Team Quaternary:

ST10019838 - Damian Dare

ST10019972 - Guillaume Swanevelder

ST10091991 - Christiaan Versfeld

ST10158660 - Ruan Zwarts

Module: XBCAD7319

Lecturer: Ms. Mmaphuti Matau

Documentation: DEVOPS

# Table of Contents

[Table of Contents 2](#_Toc182131013)

[INTRODUCTION 3](#_Toc182131014)

[DEVOPS 3](#_Toc182131015)

[**1.1** GitHub Actions Pipeline 3](#_Toc182131016)

[1.2 Detailed Description of Each Pipeline Step 5](#_Toc182131017)

[Testing and Reports 6](#_Toc182131018)

[Security and Static Code Analysis Tools 6](#_Toc182131019)

[REFERENCE LIST 7](#_Toc182131020)

# INTRODUCTION

DevOps practices are crucial in modern software development, enabling rapid and reliable application deployment through Continuous Integration and Continuous Deployment (CI/CD). This project employs DevOps methodologies to streamline development processes, improve code quality, and maintain application security. By leveraging GitHub Actions, a powerful CI/CD tool, the team ensures that every code change undergoes rigorous testing, static analysis, and security checks before being deployed to production.

The GitHub Actions pipeline integrates automated workflows for unit testing, code analysis, security scans, and deployment, creating a robust development environment that fosters collaboration and minimizes errors. Each stage of the pipeline has been carefully designed to meet the project's quality, performance, and security requirements. This documentation provides a detailed description of the DevOps pipeline, its structure, and how each phase contributes to the stability and maintainability of the application. Additionally, it includes a flowchart illustrating the steps in the GitHub Actions pipeline, along with reports from testing and analysis tools to validate the pipeline's effectiveness.

By incorporating DevOps into the project, the team can deliver consistent, high-quality updates to the application, reducing the risks associated with manual deployment and enhancing overall project reliability.

# DEVOPS

The DevOps practices for this project focus on automating and streamlining the development, testing, and deployment processes using GitHub Actions. This continuous integration and continuous delivery (CI/CD) pipeline ensures that code updates are consistently tested, built, and deployed, promoting high-quality, secure, and reliable application development.

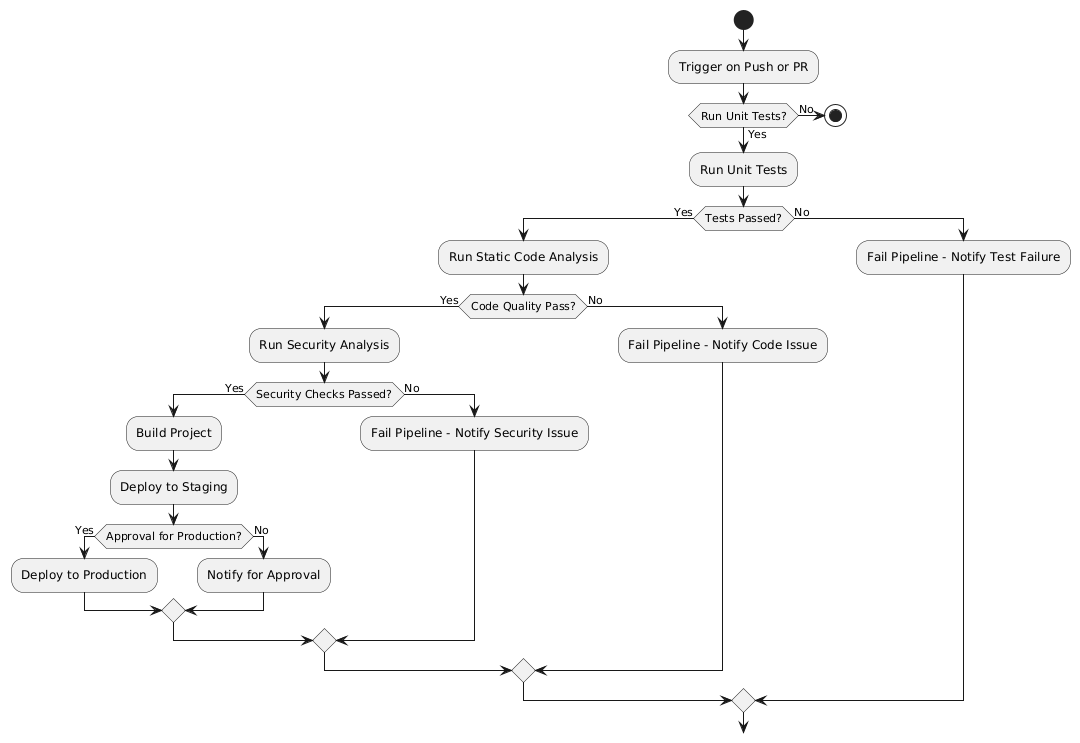
## GitHub Actions Pipeline

The GitHub Actions Pipeline is structured to follow best practices in CI/CD, covering code integration, automated testing, static code analysis, security checks, and deployment. Each step is essential for maintaining code integrity, application security, and streamlined deployment to cloud infrastructure.

The GitHub Actions pipeline for this project is configured as follows:

1. **Code Integration:** Triggered on every code push, pull request, or merge to ensure code changes are integrated consistently.
2. **Unit Testing:** Runs automated tests to verify that each component functions correctly in isolation.
3. Static Code Analysis: Uses third-party tools like SonarQube or CodeClimate to analyse code for potential issues, such as code smells, performance bottlenecks, or poor coding practices.
4. **Security Testing:** Uses a security analysis tool (e.g., GitHub's CodeQL or Snyk) to check for vulnerabilities in dependencies or code.
5. **Build and Deployment:** Builds the project and deploys it to staging for review. After approval, it is deployed to production, with automated testing at each step.
6. **Monitoring and Notifications:** Sends notifications on pipeline results and monitors key metrics to assess performance and stability.

This flow chart depicts the CI/CD pipeline in GitHub Actions, illustrating each step and its sequential dependencies.



(PlantText, 2024)

## Detailed Description of Each Pipeline Step

1. **Trigger on Code Push or PR:** The pipeline begins on every code push, pull request, or merge to the main branches. This ensures that each code change is validated through testing and analysis.
2. **Run Unit Tests:** Unit tests confirm that each component of the code functions as intended. Early detection of regressions is aided by automated testing.

Output: A test report is generated, summarizing passing and failing tests, allowing developers to identify and fix issues.

1. **Static Code Analysis:** Third-party tools like SonarQube or CodeClimate assess code for maintainability and performance issues. This step enforces coding standards and best practices, improving code quality and readability.

Output: A report highlighting code issues such as duplications, code smells, and complexity.

1. **Security Testing:** Security checks run using tools like GitHub's CodeQL or Snyk, which scan code and dependencies for known vulnerabilities. This process reduces the risk of introducing security flaws into production.

Output: A security report lists identified vulnerabilities, their severity, and recommended actions.

1. **Build and Deployment:** If tests and analyses pass, the code is built and deployed to the staging environment. This deployment allows for final testing before moving to production.

Output: A successful build and staging deployment, enabling the team to review the application.

1. **Approval for Production Deployment:** Following staging, the deployment to production is started by an automated or manual approval process. Updates are pushed to live environments with this last deployment.

Output: Notifications are sent to confirm deployment success, marking the process complete.

1. **Monitoring and Notifications:** After deployment, monitoring tools keep track of application health and key metrics. Alerts are set up to notify the team of potential issues.

Output: Real-time alerts on application status, performance, and availability.

# Testing and Reports

1. **Unit Test Report:** A unit test report that details the results of every test is produced by each pipeline run. Test failures give developers information to help them quickly fix problems
2. **Static Code Analysis Report:** Following each study, a static analysis report is produced that includes scores for maintainability, duplication, and code smells.
3. **Security Testing Report:** A security report with identified vulnerabilities, impacted locations, and suggested corrective measures is produced by the pipeline. The security of the application depends on this report.

# Security and Static Code Analysis Tools

1. Static Code Analysis: To preserve code quality and find problems like code smells, vulnerabilities, and duplications, use SonarQube or CodeClimate.

Report: These tools generate reports highlighting code quality metrics, including cyclomatic complexity, maintainability, and technical debt.

1. Security Testing: GitHub CodeQL or Snyk, which identifies vulnerabilities in both code and dependencies.

Report: Each pipeline run outputs a security report with detected vulnerabilities, categorized by severity level (e.g., critical, high, medium, low).

# REFERENCE LIST

PlantText, 2024. *PlantUML.* [Online]   
Available at: https://www.planttext.com/  
[Accessed 24 10 2024].