Especificación del Pipeline de CI/CD

Plataforma de Aprendizaje para Todos

Revisor	Versión	Fecha
Fabricio Chuquispuma	V1.0	29 nov 2024

1. Introducción

El pipeline está diseñado para construir, analizar, probar y desplegar un proyecto compuesto por un backend en Java y un frontend en Node.js. También incluye análisis estático de código con SonarQube y soporte para ejecución local.

2. Requisitos previos

Antes de usar este pipeline, asegúrate de cumplir con los siguientes requisitos:

2.1. Herramientas necesarias configuradas en Jenkins:

- JDK: Configurado como JAVA.
- Maven: Configurado como maven.
- Node.js: Configurado como nodejs.
- SonarQube Scanner: Configurado como sonar-scanner en las herramientas globales de Jenkins.

2.2. Repositorio Git:

- URL: https://github.com/GHOPsT/TDW---Grupo-1.git
- Ramas: main.

2.3. SonarQube:

- Servidor disponible en http://localhost:9000.
- Token de autenticación: extraido del sonar qube

3. Desglose de las Etapas

3.1. Git Checkout: Clona el repositorio desde GitHub.

Acciones:

- Descarga el código de la rama main sin registrar cambios en el changelog ni activar polling.

3.2. **Build Frontend:** Construye el proyecto frontend usando npm.

Acciones:

- Ejecuta npm install para instalar dependencias.
- Ejecuta npm run build para construir el proyecto.

3.3. Build Backend: Compila el código backend usando Maven.

Acciones:

- Ejecuta mvn clean compile para compilar.

3.4. **SonarQube Analysis Backend:** Realiza análisis estático de código en el backend.

Acciones:

- Utiliza SonarQube Scanner con configuración específica.
- Variables importantes:
- Proyecto: jenkins-back.
- Directorio de fuentes: ..
- Binarios: target.

3.5. SonarQube Analysis Frontend: Realiza análisis estático de código en el frontend.

Acciones:

- Utiliza SonarQube Scanner con configuración específica.

Variables importantes:

- Proyecto: jenkins-front.

3.6. **Test Backend:** Ejecuta pruebas unitarias en el backend.

Acciones:

- Ejecuta mvn test.

3.7. **Test Frontend** Ejecuta pruebas en el frontend con control de errores.

Acciones:

- Ejecuta npm test.

En caso de fallos, el pipeline continúa pero marca la etapa como inestable (UNSTABLE).

3.8. **Despliegue Local:** Despliega el backend y frontend en paralelo para pruebas locales.

Acciones:

- Backend:

  Ejecuta mvn clean install.

  Inicia la aplicación usando el archivo JAR generado.

- Frontend:

  Ejecuta npm run dev para iniciar el servidor de desarrollo.

4. Pipeline script

```
pipeline {
    agent any
    tools {
        jdk 'JAVA'
        maven 'maven'
        nodejs 'nodejs'
    }
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
    stages {
        stage("Git checkout") {
            steps {
                git branch:'main', changelog:false, poll:false,
url:'https://github.com/GHOPsT/TDW---Grupo-1.git'
            }
        }
        stage("Build frontend"){
            steps{
                dir("Projects\\Development\\platform-for-all"){
                    bat "npm install"
                    bat "npm run build"
                }
            }
        }
        stage("Build backend"){
            steps{
              dir("Projects\\Development\\platform-for-all-back"){
                    bat "mvn clean compile"
                }
            }
        }
        stage("SonarQube Analysis Backend") {
            steps {
              dir("Projects\\Development\\platform-for-all-back"){
                    bat "$SCANNER_HOME/bin/sonar-scanner
-Dsonar.url=http://localhost:9000/ \

-Dsonar.login=squ_324480fe7535a685e927cb1708c11004160ce62d \
                    -Dsonar.projectKey=jenkins-back \
```

```
                    -Dsonar.projectName=jenkins-back \
                    -Dsonar.sources=. \
                    -Dsonar.java.binaries=target "
                }
            }
        }
        stage("SonarQube Analysis Frontend") {
            steps {
               dir("Projects\\Development\\platform-for-all-all"){
                    bat "$SCANNER_HOME/bin/sonar-scanner
-Dsonar.url=http://localhost:9000/ \

-Dsonar.login=squ_324480fe7535a685e927cb1708c11004160ce62d \
                    -Dsonar.projectKey=jenkins-front \
                    -Dsonar.projectName=jenkins-front \
                    -Dsonar.sources=. \
                    -Dsonar.java.binaries=target "
                }
            }
        }

        stage("Test backend"){
            steps{
              dir("Projects\\Development\\platform-for-all-back"){
                    bat "mvn test"
                }
            }
        }
        stage("Test frontend") {
            steps {
                dir("Projects\\Development\\platform-for-all") {
                    script{
                        catchError(buildResult: 'SUCCESS',
stageResult: 'UNSTABLE'){
                            bat "npm test" // Use sh for better
cross-platform compatibility
                        }
                    }
                }
            }
        }
        stage("Despliegue local"){
```

```
            parallel{
                stage("Backend despliegue"){
                    steps{
              dir("Projects\\Development\\platform-for-all-back"){
                                bat "mvn clean install"
                                bat "java -jar
target/platform-for-all-back-0.0.1-SNAPSHOT.jar"
                            }
                    }
                }
                stage("Frontend despliegue"){
                    steps{

dir("Projects\\Development\\platform-for-all"){
                                bat "npm run dev"
                            }
                    }
                }
            }
        }
    //
    } // End of stages
} // End of pipeline
```

5. Consideraciones de Configuración

5.1. Control de errores en pruebas frontend:

La etapa de pruebas frontend usa catchError para manejar fallos sin detener el pipeline completo.

5.2. Rutas con barras invertidas:

Se utilizan barras invertidas (\\) para compatibilidad con entornos Windows.