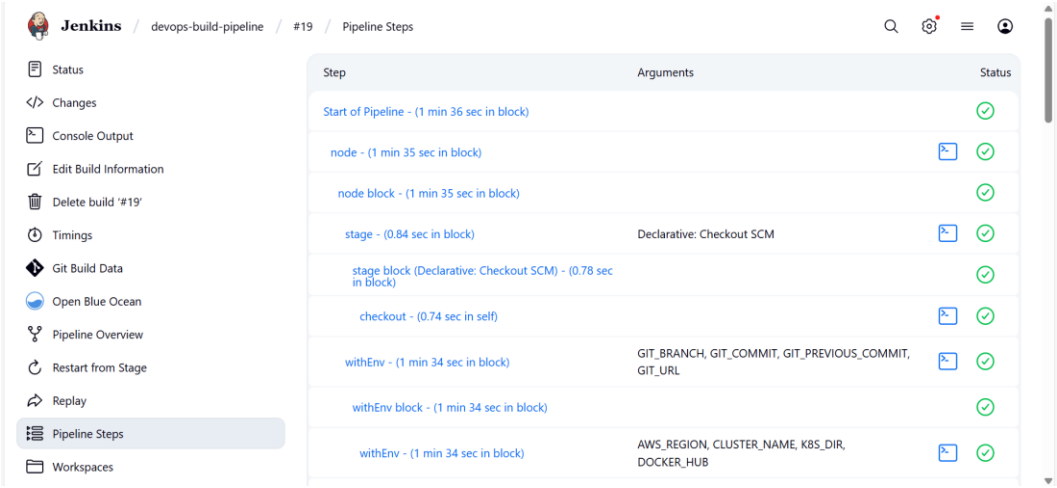


Step 29: Jenkinsfile for CI/CD pipeline

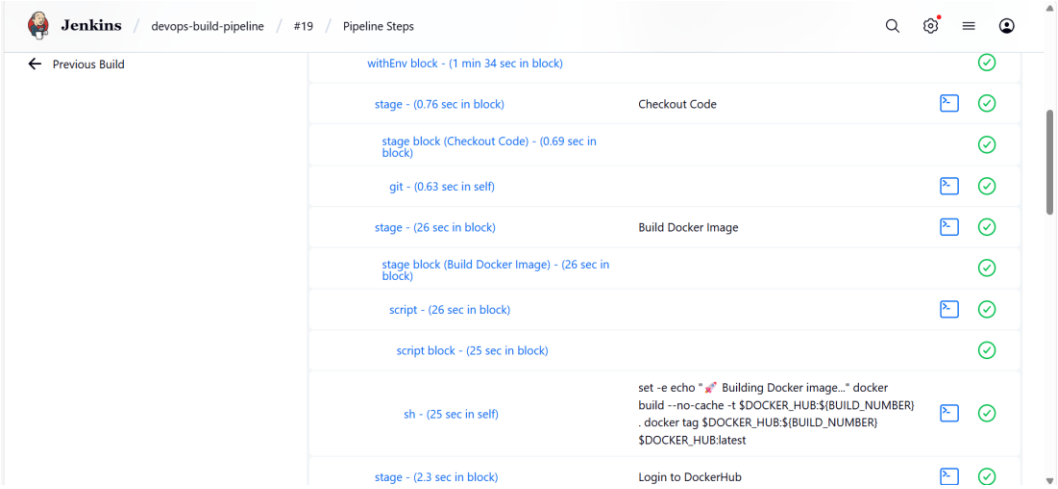
Step 30: Prometheus configuration file



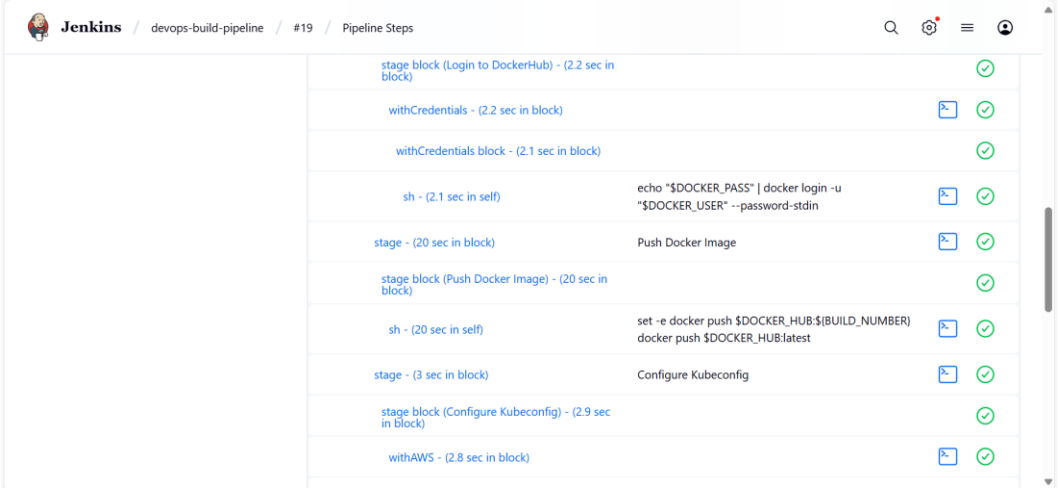
Step 31: Grafana dashboard configuration



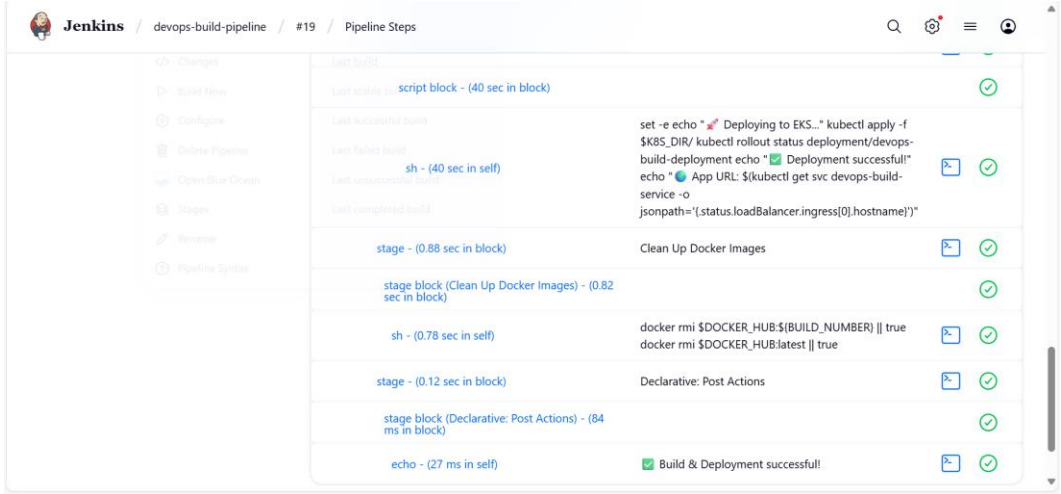
Step 32: AWS EC2 instance overview



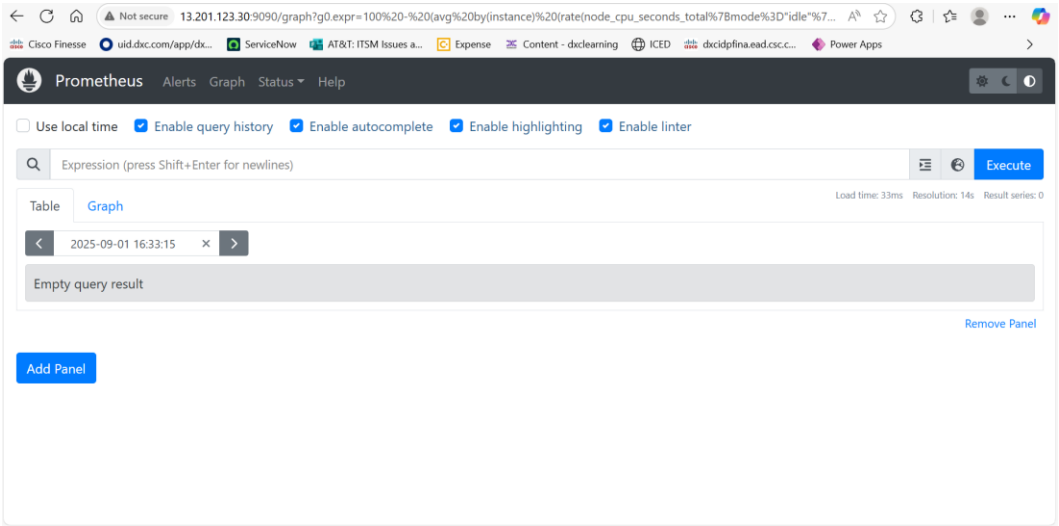
Step 33: Application logs in container



Step 34: Final deployed application dashboard



Step 35: Overall project monitoring dashboard



Conclusion

This project successfully demonstrates deploying a React application in a production-ready environment using Docker, Jenkins, AWS, Prometheus, and Grafana. The automated CI/CD pipeline ensures smooth development and deployment, while monitoring provides real-time insights into application performance.