**ODSOFT: Iteration 2 – CI/CD Practices and Implementation Overview 1201506**

**Mono-repository Strategy for Microservices Management**

The approach chosen to manage the microservices architecture was tailored to the project's team size and operational constraints. Given the small team size—comprising only one member—a mono repository structure was adopted. This decision balances simplicity in repository management with the flexibility needed for independent microservices development. In subsequent sections, we will provide more details.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

**Three-Tiered Environment Strategy: Dev, Test, and Prod**

To ensure a robust and structured CI/CD pipeline, three distinct environments were established: **Development** (Dev), **Testing** (Test), and **Production** (Prod), each serving specific objectives and facilitating a seamless software delivery process.

* **Development Environment (Dev)**

The Dev environment is dedicated to feature development, bug fixes, and experimentation. It enables rapid iteration and integration of new features.

So, we can work on feature-specific branches following the ‘**feature/\***’ naming convention, ensuring clear organization and traceability of changes.

Commits pushed to these branches trigger automated pipelines designed for linting, unit tests, and basic integration tests.

* **Testing Environment (Test)**

The Test environment is configured to closely replicate production, allowing comprehensive validation of the application’s functionality and performance.

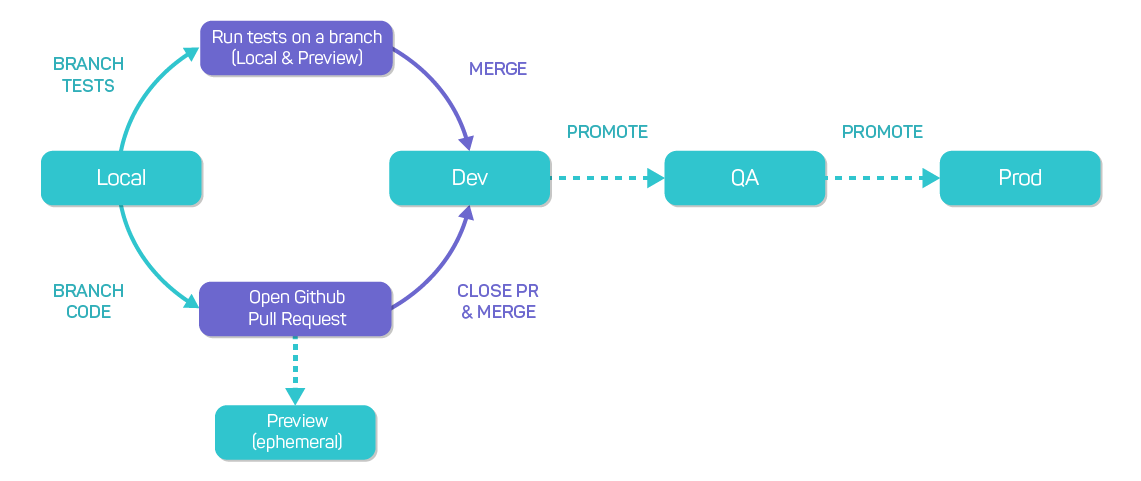
Changes are merged into ‘**release/\*’** branches, which automatically activate pipelines to execute advanced integration tests, system tests, and performance assessments.

This environment ensures the application meets all quality assurance standards before moving to production.

* **Production Environment (Prod)**

The Prod environment is the final deployment stage, where the stable version of the application is released to end-users.

Changes are merged into the **main** branch, triggering pipelines that deploy the application to production servers with monitoring and security mechanisms in place.

This environment guarantees high reliability, scalability, and minimal downtime through strategies such as zero-downtime deployments (Blue/Green) and automated rollbacks if issues are detected.

We used **Bitbucket** for version control, this branch-based workflow integrates with the CI/CD **Jenkins** pipeline, ensuring a progression of changes from development to production while maintaining quality and stability at each stage.  
  
Uma imagem com captura de ecrã, texto, Software de multimédia, software

Descrição gerada automaticamente

**Pipeline Configuration – Independent Pipeline per Microservice**

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamenteTo manage the project structure and support the three designated environments, Development, Testing, and Production, each microservice will have its own dedicated CI/CD pipeline. In Jenkins, a multibranch project was created for every microservice.

These projects were designed to scan all branches containing the specific Jenkinsfile for each microservice. Upon detecting any changes in the repository, the configured webhook automatically triggers the corresponding pipeline, ensuring that the appropriate workflow is executed based on the branch where the commit was made. Each branch is also configured to generate its own distinct metrics and outputs upon the pipeline's execution and successful completion, providing targeted insights and results to the specific stage of the development lifecycle.

**Multibranch Project Configuration**

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente  
  
  
Uma imagem com texto, captura de ecrã, file, Tipo de letra

Descrição gerada automaticamente

**Branches Overview**

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

**Branch Project Overview**

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

**Pipeline Description**

With the environments (Development, Testing, and Production) established, the next step was to design and implement tailored pipelines for each branch type—feature, release, and main. These pipelines are critical for automating the build, test, and deployment processes while ensuring that the correct configurations and environments are applied at every stage.

**Declarative Pipeline Challenges and Custom Solution**

In our Jenkins implementation, we utilized declarative pipelines for their simplicity and readability. However, declarative pipelines present a known limitation, they cannot directly call other pipeline jobs. To address this, we designed a dynamic and reusable pipeline structure that adapts based on the branch where the commit was made.

This solution employs scripted logic within the declarative pipeline to determine the branch type (feature, release, or main) and dynamically configure environment variables, paths, and processes. Below is the pipeline configuration tailored for the AuthNUserCommand microservice, as example.

This environment block configures dynamic variables to adapt the pipeline to the corresponding environment and branch.

1. ENVIRONMENT: Sets the environment (dev, test, or prod) based on the branch name (feature/, release/, or others).
2. POM\_LOC: Specifies the path to the pom.xml for Maven commands related to the microservice.
3. DOCKER\_COMPOSE\_PATH: Selects the appropriate docker-compose file for the environment.
4. CONTAINER\_ID: Reserves space for the Docker container ID, set during execution.
5. DOCKER\_CREDS: Retrieves secure credentials for Docker registry authentication.

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamenteThis block automates environment-specific configurations, making the pipeline dynamic and secure.

The ‘Check Changes’ stage checks if there are any changes in the AuthNUserCommand directory by comparing the last commit with the previous one. If no changes are detected, it skips the pipeline.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteThe ‘Print Environment’ stage outputs the current branch name (BRANCH\_NAME) and the automatically determined environment (ENVIRONMENT) to provide visibility into the pipeline execution context.

**Development Pipeline**