DevOps-Assignment

Date Submitted: 11/04/2025

T00224519 - Rew Guardiano

Contents

[Introduction 2](#_Toc191980082)

[Chapter 1: Code and Tools 2](#_Toc191980083)

[GitHub: 2](#_Toc191980084)

[Jenkins: 2](#_Toc191980085)

[Docker: 2](#_Toc191980086)

[SonarQube: 2](#_Toc191980087)

[Maven: 2](#_Toc191980088)

[GitHub Actions: 3](#_Toc191980089)

[1.2 Flow Diagram 3](#_Toc191980090)

[Chapter 2: Continuous Integration 4](#_Toc191980091)

[2.1 Setup of GitHub Repository 4](#_Toc191980092)

[2.2 Jenkins 4](#_Toc191980093)

[2.2.1 Automate and build test execution 4](#_Toc191980094)

[2.2.2 Configure Workflow file to: 4](#_Toc191980095)

[Chapter 3: Building the Code and Configuring the Build Pipeline 5](#_Toc191980096)

[Chapter 4: End-to-end automation of the application delivery lifecycle 6](#_Toc191980097)

[Chapter 5: Cloud Provisioning and Configuration Management 7](#_Toc191980098)

[Chapter 6: Deploying Application (AWS, Azure, and Docker) 8](#_Toc191980099)

[Chapter 7: Monitoring Infrastructure and Applications 9](#_Toc191980100)

[Chapter 8: Orchestrating Application Deployment 10](#_Toc191980101)

[References 11](#_Toc191980102)

## Introduction

The tools used in this assignment are:

* GitHub
* Jenkins
* Docker
* SonarQube
* Maven
* GitHub Actions
* Jira

## Chapter 1: Code and Tools

GitHub: serves as the main source code repository for managing the version history and application code. It offers issue tracking, pull requests, and collaborative tools.

Jenkins: As the Continuous Integration (CI) tool, Jenkins will automate the application development, testing, and deployment process. It will be set up to run unit tests, start automated deployment pipelines, and grab code from GitHub.

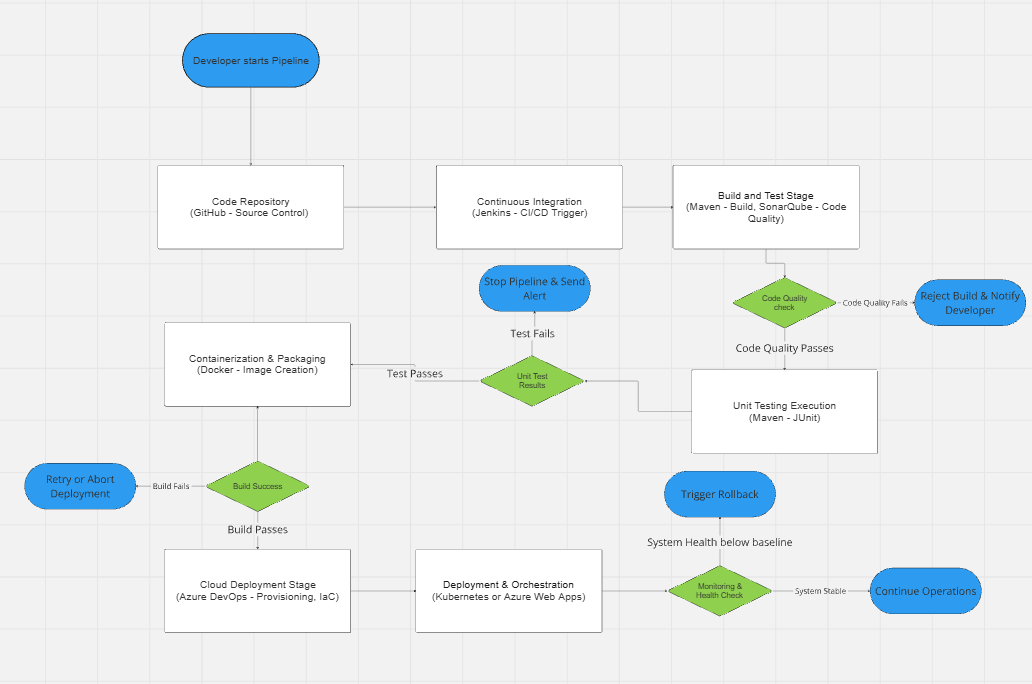
Docker: allows the program to be containerized, guaranteeing consistency across many environments.

SonarQube: included in the process to assess the quality of the code, find security flaws, and guarantee adherence to coding standards. SonarQube will be integrated in GitHub Actions will be set up to perform static code analysis following every commit.

Maven: Mostly used for Java applications, Maven is a strong build automation tool. It manages builds, packaging, and project dependencies. The Java application in this project will be compiled using Maven, unit tested, and packaged as a deployable artifact. The repository's pom.xml file lists the plugins and dependencies required for the build procedure.

GitHub Actions: CI/CD capabilities are inherent to GitHub due to GitHub Actions. Jenkins will manage the main CI/CD pipeline, but GitHub Actions can be used for other automatic processes including launching builds in response to pull requests, performing security scans, and checking dependencies. Its smooth integration with GitHub repositories improves the effectiveness of automation.

### 1.2 Flow Diagram



## Chapter 2: Continuous Integration

## 2.1 Setup of GitHub Repository

The repository used was a well-maintained Java application from GitHub that was forked and is used as the test application. In the setup of the repository, ensuring that a pom.xml file was included in the repository for Maven builds. This repository will be connected within Jenkins.

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 1. GitHub Repository of Java Application.*

## 

## 2.2 Jenkins

### 2.2.1 Setup

To provide a portable and isolated environment to connect easily with other applications, Jenkins will be set up inside a Docker container. Jenkins was launched using the following command docker desktop:

A screen shot of a computer

AI-generated content may be incorrect.

*Figure 2. Command to pull docker image*

A black background with white text

AI-generated content may be incorrect.

*Figure 3. Command to run the Jenkins Container*

### 2.2.2 Automate and build test execution

Created Freestyle project item in Jenkins which is in localhost:8080 A screenshot of a computer

AI-generated content may be incorrect.

*Figure 4. Creating project item*

Jenkins integration between the GitHub Repository was accomplished through the connection with docker. This involved installing maven, java and pulling and pushing the repository in the Jenkins docker container

A screenshot of a computer program

AI-generated content may be incorrect.

*Figure 5. Docker terminal configuration of Jenkins container*

- Setup Maven and run the Unit test

- Perform the SonarQube analysis of the repository with the code.

- Build the docker repo (in docker website)

## Chapter 3: Building the Code and Configuring the Build Pipeline

* Configure GitHub Actions and Jenkins for CI/CD
* Automate tests using **Maven & JUnit**

## Chapter 4: End-to-end automation of the application delivery lifecycle

* Implement **Docker** for containerization
* Automate builds and security scans using **SonarQube**

## Chapter 5: Cloud Provisioning and Configuration Management

Use **Terraform or Ansible** for setting up cloud resources. Ask about this

## Chapter 6: Deploying Application (AWS, Azure, and Docker)

* Deploy via GitHub Actions
* Use **Docker Swarm or Kubernetes**

## Chapter 7: Monitoring Infrastructure and Applications

* Use **Prometheus, Grafana, or ELK Stack** for logging and monitoring
* Configure alert notifications

## Chapter 8: Orchestrating Application Deployment

Implement **CI/CD best practices** for **zero-downtime deployments**

# References

**There are no sources in the current document.**