Experiment 6: Continuous Integration with Jenkins: Setting Up a CI Pipeline, Integrating Jenkins with Maven/Gradle, Running Automated Builds and Tests

1. Overview

What is a CI Pipeline?

A **Continuous Integration (CI) Pipeline** automates the process of building, testing, and integrating code changes every time code is committed to the repository. This pipeline:

- Automatically checks out the latest code.
- Compiles the application.
- Runs tests to catch errors early.
- Notifies the team of build/test results.

Why Use Jenkins for CI?

- **Automation:** Jenkins automates the build and test cycle, reducing manual intervention.
- **Immediate Feedback:** Developers get rapid notifications of any integration issues.
- **Extensibility:** With hundreds of plugins available, Jenkins can integrate with version control systems, build tools (Maven, Gradle), testing frameworks, and more.
- **Pipeline as Code:** Using Jenkins Pipelines (defined in a Jenkinsfile), you can manage the CI process as part of your source code repository.

2. Setting Up a CI Pipeline with Jenkins (Freestyle Project)

This section explains how to create a CI pipeline as a Freestyle project that integrates with a Maven or Gradle project.

Step 1: Create a New Jenkins Job

1. **Log into Jenkins:**

o Open your web browser and navigate to your Jenkins URL (e.g., http://localhost:8080 or your cloud instance URL).

o Log in with your admin credentials.

2. **Create a New Job:**

o On the Jenkins dashboard, click on **"New Item"**.

o **Enter an Item Name:** For example, Maven-CI (or Gradle-CI if you prefer Gradle).

o **Select "Freestyle project"**.

o Click **"OK"**.

Step 2: Configure Source Code Management (SCM)

1. **Select SCM:**

o In the job configuration page, scroll down to the **"Source Code Management"** section.

o Select **"Git"** (if using Git for version control).

2. **Enter Repository Details:**

o **Repository URL:** Enter the URL of your Git repository (for example, https://github.com/yourusername/your-maven-project.git).

o **Credentials:** If your repository is private, click **"Add"** to provide the necessary credentials.

o Optionally, specify the **Branch Specifier** (e.g., */main).

Step 3: Add Build Steps

A. For a Maven Project

1. **Add Maven Build Step:**

o Scroll down to **"Build"** and click on **"Add build step"**.

o Select **"Invoke top-level Maven targets"**.

o **Goals:** In the Goals field, enter:

o clean package

This command instructs Maven to clean the previous build artifacts, compile the code, run tests, and package the application into a JAR/WAR file.

o Optionally, set the **POM File** location if it is not in the default location (pom.xml).

Step 4: Configure Post-build Actions

1. **Publish Test Results:**

o Scroll down to the **"Post-build Actions"** section.

o Click **"Add post-build action"** and select **"Publish JUnit test result report"**.

Step 5: Save and Run the Job

1. **Save the Configuration:**

o Click **"Save"** at the bottom of the job configuration page.

2. **Trigger a Build:**

o On the job's main page, click **"Build Now"**.

o The build will be added to the build history on the left side.

3. **Monitor Build Output:**

o Click on the build number (e.g., #1) and then click **"Console Output"**.

o Verify that Jenkins successfully checks out the code, runs the build commands (Maven or Gradle), and executes tests.
o Look for **"BUILD SUCCESS"** or the equivalent output to confirm that the build and tests passed.

3. Setting Up a CI Pipeline with Jenkins (Pipeline as Code)

For greater flexibility and version-controlled CI configuration, you can use a **Jenkins Pipeline** defined in a Jenkinsfile.

Step 1: Create a Pipeline Job

1. **Log into Jenkins** and click on **"New Item"**.
2. **Enter an Item Name:** For example, Pipeline-CI.
3. **Select "Pipeline"** and click **"OK"**.

Step 2: Define the Pipeline Script

1. **Configure the Pipeline:**

o In the job configuration page, scroll to the **"Pipeline"** section.
o Choose **"Pipeline script"** (or "Pipeline script from SCM" if you want to load the script from your repository).

2. **Enter the Pipeline Script:**

Below are sample pipeline scripts for Maven and Gradle projects.

Example for a Maven Project:

```
pipeline {
agent any
stages {
        stage('Checkout')
        {       steps {
                        // Check out code from Git repository
                        git url: 'https://github.com/yourusername/your-mavenproject.
                        git', branch: 'main'
                        }

}
stage('Build')
        { steps {
                // Run Maven build
                sh 'mvn clean package'
                }
        }
stage('Test') {
```

```
        steps {
            // Optionally, separate test execution if needed
            sh 'mvn test'
            }
        }
}
post {
        always {
            // Archive test reports
            junit '**/target/surefire-reports/*.xml'
            }
        success {
            echo 'Build and tests succeeded!'
            }
        failure {
            echo 'Build or tests failed.'
            }
        }
        }
```

3. **Save the Pipeline Script:**

o After entering your pipeline script, click **"Save"**.

Step 3: Run the Pipeline

1. **Trigger the Build:**

o On the Pipeline job's main page, click **"Build Now"**.

o Monitor the build progress through the Pipeline visualization or by clicking on the build number and then **"Console Output"**.

2. **Verify the Results:**

o Confirm that each stage (Checkout, Build, Test) executes successfully.

o Review the archived test reports to verify that tests have run and passed.