DevOps CI/CD Project Report

GitHub Actions CI/CD Pipeline with Security Integration

Name:	Jiya Singhal
Roll Number:	10043
Course:	DevOps
GitHub URL:	https://github.com/jiya-singhal/devops-ci-cd
Submission Date:	January 2026

1. Problem Background & Motivation

In software development, a lot of problems happen because things are done manually. Developers sometimes forget to test their code before pushing it. Security issues in the code or in external libraries go unnoticed. When code is deployed, it might work on one computer but fail on another because of different environments.

These problems cost companies a lot of time and money. Studies show that bugs found in production are much more expensive to fix than bugs found during development. This is where CI/CD comes in.

CI/CD stands for Continuous Integration and Continuous Delivery. It means automating the process of building, testing, and deploying code. Every time a developer pushes code, the pipeline automatically checks everything. This catches problems early when they are easier to fix.

My goal in this project is to build a complete CI/CD pipeline that includes security checks. This is called DevSecOps - adding security into the DevOps pipeline instead of checking security at the end.

2. Application Overview

I built a simple Java application using Spring Boot. Spring Boot is a popular framework for building web applications in Java. I chose Java because the assignment recommended it and it is widely used in the industry.

My application has:

- A /health endpoint - returns OK if the app is running
- A /hello endpoint - returns a greeting message
- A /version endpoint - shows the app version
- A Calculator service - does basic math (add, subtract, multiply, divide)

I included the Calculator service so I could write unit tests for it. The tests check if the math operations work correctly. This demonstrates how the pipeline catches bugs through automated testing.

Technologies used:

Technology	Version	Purpose
Java	17	Programming language
Spring Boot	3.2	Web framework
Maven	3.9	Build tool
JUnit	5	Testing framework
Docker	-	Containerization

GitHub Actions	-	CI/CD platform

3. CI/CD Architecture Diagram

Below is the flow of my CI/CD pipeline. Each box is a stage. If any stage fails, the pipeline stops and doesn't continue to the next stage.

Pipeline Flow:

```
Code Push → Checkout → Setup Java → Linting → Unit Tests → Build JAR ↓ Docker Hub
← Runtime Test ← Image Scan ← Docker Build ← Security Scans (push) (verify)
(Trivy) (CodeQL + OWASP)
```

The pipeline has three main phases: Build & Test (checks if code works), Security (checks for vulnerabilities), and Container (packages and publishes the app).

4. CI/CD Pipeline Design & Stages

My pipeline has 11 stages. Each stage has a specific job. Here is what each stage does and why I included it:

Stage	Tool	What it Does	Why it Matters
Checkout	actions/checkout	Gets code from GitHub	Need code to build
Setup Java	actions/setup-java	Installs Java 17	Need Java to compile
Linting	Checkstyle	Checks code style	Keeps code clean
Unit Tests	JUnit 5	Runs tests	Catches bugs early
Build	Maven	Creates JAR file	Packages the app
SAST	CodeQL	Scans code for security issues	Finds vulnerabilities
SCA	OWASP DC	Scans dependencies	Finds risky libraries
Docker Build	Docker	Creates container	Packages for deployment
Image Scan	Trivy	Scans container	Finds OS vulnerabilities
Runtime Test	curl	Tests if container works	Verifies it runs
Push	Docker Hub	Publishes image	Ready for deployment

Why this order?

I put fast stages first. Linting takes 5 seconds but CodeQL takes 5 minutes. If linting fails, why wait for security scans? This is called "fail-fast". Also, security scans run before Docker push so vulnerable images never get published.

5. Security & Quality Controls

Security is built into my pipeline at multiple points. This is called "shift-left security" because we check security early (left side of the timeline) instead of at the end.

SAST (Static Application Security Testing):

I use GitHub CodeQL for SAST. It analyzes my source code without running it. It looks for patterns that indicate security problems like SQL injection or XSS attacks. The results show up in the GitHub Security tab.

SCA (Software Composition Analysis):

I use OWASP Dependency Check for SCA. It scans all the libraries my project uses (listed in pom.xml) and checks if any have known vulnerabilities. For example, it would catch issues like Log4Shell.

Container Scanning:

I use Trivy to scan my Docker image. Even if my code is secure, the base image (Alpine Linux) might have vulnerable packages. Trivy checks for these.

Other Security Measures:

- Docker container runs as non-root user (limits damage if hacked)
- Secrets stored in GitHub Secrets (not in code)
- Multi-stage Docker build (smaller image = less attack surface)

6. Results & Observations

Pipeline Execution:

My pipeline runs successfully on every push to the main branch. All stages pass and the final Docker image is pushed to Docker Hub.

Metric	Result
Total pipeline time	~6-8 minutes
Unit tests	12 tests, all passing
Checkstyle violations	0
CodeQL findings	No critical issues
OWASP findings	No critical CVEs
Trivy findings	No critical vulnerabilities
Final image size	~180 MB

What I Observed:

- First run of CodeQL is slow because it builds a database, but later runs are faster
- Maven caching helps - dependencies don't download every time
- Multi-stage Docker build reduced image size significantly
- The /health endpoint is essential for the runtime test to work

7. Limitations & Future Improvements

Current Limitations:

- Only one environment (no separate staging/production)
- Security scans don't fail the build on warnings, only critical issues
- No automatic rollback if deployment fails
- No performance testing included

Future Improvements:

- Add deployment to Kubernetes
- Add DAST (Dynamic Application Security Testing)
- Add automatic dependency updates with Dependabot
- Add load testing with tools like JMeter
- Add Slack notifications for pipeline status

8. Conclusion

In this project, I successfully built a CI/CD pipeline using GitHub Actions. The pipeline automatically builds, tests, scans for security issues, containerizes, and publishes my application whenever I push code.

I learned that DevOps is not just about tools - it's about catching problems early and automating repetitive tasks. The shift-left approach means security is everyone's responsibility, not just a final check before release.

The most important thing I understood is WHY each stage exists. Linting prevents messy code, tests prevent bugs, SAST finds code vulnerabilities, SCA finds library vulnerabilities, and container scanning finds OS vulnerabilities. Together, they create multiple layers of protection.

References:

- GitHub Actions Documentation - https://docs.github.com/en/actions
- OWASP Top 10 - https://owasp.org/www-project-top-ten/
- Docker Documentation - https://docs.docker.com/
- Spring Boot Guide - https://spring.io/projects/spring-boot