Demo Project: Complete the CI/CD Pipeline (Docker-Compose, Dynamic versioning)

This guide demonstrates how to configure a Jenkins pipeline to dynamically set the Docker image name based on the application version. The process involves versioning the application, building a Docker image, and committing the updated version to the repository for subsequent pipelines.

Step 1: Increment Version in Jenkins Pipeline

1. **Remove Static Versioning in Jenkinsfile**:

   Remove any hardcoded versioning from the Jenkinsfile. For example, remove lines like:

   ```
   environment {
       IMAGE_NAME = 'eduardobautistamaciel/demo-app:java-maven-2.0'
   }
   ```

2. **Add Increment Version Step**:

   Implement dynamic versioning using Maven's build-helper plugin in your Jenkinsfile. The pipeline step should look similar to this:

```
        stage('increment version') {
            steps {
                script {
                    echo 'incrementing app version...'
                    sh 'mvn build-helper:parse-version versions:set \
                        -
DnewVersion=\\\${parsedVersion.majorVersion}.\\\${parsedVersion.minorVersion}.\\\${parsedVersion.nextIn
crementalVersion} \
                        versions:commit'
                    def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
                    def version = matcher[0][1]
                    env.IMAGE_NAME = "$version-$BUILD_NUMBER"
                }
            }
        }
```

- **Purpose**: This step automatically increments the version in the Maven `pom.xml` file to ensure each pipeline execution uses a unique Docker image tag.

Step 2: Commit the Version Update

1. **Commit the New Version**:

   Add a stage to commit the updated version to the repository so the next pipeline starts with the new version. For example:

```
        stage('commit version update'){
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'gitlab-credentials',
passwordVariable: 'PASS', usernameVariable: 'USER')]){
                        def encodedPassword = URLEncoder.encode(PASS, 'UTF-8')
                        sh 'git remote set-url origin
https://${USER}:${encodedPassword}@gitlab.com/twn-devops-projects/aws/java-maven-app.git'
                        sh 'git add .'
                        sh 'git commit -m "ci: version bump"'
                        sh 'git push origin HEAD:jenkins-jobs'
                    }
                }
            }
        }
```

- **Purpose**: Ensures version consistency and avoids repetitive builds starting with the same version.

Complete Jenkinsfile Overview

Your Jenkinsfile should look like this after adding dynamic versioning and version updates:

```groovy
#!/usr/bin/env groovy

library identifier: 'jenkins-shared-library@main', retriever: modernSCM([
    $class: 'GitSCMSource',
    remote: 'https://gitlab.com/twn-devops-projects/aws/jenkins-shared-library',
    credentialsId: 'gitlab-credentiales'
])


pipeline {
    agent any
    tools {
        maven 'maven-3.9'
    }
    stages {
        stage('increment version') {
            steps {
                script {
                    echo 'incrementing app version...'
                    sh 'mvn build-helper:parse-version versions:set \
                        -
DnewVersion=\\\${parsedVersion.majorVersion}.\\\${parsedVersion.minorVersion}.\\\${parsedVersion.nextIn
crementalVersion} \
                        versions:commit'
                    def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
                    def version = matcher[0][1]
                    env.IMAGE_NAME = "$version-$BUILD_NUMBER"
                }
            }
        }
        stage('build app') {
            steps {
                echo 'building application jar...'
                buildJar()
            }
        }
        stage('build image') {
            steps {
                script {
                    echo 'building the docker image...'
                    buildImage(env.IMAGE_NAME)
                    dockerLogin()
                    dockerPush(env.IMAGE_NAME)
                }
            }
        }
        stage("deploy") {
            steps {
                script {
                    echo 'deploying docker image to EC2...'

                    def shellCmd = "bash ./server-cmds.sh ${IMAGE_NAME}"
                    def ec2Instance = "ec2-user@52.71.72.116"

                    sshagent(['ec2-server-key']) {
                        sh "scp server-cmds.sh ${ec2Instance}:/home/ec2-user"
                        sh "scp docker-compose.yaml ${ec2Instance}:/home/ec2-user"
                        sh "ssh -o StrictHostKeyChecking=no ${ec2Instance} ${shellCmd}"
                    }
                }
            }
        }
        stage('commit version update'){
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId: 'gitlab-credentials',
passwordVariable: 'PASS', usernameVariable: 'USER')]){
                        def encodedPassword = URLEncoder.encode(PASS, 'UTF-8')
                        sh 'git remote set-url origin
https://${USER}:${encodedPassword}@gitlab.com/twn-devops-projects/aws/java-maven-app.git'
                        sh 'git add .'
                        sh 'git commit -m "ci: version bump"'
                        sh 'git push origin HEAD:jenkins-jobs'
                    }
                }
            }
        }
    }
}
```

Summary of the Jenkinsfile Functionality

This Jenkinsfile automates the build, versioning, and deployment process for a Java Maven application using Docker.

1. **Shared Library**: Loads utility functions from the Shared Library GitLab repository.
2. **Pipeline Stages**:
   - **Increment Version**: Increments the application version in the `pom.xml`.
   - **Build App**: Compiles the Java application and generates a JAR file.
   - **Build Image**: Builds and pushes the Docker image to the registry with a dynamically set version tag.
   - **Deploy**: Deploys the Docker image to an EC2 instance using SSH.
   - **Commit Version Update**: Pushes the updated version back to the repository.