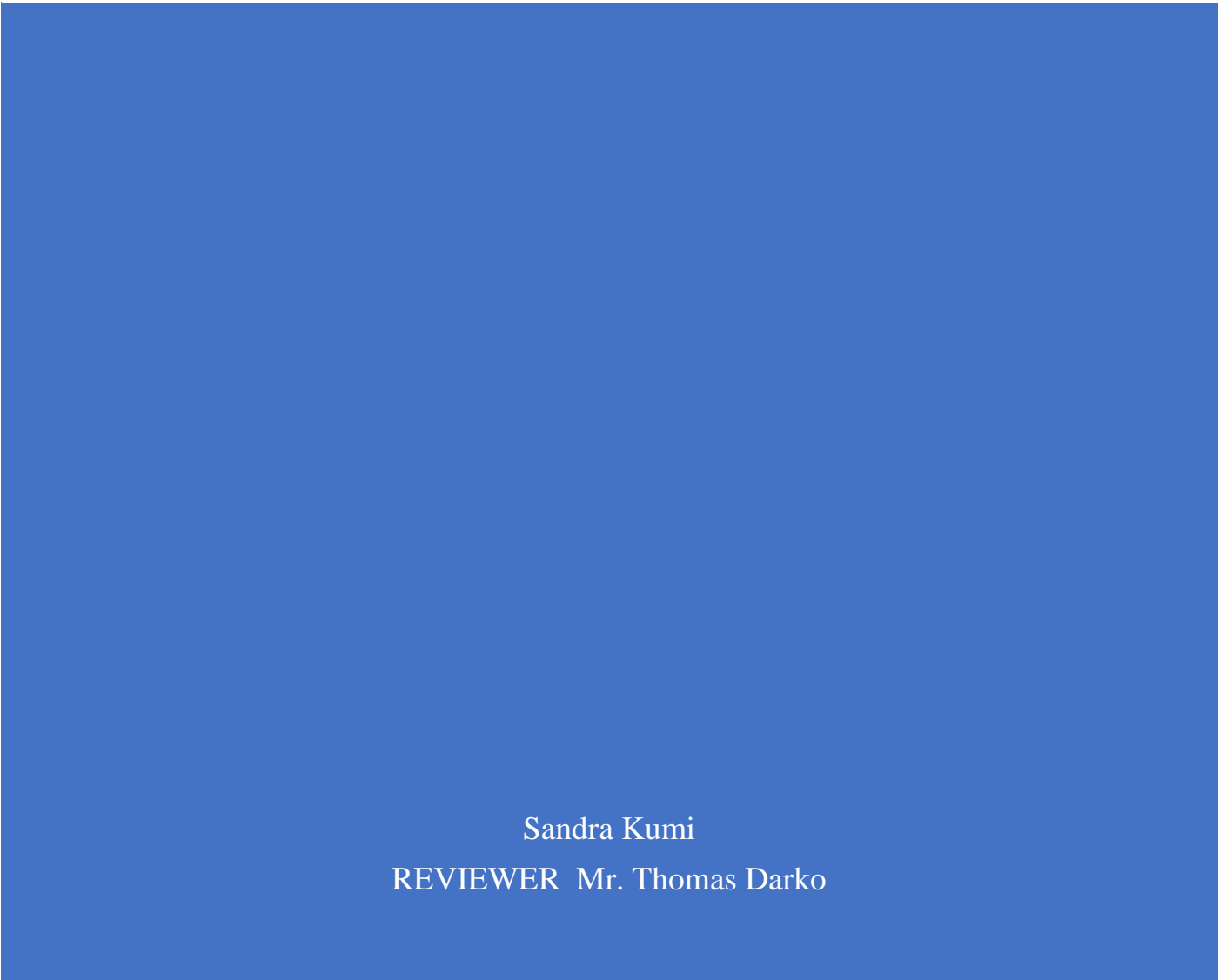


CI/CD PIPELINE AND DEPLOYMENT PROCESS SUMMARY



Sandra Kumi
REVIEWER Mr. Thomas Darko

CI/CD Pipeline and Deployment Process Summary

This document provides a summary of the Continuous Integration and Continuous Deployment (CI/CD) pipeline used for building, testing, and deploying a Java application. The pipeline is integrated with Docker and Jenkins to automate the entire process, from code changes to deployment.

Pipeline Overview

1. **Checkout Stage**:

- The pipeline starts by checking out the code from the specified branch of the GitHub repository. The 'jenkins' branch is used in this case to ensure the CI/CD process works with the correct version of the application.

2. **Build Stage**:

- In this stage, the Maven Wrapper ('mvnw') is used to compile the Java application and package it into a JAR file. The build is executed with 'mvnw clean package', which ensures that all dependencies are resolved and the project is built from scratch.

3. **Test Stage**:

- Unit tests are executed using the Maven Wrapper. This ensures that any changes to the codebase do not break existing functionality. The command 'mvnw test' runs the tests defined in the project.

4. **Docker Compose Down**:

- Before deploying, the pipeline ensures any previously running Docker containers are stopped and removed to avoid conflicts using 'docker compose down --remove-orphans'.

5. **Docker Compose Up --build**:

- Finally, Docker Compose is used to build and deploy the application in the target environment. The command ``docker compose up --build -d`` forces a rebuild of the Docker images before starting the containers in detached mode.

Docker Image and Deployment

- A Dockerfile is included in the project to package the Java application into a Docker image. The Dockerfile uses the OpenJDK 21 Alpine image as the base and copies the built JAR file to the image for execution.
- The application is then deployed as a Docker container using Docker Compose, which defines how services (such as the Java app) are built, configured, and networked together.

Takeaways and Lessons Learned

1. Automating the CI/CD pipeline with Jenkins and Docker ensures that the application is continuously integrated and deployed without manual intervention.
2. Using Docker Compose simplifies deployment by managing services as defined in the configuration file, making the deployment process consistent across different environments.
3. Key challenges included configuring the Docker environment and ensuring that Maven was correctly set up to build the project.
4. It is critical to ensure that the environment has the necessary tools (e.g., JDK, Maven) pre-installed to avoid issues during the build process.