DevOps CI/CD Project Proposal

Student Name:	[YOUR NAME HERE]
Scaler Student ID:	[YOUR STUDENT ID]
Submission Date:	January 2026
GitHub Repository:	https://github.com/[USERNAME]/devops-cicd-demo

1. Project Title

Production-Grade CI/CD Pipeline with GitHub Actions: Implementing DevSecOps Best Practices

2. Application Description

The application is a Java Spring Boot web service that provides REST API endpoints. It includes a health check endpoint for monitoring, a greeting service demonstrating request handling, and a calculator service with business logic for demonstrating unit testing capabilities. The application is containerized using Docker and follows twelve-factor app principles for cloud-native deployment.

Key Features:

- REST API endpoints for health monitoring and functionality testing
- Spring Boot 3.2 with Java 17 for modern Java development
- Calculator service demonstrating testable business logic
- Docker containerization with multi-stage builds
- Non-root container execution for security

3. CI/CD Problem Statement

In modern software development, organizations face significant challenges in ensuring code quality, security, and reliable deployments. Manual testing is error-prone and slow. Security vulnerabilities often go undetected until production. Inconsistent environments cause deployment failures. This project addresses these challenges by implementing an automated CI/CD pipeline that integrates quality checks, security scanning, and containerization at every stage of the software delivery process.

Problems Addressed:

- Manual code reviews miss security vulnerabilities (solved by SAST/CodeQL)
- Vulnerable dependencies go undetected (solved by SCA/OWASP Dependency Check)
- Container images ship with vulnerabilities (solved by Trivy scanning)
- Inconsistent builds across environments (solved by Docker containerization)
- Late defect detection increases costs (solved by shift-left testing)

4. CI/CD Stages and Justification

Stage	Tool/Technology	Justification
Code Checkout	actions/checkout@v4	Retrieves source code for all subsequent operations
Setup Runtime	actions/setup-java@v4	Provides consistent Java 17 environment with dependency caching
Linting	Maven Checkstyle	Enforces coding standards, prevents technical debt accumulation
Unit Testing	JUnit 5 + Maven	Validates business logic, prevents regressions, ensures code quality
Build	Maven	Compiles and packages application into deployable JAR artifact
SAST	GitHub CodeQL	Static analysis for OWASP Top 10 vulnerabilities in source code
SCA	OWASP Dependency Check	Scans dependencies for known CVEs in third-party libraries
Docker Build	Docker Buildx	Creates consistent, portable container image with multi-stage builds
Image Scan	Trivy	Detects OS and library vulnerabilities in container image
Runtime Test	curl + Docker	Validates container starts correctly and endpoints respond
Registry Push	Docker Hub	Publishes verified image for deployment pipelines

5. Expected Outcomes

- **Quality Assurance:** Every code change passes automated linting and testing before merge
- **Security Integration:** Vulnerabilities detected early through SAST, SCA, and container scanning
- **Deployment Reliability:** Container images validated before registry push
- **Traceability:** Complete audit trail via GitHub Actions logs and artifacts
- **Developer Experience:** Fast feedback on code quality within minutes of push

6. DevSecOps Principles Applied

This project implements DevSecOps by integrating security at every pipeline stage (shift-left security). Security is not a separate phase but embedded throughout: SAST during code analysis, SCA during dependency resolution, and container scanning before deployment. Failures at any security gate prevent progression, ensuring only secure artifacts reach production.