Final Deliverables

***Project Introduction***

***About the Application***

***The project focuses on developing and deploying a Book-My-Show clone application, which simulates an online movie and event ticket booking platform. The application allows users to:***

* ***Browse available movies and events.***
* ***Check show timings and seat availability.***
* ***Book tickets through a user-friendly interface.***
* ***Manage bookings efficiently.***

***This use case was selected because it closely mirrors a real-world business scenario, where multiple services interact, and scalability, reliability, and continuous delivery are essential. It also provides an ideal foundation for applying DevOps principles in a hands-on manner.***

***Why This Application***

***The Book-My-Show clone was chosen as the base application for several reasons:***

1. ***Real-world relevance – Online ticket booking is a well-understood domain, making it easy to relate DevOps practices to actual business outcomes.***
2. ***Complexity of components – The app requires backend services, frontend UI, databases, and external integrations, making it a great candidate to practice containerization, deployment, and monitoring.***
3. ***Scalability needs – As user load grows, the application must scale seamlessly, which highlights the importance of Kubernetes, monitoring, and load balancing.***
4. ***End-to-end DevOps demonstration – The project covers the complete DevOps lifecycle:***
   * ***Plan (using Jira for project/task tracking)***
   * ***Code & Build (using GitHub, Jenkins, Maven)***
   * ***Test (using SonarQube for quality, Trivy/OWASP for security scanning)***
   * ***Release & Deploy (using Docker, Kubernetes, AWS EKS)***
   * ***Monitor & Improve (using Prometheus and Grafana)***

***Tool Stack Used***

***The project leverages a comprehensive DevOps toolchain:***

***🔹 Project Management & Collaboration***

* ***Jira – For defining epics, stories, and tasks, and for tracking progress from backlog to completion.***
* ***GitHub – For source code management, pull requests, and collaboration.***

***🔹 CI/CD Automation***

* ***Jenkins – Automates the CI/CD pipeline, including building, testing, containerizing, and deploying the app.***
* ***Jenkinsfile – Defines pipeline stages (build, test, quality scan, Docker image build, push to registry, deployment).***

***🔹 Code Quality & Security***

* ***SonarQube – Performs static code analysis and enforces Quality Gates for code maintainability, reliability, and security.***
* ***dependencies for known vulnerabilities.***

***🔹 Containerization & Image Management***

* ***Docker – Packages the application into containers for portability and consistency across environments.***
* ***DockerHub – Stores and shares Docker images, enabling smooth deployment to Kubernetes clusters.***

***🔹 Infrastructure & Orchestration***

* ***Terraform – Automates provisioning of AWS resources, ensuring infrastructure is reproducible and version-controlled.***
* ***Amazon EKS (Elastic Kubernetes Service) – Provides a managed Kubernetes environment for deploying and managing containerized workloads.***
* ***Kubernetes – Handles application deployment, scaling, and load balancing via Pods, Deployments, and Services.***

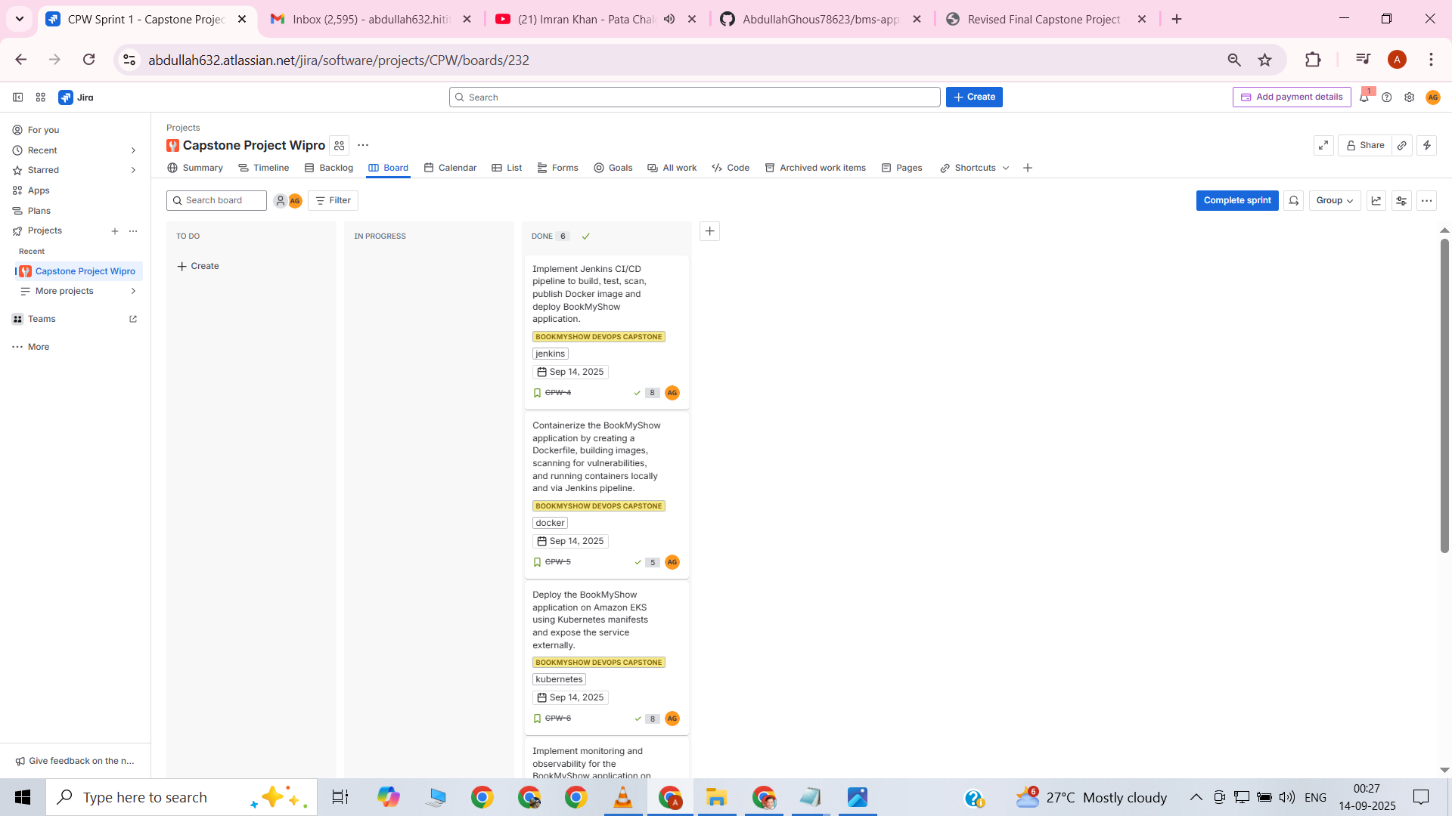
***🔹 Monitoring & Alerting***

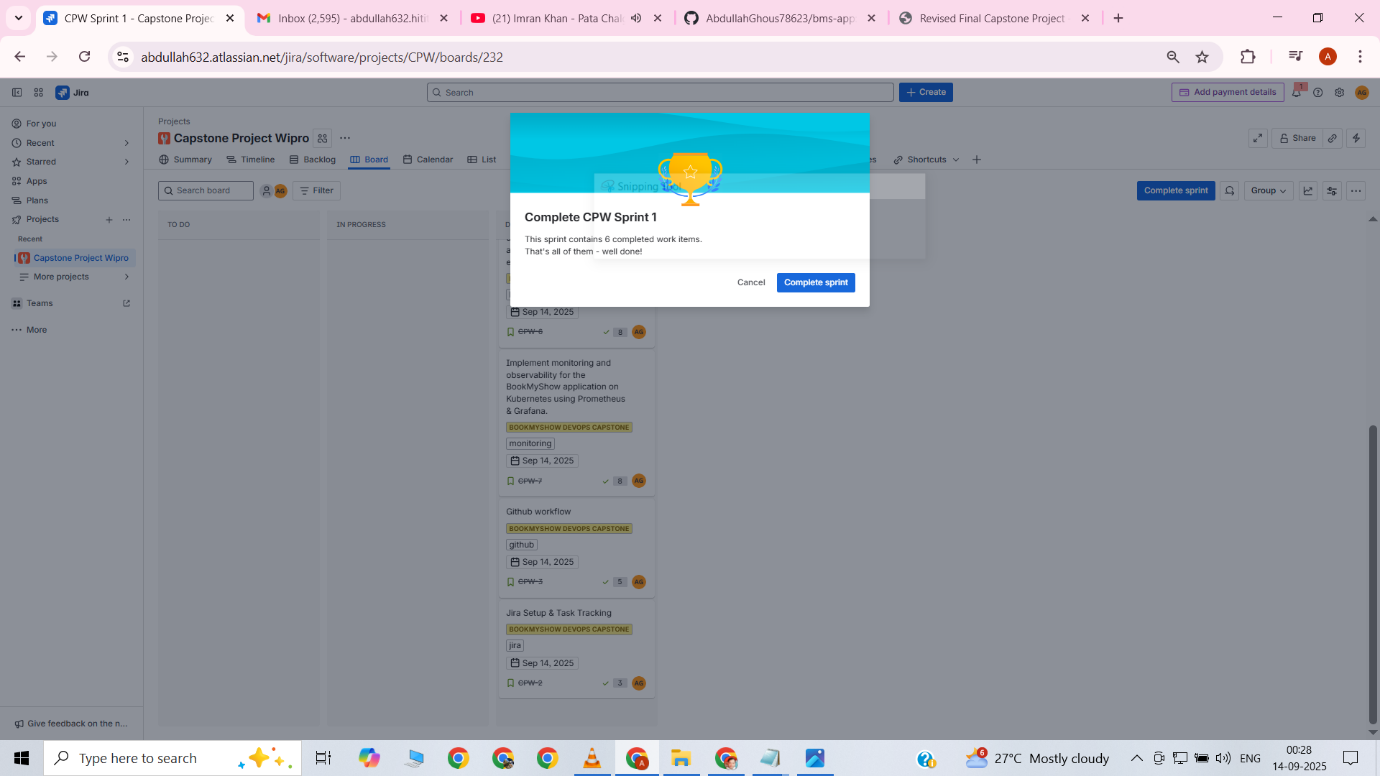
* ***Prometheus – Collects real-time metrics from applications and infrastructure.***
* ***Grafana – Visualizes metrics on interactive dashboards for monitoring performance and availability.***
* ***Email Notifications – Alerts stakeholders on pipeline status (success/failure).***

***Expected Outcome***

***By the end of this project, the application is fully automated through a CI/CD pipeline, deployed on a scalable Kubernetes cluster, continuously monitored, and maintained with proper quality checks and security scans. The project demonstrates how DevOps practices ensure faster delivery, better collaboration, higher code quality, and improved reliability of modern applications.***

1. . Jira Board





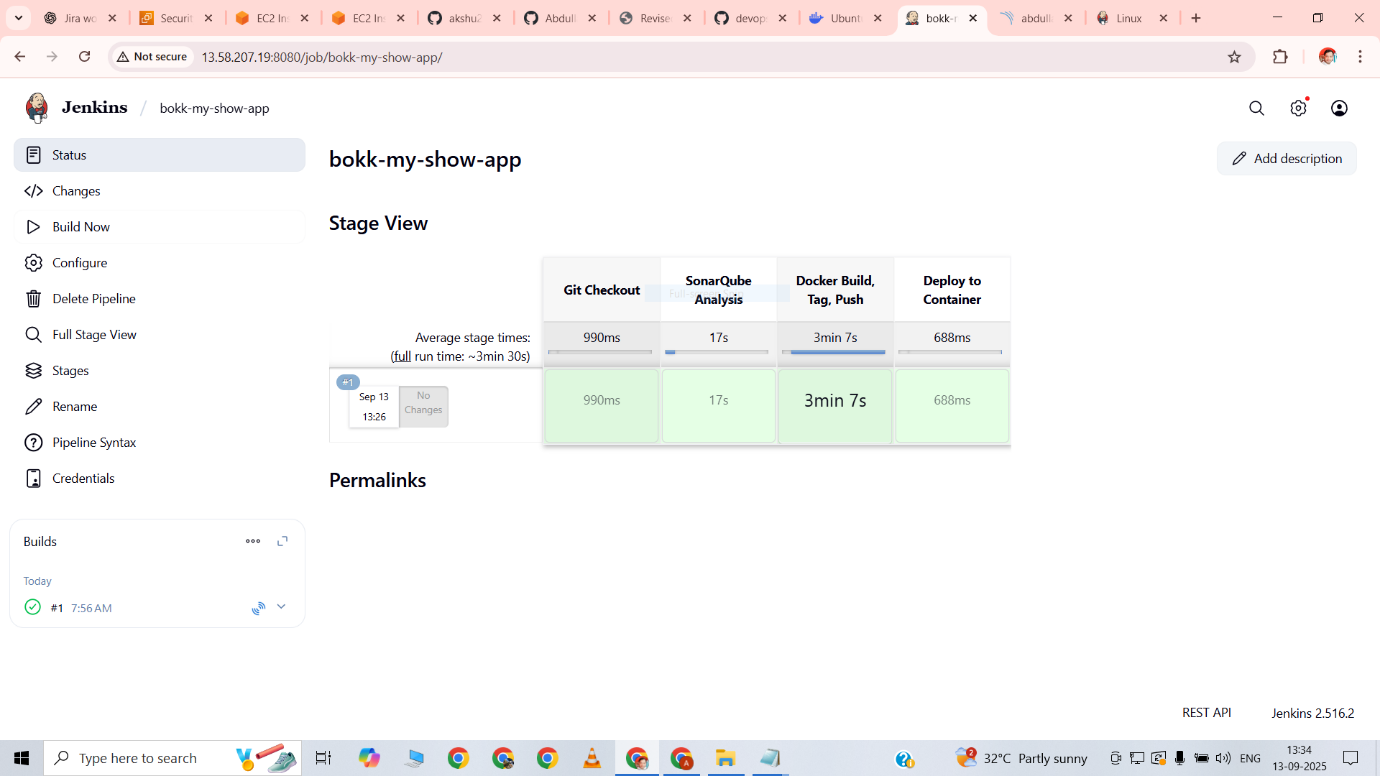
Created a dedicated Jira project board named “Automated CI/CD Pipeline with Jenkins, Terraform, and Monitoring Integration”  
Defined Epics, Stories, and Tasks to structure the project workflow  
Configured the board to follow To Do → In Progress → Done lifecycle  
This board served as the central place to track all project activities from start to finish

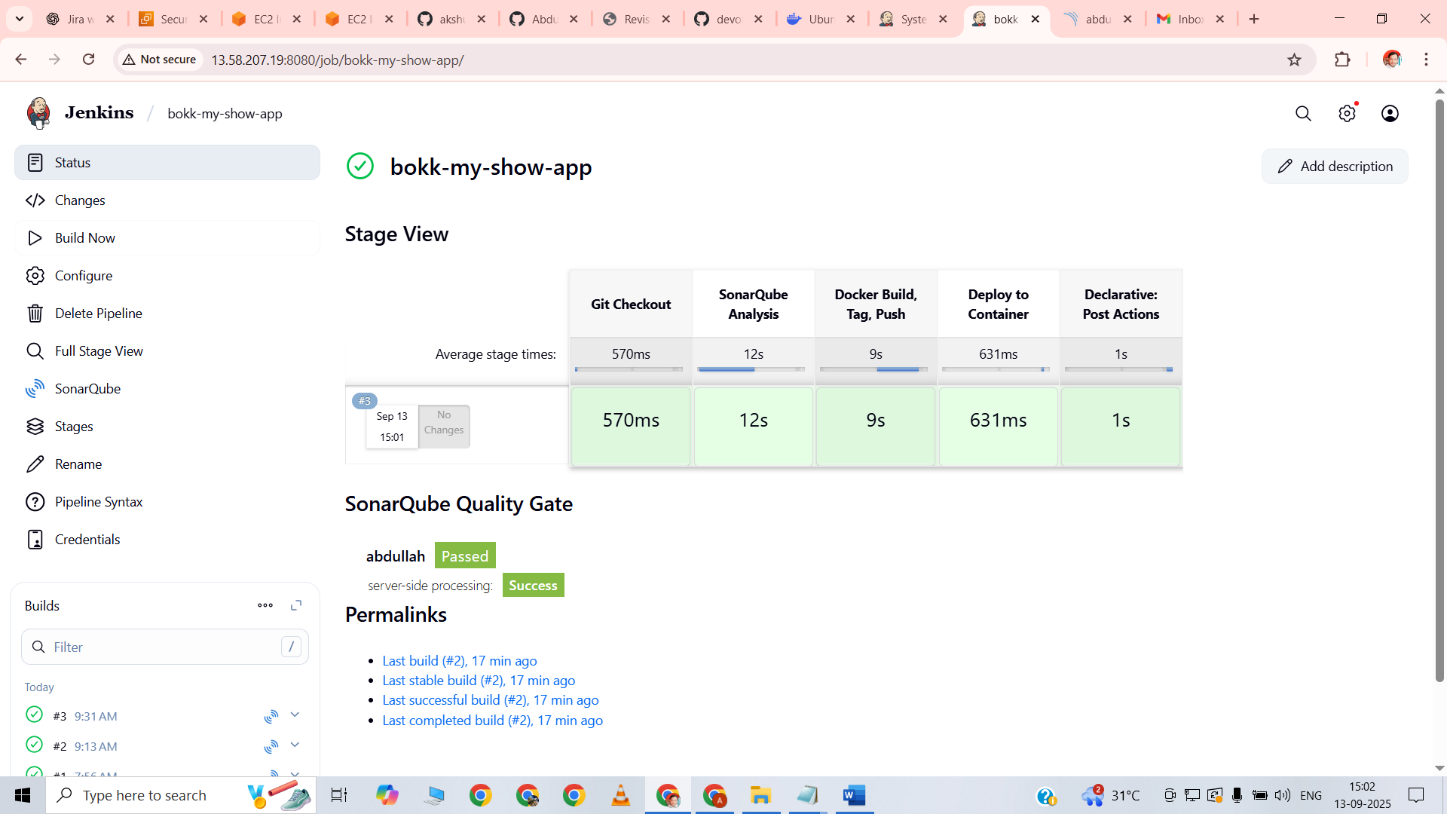
2.. GitHub PR link.

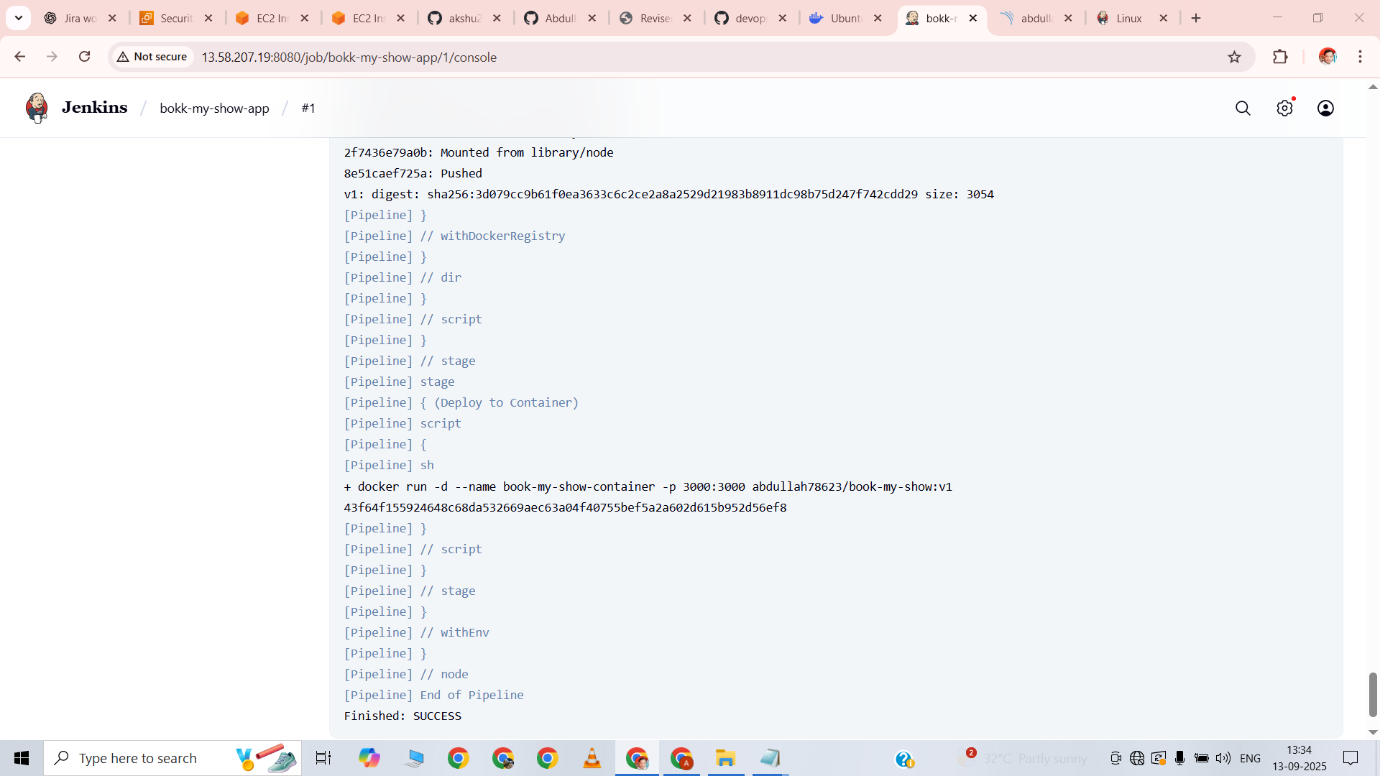
<https://github.com/akshu20791/Book-My-Show/pull/72>

Changed the Dockerfile to refine container configuration  
Created a docker-compose.yml file to run multiple services together  
Verified that the application and supporting services worked correctly with docker-compose

3.. Jenkins pipeline execution logs.







The Jenkins pipeline was executed successfully. All stages including Git Checkout, SonarQube Analysis, Docker Build & Push, and Deployment passed without errors. The console output displayed the final message “Finished: SUCCESS”, confirming the end-to-end pipeline execution.

4.1--Jenkinsfile-file-1-Deployment successfully done

pipeline {

agent any

environment {

SCANNER\_HOME = tool 'sonar-scanner'

}

stages {

stage('Git Checkout') {

steps {

git branch: 'main', url: 'https://github.com/AbdullahGhous78623/bms-app.git'

}

}

stage('SonarQube Analysis') {

steps {

withSonarQubeEnv('sonar') {

sh """

$SCANNER\_HOME/bin/sonar-scanner \

-Dsonar.projectKey=abdullah \

-Dsonar.projectName=abdullah \

-Dsonar.sources=.

"""

}

}

}

stage('Docker Build, Tag, Push') {

steps {

script {

dir('bookmyshow-app') {

withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {

sh 'docker build -t book-my-show:v1 .'

sh 'docker tag book-my-show:v1 abdullah78623/book-my-show:v1'

sh 'docker push abdullah78623/book-my-show:v1'

}

}

}

}

}

stage('Deploy to Container') {

steps {

script {

sh 'docker run -d --name book-my-show-container -p 3000:3000 abdullah78623/book-my-show:v1'

}

}

}

}

}

4.2-Jenkins—file-2—with email message of success message

pipeline {

agent any

environment {

SCANNER\_HOME = tool 'sonar-scanner'

}

stages {

stage('Git Checkout') {

steps {

git branch: 'main', url: 'https://github.com/AbdullahGhous78623/bms-app.git'

}

}

stage('SonarQube Analysis') {

steps {

withSonarQubeEnv('sonar') {

sh """

${SCANNER\_HOME}/bin/sonar-scanner \\

-Dsonar.projectKey=abdullah \\

-Dsonar.projectName=abdullah \\

-Dsonar.sources=.

"""

}

}

}

stage('Docker Build, Tag, Push') {

steps {

script {

dir('bookmyshow-app') {

withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {

sh 'docker build -t book-my-show1:v1 .'

sh 'docker tag book-my-show1:v1 abdullah78623/book-my-show1:v1'

sh 'docker push abdullah78623/book-my-show1:v1'

}

}

}

}

}

stage('Deploy to Container') {

steps {

script {

sh 'docker run -d --name book-my-show-container -p 3000:3000 abdullah78623/book-my-show1:v1"'

}

}

}

}

post {

success {

emailext (

subject: "SUCCESS: CI/CD Pipeline Passed",

body: """

Hi Abdullah,

✅ The Jenkins pipeline executed successfully.

- Code checkout ✅

- SonarQube Quality Gate ✅

- Docker Build & Push ✅

- Deployment ✅

You can check SonarQube report here:

http://13.58.207.19:9000/dashboard?id=abdullah&selectedTutorial=local

username: admin, password: 1234

Regards,

Jenkins

""",

to: 'abdullah632.hitit@gmail.com'

)

}

failure {

emailext (

subject: "FAILED: CI/CD Pipeline",

body: """

Hi Abdullah,

❌ The Jenkins pipeline has failed.

Please review the Jenkins console logs and fix the issue.

SonarQube Dashboard:

http://13.58.207.19:9000/dashboard?id=abdullah&selectedTutorial=local

username: admin, password: 1234

Regards,

Jenkins

""",

to: 'abdullah632.hitit@gmail.com'

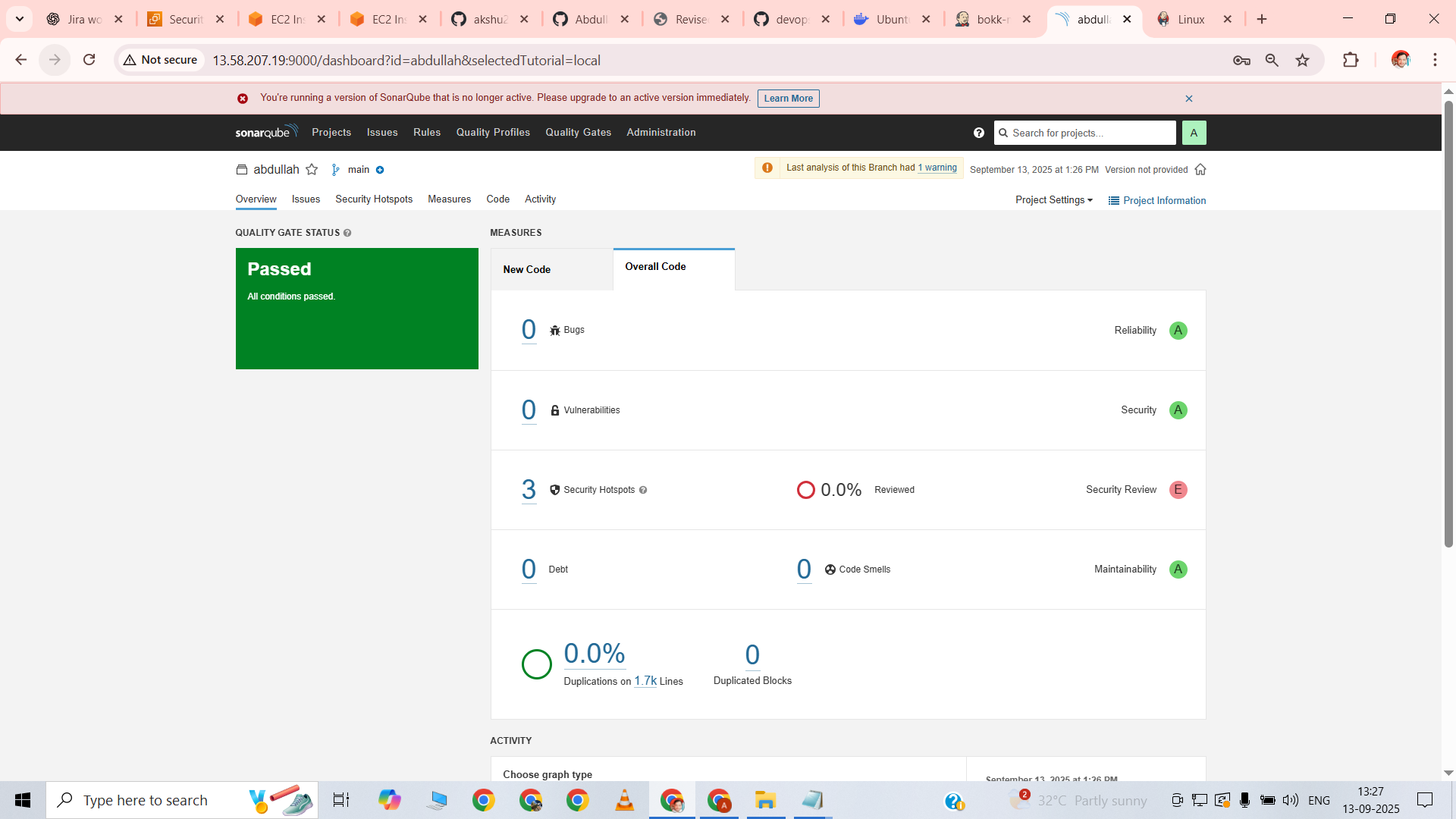
)

}

}

}

5. . SonarQube Quality Gate report.



SonarQube analysis was successfully completed, and the project passed the Quality Gate with no major code issues. This ensured code quality and maintainability before deployment.

6. DockerHub repository link.

<https://hub.docker.com/repository/docker/abdullah78623/book-my-show/tags>

docker pull abdullah78623/book-my-show:v1

The Docker image book-my-show:v1 was successfully built and pushed to Docker Hub, making it available for deployment in containers or Kubernetes.

7. Dockerfile

FROM node:18

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install chokidar@3 postcss@8.4.21 postcss-safe-parser@6.0.0 --legacy-peer-deps

RUN npm install --legacy-peer-deps

COPY . .

EXPOSE 3000

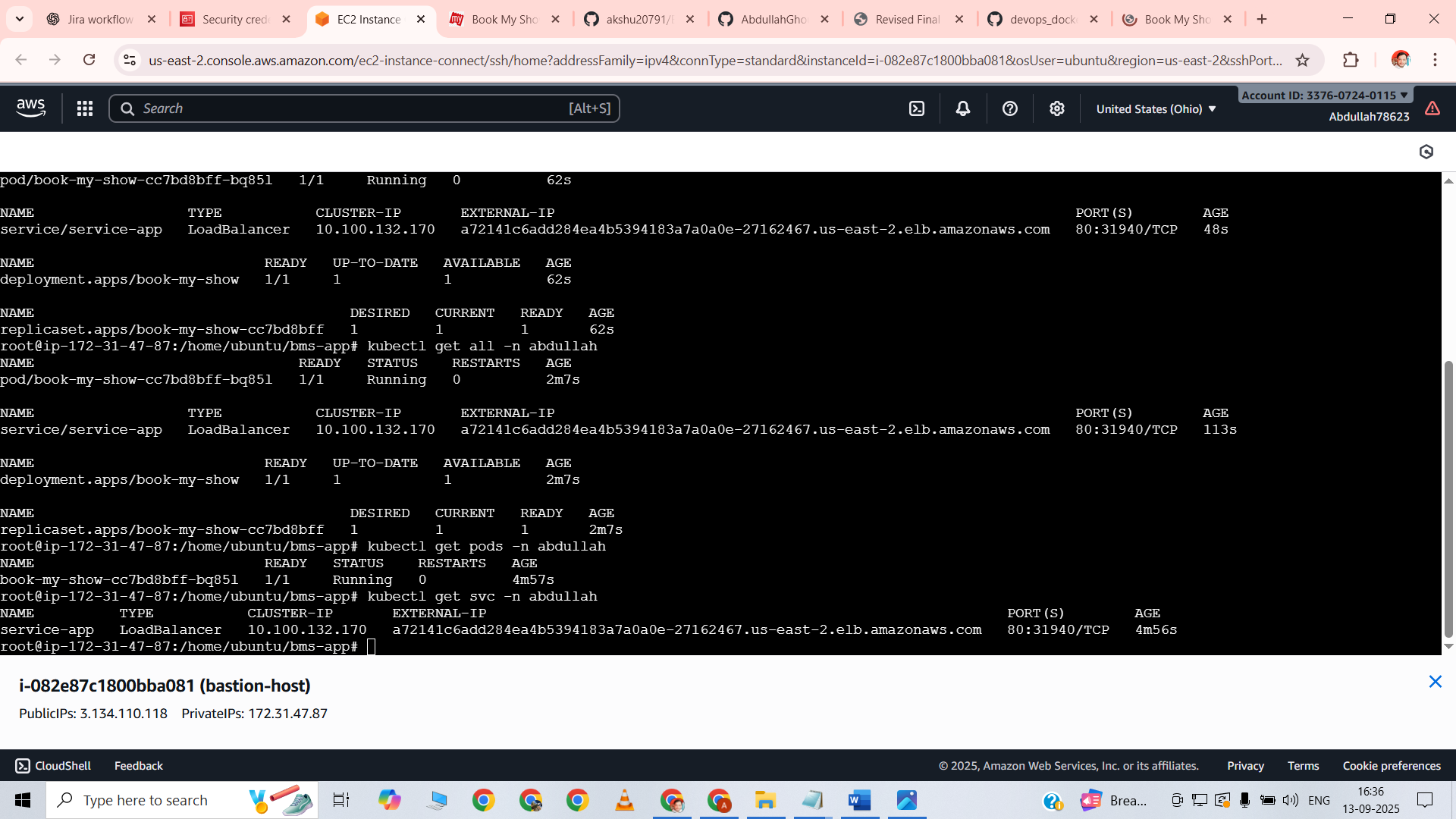
ENV NODE\_OPTIONS=--openssl-legacy-provider

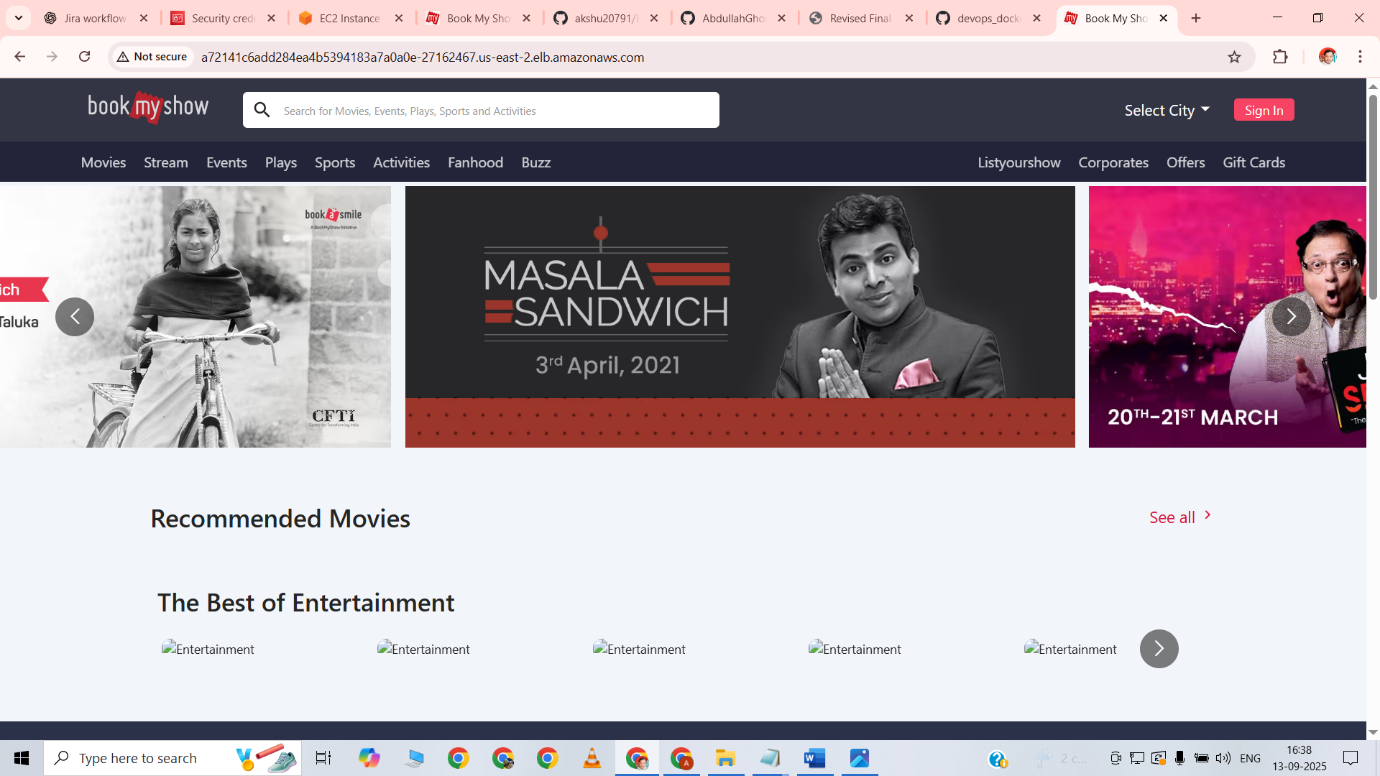
ENV PORT=3000

CMD ["npm", "start"]

The Dockerfile was created to containerize the Book-My-Show React application. It installs dependencies, sets the working directory, exposes port 3000, and defines the start command for the container.

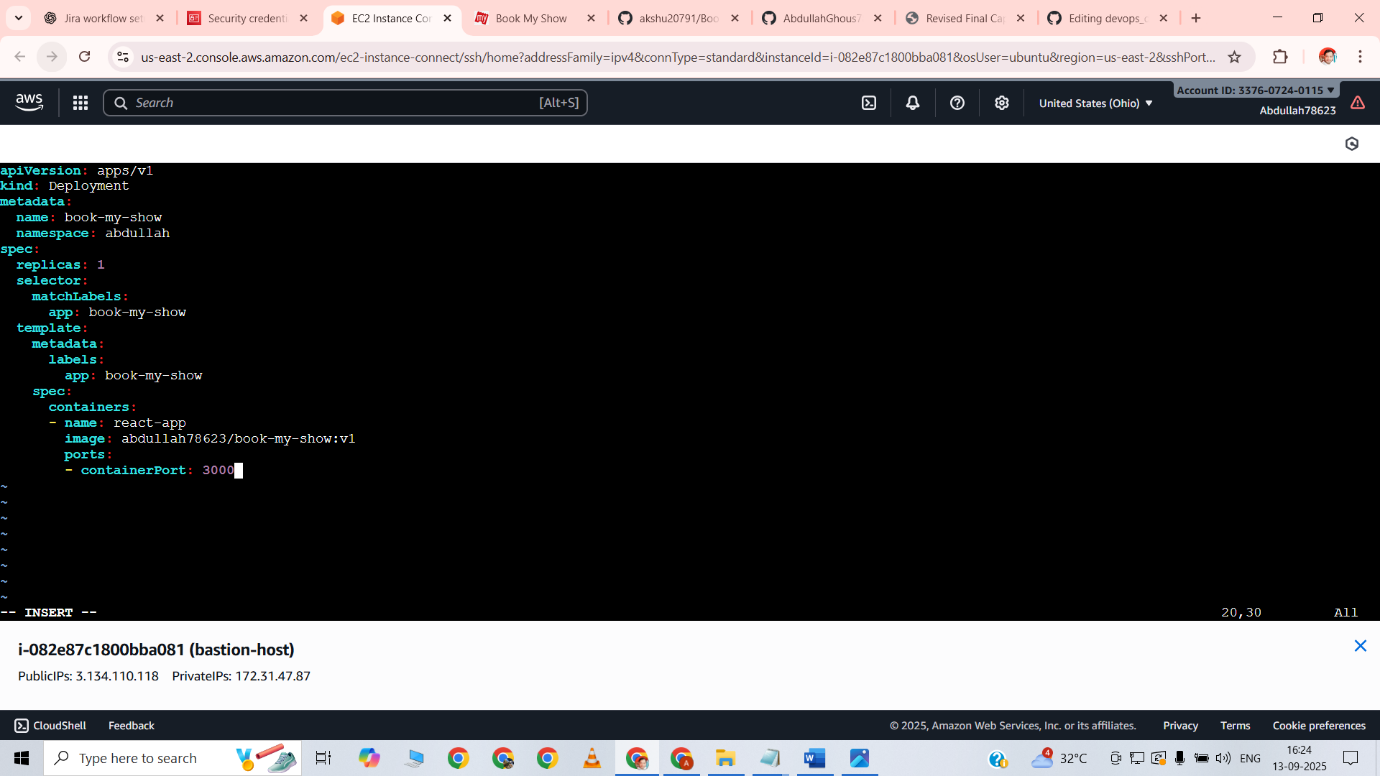
8. . EKS deployment proof





Commands executed:  
kubectl get all -n abdullah  
kubectl get pods -n abdullah  
kubectl get svc -n abdullah

**Observation:**  
All pods and services are running successfully. The Book-My-Show application is accessible through the LoadBalancer URL in a web browser.

8.1 . Kubernetes manifest files – deployment.yaml 

apiVersion: apps/v1

kind: Deployment

metadata:

name: book-my-show

namespace: abdullah

spec:

replicas: 1

selector:

matchLabels:

app: book-my-show

template:

metadata:

labels:

app: book-my-show

spec:

containers:

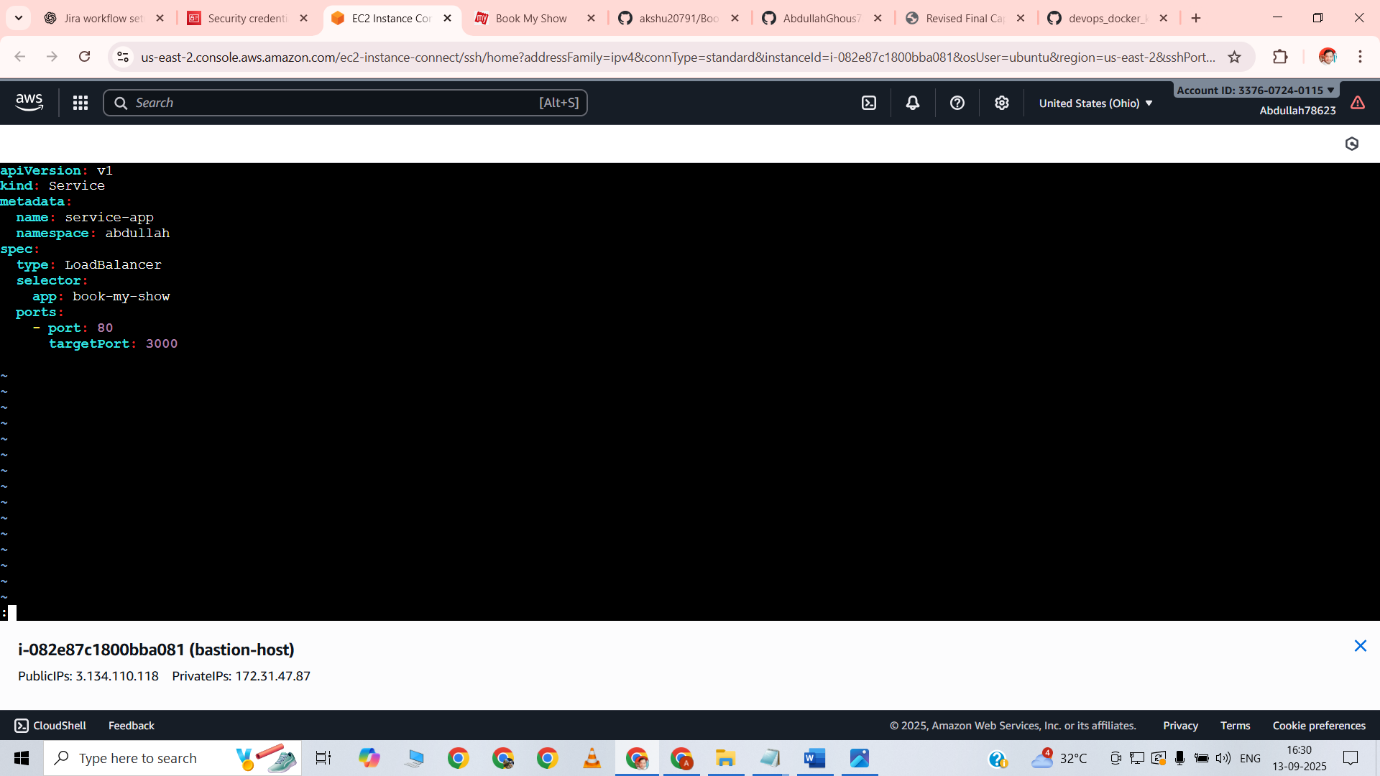
- name: react-app

image: abdullah78623/book-my-show:v1

ports:

- containerPort: 3000

8.1 . Kubernetes manifest files –service.yaml



apiVersion: v1

kind: Service

metadata:

name: service-app

namespace: abdullah

spec:

type: LoadBalancer

selector:

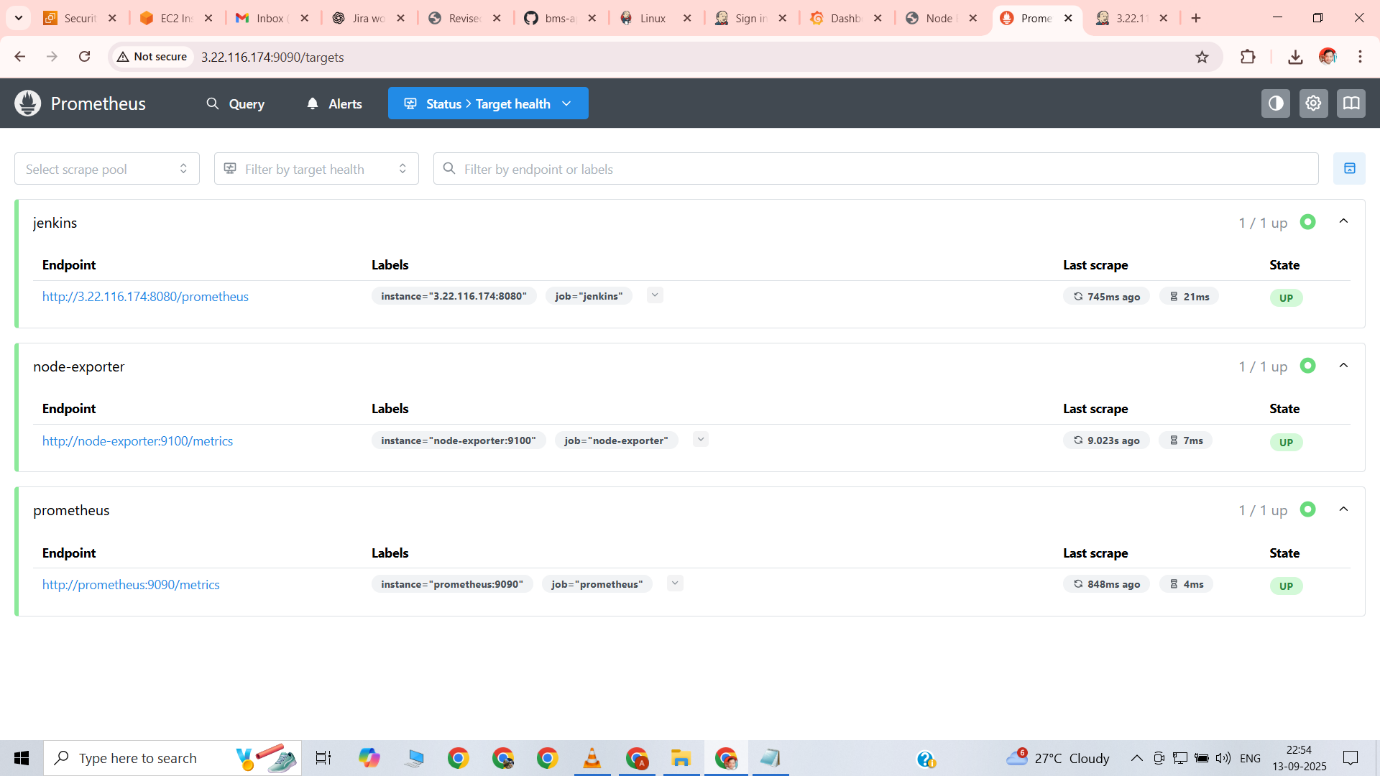
app: book-my-show

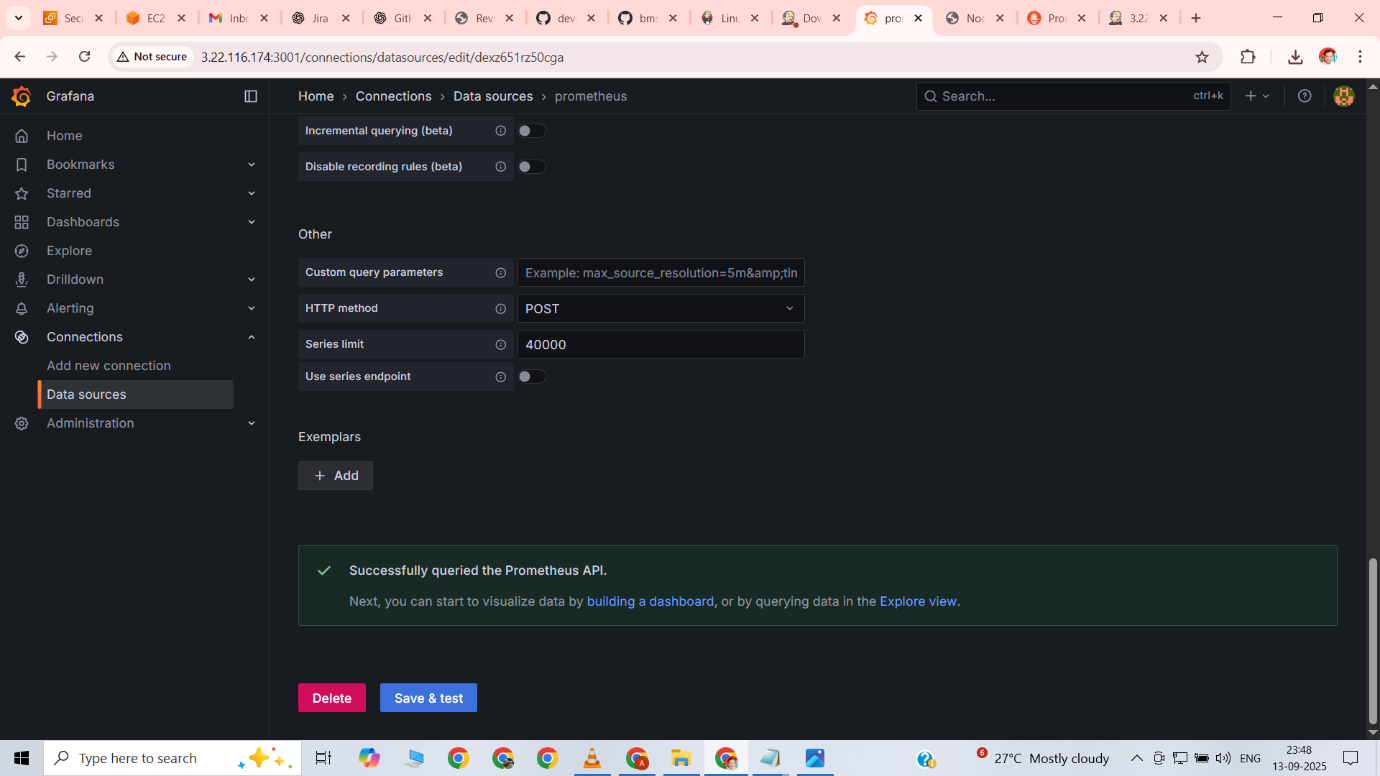
ports:

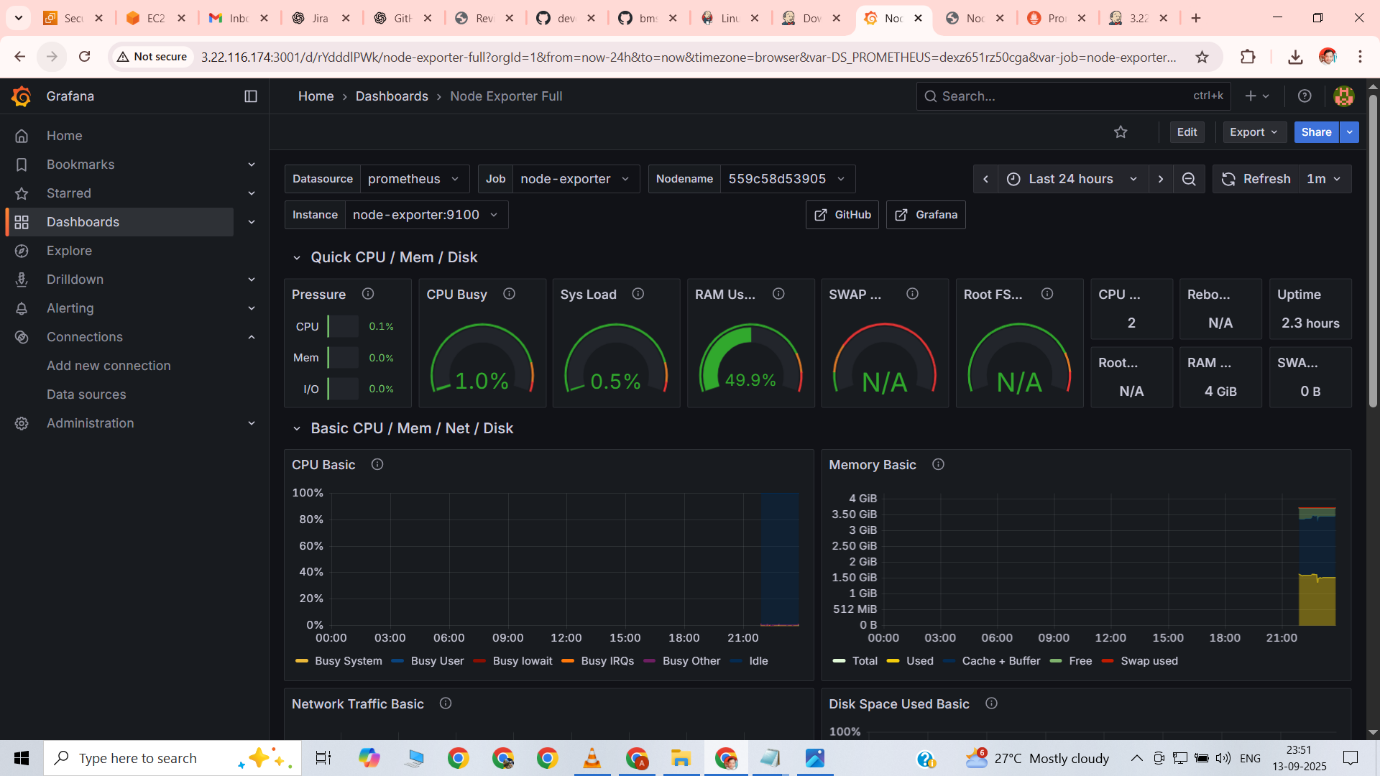
- port: 80

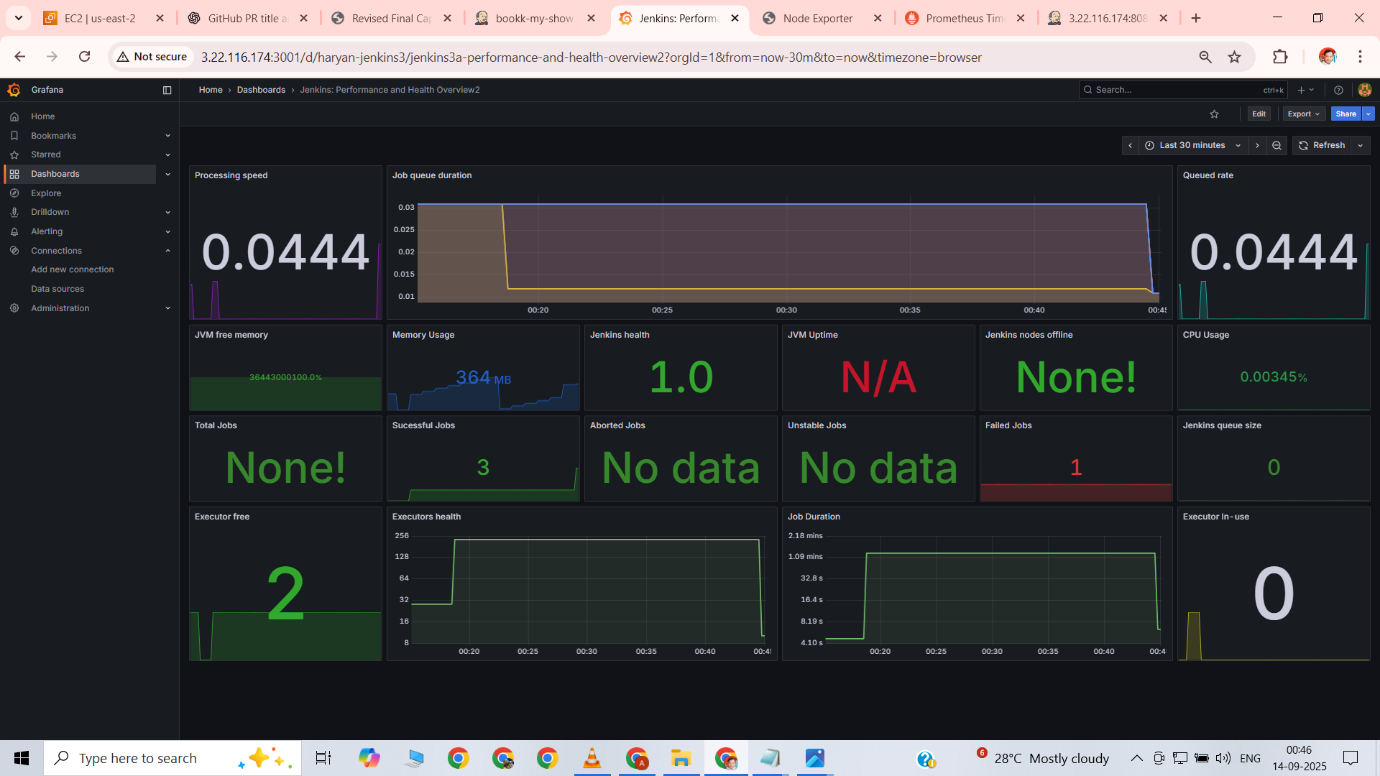
targetPort: 3000

9. Prometheus & Grafana screenshots



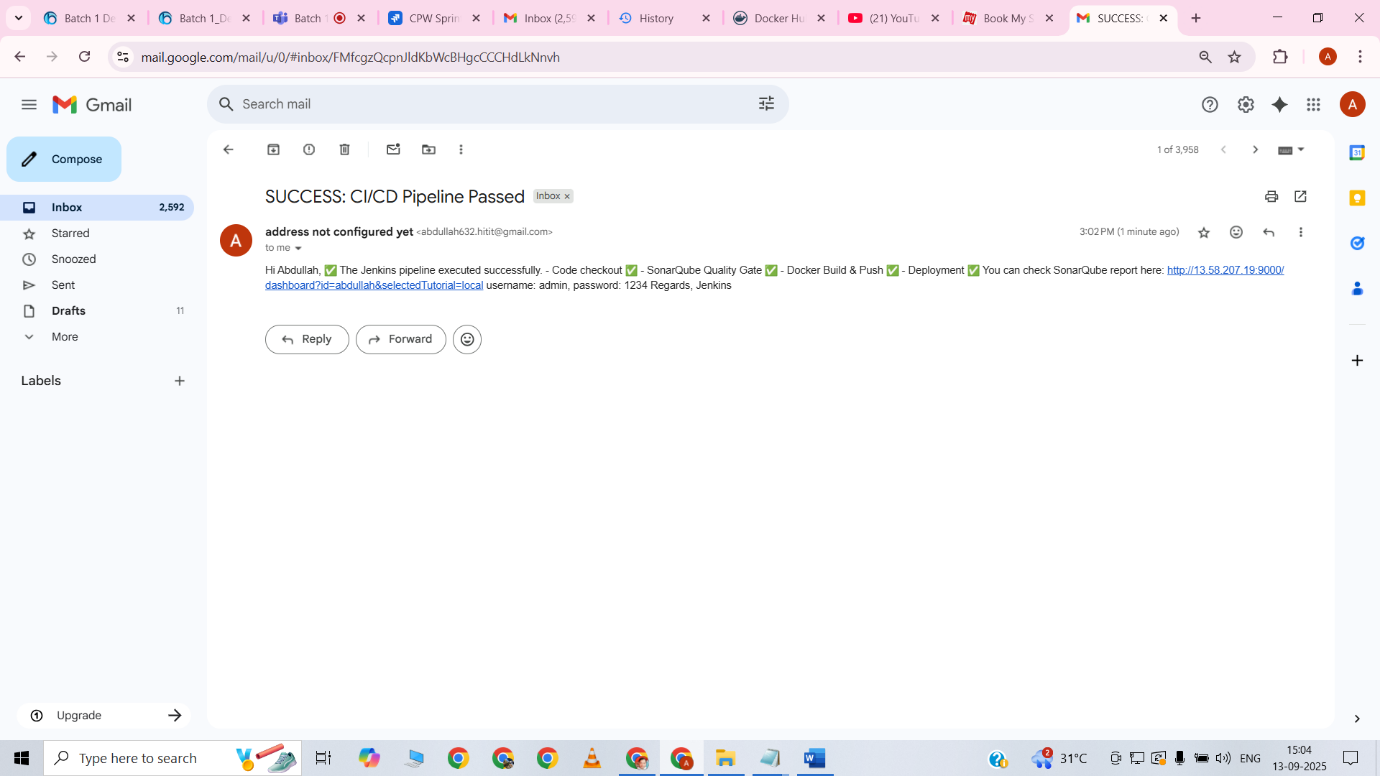






Prometheus and Node Exporter were successfully set up to collect system metrics. Jenkins metrics were integrated into Prometheus, and Grafana dashboards were configured to visualize Node health and CI/CD performance in real time.

10. Email notification screenshot.



Jenkins was configured to send email notifications upon pipeline completion. A success notification was received in the registered email, confirming that all CI/CD stages (Git checkout, SonarQube analysis, Docker build & push, and deployment) executed successfully.

Conclusion

This project successfully demonstrated the **end-to-end implementation of a modern DevOps pipeline** using a real-world application scenario. By integrating tools such as **Jira, GitHub, Jenkins, SonarQube, Docker, Kubernetes, Terraform, Prometheus, and Grafana**, the application lifecycle was fully automated — from **planning and coding** to **deployment and monitoring**.

All the defined deliverables were completed, and the application was deployed on a scalable and reliable **EKS cluster** with proper monitoring and security checks in place. This not only highlights the power of **DevOps practices** in streamlining software delivery but also proves the importance of **automation, collaboration, and continuous improvement** in modern software engineering.

The journey through this project reinforced the value of adopting DevOps in real-world enterprises:

* Faster and more reliable deployments.
* Improved code quality and security.
* Enhanced visibility and monitoring.
* Scalable infrastructure that adapts to business needs.

With all tasks successfully moved to **Done**, this project stands as a complete demonstration of **DevOps best practices applied to a real-world application**.

Name-ABDULLAH GHOUS

PROJECT-NAME-CAPSTONE\_PROJECT.