# 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.

# 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

### Log into Jenkins:

* + Open your web browser and navigate to your Jenkins URL (e.g.,

[http://localhost:8080](http://localhost:8080/) or your cloud instance URL).

* + Log in with your admin credentials.

### Create a New Job:

* + On the Jenkins dashboard, click on **“New Item”**.
  + **Enter an Item Name:** For example, Maven-CI (or Gradle-CI if you prefer Gradle).

### Select “Freestyle project”.

* + Click **“OK”**.

## Step 2: Configure Source Code Management (SCM)

### Select SCM:

* + In the job configuration page, scroll down to the **“Source Code Management”**

section.

* + Select **“Git”** (if using Git for version control).

### Enter Repository Details:

* + **Repository URL:** Enter the URL of your Git repository (for example,

https://github.com/yourusername/your-maven-project.git).

* + **Credentials:** If your repository is private, click **“Add”** to provide the necessary credentials.
  + Optionally, specify the **Branch Specifier** (e.g., \*/main).

## Step 3: Add Build Steps

### For a Maven Project

* 1. **Add Maven Build Step:**
     + Scroll down to **“Build”** and click on **“Add build step”**.

### Select “Invoke top-level Maven targets”.

* + - **Goals:** In the Goals field, enter:
    - 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.

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

### For a Gradle Project

* 1. **Add Gradle Build Step:**
     + If you are integrating a Gradle project, click on **“Add build step”** and choose **“Invoke Gradle script”** (this option might be available if you have installed a Gradle plugin in Jenkins).
     + **Tasks:** In the tasks field, enter:
     + clean build

This instructs Gradle to clean previous build outputs and then build the project, running tests along the way.

* + - **Switches:** If needed, you can add additional flags (for example, --info or -- stacktrace for more detailed output).

## Step 4: Configure Post-build Actions

### Publish Test Results:

* + Scroll down to the **“Post-build Actions”** section.
  + Click **“Add post-build action”** and select **“Publish JUnit test result report”**.
  + **Test Report XMLs:** In the field, enter the pattern that matches your test report files. For example:
  + \*\*/target/surefire-reports/\*.xml

for Maven, or a similar path for Gradle projects.

## Step 5: Save and Run the Job

### Save the Configuration:

* + Click **“Save”** at the bottom of the job configuration page.

### Trigger a Build:

* + On the job’s main page, click **“Build Now”**.
  + The build will be added to the build history on the left side.

### Monitor Build Output:

* + Click on the build number (e.g., #1) and then click **“Console Output”**.
  + Verify that Jenkins successfully checks out the code, runs the build commands (Maven or Gradle), and executes tests.
  + Look for **“BUILD SUCCESS”** or the equivalent output to confirm that the build and tests passed.

# 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

### Configure the Pipeline:

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

### 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 {

git url: 'https://github.com/yourusername/your-maven- project.git', branch: 'master'

}

}

stage('Build') { steps {

// Run Maven build

bat 'mvn clean package'

}

}

stage('Test') { steps {

// Optionally, separate test execution if needed bat 'mvn test'

} } }

post {

success {

echo 'Build and tests succeeded!'

}

failure {

echo 'Build or tests failed.'} } }

### Example for a Gradle Project:

pipeline {

agent any

stages {

stage('Checkout') { steps {

// Check out code from Git repository

git url: 'https://github.com/yourusername/your- gradle-project.git', branch: 'master'

} }

stage('Build') { steps {

// Run Gradle build

bat './gradlew clean build'

} }

stage('Test') { steps {

// Run tests (if not already run in the build stage) bat './gradlew test'

} } }

post {

success {

echo 'Build and tests succeeded!'

}

failure {

echo 'Build or tests failed.'

} } }

### Save the Pipeline Script:

* + After entering your pipeline script, click **“Save”**.

## Step 3: Run the Pipeline

### Trigger the Build:

* + On the Pipeline job’s main page, click **“Build Now”**.
  + Monitor the build progress through the Pipeline visualization or by clicking on the build number and then **“Console Output”**.

### Verify the Results:

* + Confirm that each stage (Checkout, Build, Test) executes successfully.
  + Review the archived test reports to verify that tests have run and passed.