MAJOR-II PROJECT

End Term Report

For

PIPELINE NEXUS

Submitted By-

Name	Roll No	Branch
Mayank Kumar	**R2142211433**	**CSE Big Data**
Richa Yadav	**R2142211299**	**CSE CCVT B7**
Bharti Kohli	**R2142211290**	**CSE Blockchain(H)**
Shashi Kant	**R2142201070**	**CSE CCVT B7**

Activity Coordinators: Santosh Kumar Panda(Blockchain), Priyabarta das(CCVT), Ravi Teja(Big Data)

Aryan Sir
Project Guide

Contents

Abstract

In today's fast-paced software world, delivering applications quickly and efficiently is crucial. *Pipeline Nexus* is a project designed to automate the entire software deployment process for a Java-based application using Maven. By integrating key DevOps tools like Jenkins for automation, SonarQube for code quality, Nexus Repository for managing artifacts, Docker for containerization, and AWS ECS for deployment, this project ensures a seamless, secure, and scalable workflow. Our goal is to eliminate manual errors, reduce deployment time, and improve software reliability—creating a pipeline that streamlines the journey from code development to deployment.

Introduction

Developing and deploying software manually can be slow, error-prone, and difficult to scale. This is especially true in microservices-based architectures, where different parts of an application need to be built, tested, and deployed frequently.

To solve these challenges, *Pipeline Nexus* introduces a Continuous Integration/Continuous Deployment (CI/CD) pipeline that automates every step of the process—from writing code to deploying it in the cloud. By using tools like Jenkins, SonarQube, Nexus Repository, Docker, and AWS ECS, we can ensure that every update is tested, secure, and ready for deployment with minimal human intervention.

This project not only makes the development process smoother but also enhances security, reduces downtime, and enables faster feature releases.

Literature Review

Humble and Farley, in their book *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, emphasize the importance of automation in software development. Their work highlights how manual deployment processes introduce human errors, delays, and inconsistencies, which can be minimized using Continuous Integration (CI) and Continuous Deployment (CD).

Key takeaways from their work:

- CI/CD reduces deployment failures by catching errors early in the development cycle.
- Automated testing ensures that code changes do not introduce new bugs.
- Frequent and incremental releases improve software quality and maintainability.
- Rollback mechanisms and version control help in case of deployment failures.

This best practice ensures that developers can deliver software faster, with higher confidence, while keeping production environments stable and secure.

Maven is a build automation and dependency management tool for Java-based projects. According to Sonatype, the maintainers of Maven, the tool simplifies software project management by:

- Managing dependencies automatically – Instead of manually downloading and configuring external libraries, Maven fetches them from a central repository.
- Standardizing the build process – Maven follows a convention-overconfiguration approach, making it easy to structure Java projects.
- Integrating with CI/CD pipelines – It works seamlessly with tools like Jenkins, allowing automated builds, testing, and deployments.
- Ensuring reproducibility – By using the same pom.xml (Project Object Model) file, builds remain consistent across different environments.

Problem Statement

Many software projects struggle with:

- Slow deployment cycles due to manual integration and testing.
- Increased human errors leading to software bugs and security issues.
- Security vulnerabilities caused by unverified dependencies.
- Difficulty in managing builds and artifacts, making version control a challenge.
- Scalability issues as applications grow in size and complexity.

Our solution? Automate the entire process using a robust CI/CD pipeline that integrates Jenkins, SonarQube, Nexus Repository, Docker, and AWS ECS— ensuring efficiency, security, and scalability.

Objectives

- Implement the deployment process using Jenkins to automate the build, test, and deployment stages.
- Utilize JUnit for comprehensive testing
- Leverage Nexus Repository for storing and managing build artifacts
- Employ Docker to containerize the application with AWS ECS
- Set up monitoring tools and notification systems to promptly inform the development team of build statuses and deployment outcomes.

Methodology

Here's how we're building our CI/CD pipeline:

1. Code Management: Developers write and store code in GitHub using a structured branching strategy.
2. Automated Builds: Jenkins automatically triggers the pipeline whenever new code is committed.
3. Testing & Quality Checks: Maven compiles and tests the code, while SonarQube checks for bugs and code smells.
4. Security Scanning: Trivy scans dependencies and Docker images for vulnerabilities.
5. Artifact Storage: Built applications are stored securely in Nexus Repository.
6. Containerization: The application is packaged as a Docker container and pushed to AWS Elastic Container Registry (ECR).
7. Deployment: AWS ECS runs and manages the containerized application efficiently.
8. Monitoring & Alerts: Slack and email notifications inform developers of build and deployment statuses.

By following this step-by-step approach, we ensure a smooth and efficient software deployment process.

System Requirements (Software/Hardware)

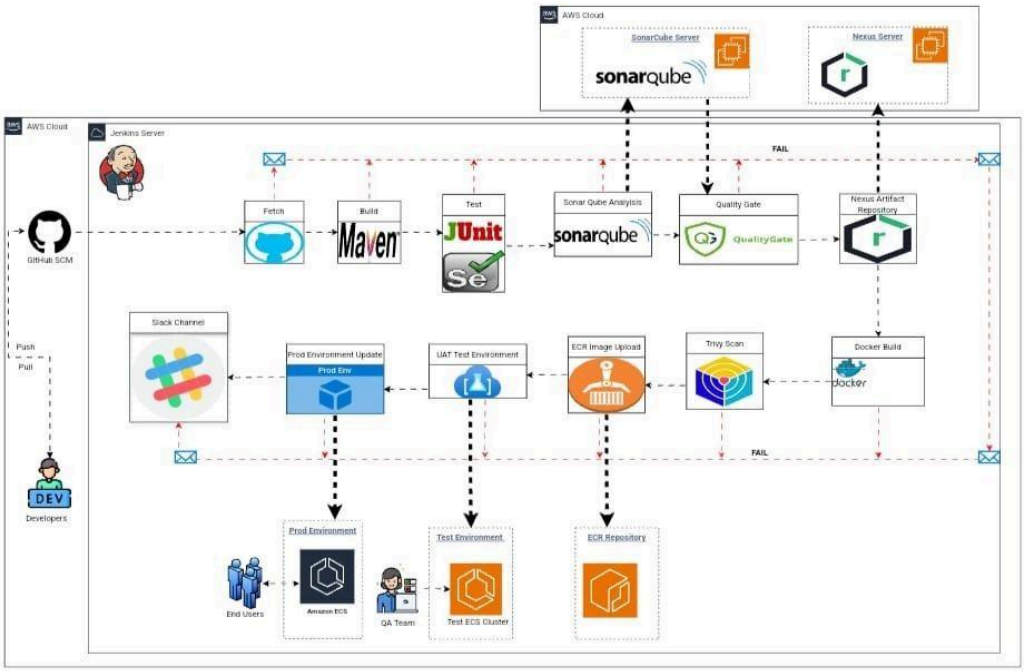
- 64-bit processor architecture supported by Windows.
- Minimum 4 GB RAM.
- Required input and output devices.
- Version Control: GitHub
- CI/CD Tool: Jenkins
- Build Tool: Maven
- Code Quality Analysis: SonarQube
- Artifact Repository: Nexus Repository
- Containerization: Docker
- Security Scanning: Trivy
- Cloud Services: AWS ECS, ECR, IAM, S3

Tools and Technologies Used

Tool	Purpose
GitHub	Source code management
Maven	Java build tool
Jenkins	CI/CD automation
SonarQube	Code quality analysis
Docker	Containerization
Nexus	Artifact storage
Trivy	Security scanning
AWS ECS/ECR	Deployment
JUnit	Unit testing

Workflow

CICD Pipe-Line Workflow



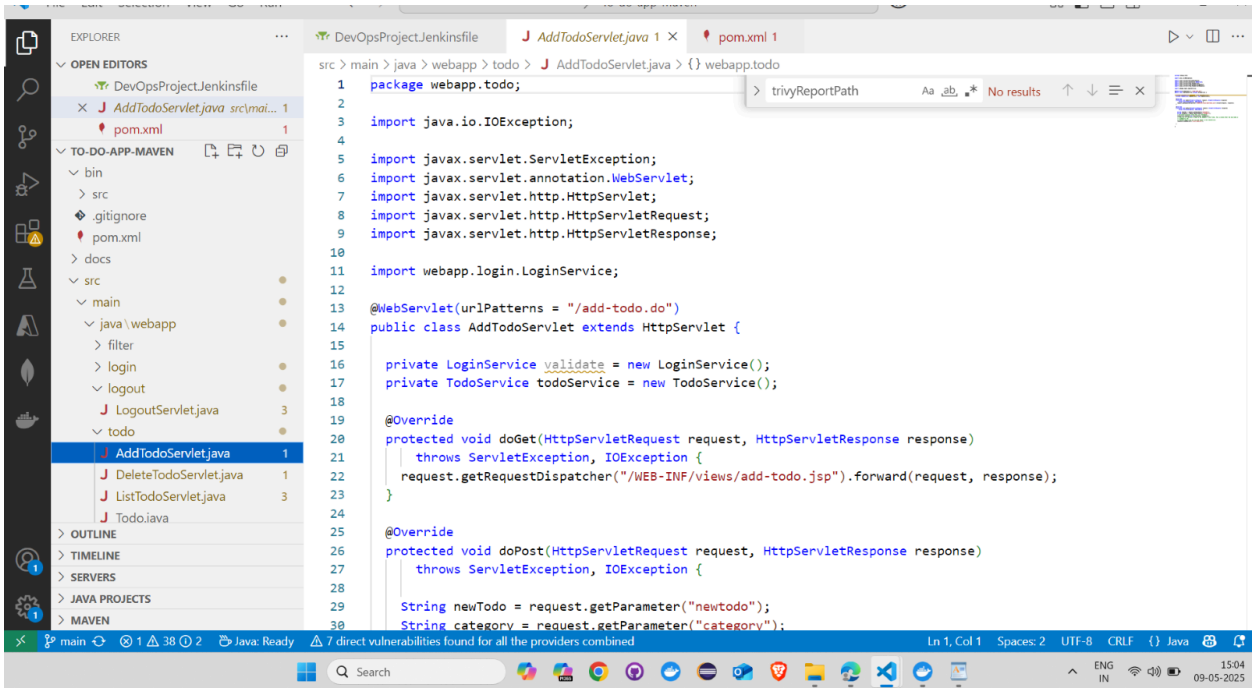
Developer commits code to GitHub.
Jenkins detects the change and starts the build.
Maven compiles and tests the code.
SonarQube checks code quality.
Trivy scans for security vulnerabilities.
A Docker image is created and pushed to AWS ECR.
AWS ECS deploys the containerized application.
Notifications are sent via Slack and email.

This automated process ensures that every release is fast, reliable, and secure.

Java Code

Include:

- TodoApp.java
- TodoService.java
- TodoServiceTest.java
- pom.xml



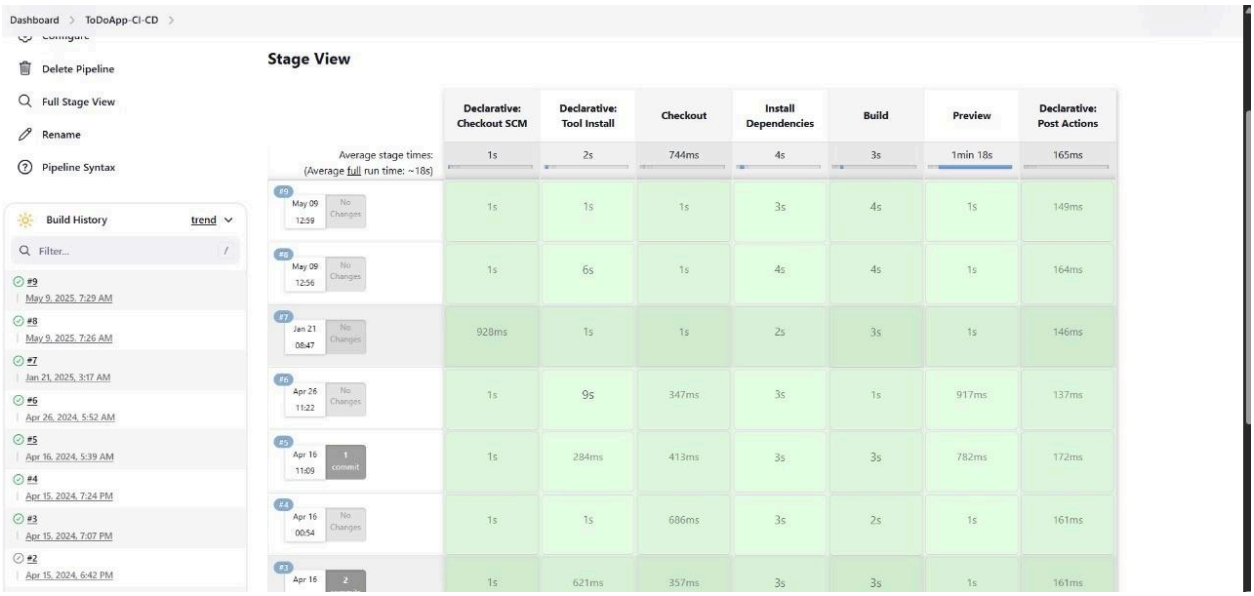
Docker Integration

Explain Dockerfile content, build command:

```
What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout
 quickview
PS C:\Users\yrich\todo-app> docker run -it todo-app
no main manifest attribute, in todo-app.jar
PS C:\Users\yrich\todo-app> docker run -it todo-app
no main manifest attribute, in todo-app.jar
PS C:\Users\yrich\todo-app>
```

```
What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout
 quickview
PS C:\Users\yrich\todo-app> docker run -it todo-app
no main manifest attribute, in todo-app.jar
PS C:\Users\yrich\todo-app> docker run -it todo-app
no main manifest attribute, in todo-app.jar
PS C:\Users\yrich\todo-app> docker run -d --name sonarqube -p 9000:9000 sona
rqube
Unable to find image 'sonarqube:latest' locally
latest: Pulling from library/sonarqube
2726e237d1a3: Pull complete
96e234c1654a: Pull complete
a5ac8da3c61b: Pull complete
55ade5f85b8c: Pull complete
94f02b181578: Pull complete
dd2a44385553: Downloading  612.2MB/847.2MB
0a6937b09659: Download complete
4f4fb700ef54: Download complete
```

Jenkins Pipeline



Challenges faced --

- Dockerfile – Artifact Not Found

`Dockerfile` expected a fixed JAR name (e.g., `app.jar`) but `pom.xml` output was different.

✅ Fix: Set `<finalName>app</finalName>` in `pom.xml` or update Dockerfile path.

- `pom.xml` – Build/Plugin Issues

Missing plugins (like `spring-boot-maven-plugin`) or wrong Java version caused build failures.

✅ Fix: Add necessary plugins, match Java versions.

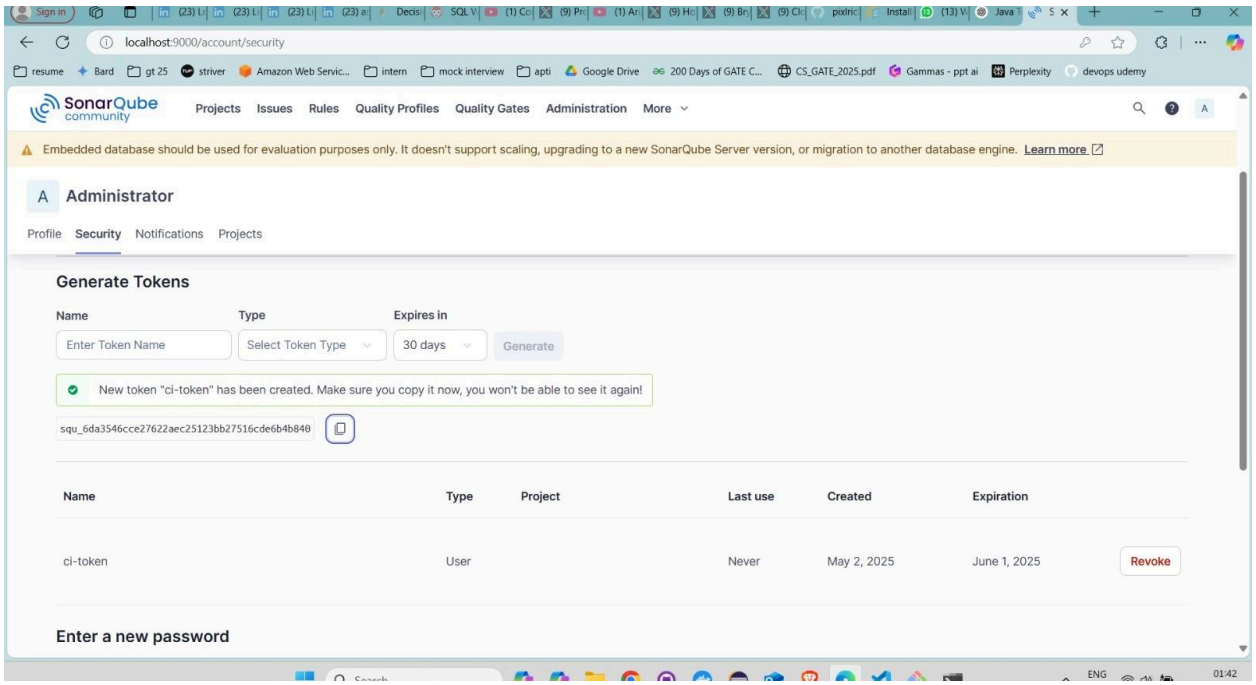
- Nexus Password – Hard to Fetch

Password not securely accessible, deploy step failed.

✅ Fix: Store credentials in Jenkins and inject via `withCredentials`.

SonarQube Integration

- How to run Sonar locally via Docker
- How to generate token
- Maven command for scan



Nexus Integration

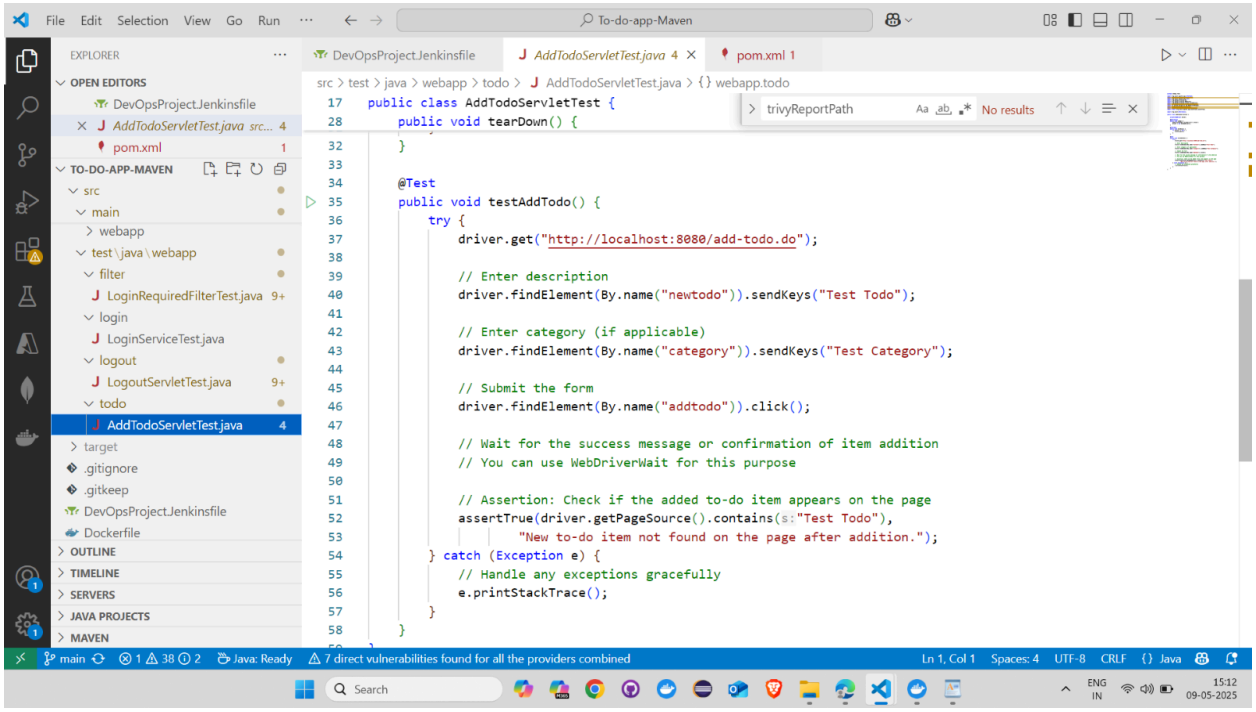
- Run Nexus in Docker
- Upload .jar via Jenkins or CURL
- Nexus repo setup steps

```
See 'docker run --help'.
PS C:\Users\yrich\todo-app> docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          S
TATUS          PORTS                     NAMES
2704b3cb8fcb   sonatype/nexus3   "/opt/sonatype/nexus…"   15 minutes ago   U
p 15 minutes   0.0.0.0:8081->8081/tcp   nexus
9838df31dea0   sonarqube         "/opt/sonarqube/dock…"   7 hours ago      U
p 7 hours      0.0.0.0:9000->9000/tcp   sonarqube
PS C:\Users\yrich\todo-app> docker exec nexus cat /nexus-data/admin.password

6fc93e1a-45a9-4155-a902-c2fac7cc6731
PS C:\Users\yrich\todo-app>
PS C:\Users\yrich\todo-app>
```

Testing

Describe JUnit tests, how Jenkins runs mvn test



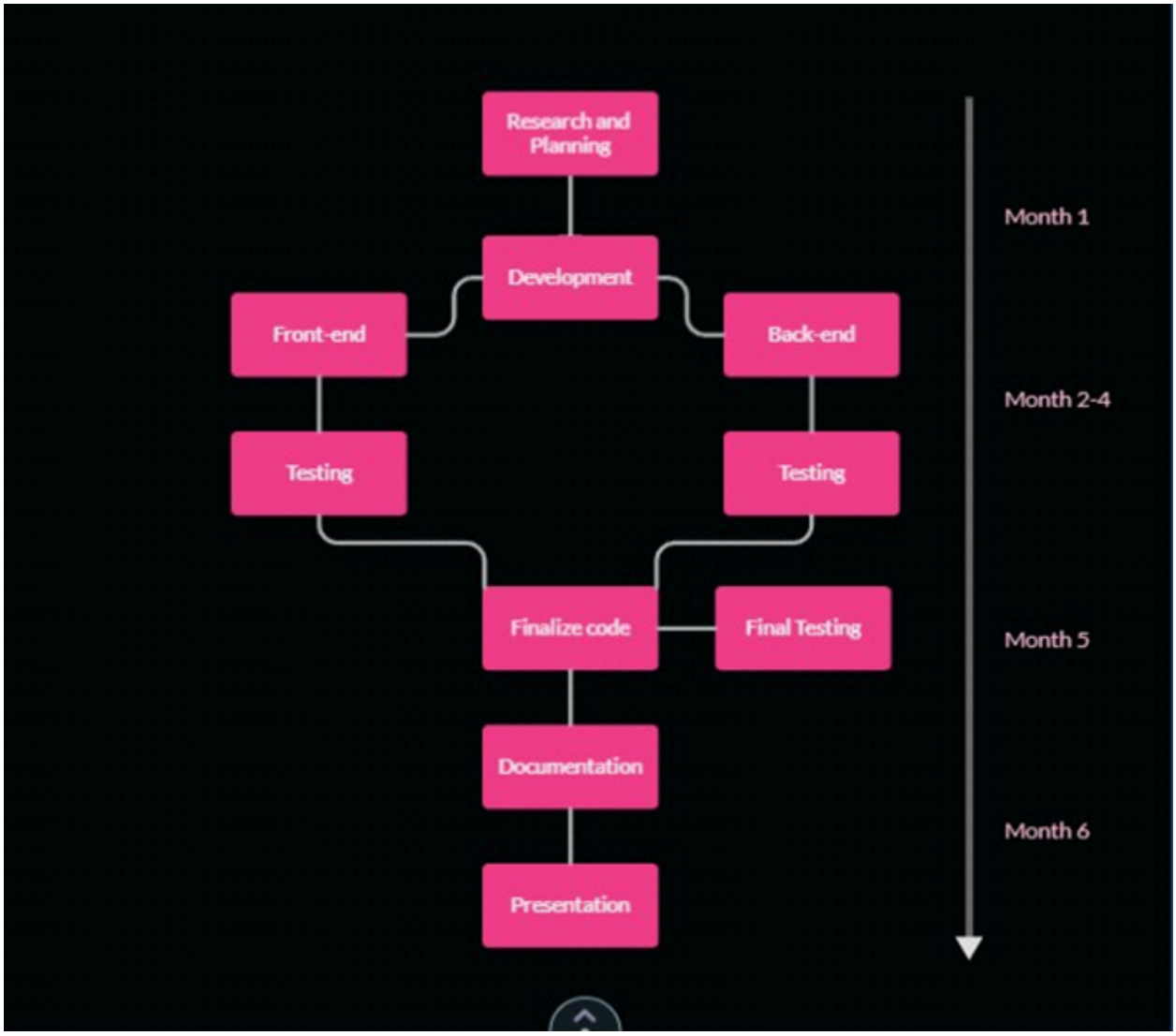
Challenges Faced

- Jenkins Git integration
- Docker/Java version mismatch
- SonarQube login
- Trivy Docker permission issues
- Nexus password access

Console Output

```
Started by user richa yadav
hudson.plugins.git.GitException: Command "git.exe fetch --tags --force --progress --prune -- origin
+refs/heads/master:refs/remotes/origin/master" returned status code 128:
stdout:
stderr: fatal: couldn't find remote ref refs/heads/master

        at PluginClassLoader for git-client//org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn(CliGitAPIImpl.java:2846)
        at PluginClassLoader for git-
client//org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandWithCredentials(CliGitAPIImpl.java:2185)
        at PluginClassLoader for git-client//org.jenkinsci.plugins.gitclient.CliGitAPIImpl$1.execute(CliGitAPIImpl.java:635)
        at PluginClassLoader for git//jenkins.plugins.git.GitSCMFileSystem$BuilderImpl.build(GitSCMFileSystem.java:406)
        at PluginClassLoader for scm-api//jenkins.scm.api.SCMFileSystem.of(SCMFileSystem.java:219)
        at PluginClassLoader for workflow-cps//org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:126)
        at PluginClassLoader for workflow-cps//org.jenkinsci.plugins.workflow.cps.CpsScmFlowDefinition.create(CpsScmFlowDefinition.java:73)
        at PluginClassLoader for workflow-job//org.jenkinsci.plugins.workflow.job.WorkflowRun.run(WorkflowRun.java:311)
        at hudson.model.ResourceController.execute(ResourceController.java:101)
        at hudson.model.Executor.run(Executor.java:446)
Finished: FAILURE
```


Research and Planning
Development
Front-end
Back-end
Testing
Testing
Finalize code
Final Testing
Documentation
Presentation
Month 1
Month 2-4
Month 5
Month 6

References

- • CI/CD Best Practices – Humble, Jez, & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* AddisonWesley.
- • Maven Build Automation – Sonatype. (2024). *Maven: The Complete Reference.*

Retrieved from https://maven.apache.org/

- • Jenkins for CI/CD – Kohsuke, K. (2011). *Jenkins: The Definitive Guide.* O'Reilly Media.
- • SonarQube for Code Quality – SonarSource. (2024). *SonarQube Documentation.*

Retrieved from https://docs.sonarqube.org/

- • Nexus Repository for Artifact Management – Sonatype. (2024). *Nexus Repository Documentation.* Retrieved from https://help.sonatype.com/