Introduction

This document provides a clear explanation of the CI/CD pipeline created for automating the build, test, and deployment processes of a Java application. The pipeline uses Jenkins, Maven, Docker, and Docker Compose.

Definitions

Continuous Integration (CI)

Definition: Continuous Integration is a development practice where code changes are automatically tested and merged into a shared repository frequently. This practice helps identify and fix bugs early, improving the quality of the software and speeding up the development process.

Continuous Deployment (CD)

Definition: Continuous Deployment is a strategy where every change that passes automated tests is automatically deployed to production. This allows for frequent releases and ensures that the software is always in a deployable state.

Jenkins

Definition: Jenkins is an open-source automation server that supports building, deploying, and automating any project. It helps automate various stages of the development lifecycle, including CI/CD processes.

Pipeline Overview

The pipeline consists of the following stages:

1. **Checkout**
2. **Build with Maven**
3. **Run Tests**
4. **Build Docker Image**
5. **Push Docker Image**
6. **Deploy with Docker Compose**

Pipeline Stages and Definitions

1. Checkout

- **Objective:** Retrieve the latest code from the Git repository.
- **Definition:** This stage uses Jenkins to pull the code from the Git repository to ensure that the pipeline works with the most recent version of the code.

```
Unset
stage('Checkout') {
    steps {
        git branch: 'master', url:
'https://github.com/virgile-am/BasicsOfDockerLab.git'
    }
}
```

2. Build with Maven

- **Objective:** Compile and package the Java application.
- **Definition:** Maven is used to compile the application code and package it into a deployable format (e.g., a JAR file). This step ensures that the code is built correctly and is ready for testing.

```
Unset
stage('Build with Maven') {
    steps {
        script {
            echo 'Building with Maven...'
            bat 'mvn clean package -DskipTests'
        }
    }
}
```

3. Run Tests

- **Objective:** Execute unit tests to verify code quality.
- **Definition:** This stage runs the unit tests to check if the code behaves as expected. Passing tests are essential for ensuring code quality before deployment.

```
Unset
stage('Run Tests') {
    steps {
        script {
            echo 'Running tests...'
            bat 'mvn test'
        }
    }
}
```

4. Build Docker Image

- **Objective:** Create a Docker image of the Java application.
- **Definition:** Docker is used to create a container image that includes the application and its dependencies. This image can then be used to run the application consistently across different environments.

```
Unset
stage('Build Docker Image') {
    steps {
        script {
            echo 'Building Docker image...'
            bat "docker build -t %DOCKER_IMAGE% ."
        }
    }
}
```

5. Push Docker Image

- **Objective:** Upload the Docker image to Docker Hub.
- **Definition:** This stage involves pushing the Docker image to a container registry (Docker Hub) so that it can be accessed and used for deployment.

```
Unset
stage('Push Docker Image') {
    steps {
        script {
            echo 'Pushing Docker image to Docker Hub...'
            withCredentials([usernamePassword(credentialsId:
'docker-hub-credentials', usernameVariable: 'DOCKER_USER', passwordVariable:
'DOCKER_PASS')]) {
                bat "docker login -u %DOCKER_USER% -p %DOCKER_PASS%"
                bat "docker push %DOCKER_IMAGE%"
                bat "docker logout"
            }
        }
    }
}
```

6. Deploy with Docker Compose

- **Objective:** Deploy the application using Docker Compose.
- **Definition:** Docker Compose is used to define and run multi-container Docker applications. This stage deploys the application and its dependencies, such as a database, in a defined environment.

```java
stage('Deploy with Docker Compose') {
    steps {
        script {
            echo 'Deploying with Docker Compose...'
            withCredentials([usernamePassword(credentialsId:
'docker-hub-credentials', usernameVariable: 'DOCKER_USER', passwordVariable:
'DOCKER_PASS')]) {
                bat "docker login -u %DOCKER_USER% -p %DOCKER_PASS%"

                withEnv([
                    "DOCKER_IMAGE=${DOCKER_IMAGE}",
                    "SPRING_PORT=${SPRING_PORT}",
                    "POSTGRES_DB=${POSTGRES_DB}",
                    "POSTGRES_USER=${POSTGRES_USER}",
                    "POSTGRES_PASSWORD=${POSTGRES_PASSWORD}"
                ]) {
                    bat "docker-compose down"
```

```
                        bat "docker-compose build app"
                        bat "docker-compose up -d"
                    }

                    bat "docker logout"
                }
            }
        }
    }
```

Summary

The pipeline automates the end-to-end workflow for building, testing, and deploying a Java application. By integrating Jenkins, Maven, Docker, and Docker Compose, it ensures a consistent and reliable process, facilitating frequent and reliable software releases.