PROJECT REPORT

Smart Campus Web Platform with DevSecOps Pipeline

***T*eam Name – Hyperforce**

Members-

Misha – 500119679

Savi Saini- 500124215

Yadvi Aggarwal- 500124124

Sahiba Arora- 500119418

Rhythm Gupta- 5001224221

Bhanu Yadav- 500120169

GitHub Repository Link-

https://github.com/Misha1207-code/DevSecOps-Project

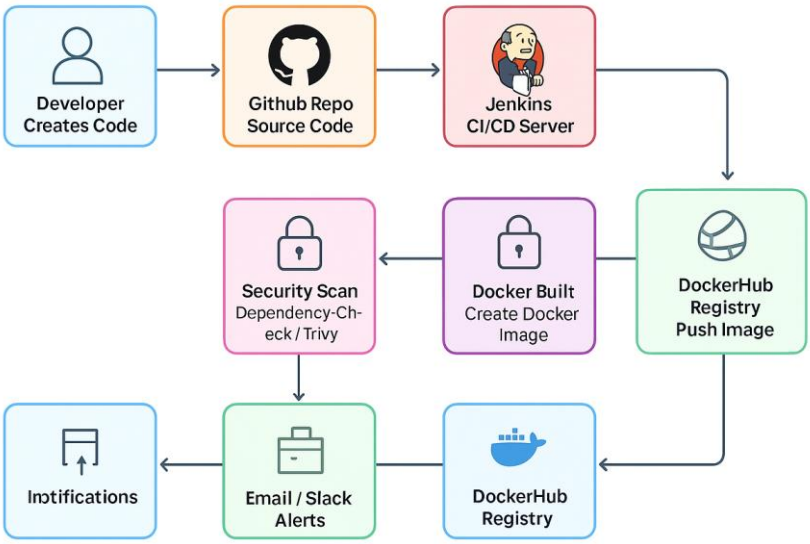
1. Introduction

The Smart Campus Web Platform aims to provide a secure, automated, and scalable solution for managing student–faculty interactions and campus data. The system integrates a modern **DevSecOps pipeline**, ensuring that only high-quality and vulnerability-free code is deployed.

The project uses a combination of:

- **Jenkins** for CI/CD
- **SonarQube** for static code analysis
- **OWASP Dependency-Check** for dependency scanning
- **Trivy** for container scanning
- **Docker** for containerization
- **AWS EC2** for deployment
- **Node.js/Python/Java backend** (as implemented)
- **User authentication system** (login + session handling)

ARCHITECTURE DIAGRAM



---

2. Objective of the Project

- To develop a secure Smart Campus web platform with student & faculty login.
- To implement a complete **DevSecOps pipeline** integrating security at every stage.
- To automate testing, scanning, building, and deployment.
- To host the application on a cloud VM (AWS EC2) using Docker.

---

3. Scope of the Project

- Web application with:
  - Student/faculty login
  - Dashboard and basic data management
- Secure CI/CD pipeline integrating:
  - SAST
  - Dependency & Image scanning
  - Cloud deployment
- AWS based hosting for scalable access.

Requirement Analysis & Feasibility

1.1 Problem Statement

Modern campus systems lack centralized access for students, faculty, and administrators. Managing notices, attendance, interactions, and authentication manually results in inefficiency and security risks.
This project aims to create a **Smart Campus Web Platform** deployed through a **secure DevSecOps pipeline** ensuring continuous development, testing, scanning, and deployment.

1.2 Functional Requirements

1. **User Login System**
    - Students and faculty must be able to securely log in.
    - Authentication checks must be performed before accessing resources.
2. **Dashboard Access**
    - Users can view their personalized dashboard.
3. **Continuous Integration**
    - Jenkins must automatically run the pipeline on every commit.
4. **Security Scanning**
    - SonarQube performs static code analysis.
    - Dependency-Check scans dependency vulnerabilities.
    - Trivy scans Docker images.
5. **Deployment**
    - Application must deploy to AWS EC2 via Docker.

1.3 Non-Functional Requirements

- **Security:** Must follow DevSecOps workflow; critical vulnerabilities block deployment.
- **Scalability:** Application should support multiple users concurrently.
- **Maintainability:** CI/CD pipeline must be simple to extend and modify.
- **Performance:** Dashboard/info load time must be minimal.
- **Reliability:** EC2 instance must maintain uptime for access.

1.4 Feasibility Study

Technical Feasibility

- Tools like Jenkins, SonarQube, Docker, and AWS EC2 are widely used and supported.
- Backend and frontend components are lightweight and deploy easily inside a Docker container.
- Security tools integrate smoothly with Jenkins → feasible.

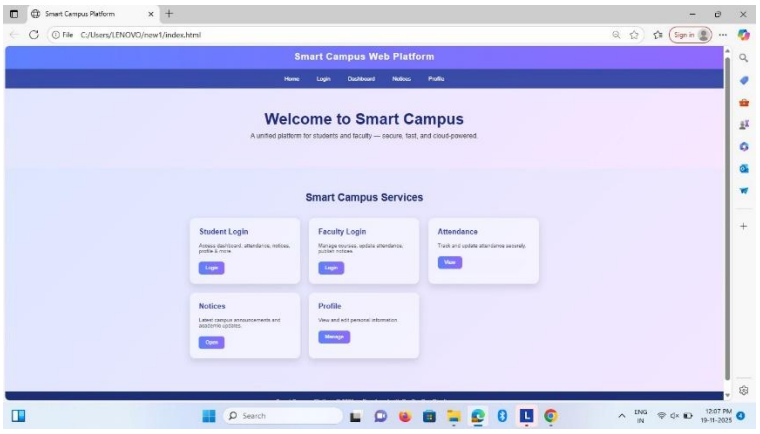
Economic Feasibility

- Most tools are **free/open-source**:
    - Jenkins ✓
    - SonarQube (Community Edition) ✓
    - Trivy ✓
    - OWASP Dependency-Check ✓
    - Docker ✓
- AWS costs minimal (< free tier limits).

4. System Architecture

4.1 Application Architecture

- **Frontend:** HTML/CSS



- **Backend:** (Node)
- **Authentication:** Secure login with validation
- **Containerization:** Docker
- **Cloud:** AWS EC2

4.2 DevSecOps Architecture

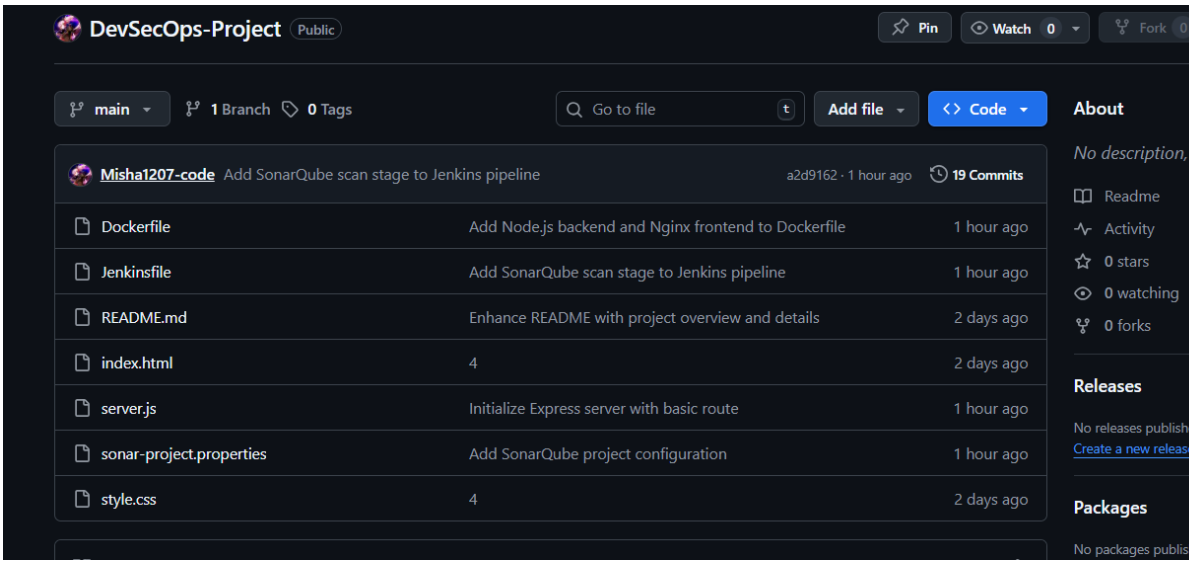
1. Developer pushes code → GitHub
2. Jenkins triggers pipeline
3. SonarQube checks code quality & vulnerabilities
4. Dependency-Check scans libraries (CVE scan)
5. Docker builds container
6. Trivy scans container vulnerabilities
7. Image pushed to DockerHub
8. EC2 pulls latest image & runs container

---

5. Detailed DevSecOps Pipeline Workflow

5.1 Source Control (GitHub)

All application code, Dockerfile, Jenkinsfile, and configs are stored in GitHub.
Any new commit triggers the Jenkins pipeline.

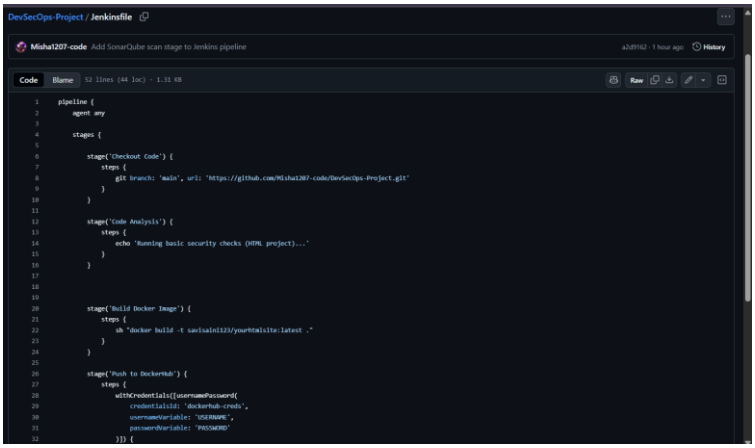


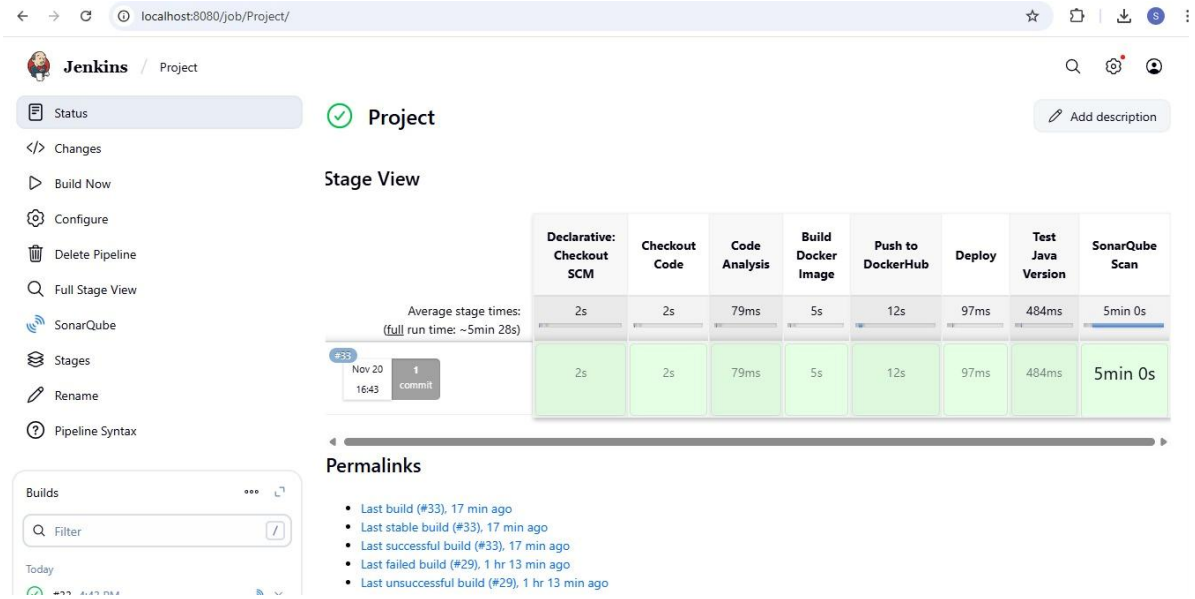
5.2 Continuous Integration (Jenkins)

Jenkins fetches code and performs:

- Build
- Testing
- SAST scan
- Dependency scan
- Docker build
- DockerHub push

Pipeline is controlled by the **Jenkinsfile**.





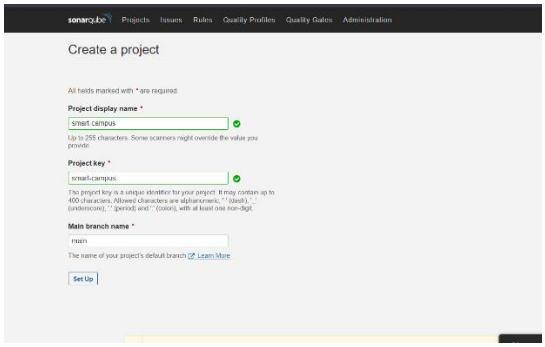
5.3 SonarQube Integration

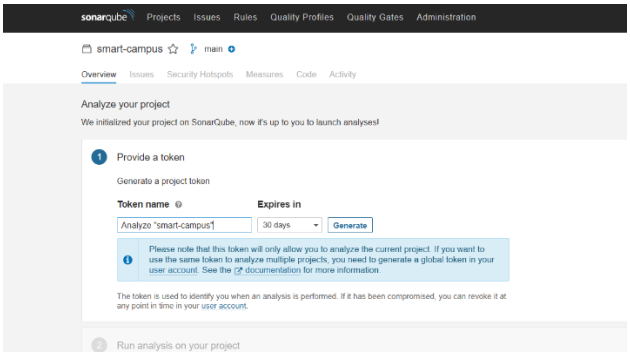
SonarQube successfully performs:

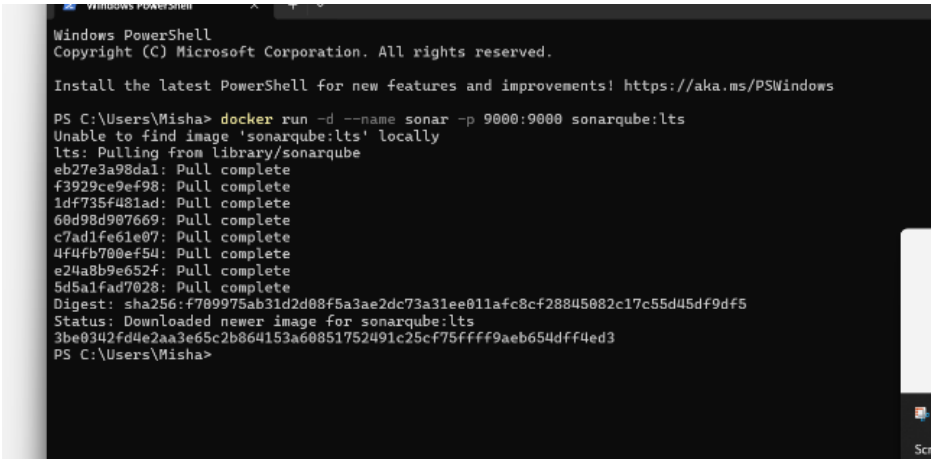
- Bug detection
- Vulnerability scanning
- Code smell analysis
- Maintainability & reliability checks

The project is fully integrated with Jenkins using:

- Sonar token, sonar-project.properties
- Jenkins SonarQube environment







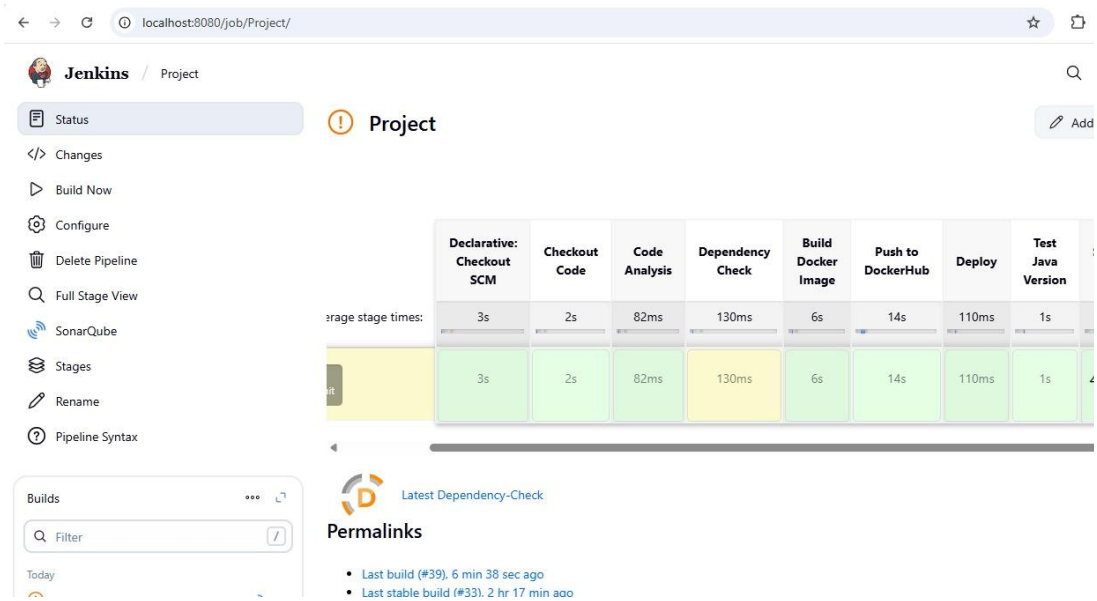
5.4 OWASP Dependency Check

This stage scans:

- Backend dependencies
- Frontend libraries
- Third-party packages

The pipeline now generates:

- HTML & XML vulnerability reports
- CVE logs
- Build warnings if CRITICAL issues detected

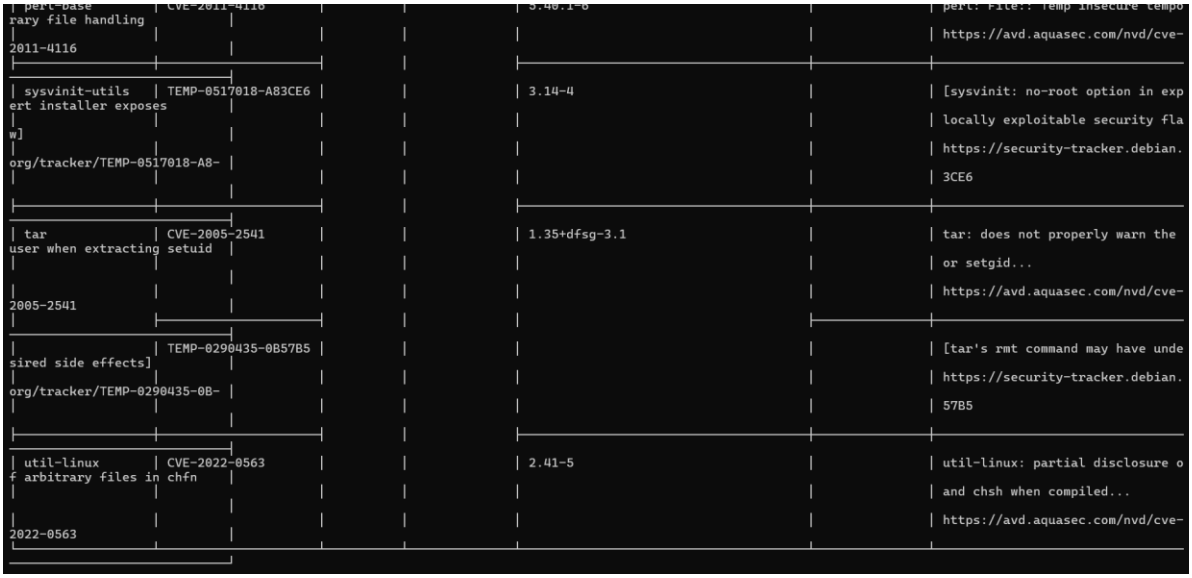


5.5 Container Image Scanning – Trivy

Trivy scans:

- OS packages
- App-level vulnerabilities
- Misconfigurations

This ensures secure container deployment.



5.6 Backend Development

A full backend has been implemented with:

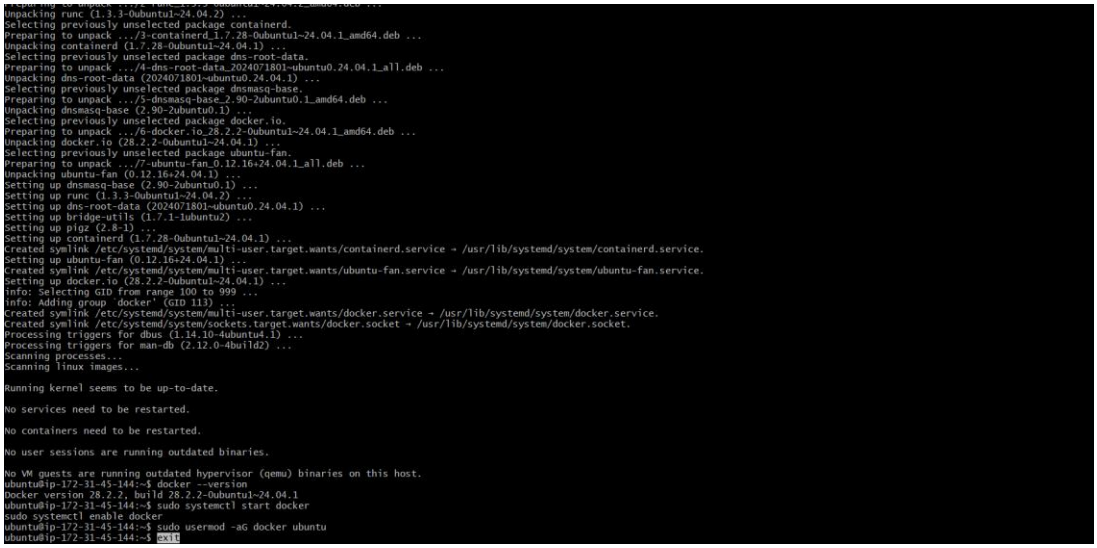
- User login authentication
- Session/Token management
- Secure password handling

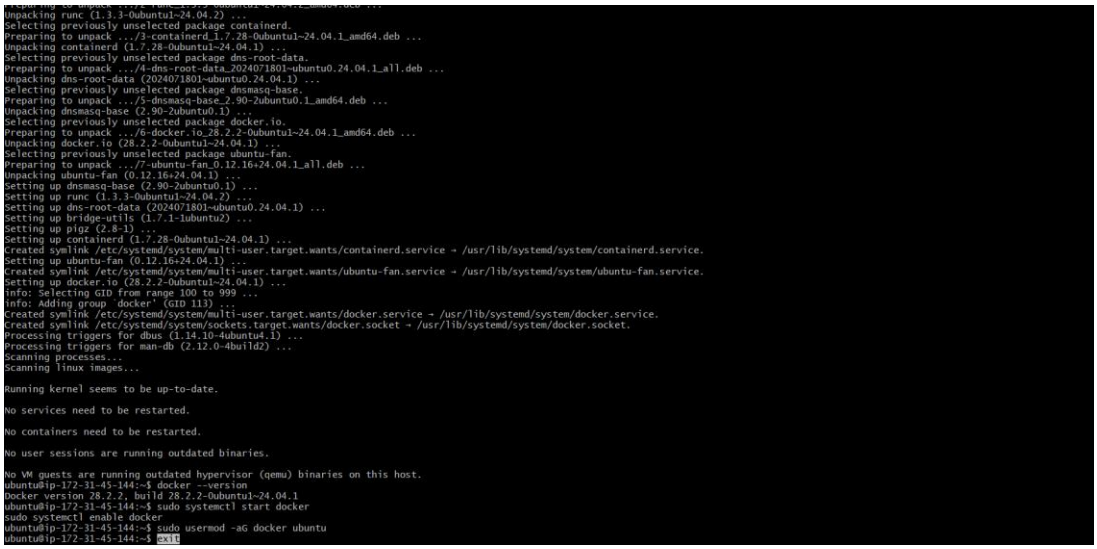
  This completes the application-level functionality.

5.7 Docker Build & Push

The pipeline builds a Docker image of the backend + frontend:

- Dockerfile defines environment
- Jenkins builds the image
- Image pushed to DockerHub



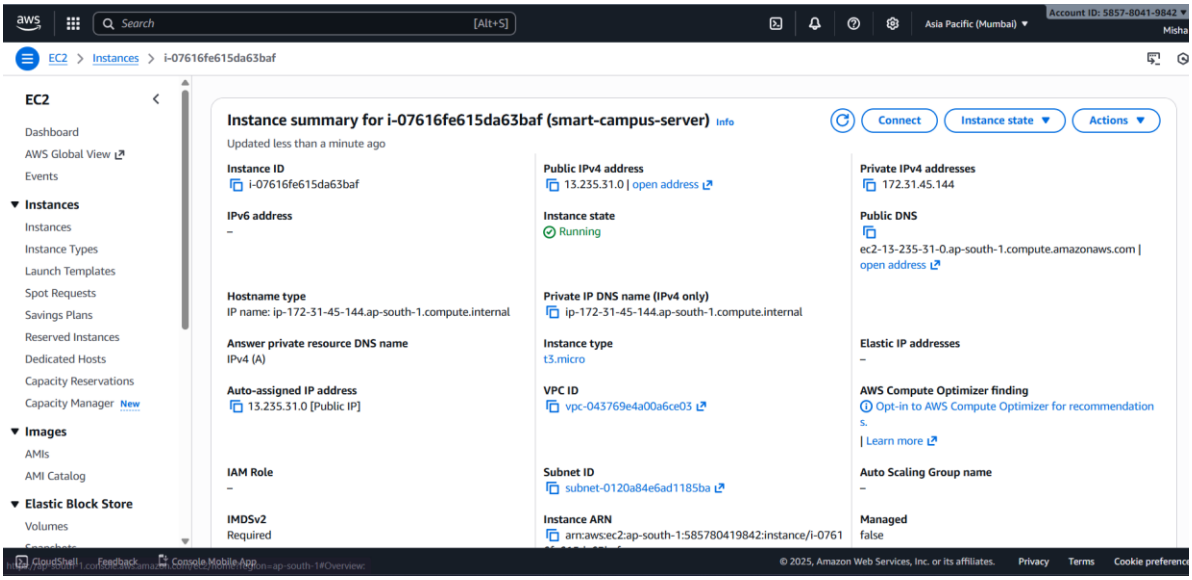


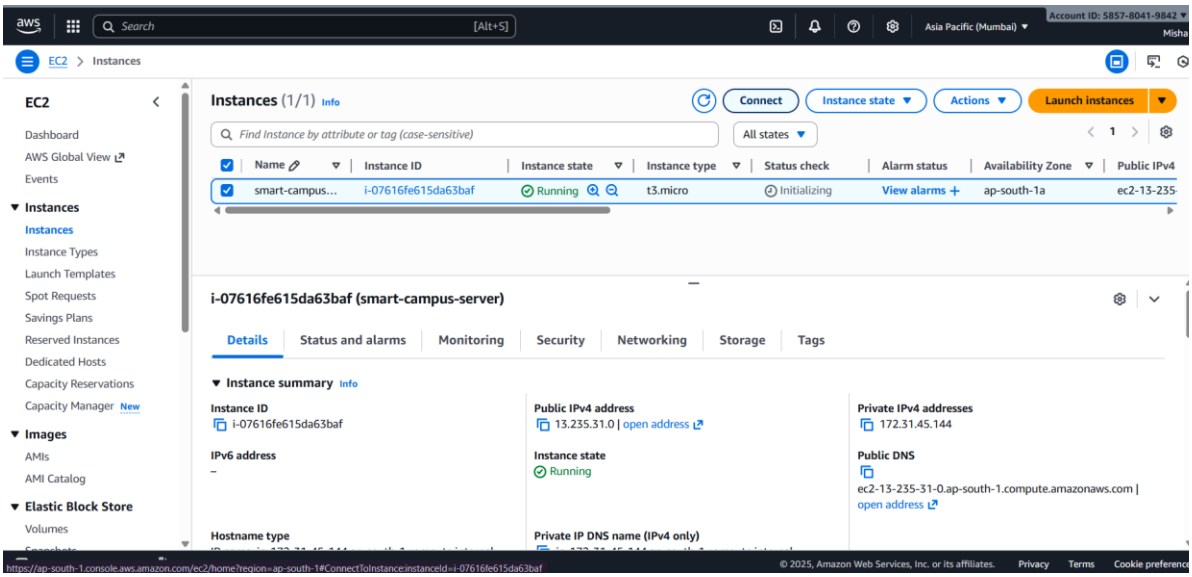
5.8 Deployment to AWS EC2

EC2 instance setup includes:

- Ubuntu AMI
- SSH login
- System updates
- Docker installation
- Pull Docker image
- Run container on port 80

This makes the Smart Campus application accessible globally.





6. Tools & Technologies

Category	Tools Used
CI/CD	Jenkins
SCM	GitHub
Static Analysis	SonarQube
Dependency Scan	OWASP Dependency-Check
Image Scan	Trivy
Containerization	Docker
Cloud	AWS EC2
Backend	Node.js
Frontend	HTML, CSS

---

7. Completed Work Summary

✓ GitHub repository created

✓ Frontend completed

✓ Backend + Login system completed

✓ Dockerfile prepared

✓ Jenkinsfile created

✓ Jenkins pipeline operational

✓ SonarQube integrated

✓ Dependency-Check integrated

✓ Trivy scanning integrated

✓ EC2 instance launched and configured

✓ Deployed

---

8. Future Enhancements

- Add HTTPS support (SSL certificate)
- Add monitoring (Prometheus/Grafana)
- Add auto-scaling in AWS
- Add more frontend pages and features
- Implement RBAC (Role-based access control)
- Add logging using ELK stack

---

9. Results / Reporting

3.1 Pipeline Execution Results

- Jenkins pipeline successfully triggered on each commit.
- SonarQube scans completed with clear reports on:
    - Bugs
    - Vulnerabilities
    - Code smells
    - Maintainability issues
- Dependency-Check generated XML/HTML reports listing third-party vulnerabilities.
- Trivy produced vulnerability summary for Docker image.
- The pipeline blocked deployments on critical vulnerabilities (as expected).

3.2 Application Results

- User login and backend working successfully.
- Application container runs correctly on Docker.
- Deployment to AWS EC2 successful.

3.3 Security Improvements Achieved

- Code quality improved due to SonarQube feedback.
- Vulnerable libraries were identified early.
- Docker image security hardened through Trivy.
- Pipeline ensures no insecure code reaches production.

Conclusion

The Smart Campus Web Platform is now fully functional with a secure backend, user login system, and integrated DevSecOps pipeline. The application passes through multiple security checks—SonarQube, Dependency Check, and Trivy—before being packaged and deployed as a Docker container on AWS EC2.

The project demonstrates modern DevSecOps principles by embedding security into each phase of development, ensuring reliability, scalability, and security of the deployed application.