

Full Stack Development

Lab Jenkins 2

Running a Maven Build

1. Lab objectives

In this lab you will be using both a local repository and to run a Maven build

2. Create and run the project

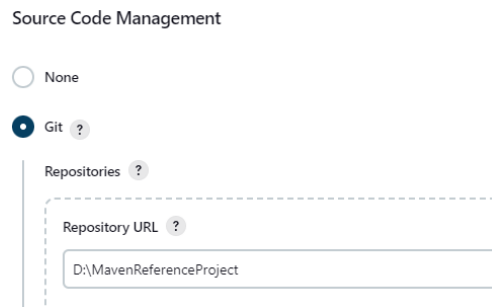
Start Jenkins and log in as in Lab 1 if necessary.

Clone the time-tracker project to a local directory of your choosing. The URL of the project is

<https://github.com/ExgnosisClasses/MavenReferenceProject>

Create a new **FreeStyle** project named **Maven1**.

In the Source Code Management section, enter the directory name where you cloned the repository. Don't use the directory in the screenshot.



Source Code Management

☐ None

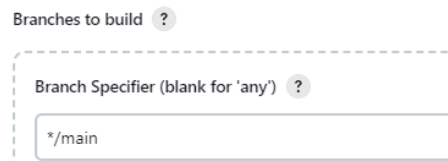
☒ Git ?

Repositories ?

Repository URL ?

D:\MavenReferenceProject

Jenkins will check out a branch to be used in the build. The default for Jenkins is “**8/master**” But this project uses “main” instead of “master”. Make the appropriate adjustment in the **Branches to Build** section.

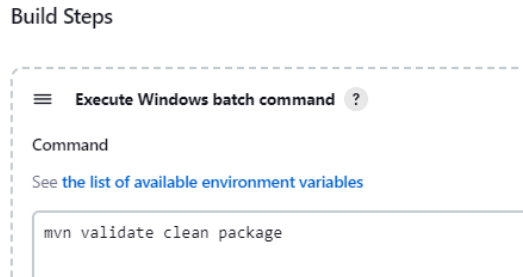


Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

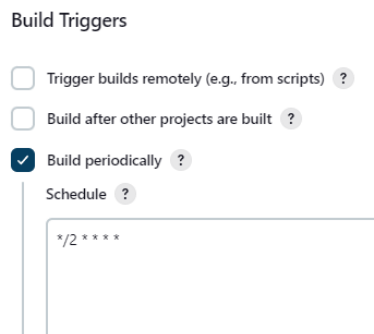
In the Build Steps in the same field you entered “Hello World from Jenkins” before, enter the command **mvn validate clean package**. Save the configuration and build the project.



Examine the output the way we did in the demo. You can experiment with running different Maven commands this way.

3. Automatic Builds

In the Maven1 project, in the project configuration, go to the **Build Triggers** section select **Build Periodically**. Enter the cron string ***/2 * * * *** which tells Jenkins to run a build every two minutes.



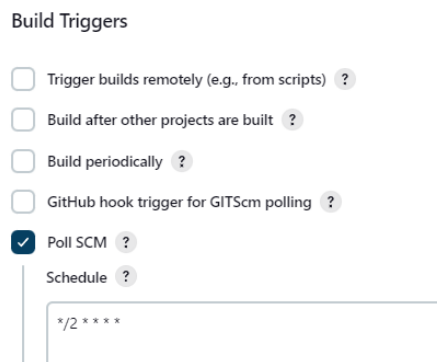
Save the configuration and wait a few minutes to see the build run. After you have confirmed that the builds are executed, go back into the project configuration and remove the trigger.

Wait a bit and confirm automatic builds have stopped.

4. Repository Polling

This build trigger will poll the repository at specified intervals. If there has been a commit since the last poll, Jenkins will run a build.

Just like in the last section, go to the **Build Triggers** section and choose **Poll SCM**. Enter the same string that you did in the previous section in the schedule to tell Jenkins how often to check for changes



Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☐ GitHub hook trigger for GITScm polling ?
- ☒ Poll SCM ?

Schedule ?

`*/* 2 * * * *`

How you have to make a change in the repository. Go to the directory you cloned the repository into and make a small change to one of the files. For example, change “Hello World” in the main method to “Hi Ya”.

Same the change and add only that changed file using **git add**. You want to avoid adding all the output files generated by Maven.

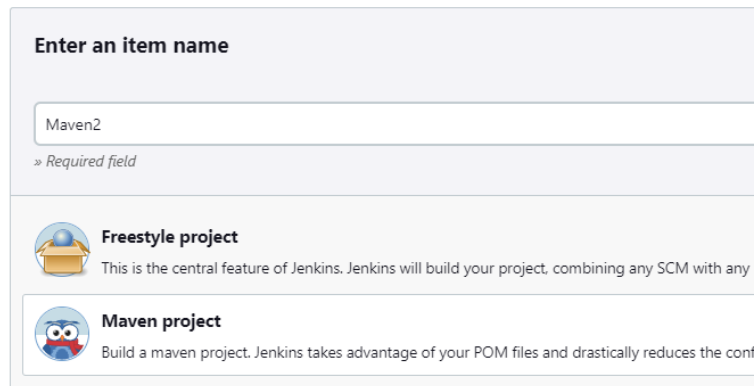
Once you commit the change, it will be picked up by Jenkins the next time it polls. Go back to the project page where you can see the build execute. You might see a “quiet period” which is just a wait period for Jenkins to make sure that it doesn’t interfere with other builds.

Now wait to see the build happen.

5. Using the Maven plugin

In the previous sections, Maven was just treated as a generic command. The Maven plugin integrates Maven more tightly with Jenkins as you saw in the demo.


Create a new Maven project called **Maven2**




Enter an item name

Maven2

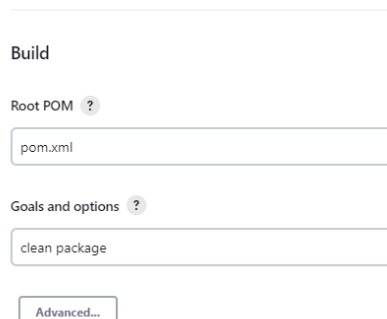
» Required field

 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any

 **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the conf

Set up the source code repository exactly as you did in the previous Maven1 project.

However, this time use the Maven build options as shown below.



Build

Root POM ?

pom.xml

Goals and options ?

clean package

Advanced...

Save the project and run it. Once it's finished, check the results to see the Maven specific artifacts.

End Lab