Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2019, 6(6):233-238**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Building Custom CI/CD Pipelines for Java Applications in GitLab

Praveen Kumar Koppanati

praveen.koppanati@gmail.com

Abstract: For modern software development, especially for Java applications Continuous Integration and Continuous Deployment (CI/CD) pipelines is a vital aspect. This paper aims to provide an in-depth exploration of building custom CI/CD pipelines in GitLab specifically for Java applications. In this guide, we are going to touch upon proper ways on how you should integrate GitLab CI/CD with Java development tools and strategies for pipeline optimization towards effective testing, as well as deployment; analyze Docker methodology where environment consistency is a concern. This will emphasize the key areas to be considered for security, performance and scalability. The analysis also covers automation techniques for testing and deployment, ensuring high code quality and fast iteration cycles. Developers can therefore use GitLab CI/CD and Java tools in tandem to simplify their workflow, with a consistent stream of production-ready software delivered efficiently.

Keywords: CI/CD, GitLab, Java applications, Continuous Integration, Continuous Deployment, Automation Testing, DevOps, automation, Docker, software development, testing, deployment.

1. Introduction

Software development practices have significantly evolved over the past few decades, with automation taking center stage in many development workflows. A transformation of such a practice in the development cycle has been around the introduction of Continuous Integration (CI) and Continuous Deployment (CD) pipelines. A CI/CD pipeline automates the testing, building and deployment of code so that changes in production are quicker but also free from human errors.

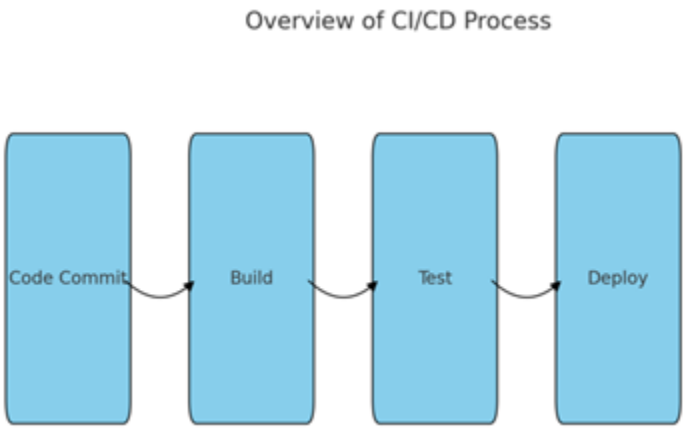

***Fig. 1** Overview of CI/CD Process*

In the context of Java applications, building a custom CI/CD pipeline requires a deep understanding of both the CI/CD tools available and Java-specific testing, building, and deployment strategies. GitLab CI/CD is one of the leading platforms that offers a fully integrated CI/CD solution in just one project by using native Git repository, enabling us to have automation across all stages and fields of software development. In this article, we will see how to create custom CI/CD pipelines for java applications using GitLab. We will also discuss the key points of consideration and best practices along with some tools that could help in improving our pipeline.

2. Understanding GitLab CI/CD

GitLab CI/CD is tightly integrated with GitLab, a Git-based source code management (SCM) system. It provides a powerful, flexible system for automating code testing, building, and deployment. GitLab CI/CD operates through pipelines defined in a .GitLab-ci.yml file located at the root of a repository. This file defines the various stages of the pipeline (e.g., build, test, deploy), and the specific tasks or jobs to be executed within each stage.

Key Concepts in GitLab CI/CD: Before delving into the details of building a custom CI/CD pipeline, it's important to understand several core concepts:

• **Jobs:** Individual tasks that GitLab Runner executes, such as compiling Java code, running tests, or deploying to a server.

• **Stages:** A pipeline is divided into stages such as build, test, and deploy. Jobs within a stage can run in parallel, but stages run sequentially.

• **Runners:** GitLab Runners are responsible for executing jobs defined in the pipeline. Runners can be shared or project specific.

• **Artifacts:** Intermediate files generated by one stage of the pipeline, such as compiled Java classes or test reports, can be passed on to subsequent stages as artifacts.

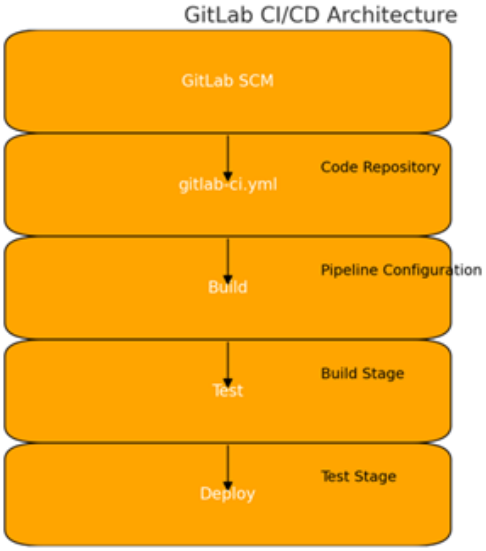

Fig.2 *GitLab CI/CD Architecture*

3. Building A Java-Specific CI/CD Pipeline

To construct an effective CI/CD pipeline for Java applications, it's essential to account for the various stages of Java development which include compilation, testing, and packaging. Tools like Maven or Gradle, along with GitLab's CI/CD, offer a smooth workflow from code commit to deployment.

Configuring GitLab Runner: The GitLab Runner is a crucial component of the CI/CD pipeline as it executes the jobs defined in the .GitLab-ci.yml file. To support Java applications, the Runner needs to be configured with a Java Development Kit (JDK). Depending on your environment, GitLab offers several types of Runners:

• **Shell Runners:** Simple but rely on the host machine's configuration.

• **Docker Runners:** Allow the use of Docker containers, providing a more isolated and consistent environment for running jobs.

A Docker Runner is often preferred for Java applications due to the ability to define and use specific Docker images that contain JDK, Maven, or Gradle. This ensures that the build environment is consistent across different stages of the pipeline and different team members.

```
image: maven:3.6.1-jdk-8
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - mvn compile

test_job:
  stage: test
  script:
    - mvn test
```

In the example above, a Maven-based Java application is built and tested using Docker images that have Maven and JDK pre-installed.

4. Testing And Quality Assurance in The Pipeline

Testing is an essential aspect of any CI/CD pipeline, and for Java applications, several testing frameworks can be used within the pipeline, such as JUnit, TestNG, and Mockito. Additionally, static code analysis tools like SonarQube and Checkstyle can be integrated to ensure code quality.

Unit Testing with Junit: JUnit is one of the most widely used testing frameworks for Java applications. By integrating JUnit tests into the CI/CD pipeline, you can automatically run tests every time code is pushed to the repository, ensuring that new changes don't break existing functionality.

```
test_job:
  stage: test
  script:
    - mvn test
  artifacts:
    paths:
      - target/surefire-reports/
```

JUnit generates reports in the target/surefire-reports directory. These reports can be archived as artifacts, allowing developers to access test results easily.

Code Quality with SonarQube: SonarQube is a popular tool for continuous code quality inspection. It can be integrated into the pipeline to perform static code analysis, detecting code smells, bugs, and potential security vulnerabilities.

```
sonarqube_scan:
  stage: test
  script:
    - mvn sonar:sonar
  dependencies:
    - build_job
```

By running SonarQube scans as part of the test stage, Java applications can be continuously analyzed for code quality issues, improving long-term maintainability.

5. Deployment Automation

Deployment in a CI/CD pipeline can take many forms, depending on the infrastructure used to host the Java application. GitLab CI/CD supports various deployment strategies, including:

• Traditional deployment to virtual machines or on-prem servers
• Containerized deployment using Docker and Kubernetes
• Cloud-native deployment to platforms like AWS, Azure, or Google Cloud

Deploying to a Traditional Server: In many enterprise environments, Java applications are deployed to traditional servers. GitLab CI/CD can automate this process by using SSH to connect to the server and deploy the application.

```
deploy_job:
  stage: deploy
  script:
    - scp target/myapp.jar user@server:/opt/myapp/
    - ssh user@server 'systemctl restart myapp'
```

This simple script transfers the built JAR file to the server and restarts the application, providing a fully automated deployment pipeline.

Deploying with Docker: Docker is increasingly being used for Java application deployments due to its ability to create lightweight, consistent environments. GitLab CI/CD can be configured to build a Docker image of the Java application and push it to a Docker registry.

```
docker_build_job:
  stage: deploy
  script:
    - docker build -t myapp .
    - docker push registry.example.com/myapp:latest
```

Using Docker enables developers to ensure that the application runs in the same environment in production as it does during development and testing.

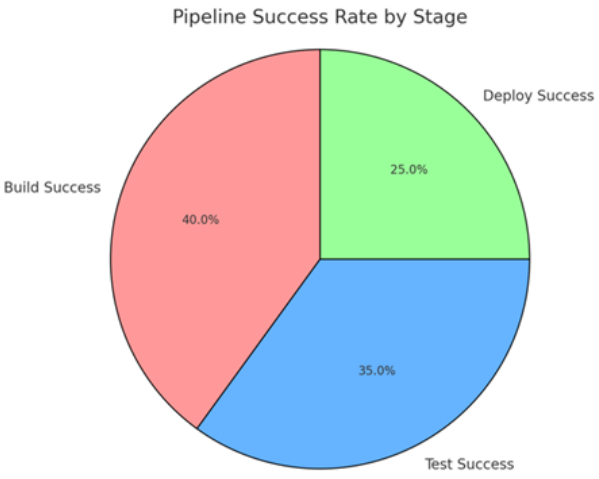

Fig 3: *Pipeline Success Rate by Stage*

6. Advanced Pipeline Features

While the basic pipeline covers build, test, and deployment, GitLab CI/CD offers several advanced features that can enhance the pipeline.

Pipeline Triggers: GitLab supports pipeline triggers that can initiate pipelines based on external events. This can be useful in cases where you want to trigger the pipeline after a successful deployment or based on events in another system.

Parallel Testing: For larger projects with extensive test suites, testing can become a bottleneck. GitLab CI/CD allows for parallel job execution, enabling multiple tests to run simultaneously.

```
test_job_1:
  stage: test
  script:
    - mvn test -Dtest=Class1

test_job_2:
  stage: test
  script:
    - mvn test -Dtest=Class2
```

By dividing the test suite into separate jobs, overall test execution time can be reduced.

7. Security Considerations

Security should be a primary concern in any CI/CD pipeline. For Java applications, special attention must be given to:

• **Dependency management:** Tools like OWASP Dependency-Check can be integrated into the pipeline to detect vulnerabilities in third-party libraries.

• **Credential management:** Sensitive information such as API keys and passwords should be stored securely, using GitLab's CI/CD environment variables or Vault integration.

```
secrets:
  stage: test
  script:
    - export SECRET_KEY=$CI_SECRET_KEY
```

Using environment variables prevents sensitive information from being hardcoded into the pipeline scripts.

8. Conclusion

GitLab with a custom CI/CD pipeline for Java applications create less human intervention automated process that increased software delivery velocity and quality of the delivered product. Utilizing the powerful CI/CD features of GitLab, developers can write pipelines that specifically cater towards their project requirements and provide a consistent approach with less manual involvement.

GitLab CI/CD helps teams keep code quality high with advanced tools like Docker, SonarQube and automated deployments all while enabling rapid iteration cycles. Flexible, automated pipelines will become even more essential for modern software development as the industry evolves.

References

[1]. Stolberg S. Enabling Agile Testing through Continuous Integration. 2009 Agile Conference. 2009;. https://doi.org/10.1109/AGILE.2009.16

[2]. Arefeen, M. S., & Schiller, M. (2019). Continuous integration using GitLab. Undergraduate Research in Natural and Clinical Science and Technology Journal, 3, 1-6.

[3]. M. Meyer, "Continuous Integration and Its Tools," in IEEE Software, vol. 31, no. 3, pp. 14-16, May-June 2014, doi: 10.1109/MS.2014.58

[4]. M. Kettani and Y. Benferhat, "Best Practices for Continuous Integration with GitLab CI/CD ," 2017 International Conference on Information and Communication Technology Convergence (ICTC), 2017, pp. 240-246. DOI: 10.1109/ICIW.2017.33

[5]. S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," 2018 Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka, 2018, pp. 156-161, doi: 10.1109/MERCon.2018.8421965

[6]. B. Kratz, "Containerizing Java Applications with Docker for CI/CD Pipelines," 2018 IEEE International Conference on Cloud Computing (CLOUD), 2018, pp. 825-830. DOI: 10.1109/ICCW.2018.00825

[7]. Ståhl D, Bosch J. Modeling continuous integration practice differences in industry software development. Journal of Systems and Software. 2014;87:48-59. https://doi.org/10.1016/j.jss.2013.08.032

[8]. K. Beck, "Continuous Integration," in Extreme Programming Explained: Embrace Change, Addison-Wesley, 1999.

[9]. M. Fowler, "Continuous Integration," ThoughtWorks, [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html.

[10]. Docker Inc., "What is Docker?," Docker Docs, [Online]. Available: https://docs.docker.com/get-started/overview/.

[11]. E. Gamma and K. Beck, JUnit: A Cook's Tour, Addison-Wesley, 2002.

[12]. OWASP, "OWASP Dependency-Check," OWASP, [Online]. Available: https://www.owasp.org/index.php/OWASP_Dependency_Check.

[13]. GitLab Inc., "GitLab CI/CD Documentation," GitLab, [Online]. Available: https://docs.GitLab.com/ee/ci/.

[14]. SonarQube, "SonarQube Documentation," SonarQube, [Online]. Available: https://docs.sonarqube.org/latest/.