DevOps CI/CD Project Report

Automated Pipeline with DevSecOps using GitHub Actions

Name:	Kushi Varadaraj
Roll Number:	10035
Course:	DevOps
GitHub URL:	https://github.com/kushivaradaraj/devops-project
Submission Date:	January 2026

1. Problem Background & Motivation

Software development without automation leads to many issues. Developers might push code without running tests, leading to broken builds. Security vulnerabilities in the code or in third-party libraries often go undetected until the application is already in production, where fixing them is expensive.

Another common problem is "it works on my machine" - code that runs fine on a developer's laptop but fails when deployed somewhere else. This happens because of differences in environments, versions, and configurations.

CI/CD (Continuous Integration and Continuous Delivery) solves these problems by automating the entire process. Every time code is pushed, it is automatically built, tested, checked for security issues, and packaged. This is much faster and more reliable than doing everything manually.

In this project, I implemented a CI/CD pipeline that also includes security scanning. This approach is called DevSecOps, where security is integrated into the development process from the beginning, not added at the end.

2. Application Overview

For this project, I created a web application using Java and Spring Boot. Spring Boot is one of the most popular frameworks for building Java applications, so I chose it to make the project more relevant to real-world scenarios.

The application has these components:

- /health endpoint - Returns OK to show the app is running
- /hello endpoint - Returns a greeting to the user
- /version endpoint - Returns the current version number
- Calculator service - Performs basic math operations

The Calculator service is used to demonstrate unit testing. I wrote tests that verify each operation (add, subtract, multiply, divide) works correctly. These tests run automatically in the pipeline.

Technology Stack:

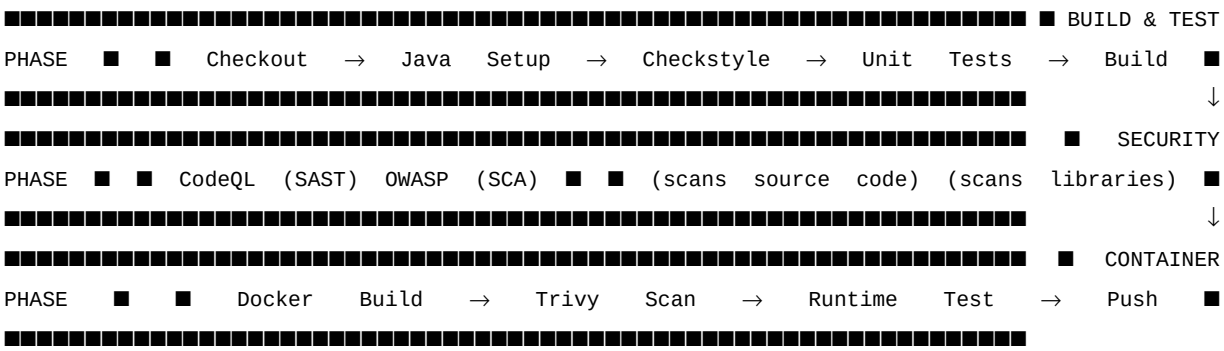
Component	Technology	Why I Chose It
Language	Java 17	Recommended in assignment
Framework	Spring Boot 3.2	Industry standard
Build Tool	Maven	Widely used with Java
Testing	JUnit 5	Default for Spring Boot
Container	Docker	For consistent deployment

CI/CD	GitHub Actions	Free and integrated with GitHub

3. CI/CD Architecture Diagram

The pipeline is triggered whenever I push code to GitHub. It goes through several stages, and if any stage fails, the pipeline stops immediately. This prevents bad code from moving forward.

Pipeline Architecture:



4. CI/CD Pipeline Design & Stages

Each stage in my pipeline has a specific purpose. I will explain what each stage does and why it is important:

Stage	Tool	Purpose	Why Important
Checkout	actions/checkout	Download code	Need source code
Setup Java	actions/setup-java	Install JDK	Required for build
Lint	Checkstyle	Check code style	Maintain quality
Test	JUnit	Run unit tests	Find bugs early
Build	Maven	Create JAR	Package application
SAST	CodeQL	Scan code	Find security issues
SCA	OWASP	Scan libraries	Find vulnerable deps
Docker	Docker	Build image	Create container
Scan	Trivy	Scan image	Check for CVEs
Test	curl	Run container	Verify it works
Push	Docker Hub	Publish image	Deploy ready

Stage Ordering:

The stages are ordered carefully. Fast checks like linting run first, so if there's a simple mistake, we know immediately without waiting for slow security scans. This is the "fail-fast" principle. Security stages run before Docker push, ensuring we never publish a vulnerable image.

5. Security & Quality Controls

My pipeline includes security at multiple points. This is called shift-left security because we check for security issues early in the process.

SAST - Static Application Security Testing:

CodeQL scans my source code for security vulnerabilities. It looks for common problems like SQL injection, cross-site scripting (XSS), and hardcoded passwords. It does this without running the code - it just analyzes the patterns.

SCA - Software Composition Analysis:

OWASP Dependency Check scans all the libraries my project uses. These are listed in the pom.xml file. It checks each library against a database of known vulnerabilities (CVEs). This is important because most applications use many external libraries.

Container Scanning:

Trivy scans the final Docker image. Even if my code is secure, the base operating system image might have vulnerabilities. Trivy checks for these and reports any issues found.

Additional Security Measures:

- Non-root user in Docker container
- Secrets stored securely in GitHub (not in code)
- Multi-stage Docker build for smaller, safer images

6. Results & Observations

After setting up everything, my pipeline runs successfully. Here are the results I observed:

What I Measured	Result
Pipeline duration	Around 6-8 minutes total
Number of unit tests	12 tests
Test pass rate	100%
Checkstyle issues	None
CodeQL critical findings	None
OWASP critical findings	None
Trivy critical findings	None
Docker image size	About 180 MB

Things I Noticed:

- CodeQL is slow the first time because it builds a database
- Caching Maven dependencies makes builds much faster
- The health endpoint is needed for runtime testing to work
- Multi-stage Docker builds really do make smaller images

7. Limitations & Future Improvements

What Could Be Better:

- No separate staging and production environments
- Security scans only fail on critical issues, not warnings
- No automatic rollback mechanism
- No performance or load testing
- No notifications when pipeline fails

What I Would Add Next:

- Kubernetes deployment stage
- DAST (Dynamic testing of running application)
- Dependabot for automatic library updates
- Slack or email notifications
- Performance testing with JMeter or similar tool

8. Conclusion

This project helped me understand how CI/CD pipelines work in practice. I learned that automation is not just about convenience - it's about catching problems early before they

become expensive to fix.

The most valuable lesson was understanding why each stage exists. It's not enough to just copy a pipeline configuration. You need to know what each part does and what problems it prevents. For example, SAST finds bugs in your code, SCA finds bugs in libraries you use, and Trivy finds bugs in the container OS.

DevSecOps means security is not an afterthought. By building security checks into the pipeline, every code change is automatically checked. This is much better than only checking security before a big release.

References:

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