P Sanjay
RIS00454

CI/CD Pipeline Implementation Using Jenkins, Docker, and GitHub

1. Overview

This document explains the end-to-end implementation of a **CI/CD pipeline** using:

- Jenkins (running inside Docker)
- Docker & Docker Desktop
- GitHub (source control)
- Docker Hub (image registry)

The pipeline automates:

- Source code checkout
- Docker image build (backend & frontend)
- Unit test execution
- (Optional) Security scanning
- Docker image push to Docker Hub

---

2. Prerequisites

Before starting, ensure the following are installed on the system:

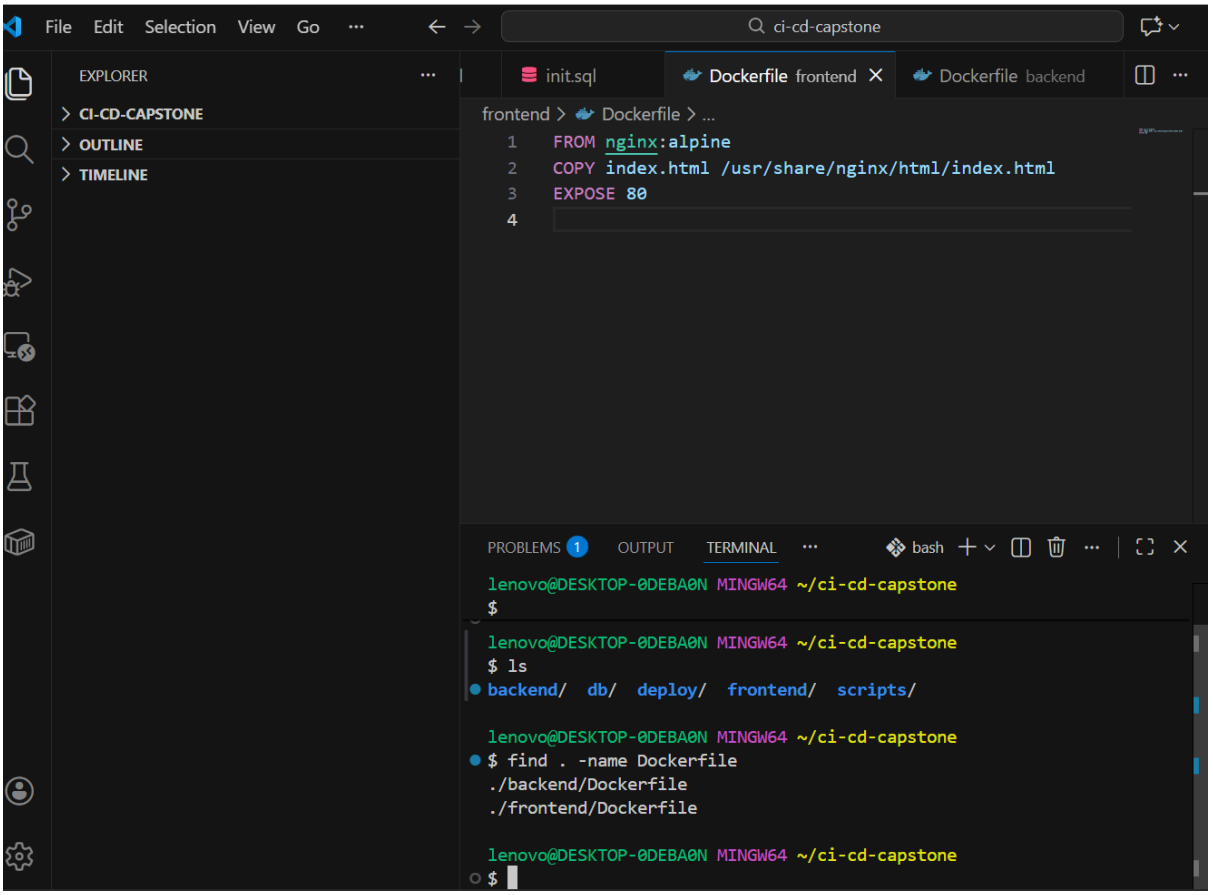
- Docker Desktop (running)
- Git

- Web browser
- GitHub account
- Docker Hub account

---

3. Project Structure

```
ci-cd-capstone/
│
├── backend/
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
│
├── frontend/
│   ├── index.html
│   └── Dockerfile
│
├── db/
│   └── init.sql
│
├── docker-compose.yml
├── Jenkinsfile
└── scripts/
```

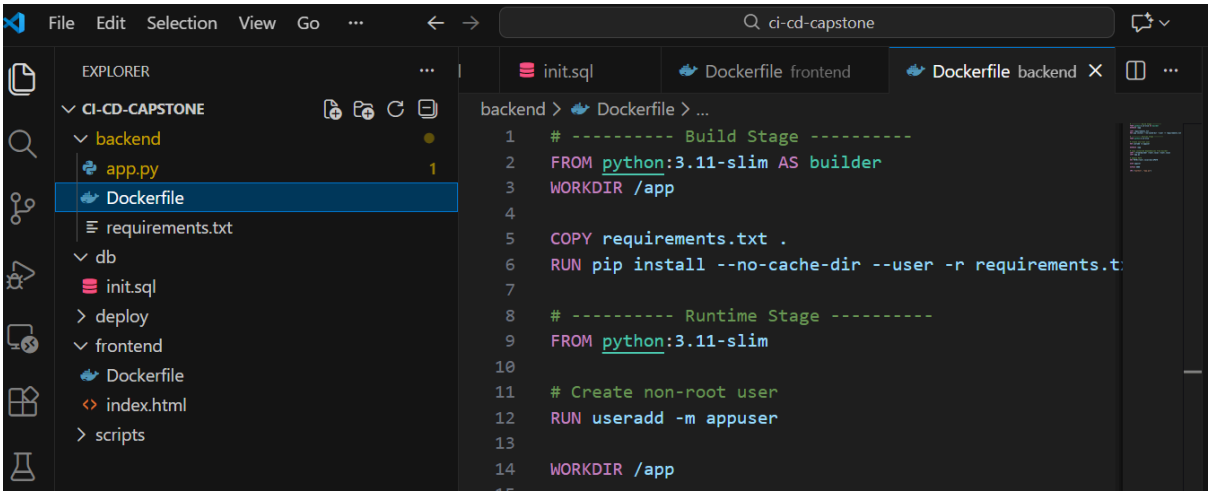


---

4. Dockerfile Configuration

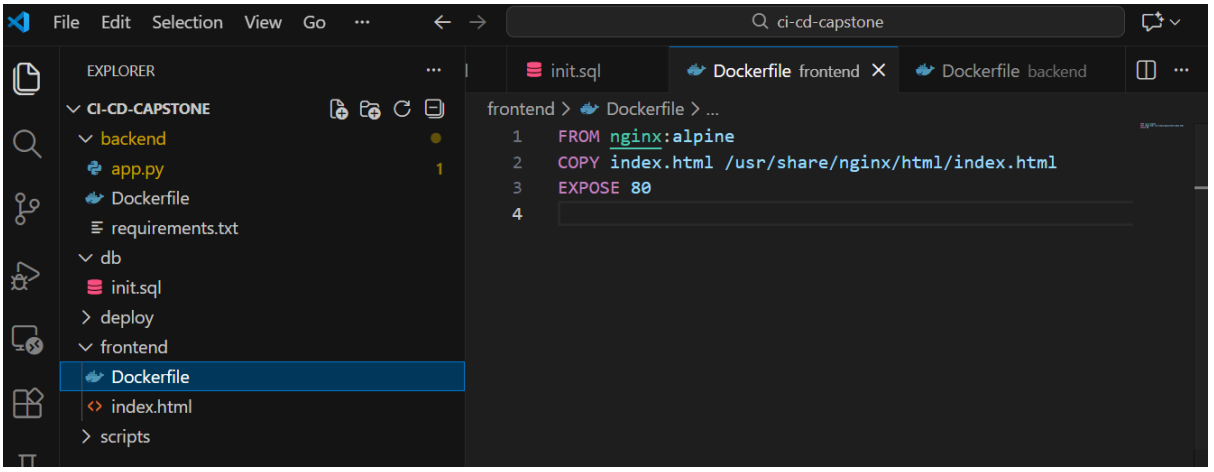
4.1 Backend Dockerfile

- Uses Python 3.11 slim image
- Multi-stage build
- Runs application as non-root user



4.2 Frontend Dockerfile

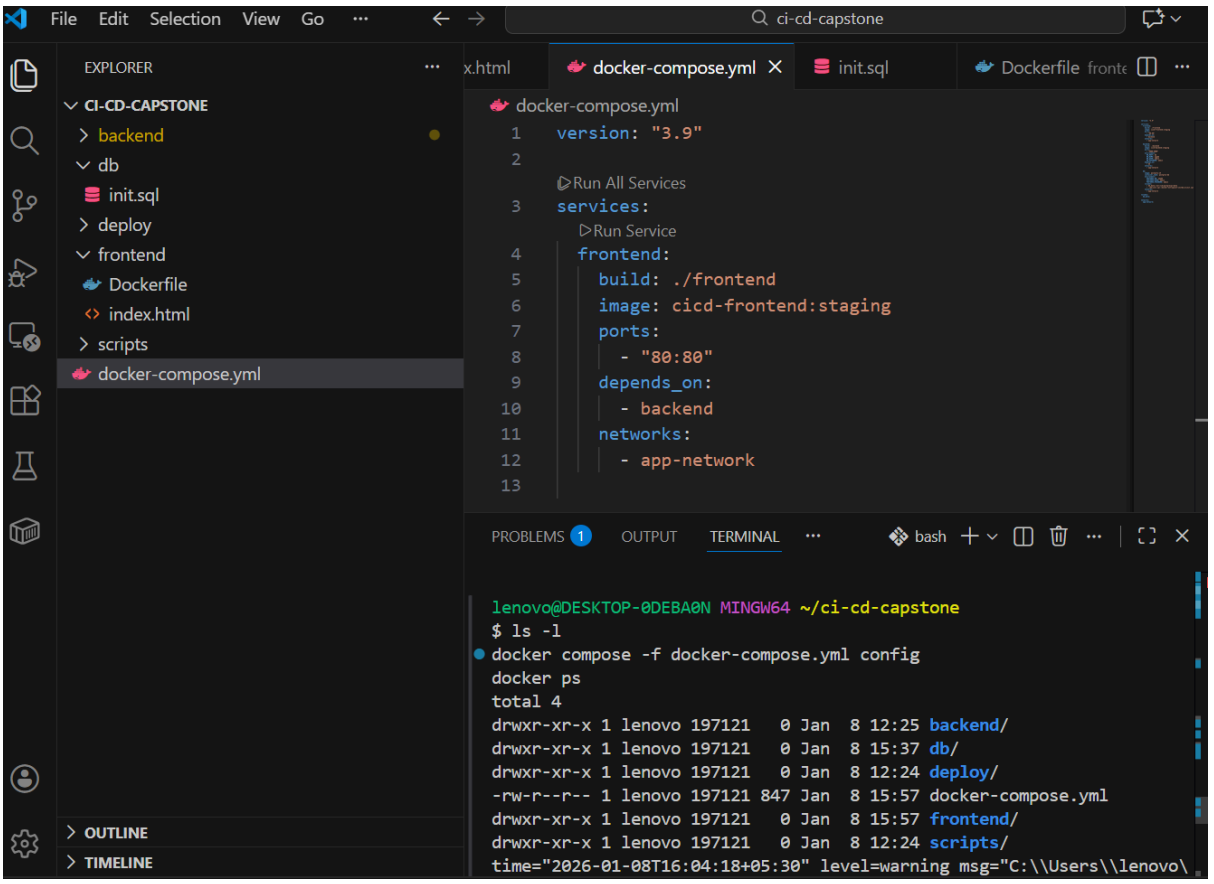
- Uses `nginx:alpine`
- Serves static HTML content



---

5. Docker Compose Configuration

Docker Compose is used to define and validate services locally.



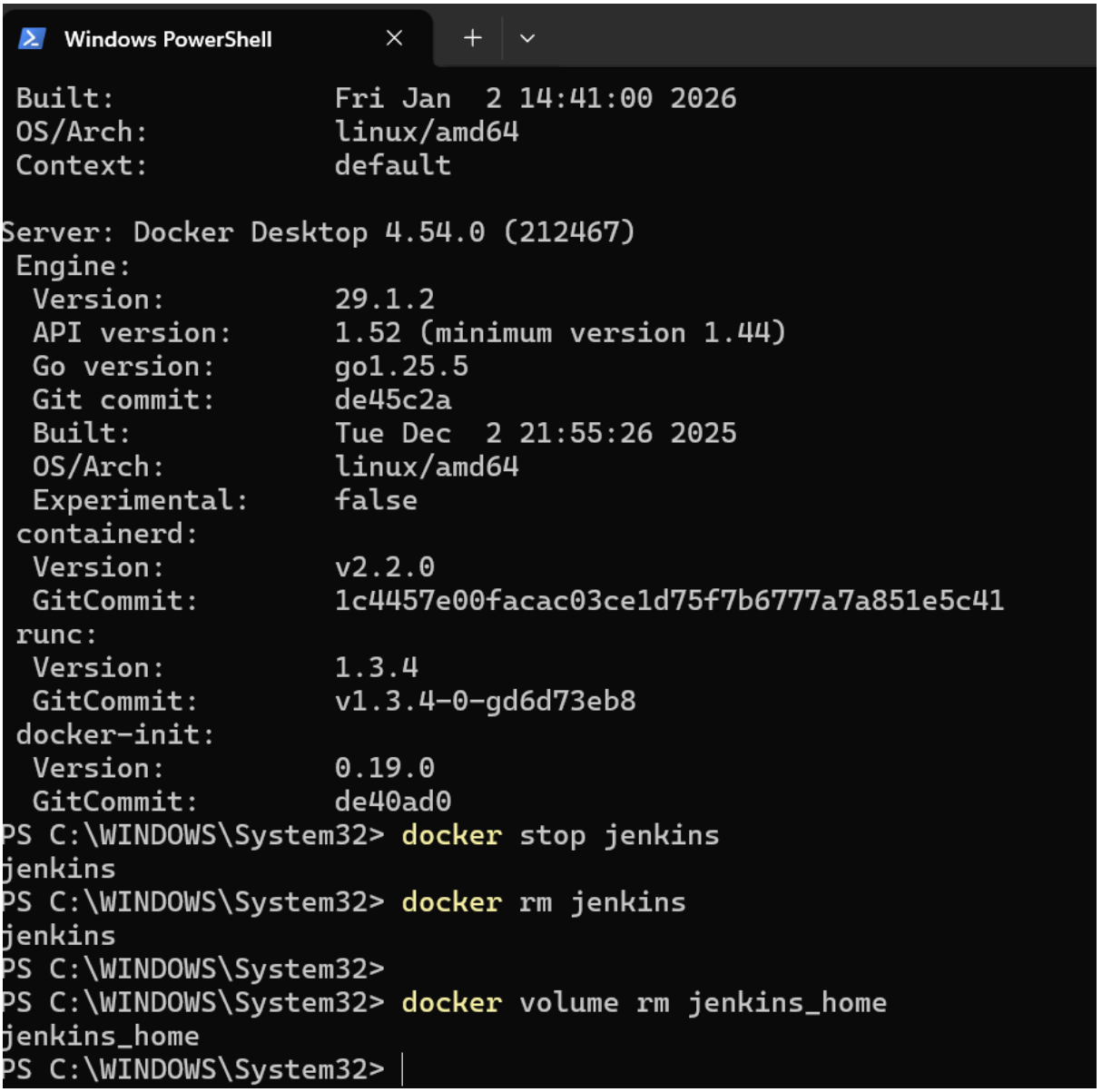
---

6. Jenkins Setup Using Docker

6.1 Remove Existing Jenkins (Clean Start)

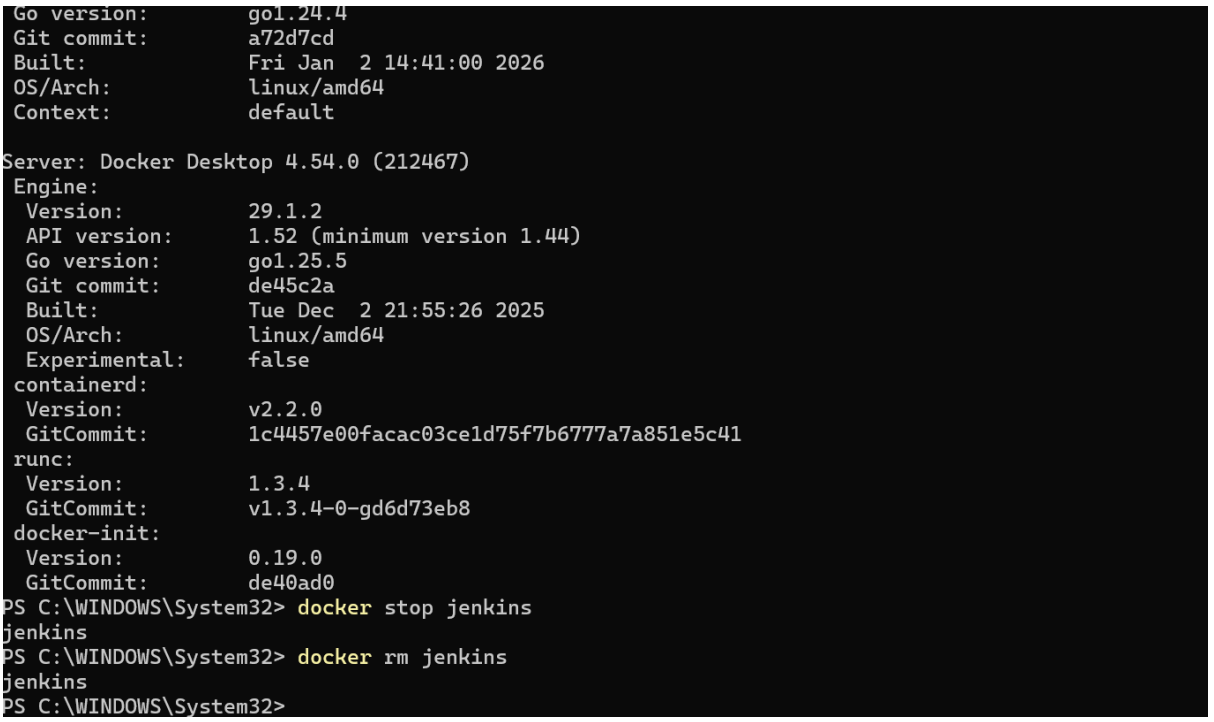
Commands executed:

```
docker stop jenkins
docker rm jenkins
docker volume rm jenkins_home
```



6.2 Run Jenkins Container

```
docker run -d `
  -p 8080:8080 `
  -p 50000:50000 `
  --name jenkins `
  -v jenkins_home:/var/jenkins_home `
  -v /var/run/docker.sock:/var/run/docker.sock `
  jenkins/jenkins:lts
```

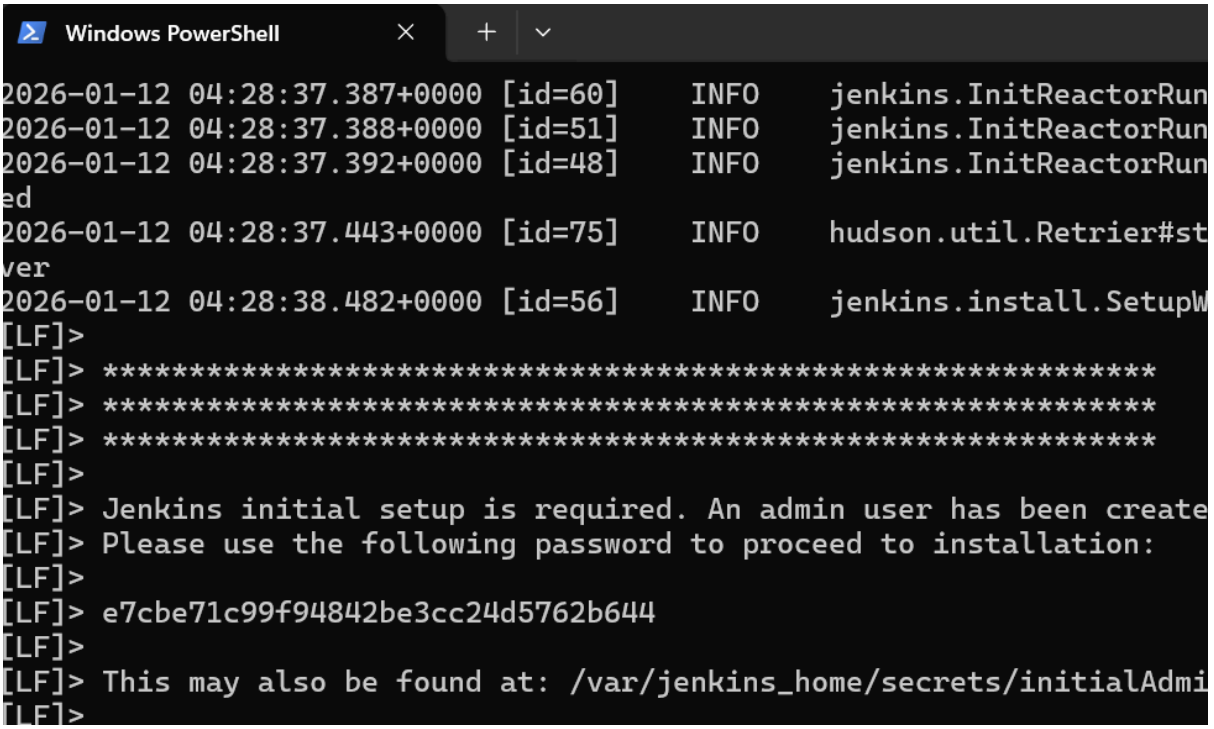


---

7. Jenkins Initial Setup

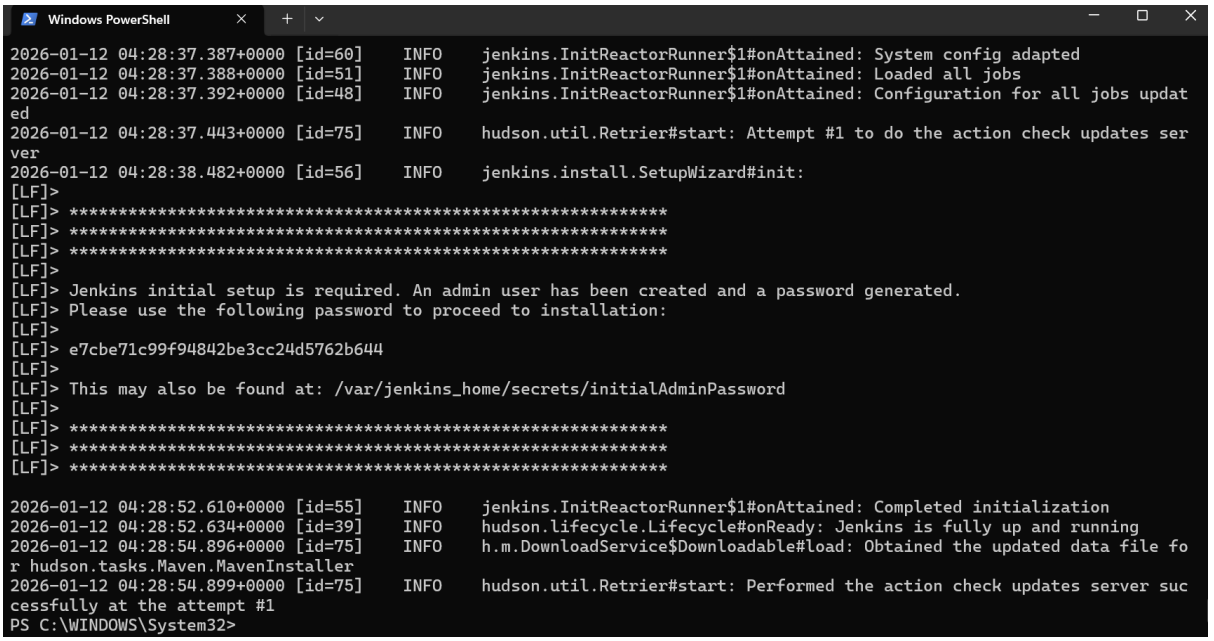
7.1 Retrieve Initial Admin Password

```
docker exec -it jenkins cat
/var/jenkins_home/secrets/initialAdminPassword
```



7.2 Jenkins Unlock Screen

- Access: `http://localhost:8080`
- Paste initial admin password



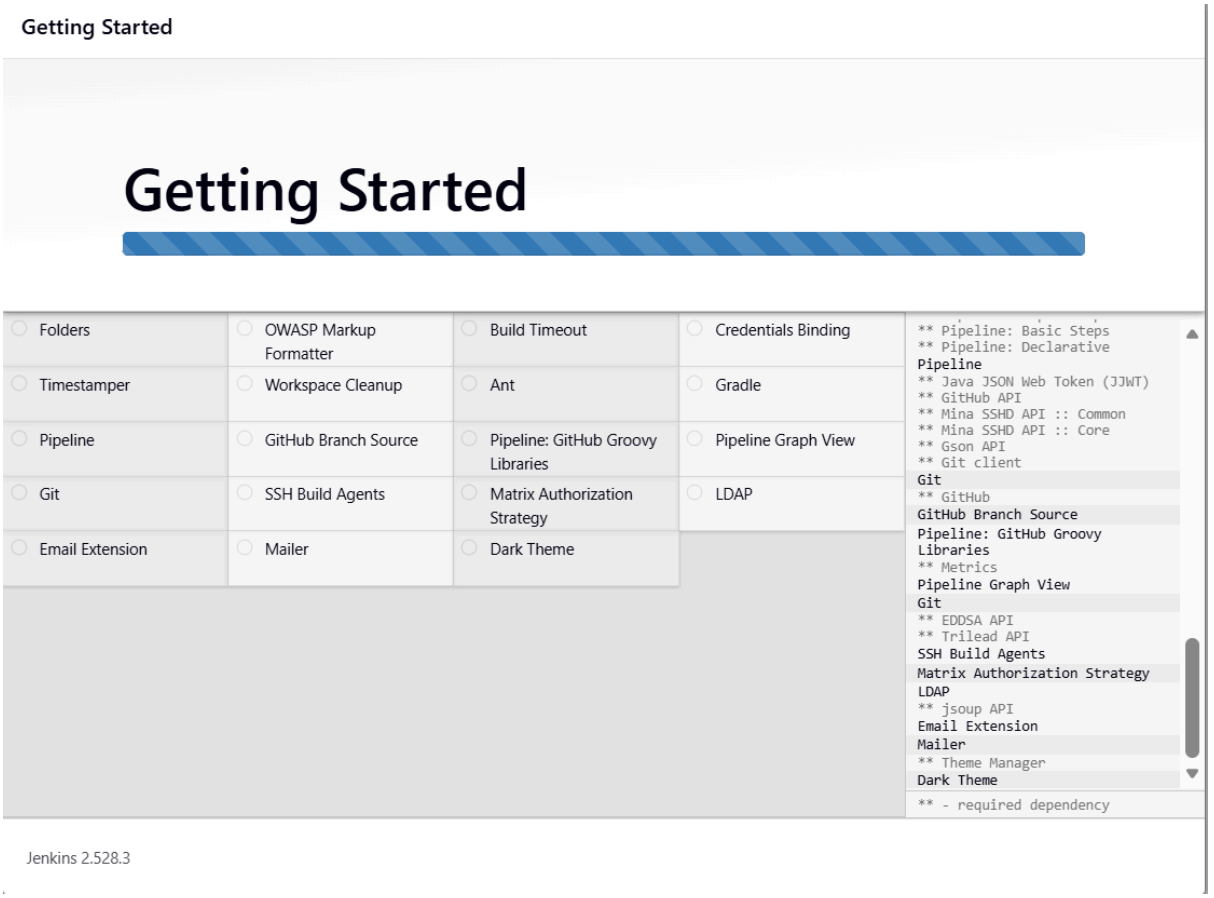
7.3 Install Suggested Plugins

- Selected **Install Suggested Plugins**

```
PS C:\WINDOWS\System32> docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
e7cbe71c99f94842be3cc24d5762b644
PS C:\WINDOWS\System32>
```

7.4 Create First Admin User

Admin user details were configured.

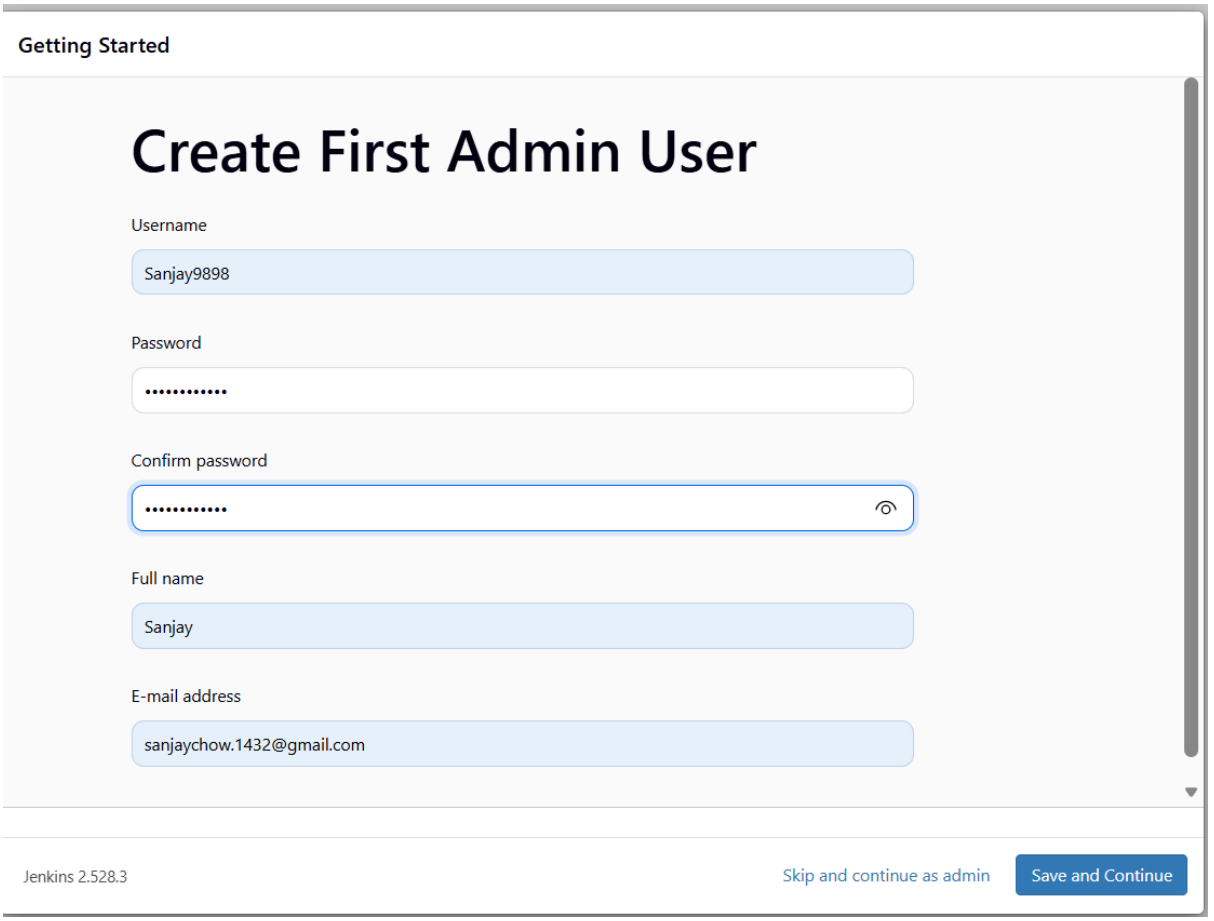


---

8. Jenkins Credentials Configuration

8.1 Docker Hub Credentials

- Type: Username & Password
- Username: `sanjay9898`
- Password: Docker Hub Personal Access Token
- ID: `docker-creds`



---

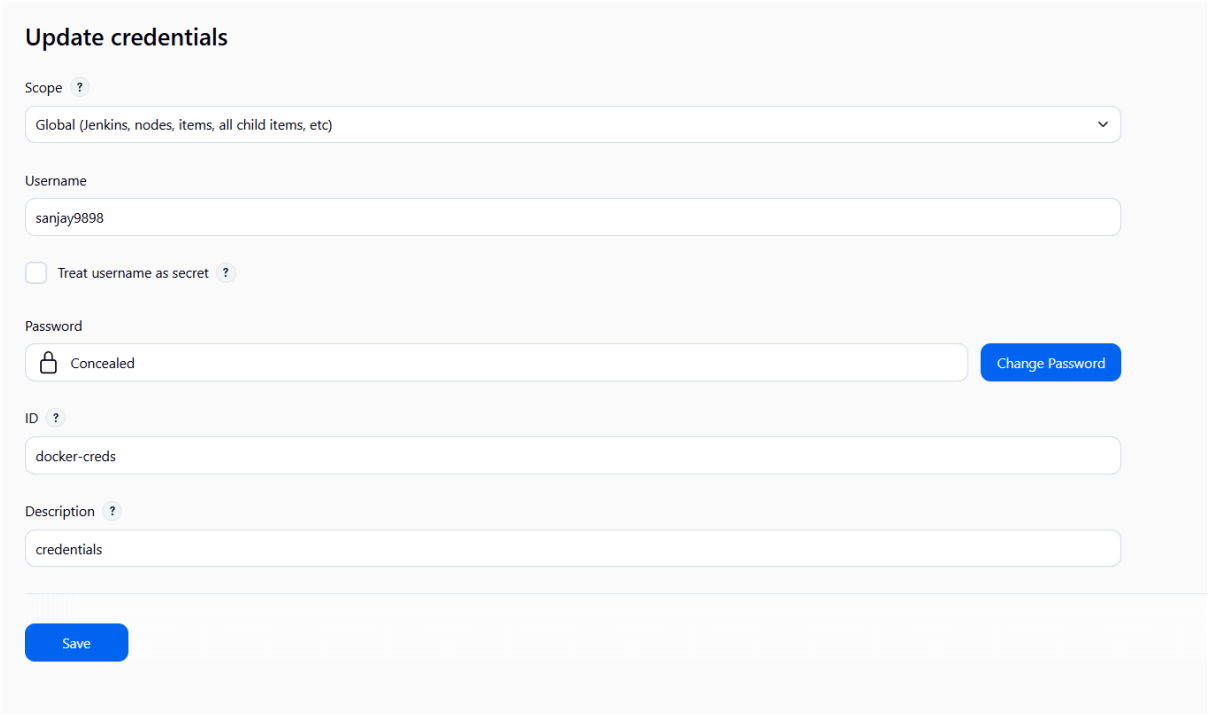
9. Jenkins Pipeline Job Creation

9.1 Create Pipeline Job

Steps:

1. Click **New Item**
2. Enter name: `ci-cd-pipeline`

3. Select **Pipeline**
4. Click **OK**



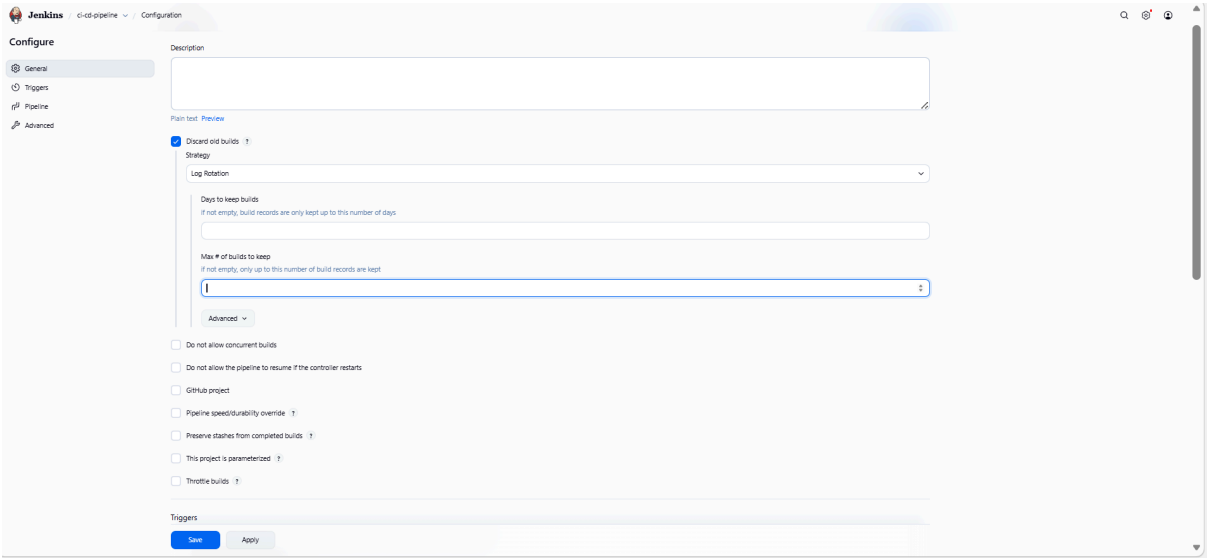
9.2 Configure Pipeline SCM

- Definition: **Pipeline script from SCM**
- SCM: Git

Repository URL:

```
https://github.com/Sanjay12223/ci-cd-capstone.git
```

-
- Credentials: Docker/Git credentials
- Script Path: `Jenkinsfile`



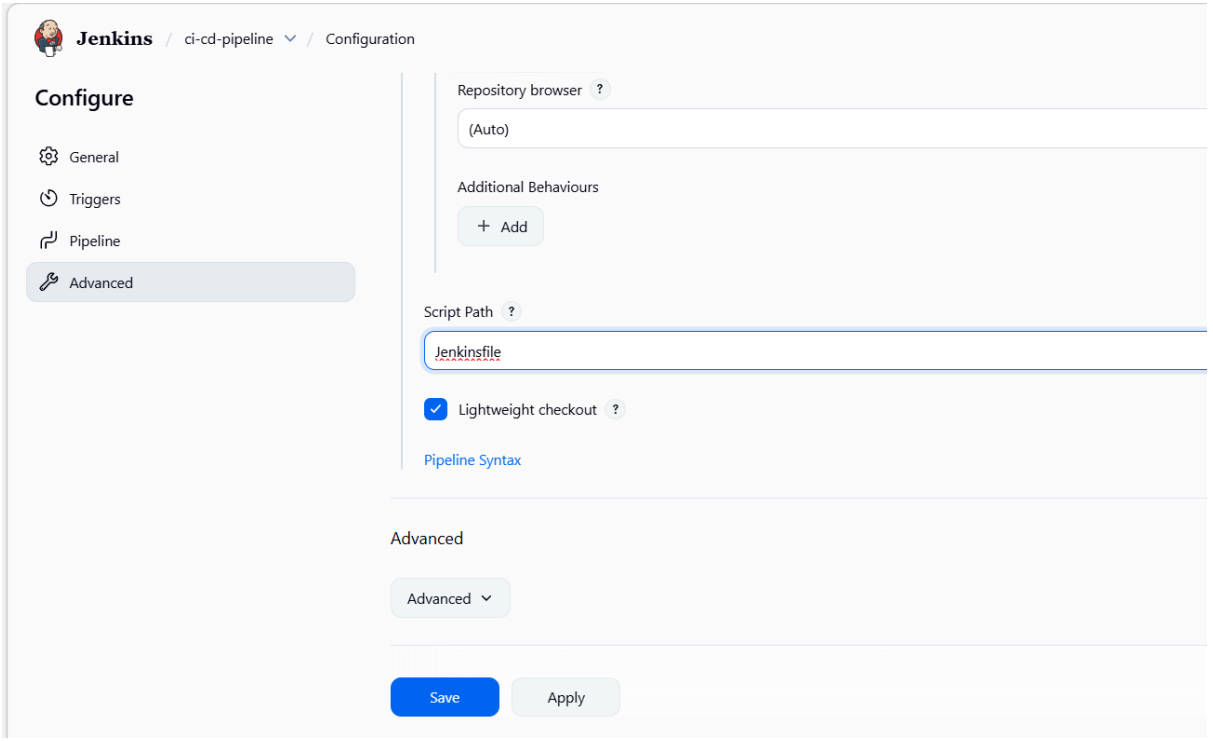
- Script path & lightweight checkout – *(Screenshot 2026-01-12 122406.png)*

---

10. Jenkinsfile (Pipeline Logic)

Pipeline stages implemented:

1. Checkout Source Code
2. Build Backend Docker Image
3. Build Frontend Docker Image
4. Run Unit Tests
5. (Optional) Security Scan
6. Push Images to Docker Hub

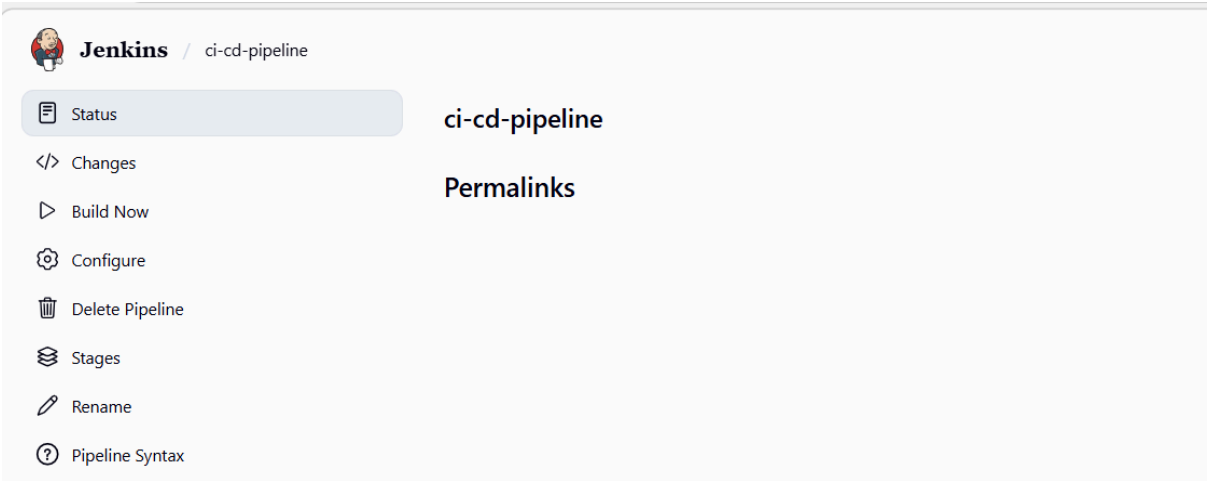


---

11. Pipeline Execution

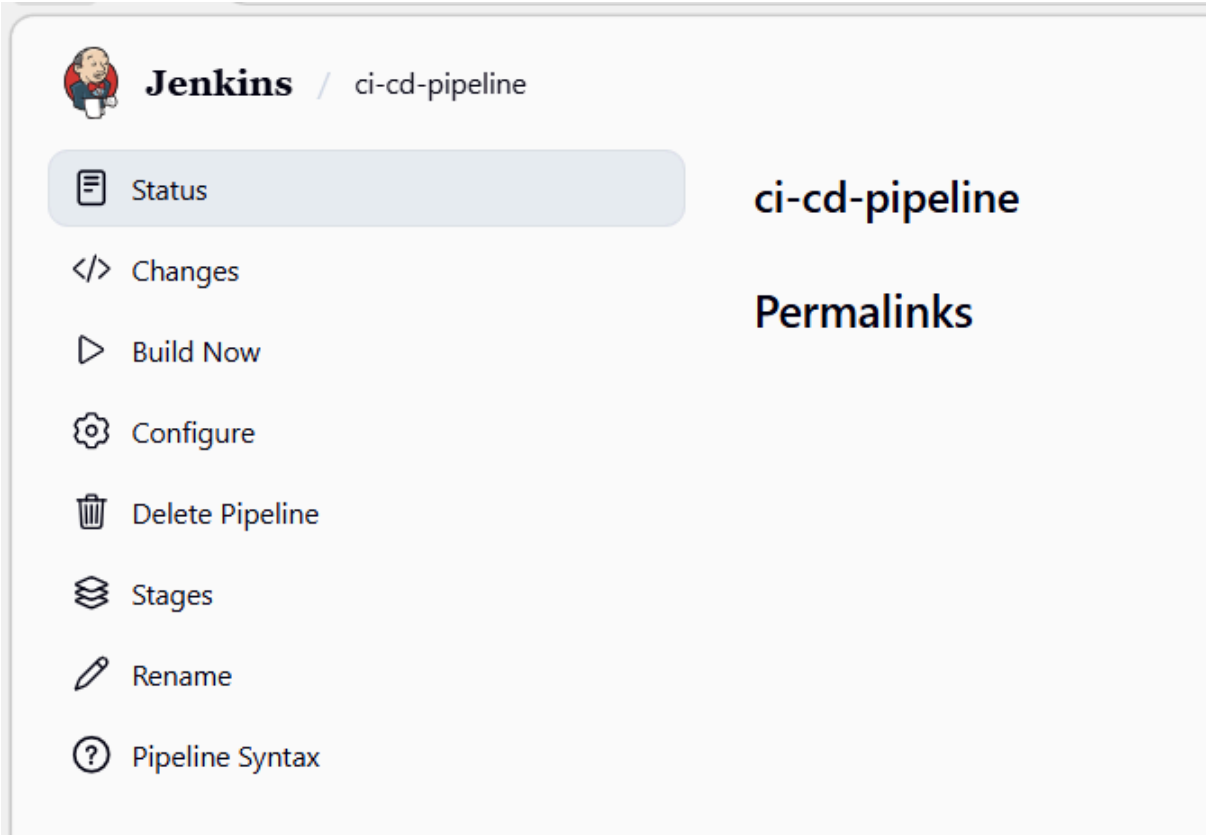
11.1 Trigger Build

- Click **Build Now**



11.2 Pipeline Status Page

- Pipeline stages visible
- Execution logs available per stage



---

12. Unit Test Verification

Status: COMPLETED SUCCESSFULLY

Evidence from console output:

```
docker run --rm <backend-image> python -c "print('Tests passed')"
```

Output:

```
Tests passed
```

This confirms:

- Backend container runs successfully

- Unit test stage executed correctly

---

13. Final Outcome

✔ Completed Successfully

- Jenkins installed via Docker
- Docker socket integration configured
- GitHub repository integrated
- Docker images built successfully
- Unit tests executed successfully
- CI pipeline validated end-to-end

⛔ Optional / Future Enhancements

- Add Trivy installation for security scans
- Add deployment stage (Kubernetes / Docker Compose)
- Enable GitHub webhook triggers

---

14. Conclusion

This CI/CD pipeline demonstrates a **production-grade Jenkins + Docker workflow**, suitable for enterprise use.
The pipeline ensures consistent builds, automated testing, and containerized delivery using industry best practices.