Maven

Certain everyday tasks are necessary to build any software automation project such as downloading the required dependencies, putting additional jars, compiling source code into binary code, running tests, packaging of compiled code into different artifacts such as Jar, Zip, and War files, and deployment of these artifacts to an application server or repository. These are very tedious tasks to do manually every time, so we need a tool that helps us to achieve all these tasks in minimized time to reduce human efforts. Here comes ***Maven*** into the picture, a build lifecycle management tool that helps automate all these tedious tasks from compiling resources to building the artifacts. Now, we know that Jenkins is one of the most used *CI/CD tools,* and the integration of Jenkins Maven is kind of a must to support the complete *CI/CD* for Java-based projects.

Subsequently, let's quickly understand the few necessary details of Maven and then understanding the ***Jenkins Maven integration*** in detail by covering the details in the following topics:

- *What is Maven?*
  - *And what is the difference between Maven and Jenkins?*
- *What is Maven Plugin for Jenkins?*
  - *How to install Maven Plugin in Jenkins?*
- *How to integrate Maven with Jenkins?*
  - *Also, how to setup Java Path in Jenkins?*
  - *How to set up Maven Path in Jenkins?*
- *What is a Maven Project in Jenkins?*
  - *How to create a Maven project in Jenkins?*
  - *How to execute a Maven project in Jenkins?*

What is Maven?

Maven is a powerful build management tool for Java projects to help execute a build life cycle framework. The basis of the Maven is the concept of ***POM*** *(Project Object Model)* in which all configurations can be done with the help of a ***pom.xml*** file. It is a file that includes all the project and configuration-related information such as source directory, dependencies with its version, plugins, and builds information, test source directory, etc. A few of the key features of Maven are:

- *Maven describes how the project is built. It builds a project using the Page Object Model.*
- *Maven automatically downloads, update as well as validate the compatibility between dependencies.*
- *In Maven, dependencies are retrieved from the dependency repository, whereas plugins are retrieved from plugin repositories, so mavens keep proper isolation between project dependencies and plugins.*
- *Maven can also generate documentation from the source code, compiling and then packaging compiled code into JAR files or ZIP files.*

As we understood, both *Maven* and *Jenkins* are automation tools, which are used for the automation of various stages of builds. Still, they are not alternatives to each other. Before understanding *Maven and Jenkins's integration,* let's quickly understand some differences between *Jenkins and Maven*.

What is the difference between Maven and Jenkins?

As we discussed, even though Maven and Jenkins don't have any direct comparisons as both of them fulfill different purposes. Still, we can compare them on the following parameters:

Comparison Parameter	***Jenkins***	***Maven***
Purpose	Jenkins is a CI/CD tool, which uses various other tools/plugins to automate the complete build cycle.	Maven is a build lifecycle management tool that helps in dependency management and creating automated builds.
Programming Models	Jenkins uses groovy for pipeline creation or uses the GUI for standalone jobs.	Maven works on an XML based structure for performing various actions in a build lifecycle.
Dependency Management	Jenkins can provide dependency between various jobs, but this is more of an action dependency.	Maven provides capabilities for resource dependency management, including third-party dependencies for builds.
Scope	Jenkins can do much more than build and deploy, such as server management, database management. It uses integration with various tools for this purpose.	Maven is more restricted around code and builds. This serves as a plugin in Jenkins for building Maven-based java projects.

Let's now quickly see how we can integrate Maven with Jenkins:

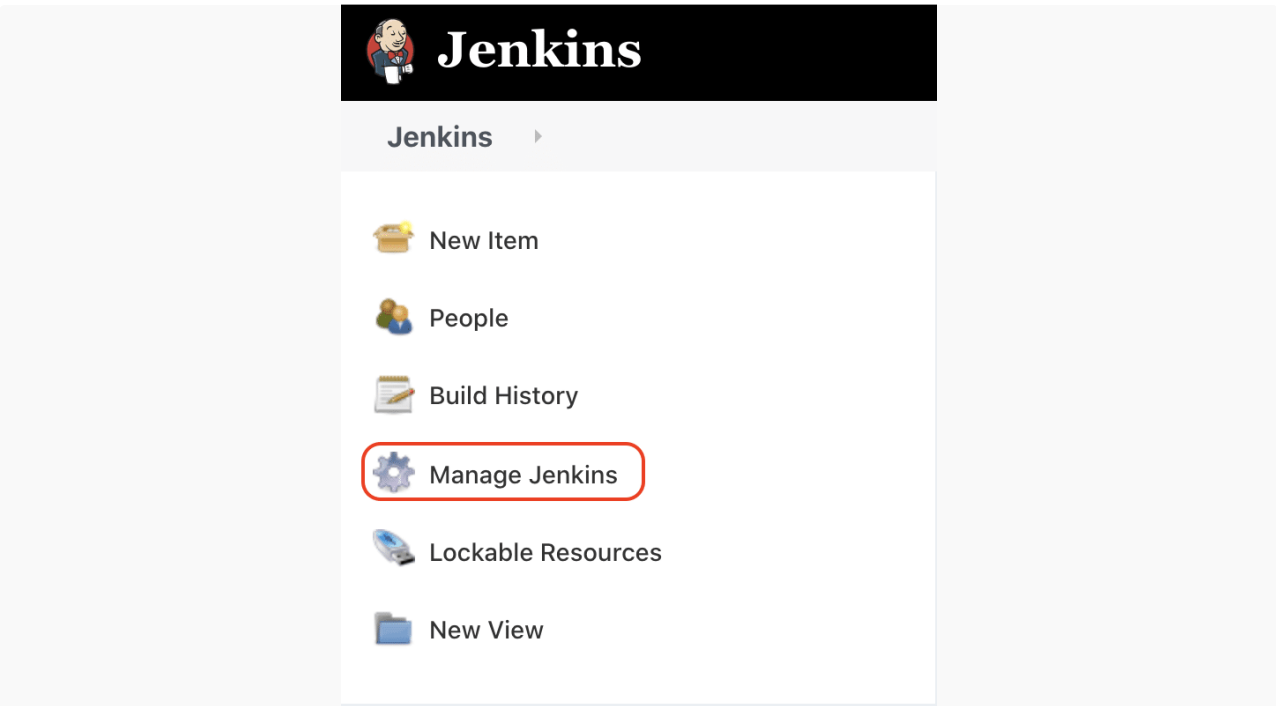
What is Maven Plugin for Jenkins?

The *Maven Plugin* is a plugin that provides the capabilities to *configure, build, and run Maven-based projects in Jenkins*. This is a must pre-requisite for the integration of *Maven with Jenkins*. Let's see how to can install *Maven's integration plugin in Jenkins:*

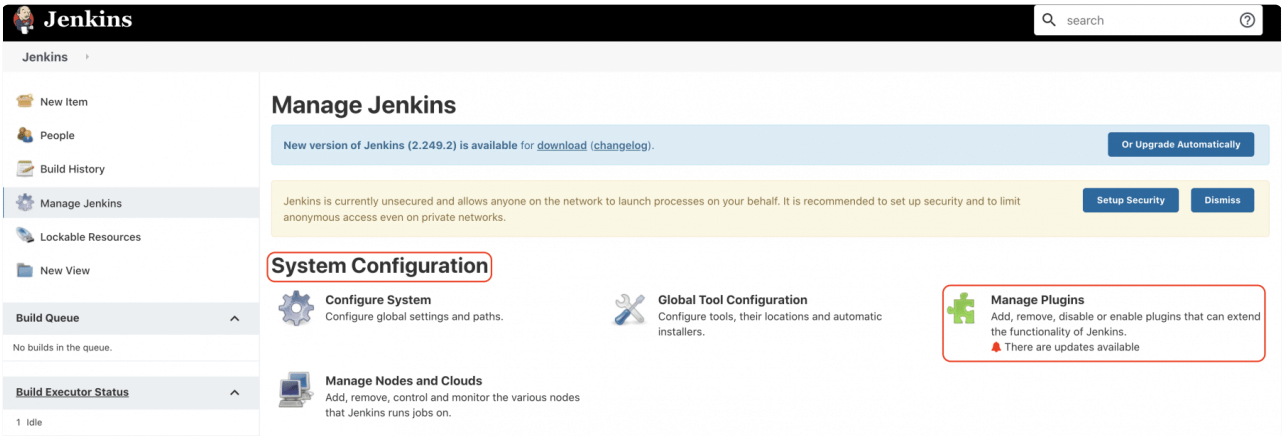
How to install Maven Plugin in Jenkins?

Follow the steps as mentioned below to install the *Maven plugin in Jenkins:*

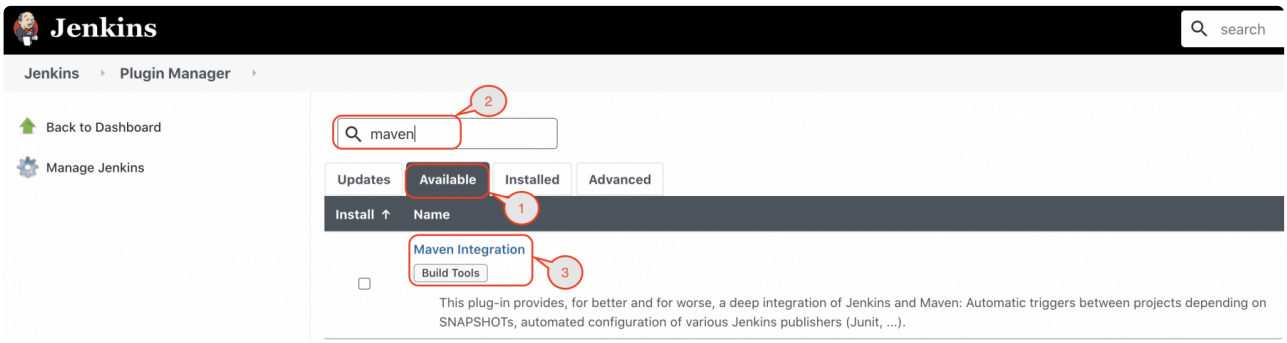
1st Step: Click on the ***Manage Jenkins*** link in the left menu bar, as highlighted below:



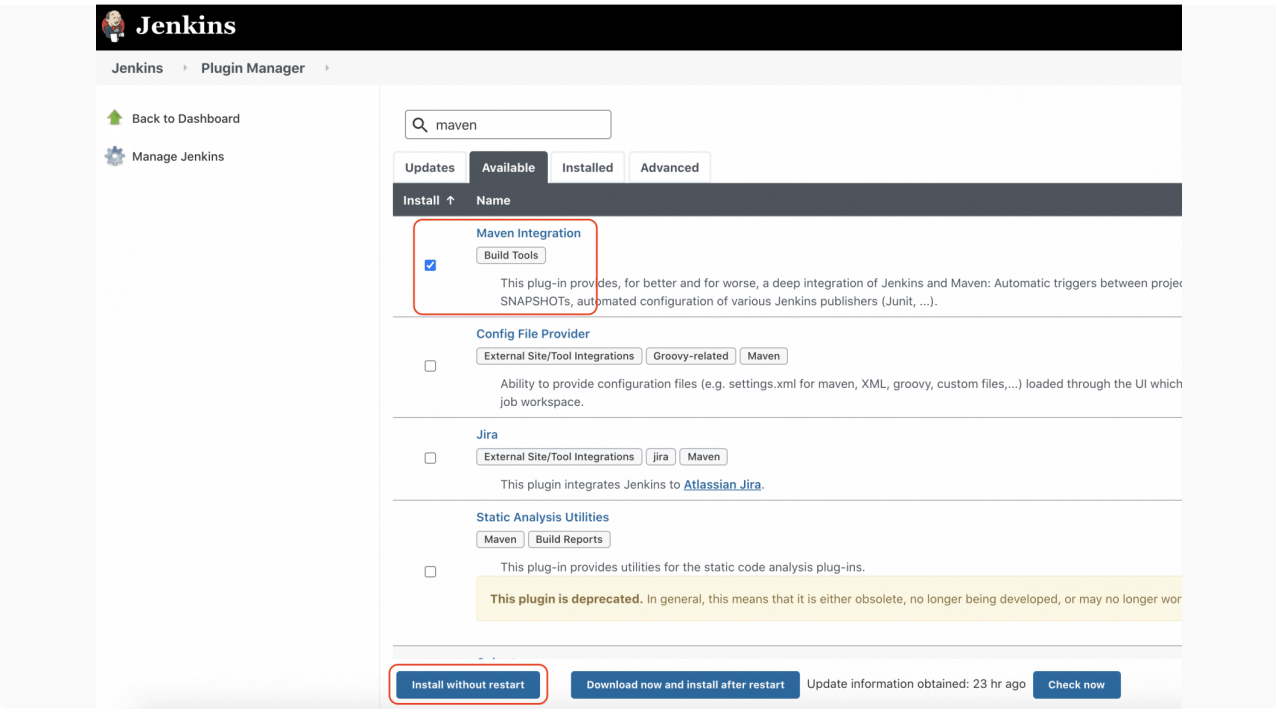
2nd Step: Under the ***System Configuration*** section, click on the ***Manage Plugins*** options:



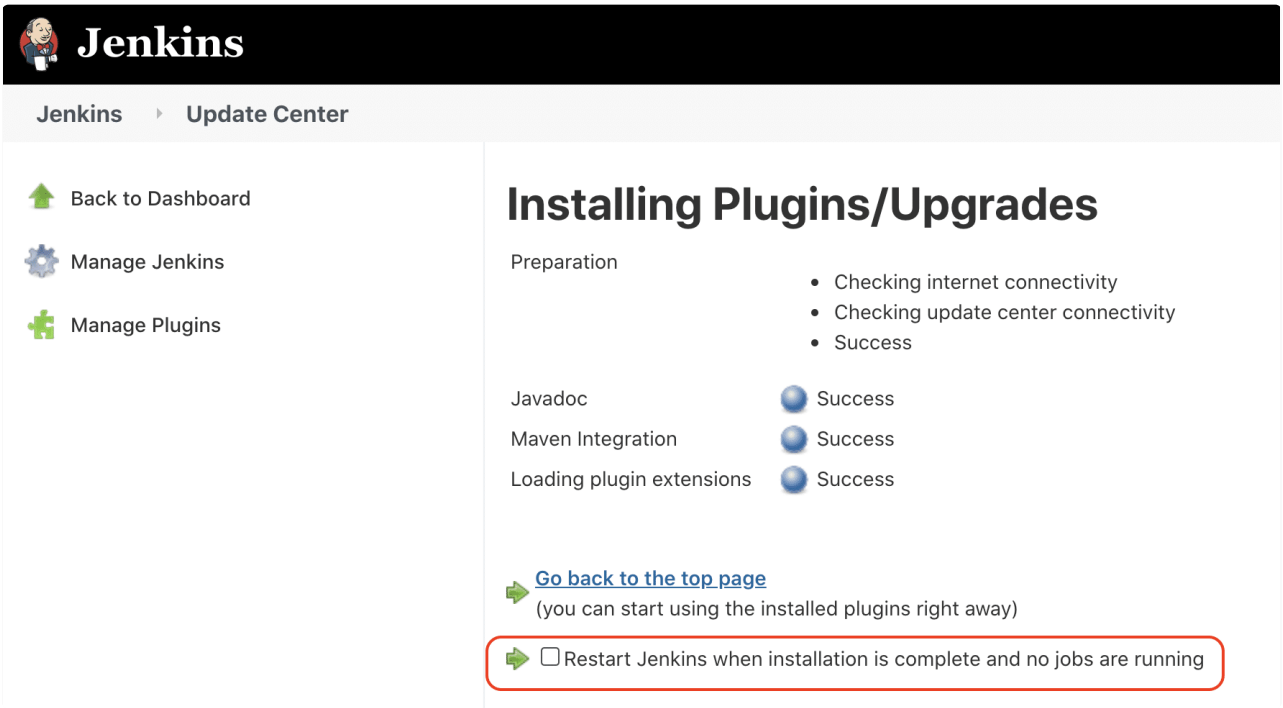
3rd Step: Under the *Plugin Manager,* click on the ***Available*** tab *(marker 1)* and search for the ***maven*** plugin *(marker 2)*. It will show the ***Maven Integration*** plugin as a result *(marker 3):*



4th Step: Select the checkbox in front of the ***Maven Integration plugin*** and click on the ***Install without restart*** button:



5th Step: Once the plugin installs successfully, click the checkbox to restart *Jenkins:*



6th Step: After the restart of *Jenkins,* the *Maven Jenkins plugin* will be installed successfully and ready for configuration.

After the installation of the *Maven Jenkins plugin,* let's see how we can configure and integrate the *Maven with Jenkins:*

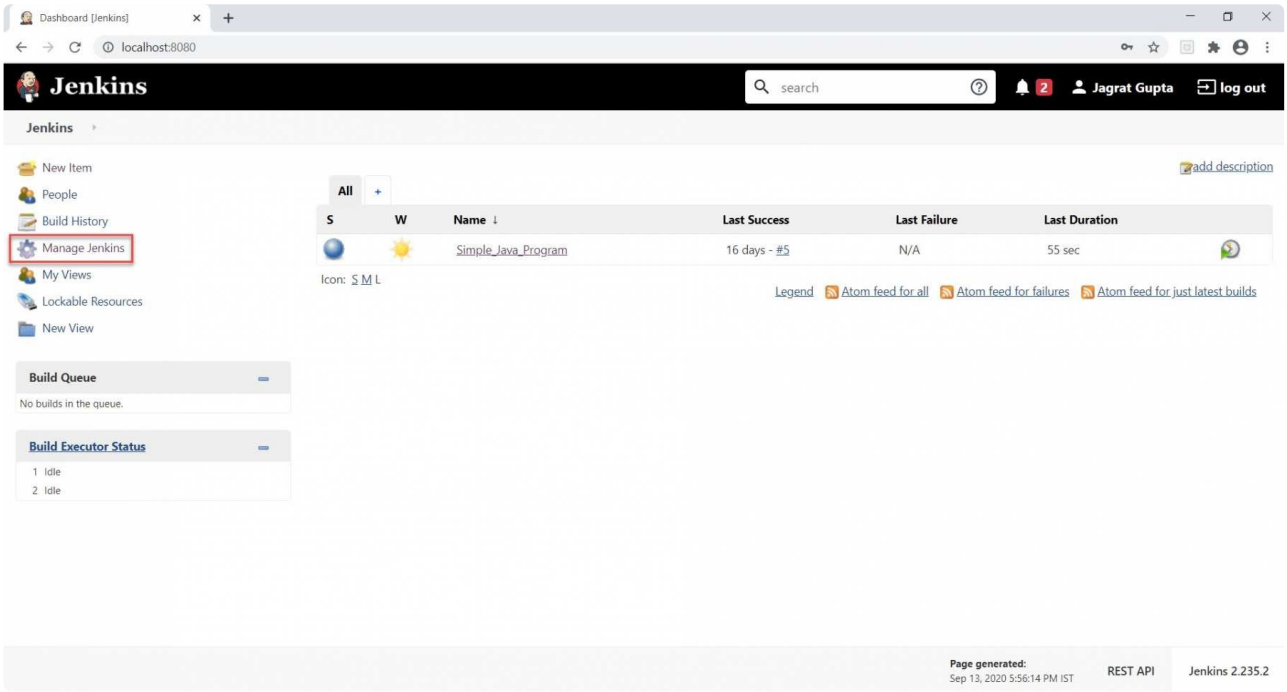
How to integrate Maven with Jenkins?

The reason behind integrating *Maven* with *Jenkins* is so that we can execute Maven commands through *Jenkins* as we will majorly use *Maven* for *Java* projects. Hence, ***JDK*** also comes as a pre-requisite for this setup. So, let's fir quickly see how to can specify the java path in *Jenkins:*

How to setup Java Path in Jenkins?

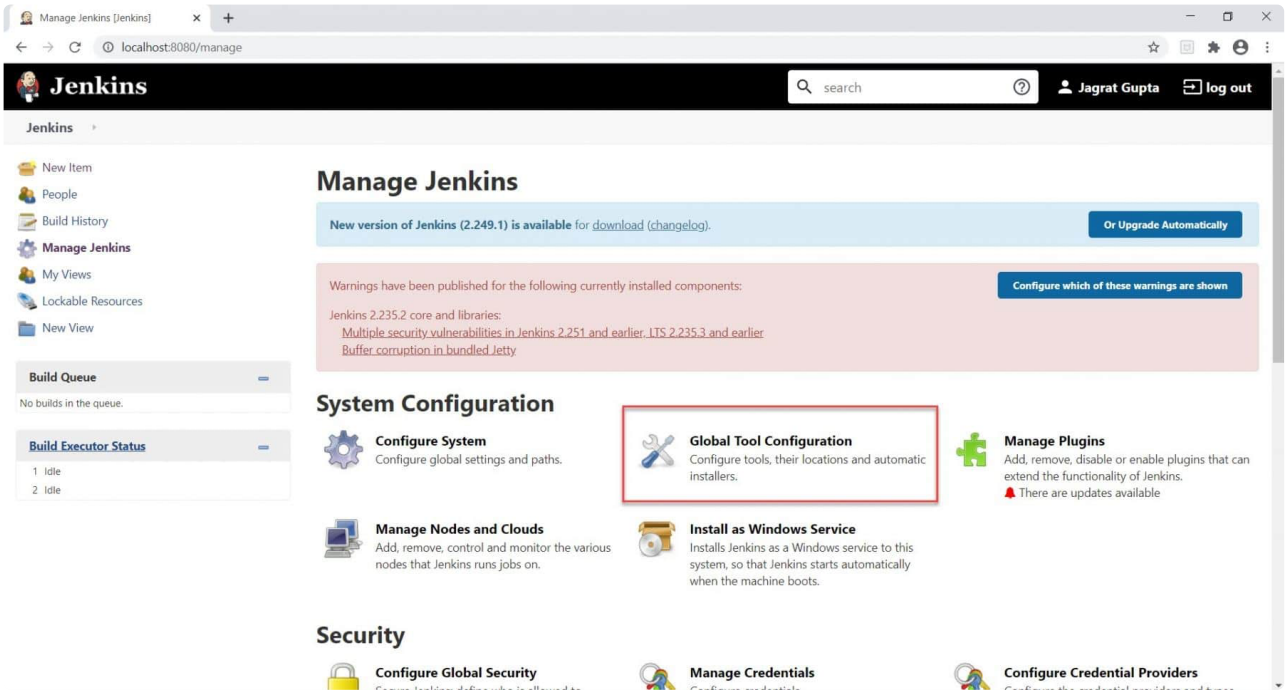
Maven integration with *Jenkins* starts with setting up the *Java path in Jenkins*. Kindly follow the below steps to setup *Java* path in *Jenkins:*

Step 1: Open the *Jenkins* and go to *Jenkins Dashboard*. After that, click on the ***Manage Jenkins*** link as shown below:



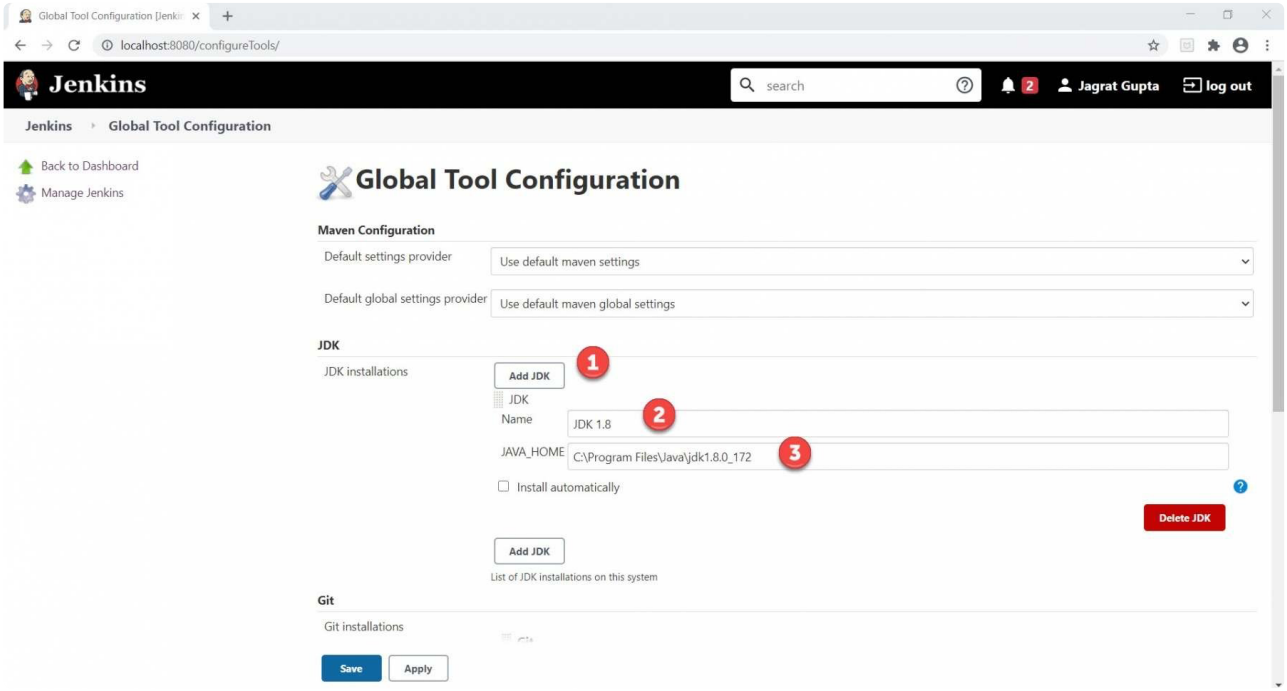
As soon as we click on the ***"Manage Jenkins"*** link, we will redirect towards the *Manage Jenkins* page in which we can see different types of options, and from here, we can see the ***"Global Tool Configuration"*** option.

Step 2: Now click on the ***Global Tool Configuration*** link as highlighted below:



As soon as we click on *Global Tool Configuration,* we will be redirected to the *Global tool configuration page* to specify different configurations.

Step 3: After that, we need to set the *JDK* path in *Jenkins*. To set the *JDK* path in *Jenkins,* please follow the below-highlighted steps:

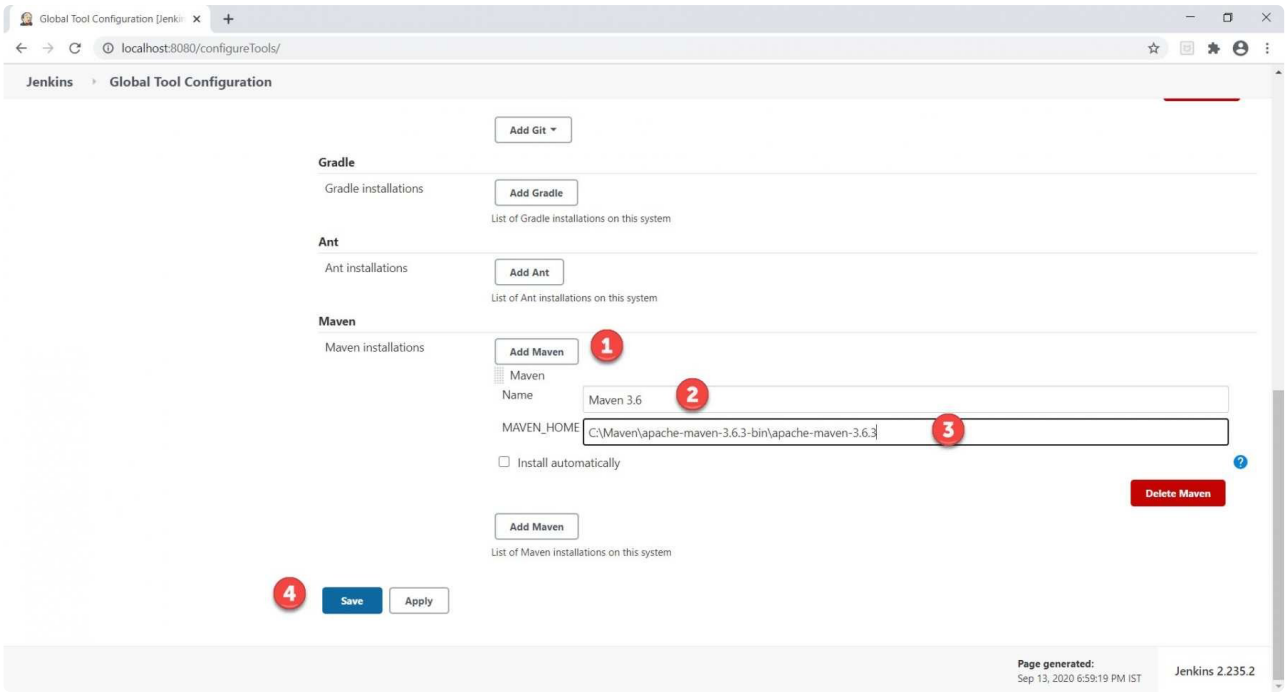


- *Click on the **Add JDK** button. Kindly note that by default, **"Install Automatically"** will be checked, so since we are going to use the JDK installed in our local machine, **"Install automatically"** will install the latest version of JDK, and you will also need to provide credentials to download relevant JDK.*
- *Give JDK's name as we gave as JDK 1.8, as this is currently installed in my machine.*
- *Give the path of JDK in JAVA_HOME textbox.*

After this, the *JDK* path is properly set up in *Jenkins*. Now, the next task is to set up the *Maven* path in *Jenkins*.

How to setup Maven Path in Jenkins?

In the previous section, we saw how to set up the *Java* path in *Jenkins,* and now, in this section, we will set up the *Maven* path in *Jenkins*. Please follow the below steps to set up the *Maven* path in *Jenkins*.



1. *Click on the **Add Maven** button. Kindly note that by default, **"Install Automatically"** will be checked, so we will uncheck it because we don't want that Jenkins will automatically install the latest version of Maven.*
2. *Give the name of Maven as we gave as **Maven 3.6,** as this is the version set up in my machine.*
3. *Give the path of Maven in the **MAVEN_HOME** textbox.*
4. *Click on the **Save** button.*

Now that we have configured both *Java* and *Maven* in *Jenkins,* Let's see how to create and execute a *Maven project in Jenkins?*

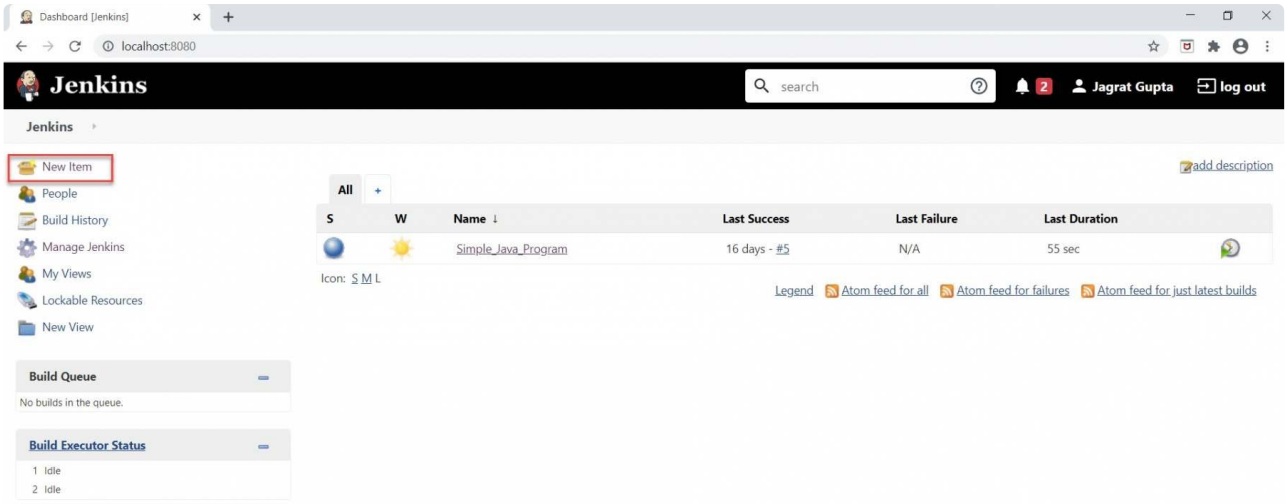
What is a Maven project in Jenkins?

Jenkins provides a particular job type, which explicitly provides options for configuring and executing a Maven project. This job type is called the ***"Maven project."*** Let's see how we can create a *Maven project in Jenkins* and run the same.

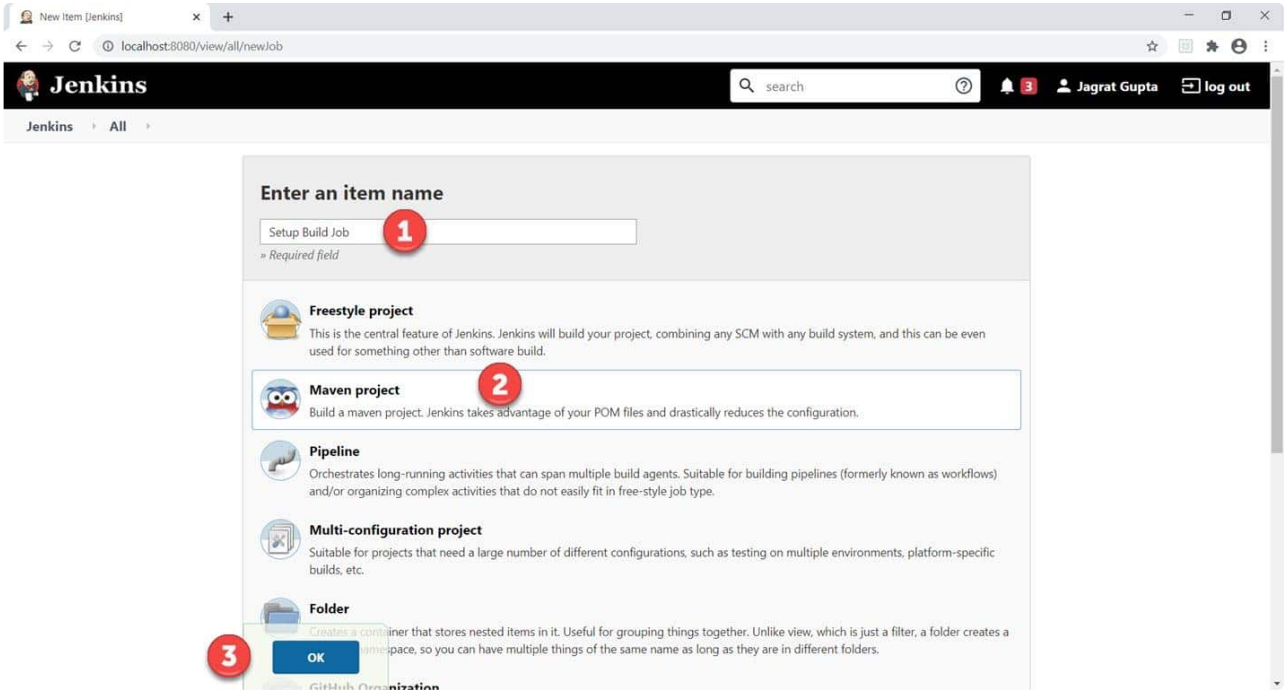
How to create a Maven project in Jenkins?

We know that the ***pom.xml*** file is the heart of *Maven* projects. For demonstration purposes, we already created a maven project, which is pushed into the ***GitHub repository.*** Kindly visit the link ***Understanding GitHub*** to understand more about *Git* and *GitHub*. For creating a Maven project in *Jenkins,* follow the steps as mentioned below:

Step 1: Firstly, we need to create a job. To create this, click on the ***"New Item"*** option as highlighted below:

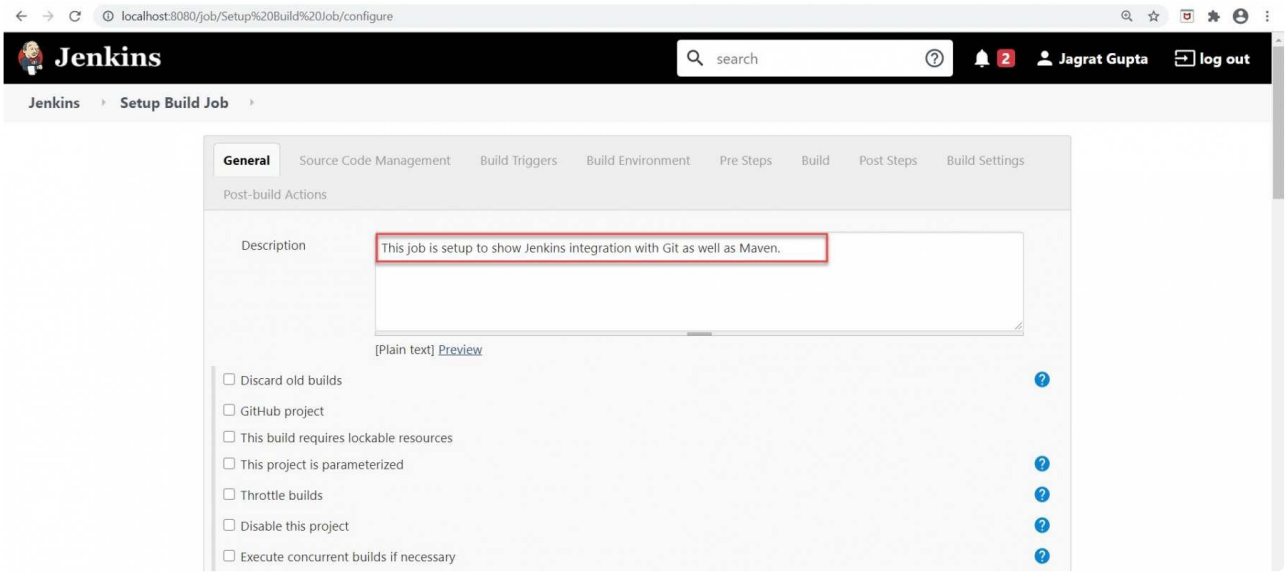


2nd Step: Now, do the following steps to create a new maven project:



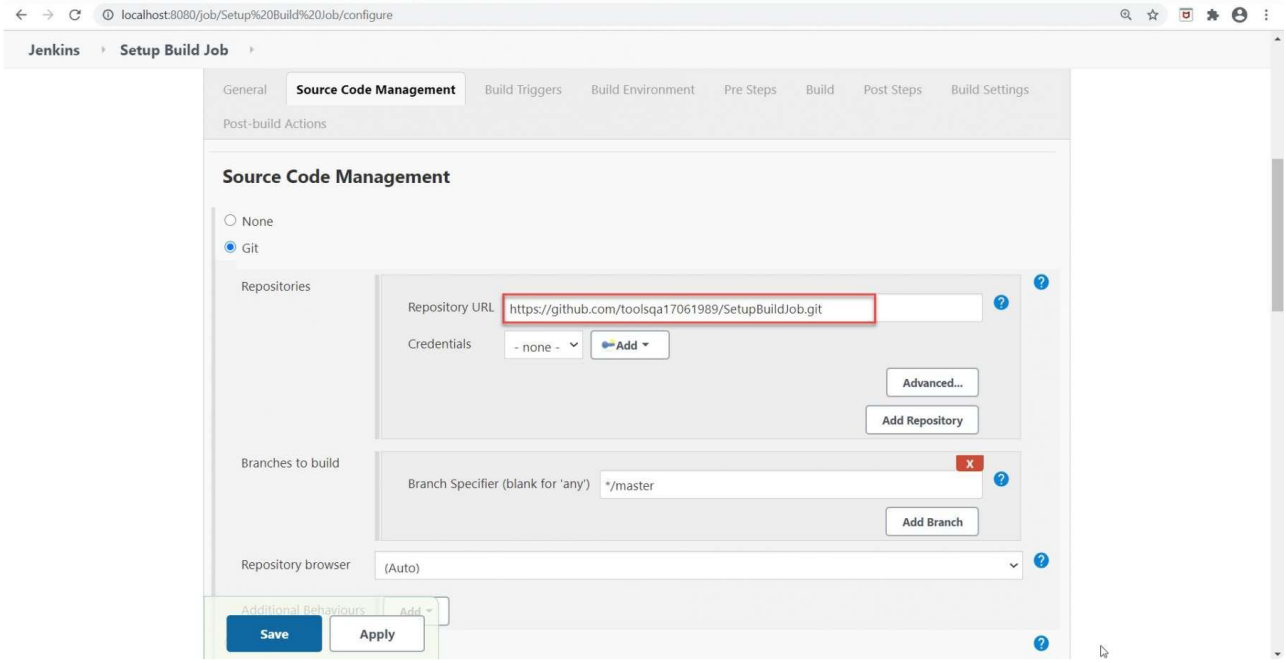
1. *Give the Name of the project.*
2. Click on the **Maven project.** Kindly note that If this Maven Project option is not visible, we need to check whether the **"Maven Integration"** plugin is installed in Jenkins. If not installed, then install it and restart Jenkins. For more details, please refer to our article***"Install Jenkins"***.
3. *Click on the **OK** button.*

3rd Step: Describe the project in the description section.

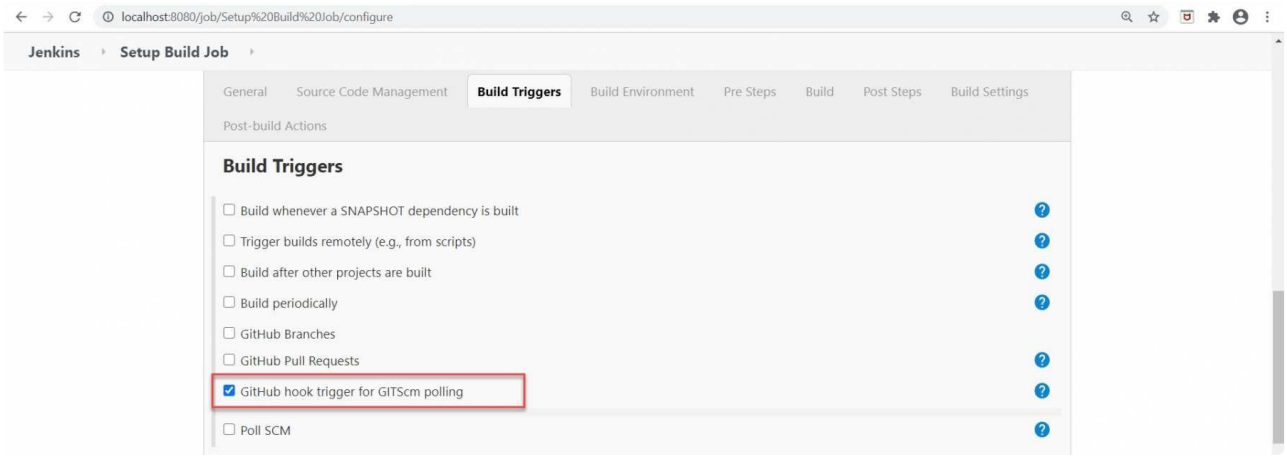


Now, go to the further sections like ***Source Code Management*** and ***Build Triggers*** section.

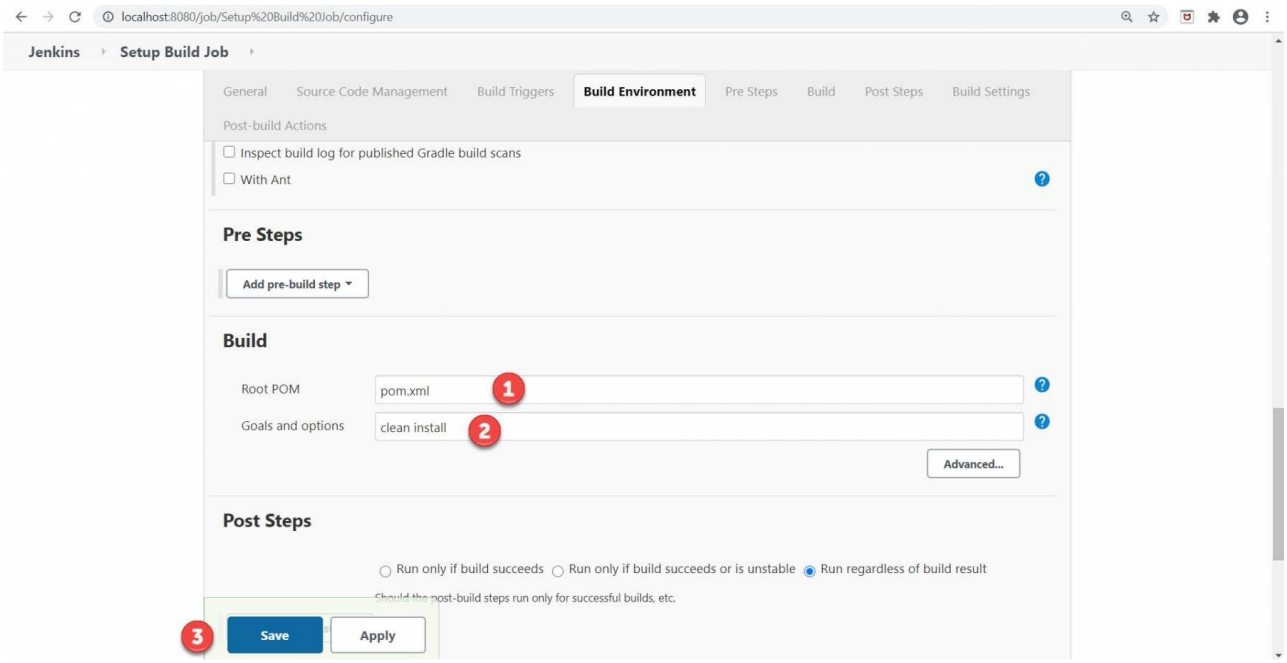
4th Step: Select the Git option in ***"Source Code Management "*** as per our requirement because we will pull our Maven project from the *GitHub* repository.



Also, if we need to select the ***"GitHub hook trigger for GITScm polling"*** option in the ***"Build Triggers"*** section, we will trigger our build with the help of webhooks.



5th Step: Now perform the following highlighted steps to move further:

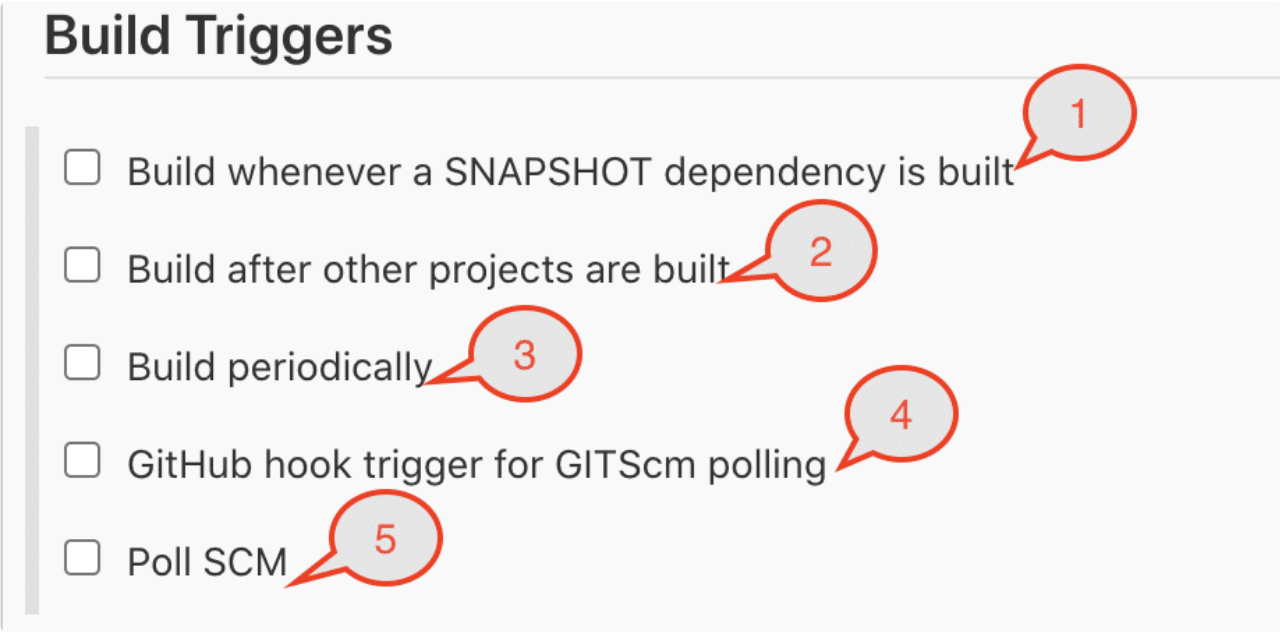


1. *Firstly, give the relative path of **pom.xml** in the **Root POM** textbox, as we do have the pom.xml at the root of the project, so we directly provided the file's name.*
2. *Secondly, type **"clean install"** in the **Goals and options** textbox as **"maven clean," "maven install,"** as well, as **"maven test"** are the maven commands while running maven build but here **"clean install"** command is sufficient to trigger build from Jenkins.*
3. *Finally, click on the **Save** button.*

So, our *Maven project* setup finishes and is ready to run. In the further subsection, we will see how to execute a *Maven* project in *Jenkins*.

How to execute a Maven project in Jenkins?

As our maven project is already setup in the previous section, we are now ready to execute it. After setup, a *Maven* project is similar to other job types in *Jenkins. Jenkins* provides multiple ways to execute jobs. Apart from the manual execution, a few of the options to automatically executing the jobs are highlighted below:



You can select one or all of the above-mentioned options to trigger the build automatically. Let's understand under what all conditions these options will trigger the build:

Build Trigger Option	***Behavior***
Build whenever a SNAPSHOT dependency is built	If checked, Jenkins will parse the POMs of this project and check if any of its snapshot dependencies are built on this Jenkins. If so, Jenkins will set up a build dependency relationship so that whenever the dependency job builds, and a new SNAPSHOT jar creates, Jenkins will schedule a build of this project.This is convenient for automatically performing continuous integration. Jenkins will check the snapshot dependencies from the element in the POM, as well as s and ``s used in POMs.

Build after other projects are built	Set up a trigger so that new build schedules for this project when some other projects finish building. This is convenient for running an extensive test after a build is complete, for example.This configuration complements the ***"Build other projects"*** section in the ***"Post-build Actions"*** of an upstream project but is preferable when you want to configure the downstream project.
Build periodically	This feature is primarily for using Jenkins as a CRON replacement, and it is ***not ideal for continuously building software projects.*** When people initially start continuous integration, they often use the idea of regularly scheduled builds like nightly/weekly that they use this feature. However, the point of continuous integration is to start a build as soon as a change is made to provide quick feedback. To do that, you need to hook up the SCM change notification to Jenkins.
GitHub hook trigger for GITScm polling	If Jenkins receives/ gets PUSH GitHub hook from repo defined in the Git SCM section, it will trigger Git SCM polling logic. In fact, polling logic belongs to Git SCM.
Poll SCM	Configure Jenkins to poll changes in SCM. Note that this will be an expensive/ cost-incurring CVS operation, as every polling requires Jenkins to scan the entire workspace and verify it with the server.

You can select any of these options for the auto-execution of *Jenkins* jobs. We will cover the details of all these options in future articles.

Key Takeaways:

- *Maven is a powerful build management tool for Java projects to assist with a complete life cycle build framework. Moreover, its basis is the concept of **POM** (Project Object Model) in which all configurations can be done with the help of a **pom.xml** file.*
- *Additionally, we can download binary zip according to our OS and save it in any directory. After that, we need to set up a maven path in the environment variables section in the system and the Global tool configuration in Jenkins.*
- *Finally, after setting up, we need to write pom.xml text in the build section and command **"clean install"** to drive maven functionalities.*
- *Moreover, we can trigger build automatically in Jenkins either manually or various Build trigger options available with Jenkins.*