---------Running the Cucumber tests with Jenkins-------------------------------------

# **Step** 1: Set Up Cucumber with Maven

Create a Maven Project:

In Eclipse, go to File > New > Project....

Select Maven Project and proceed with the wizard to create a new Maven project.

Add Cucumber Dependencies:

Open the pom.xml file of your Maven project.

Add the following dependencies for Cucumber and JUnit:

<dependencies>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-java</artifactId>

<version>7.0.0</version>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>7.0.0</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

</dependency>

</dependencies>

# **Step** 2: Write Cucumber Tests

Define Feature Files:

Create a new directory src/test/resources in your Maven project.

Add .feature files to this directory. Example:

gherkin

Copy code

Feature: Example feature

Scenario: Example scenario

Given some precondition

When some action is performed

Then some outcome is expected

# Implement **Step** Definitions:

Create a new package (e.g., stepDefinitions) in src/test/java.

# Add **Step** definition classes to this package. Example:

java

Copy code

package stepDefinitions;

import io.cucumber.java.en.Given;

import io.cucumber.java.en.When;

import io.cucumber.java.en.Then;

public class ExampleSteps {

@Given("some precondition")

public void some\_precondition() {

// code for precondition

}

@When("some action is performed")

public void some\_action\_is\_performed() {

// code for action

}

@Then("some outcome is expected")

public void some\_outcome\_is\_expected() {

// code for outcome

}

}

Create Test Runner Class:

Create a runner class in src/test/java in a package like runners. Example:

java

Copy code

package runners;

import org.junit.runner.RunWith;

import io.cucumber.junit.Cucumber;

import io.cucumber.junit.CucumberOptions;

@RunWith(Cucumber.class)

@CucumberOptions(

features = "src/test/resources",

glue = "stepDefinitions"

)

public class TestRunner {

}

# **Step** 3: Run Cucumber Tests with Maven in Eclipse

Run Maven Build:

Right-click on the pom.xml file in your project.

Select Run As > Maven test.

This will compile your project and execute the Cucumber tests.

# **Step** 4: Configure Jenkins to Run Cucumber Tests

Create a New Jenkins Job:

Open Jenkins in your web browser.

Click on New Item from the Jenkins dashboard.

Enter a name for your job (e.g., "CucumberTests") and select Freestyle project. Click OK.

Configure Source Code Management:

Under the Source Code Management section, select the version control system you are using (e.g., Git).

Provide the repository URL and credentials if required. Example configuration for Git:

csharp

Copy code

Repository URL: https://github.com/username/repo.git

Credentials: (select or add credentials)

Add Build Steps:

# Under the Build section, click Add build **Step** and select Invoke top-level Maven targets.

In the Goals field, enter clean test. This will ensure Maven cleans the project and runs the tests.

Example configuration:

bash

Copy code

Goals: clean test

Configure Post-build Actions:

Under the Post-build Actions section, click Add post-build action and select Publish JUnit test result report.

In the Test report XMLs field, enter target/cucumber-reports/\*.xml. This assumes your Cucumber reports are configured to be output in this location.

Example configuration:

bash

Copy code

Test report XMLs: target/cucumber-reports/\*.xml

Save the Configuration:

Click Save to apply the configuration and create the job

# **Step** 5: Execute and Monitor Jenkins Job

Trigger the Build:

From the Jenkins dashboard, select your newly created job.

Click on Build Now to manually trigger the build.

To automate the build process, you can set up build triggers. For example, to trigger a build when changes are pushed to the repository:

Go to the job configuration.

Under the Build Triggers section, check Poll SCM.

Set the schedule (e.g., H/5 \* \* \* \* for polling every 5 minutes).

Monitor Test Execution:

Once the build is triggered, you can monitor its progress.

Click on the build number (e.g., #1) under the Build History section on the left side of the job page.

Click on Console Output to view the log of the build process. This will show detailed information about the execution of the Maven build, including the Cucumber tests.

View Test Results:

After the build is complete, navigate to the build's detail page.

Click on Test Result to view the results of the Cucumber tests.

Jenkins will display a summary of the test results, including the number of tests passed, failed, and skipped.

Additional Reporting:

JUnit Reports:

Ensure that your pom.xml is configured to generate JUnit XML reports. Cucumber generates these by default if you have specified the output format in the @CucumberOptions in your runner class:

java

Copy code

@CucumberOptions(

features = "src/test/resources",

glue = "stepDefinitions",

plugin = {"pretty", "junit:target/cucumber-reports/Cucumber.xml"}

)

Cucumber Reports Plugin:

For enhanced reporting, you can use the Cucumber Reports plugin.

Go to Manage Jenkins > Manage Plugins.

Install the Cucumber Reports plugin.

Configure your job to use this plugin:

Under the job configuration, add a post-build action Cucumber reports.

Set the JSON reports path to target/cucumber-reports/\*.json (or wherever your reports are generated).

Example configuration for the runner class:

java

Copy code

@CucumberOptions(

features = "src/test/resources",

glue = "stepDefinitions",

plugin = {"pretty", "json:target/cucumber-reports/Cucumber.json"}

)

By following these steps, you can effectively execute and monitor Cucumber tests in Jenkins, ensuring continuous integration and comprehensive test reporting.