Jenkins Tutorial



Jenkins Tutorial is designed for both beginners and professionals. Our Tutorial provides all the basic and advanced concepts of Jenkins, such as Jenkins installation, Jenkins Configuration, Jenkins Pipeline, etc.

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects, which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

What is Jenkins?

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

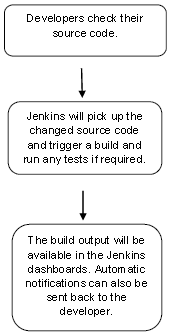
Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

**For example:** If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

* Perform a software build using a build system like Gradle or Maven Apache
* Execute a shell script
* Archive a build result
* Running software tests

**Work Flow:**



History of Jenkins

Kohsuke Kawaguchi, who is a Java developer, working at SUN Microsystems, was tired of building the code and fixing errors repetitively. In 2004, he created an automation server called **Hudson** that automates build and test task.

In 2011, Oracle who owned Sun Microsystems had a dispute with Hudson open source community, so they forked Hudson and renamed it as **Jenkins**.

Both Hudson and Jenkins continued to operate independently. But in short span of time, Jenkins acquired a lot of contributors and projects while Hudson remained with only 32 projects. Then with time, Jenkins became more popular, and Hudson is not maintained anymore.

What is Continuous Integration?

Continuous Integration *(CI)* is a development practice in which the developers are needs to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

Continuous Integration with Jenkins

Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to *develop software*, but this process has many problems.

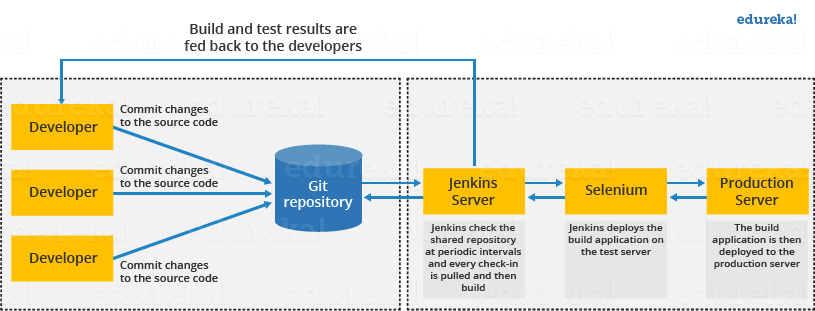
* Developer teams have to wait till the complete software is developed for the test results.
* There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
* It slows the software delivery process.
* Continuous feedback pertaining to things like architectural or coding issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
* The whole process was manual which increases the threat of frequent failure.

It is obvious from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction.

So to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.

This is what Continuous Integration (CI) is all about. Jenkins is the most mature Continuous Integration tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



**Let's see how Jenkins works**. The above diagram is representing the following functions:

* First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
* If the build fails, then the concerned team will be notified.
* If built is successful, then Jenkins server deploys the built in the test server.
* After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
* It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

Advantages and Disadvantages of using Jenkins

**Advantages of Jenkins**

* It is an open source tool.
* It is free of cost.
* It does not require additional installations or components. Means it is easy to install.
* Easily configurable.
* It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
* It is built in java and hence it is portable.
* It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
* Easy support, since it open source and widely used.
* Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

**Disadvantages of Jenkins**

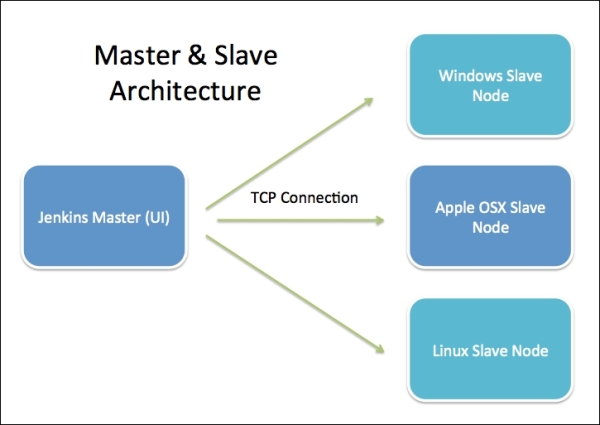
* Its interface is out dated and not user friendly compared to current user interface trends.
* Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
* CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

* Jenkins Master/Server
* Jenkins Slave/Node/Build Server



Jenkins Master

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

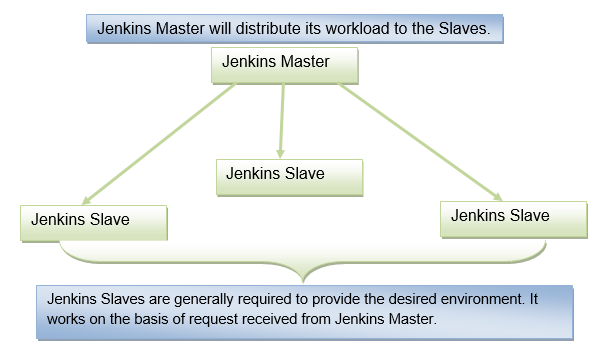
The server's job or master's job is to handle:

* Scheduling build jobs.
* Dispatching builds to the nodes/slaves for the actual execution.
* Monitor the nodes/slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master/Server instance of Jenkins can also execute build jobs directly.

Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.



The above diagram is self explanatory. It consists of a Jenkins Master which is managing three Jenkins Slaves.

Java and Tomcat Setup for Jenkins

Java Setup

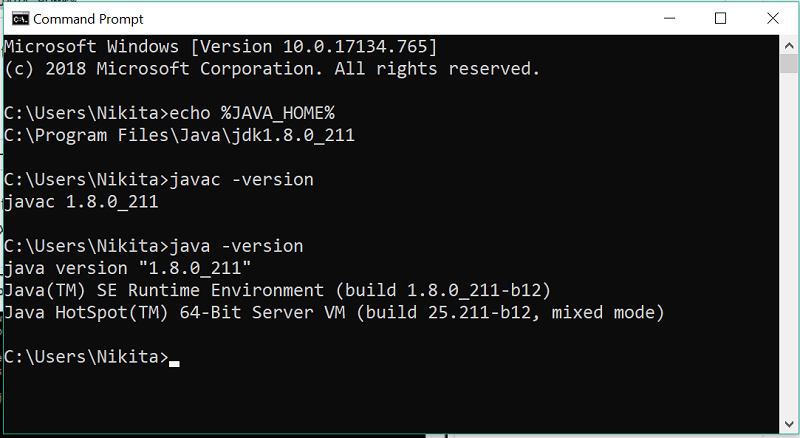
Since Jenkins is written in Java. Therefore, Java must be installed on your system. To download and install the Java, go to our previous chapter of this tutorial.

Now, to set up the Java\_Home environment variable on Windows [**click here**](https://javatutorial.net/set-java-home-windows-10).

Once the java has been installed properly on your system and Java environment variable has been set, then you can verify it by using the following commands:

1. C:\ echo %JAVA\_HOME%
3. C:\ javac -version
5. C:\ java -version

The following output should come:

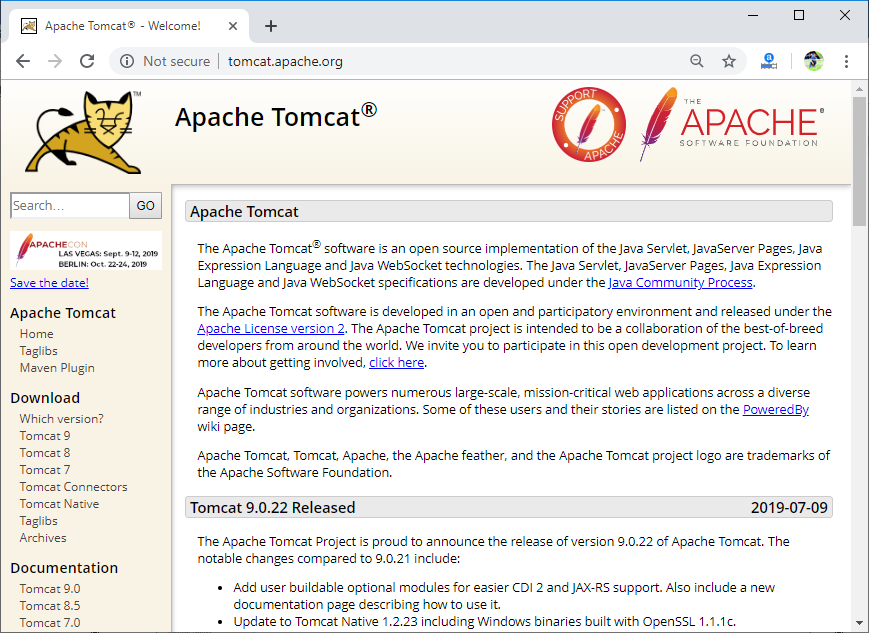


Tomcat Setup

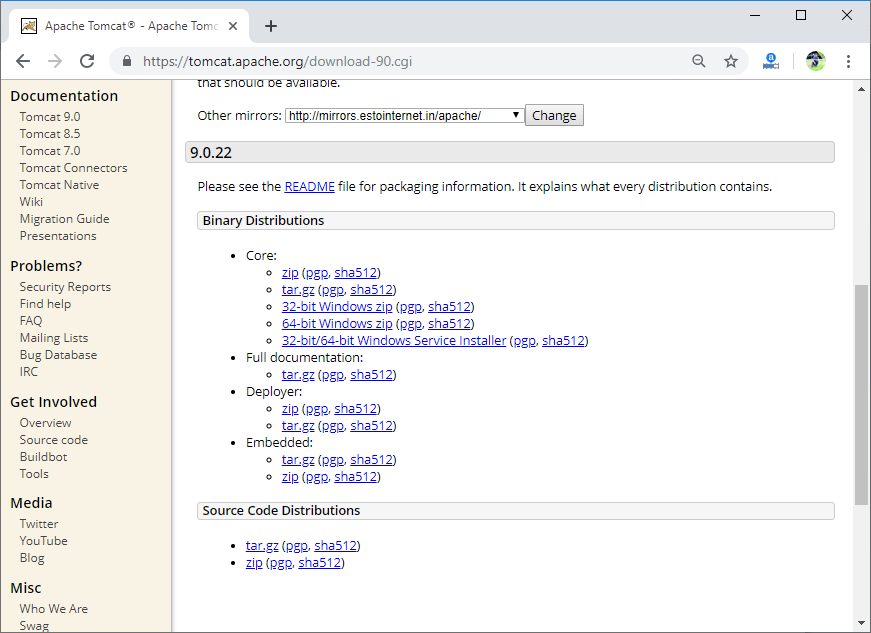
Download Tomcat:

The official website to download the tomcat is [http://tomcat.apache.org/](https://tomcat.apache.org/).

When you click the given link, you will get the home page of the official tomcat website as given below:



Go to the link <https://tomcat.apache.org/download-90.cgi> to download the latest version of tomcat.

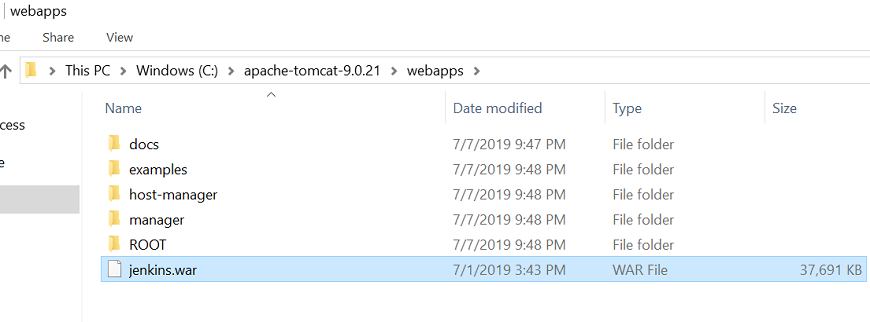


Go to the "Binary Distributions" section and download the file according to your platform. Here I am using 64-bit Windows zip for my 64-bit Windows operating system.

Unzip the contents of the downloaded zip file.

Jenkins and Tomcat Setup

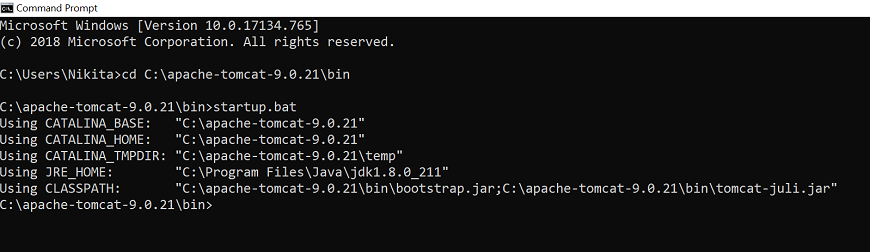
Copy the downloaded Jenkins.war file (downloaded from the previous section) and copy it to the **webapps** folder of the tomcat directory.



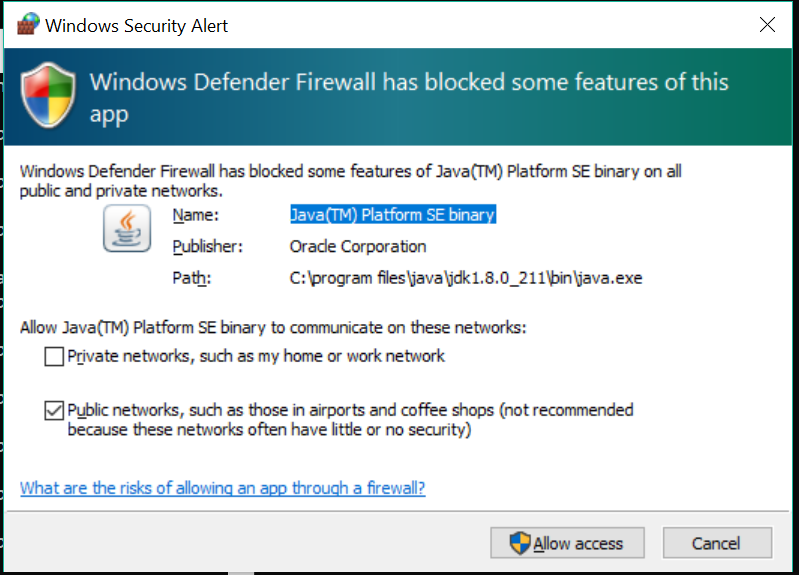
Now open the command prompt. From the command prompt, browse to the directory where the tomcat is located. Then, go to the bin directory of this tomcat folder and run the **startup.bat** file.

1. C:\apache-tomcat-9.0.21\bin>startup.bat

When you run the above command then following output will come:



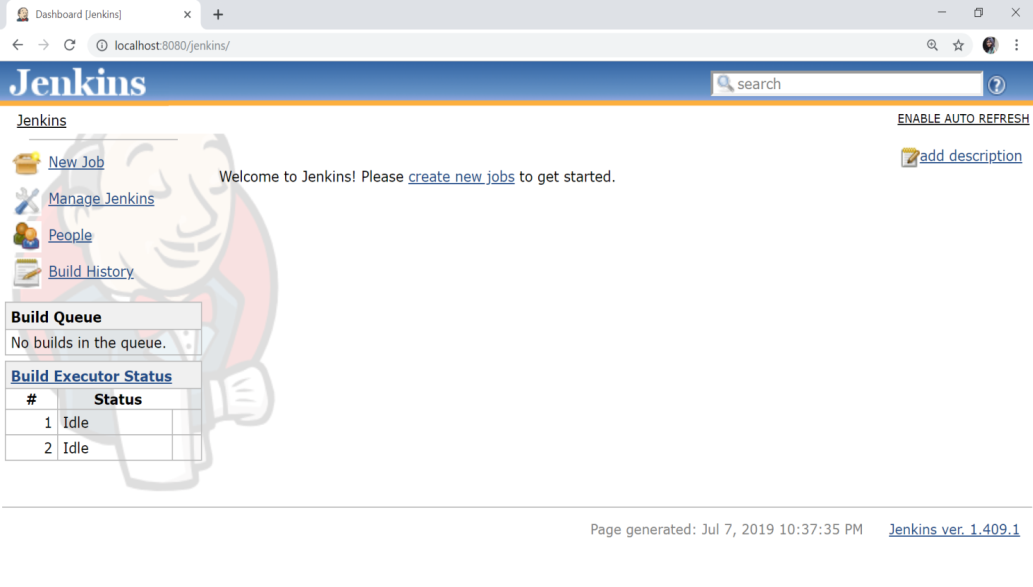
And two more windows will open. i.e., "Tomcat" window and "Windows Security Alert" window. Click on **Allow access** button in Windows Security Alert Window.


Once the processing is completed, the following line will come in the output of the Tomcat Window:

1. INFO [hudson initialization thread] hudson.TcpSlaveAgentListener.<init> JNLP slave agent listener started on TCP port 61779

Now, open the browser and go the link: <http://localhost:8080/jenkins>. The following page will open:



The setup of Jenkins with Tomcat is successfully done.

# GitHub Setup for Jenkins

Jenkins is a CI (Continuous Integration) server and this means that it needs to check out source code from a source code repository and build code. Jenkins has outstanding support for various source code management systems like Subversion, CVS etc.

Github is the fast becoming one of the most popular source code management systems. It is a web based repository of code which plays a major role in DevOps. GitHub provides a common platform for many developers working on the same code or project to upload and retrieve updated code, thereby facilitating continuous integration. Jenkins works with Git through the Git plugin.

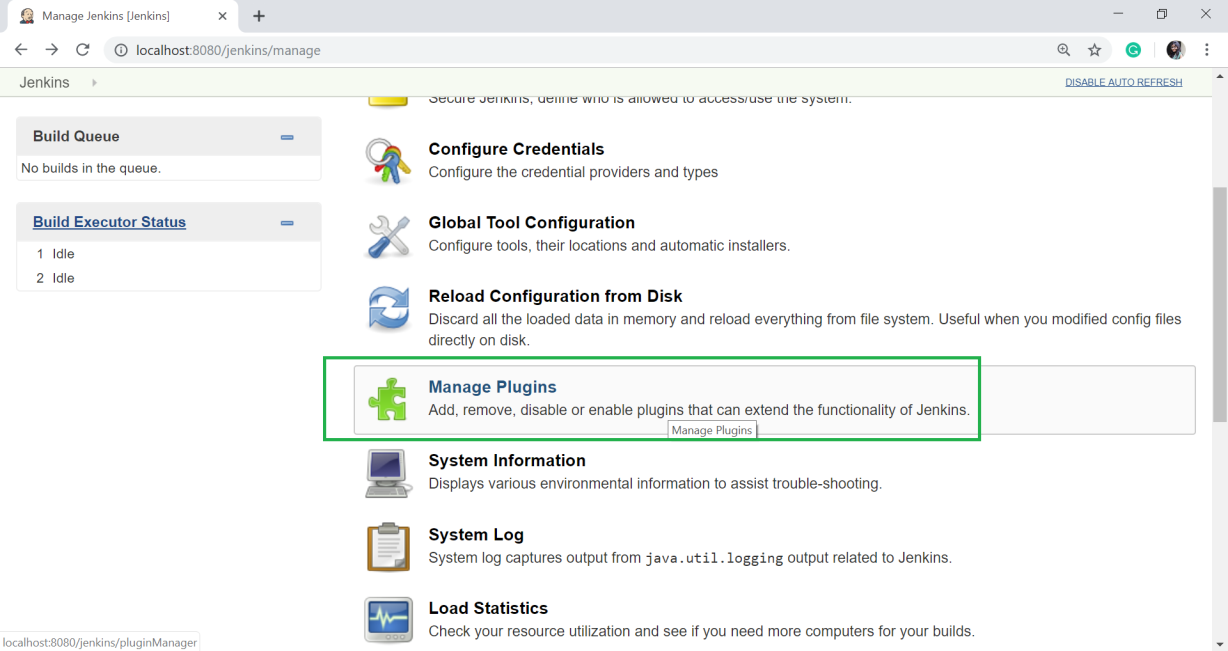
Connecting a GitHub private repository to a private instance of Jenkins can be tricky.

To do the GitHub setup, make sure that internet connectivity is present in the machine where Jenkins is installed.

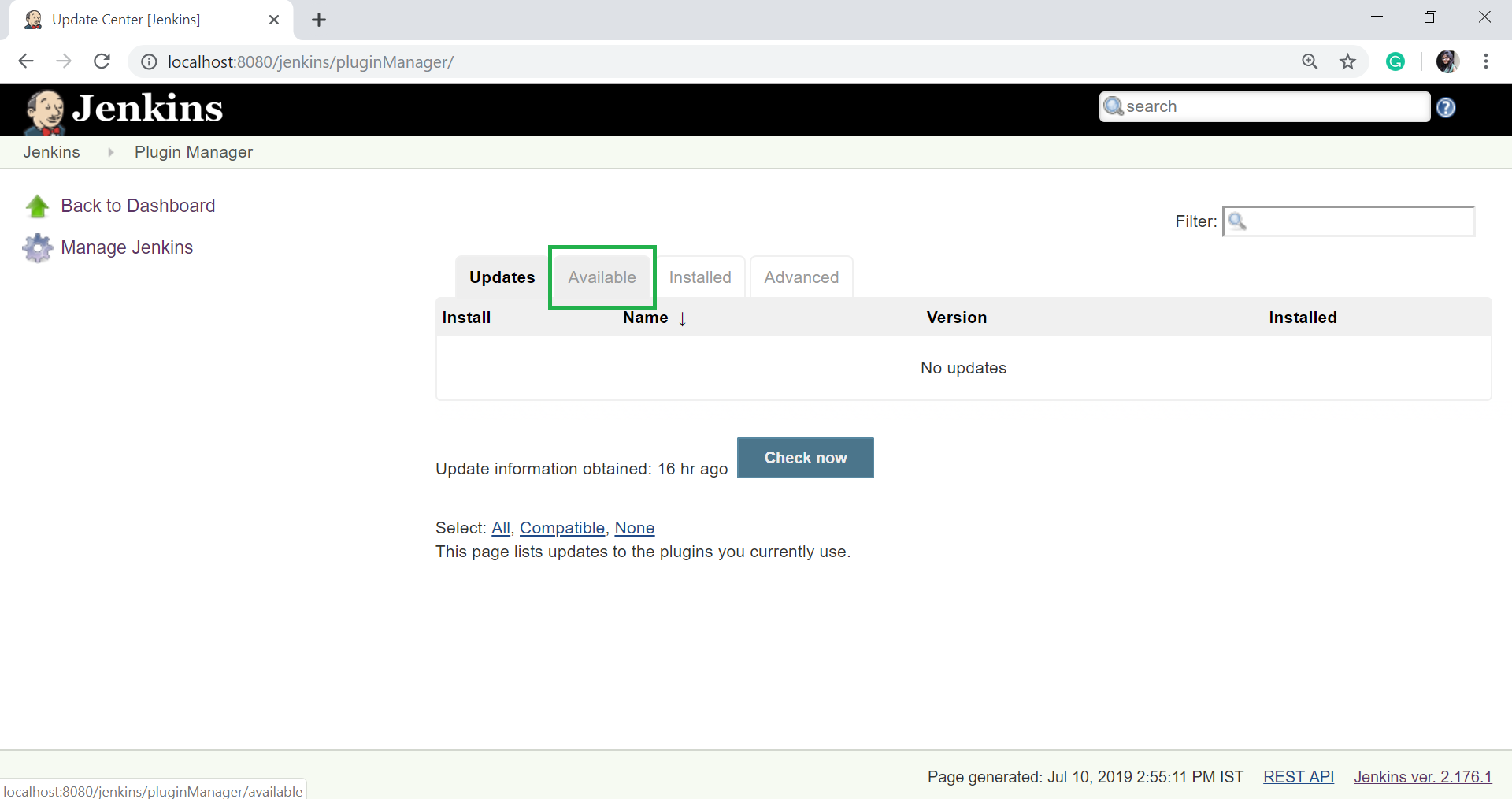
* In the Home screen of the Jenkins (Jenkins Dashboard), click on the **Manage Jenkins** option on the left hand side of the screen.



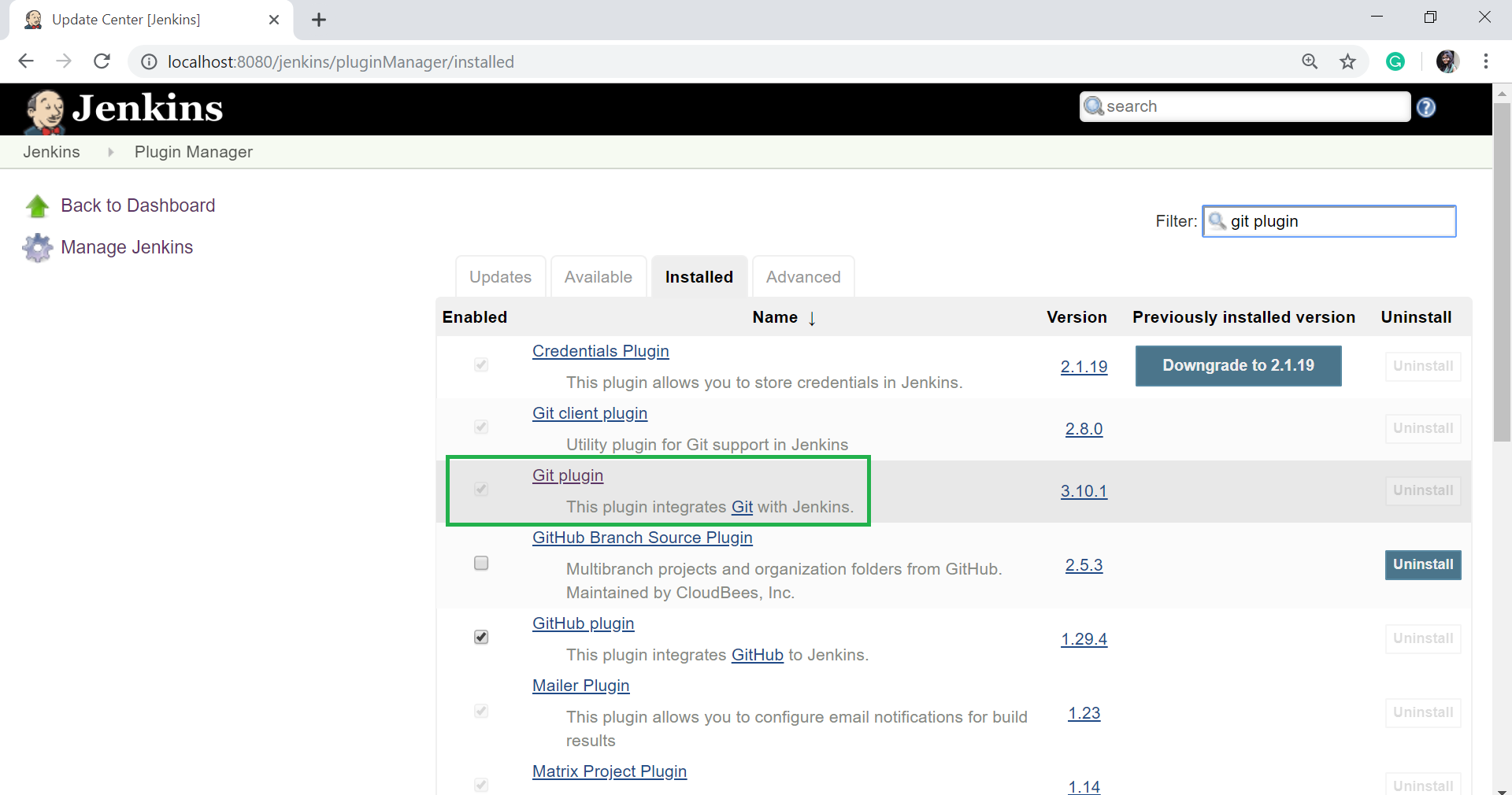
* Now, click on the **Manage Plugins** option.



* In the next page, click on the "Available tab".



* The "Available" tab gives a list of plugins which are available for downloading. In the Filter tab type, type the "Git Plugin".
* Select the Git Plugin.
* Click on the "**install without restart**". The plugin will take some time to finish downloading depending on your internet connection, and will be installed automatically.
* You can also click on "**Download now and install after restart**" button in which the git plugin is installed after restart.
* If you already have the Git plugin installed then go to "Installed" tab and in filter option type Git plugin.



* Once all the installations are completed, restart Jenkins by giving the following command in the browser. <http://localhost:8080/jenkins/restart>

After Jenkins is restarted, Git will available as an option while configuring jobs.

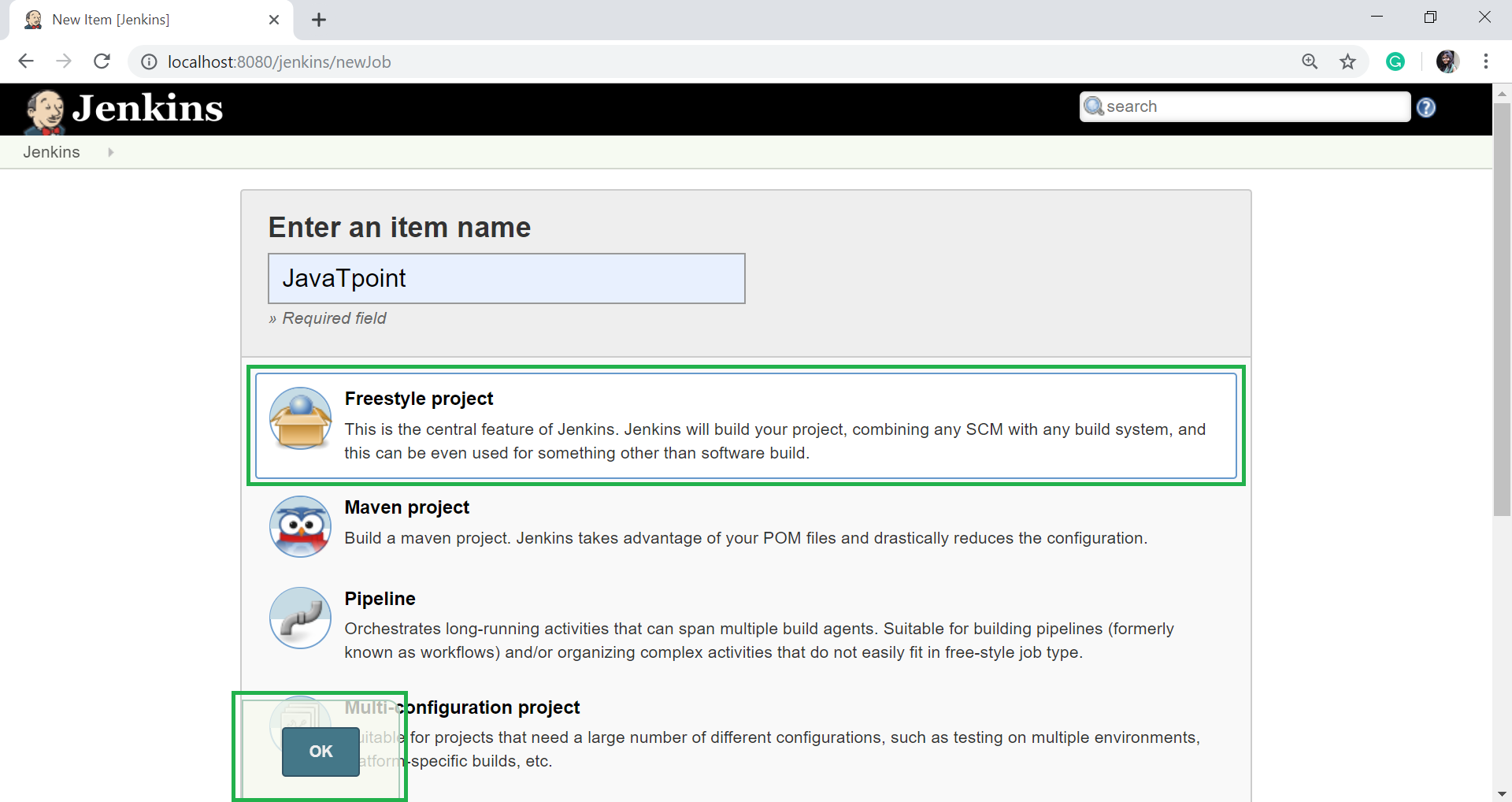
## Integrating Jenkins with GitHub

Let's see the process of integrating GitHub into Jenkins in a windows system.

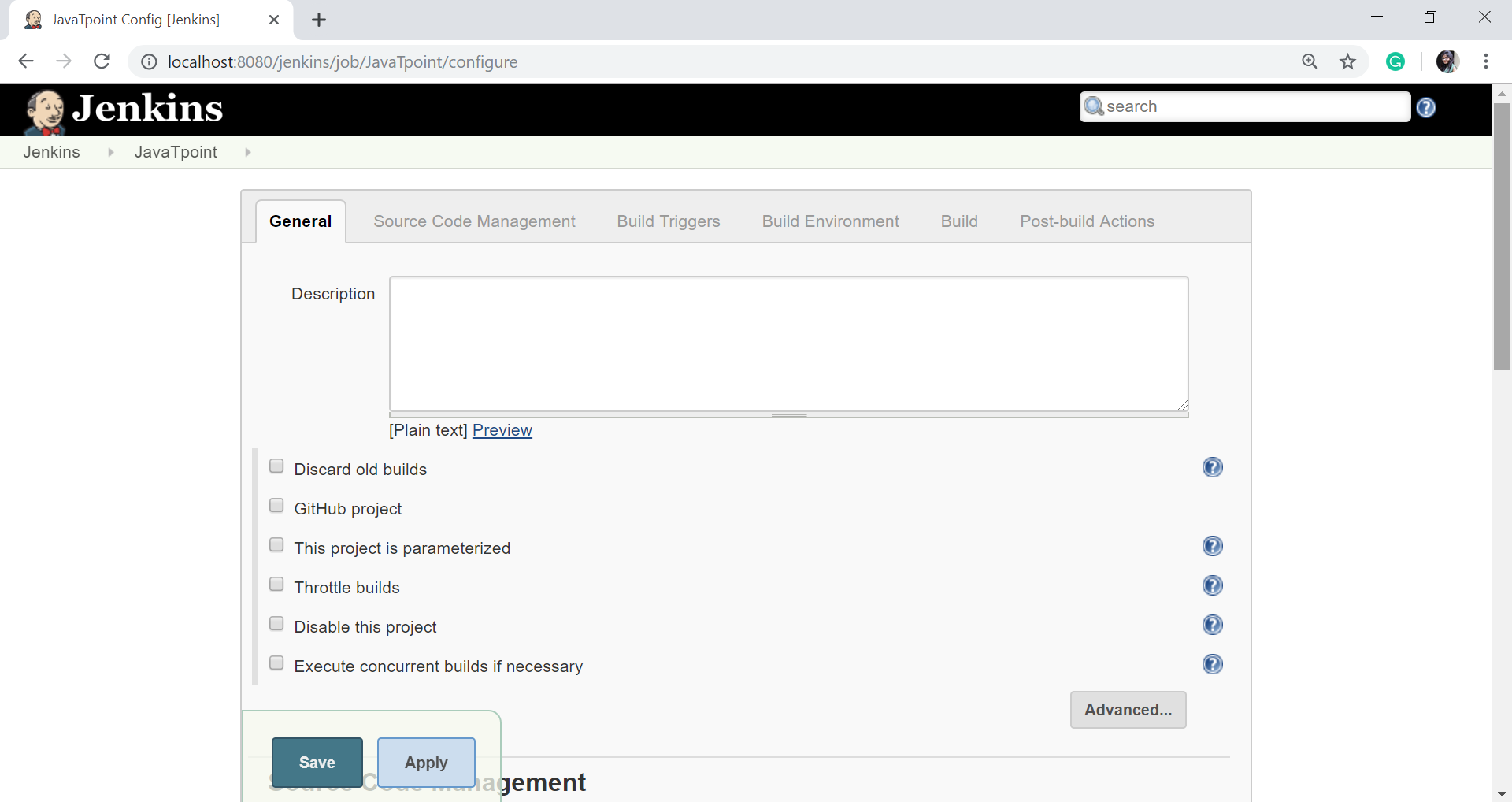
* First create a new job in Jenkins, open the Jenkins Dashboard and click on "create new jobs".



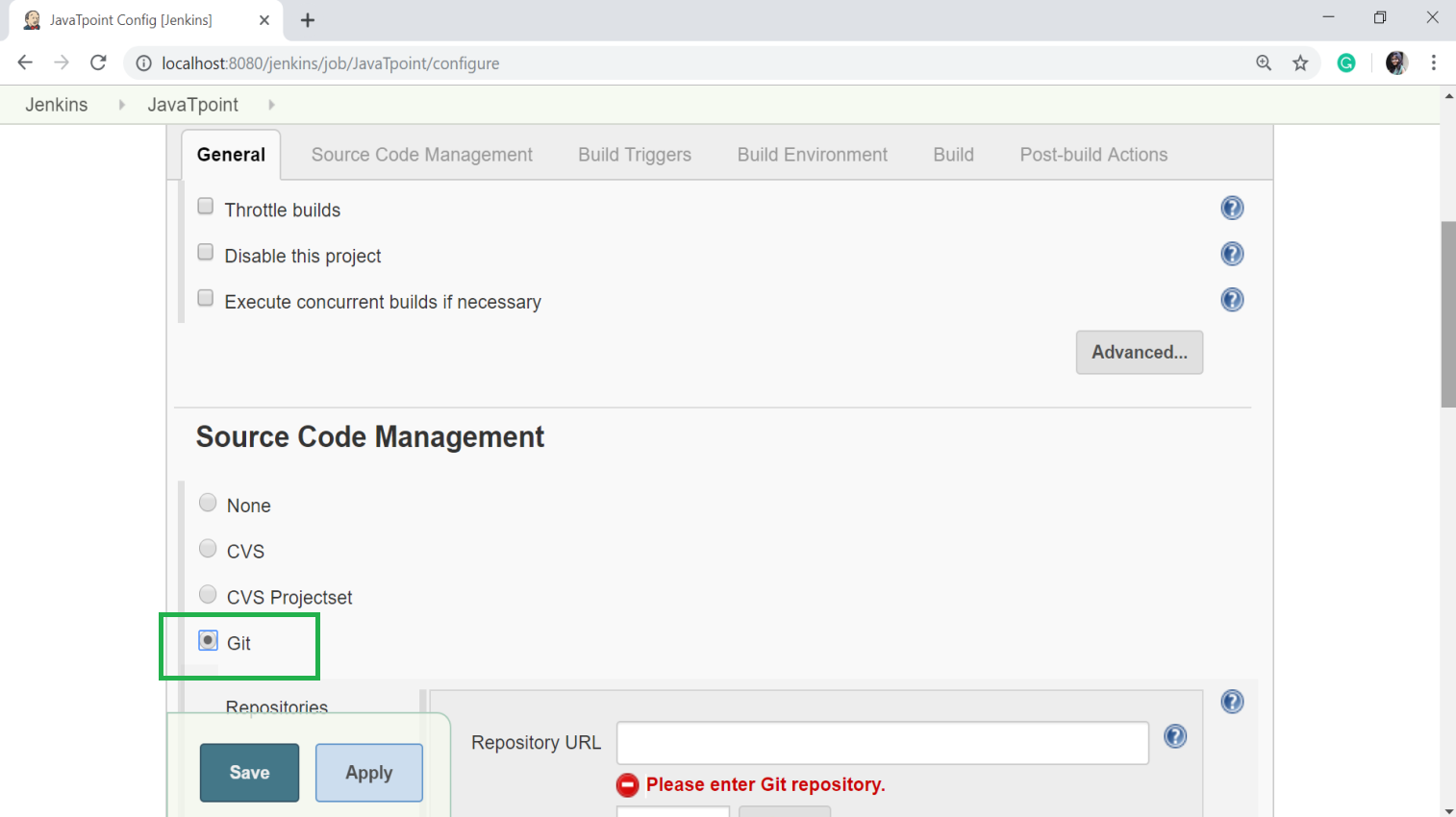
* Now enter the item name and select the job type. For example, item name is **javaTpoint**" and job type is "**Freestyle project**".
* Click on **OK**.



* Once you click OK, the page will be redirected to its project configuration. Enter the project information:

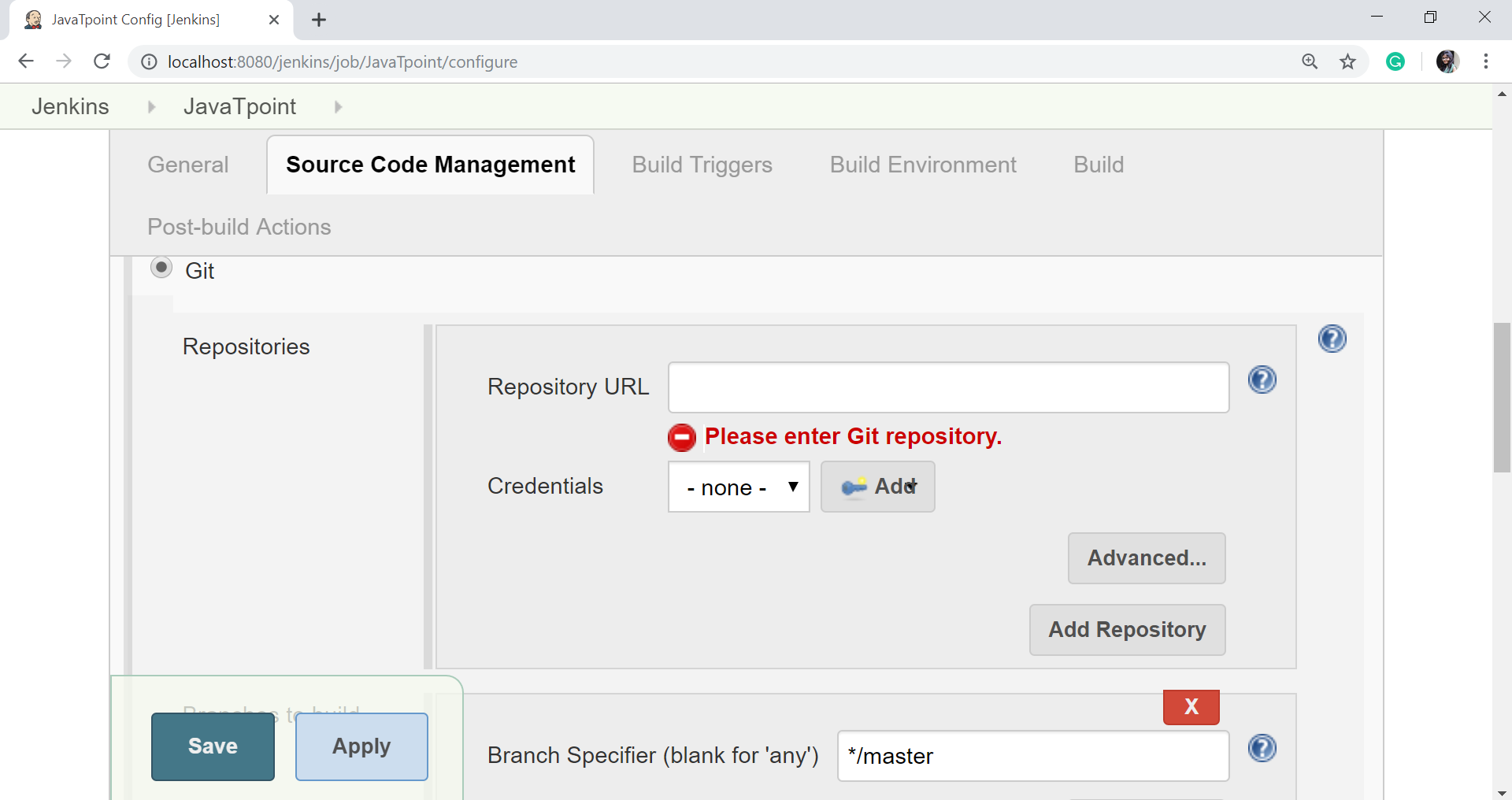


* Now, under the "Source Code Management" you will see the Git option, if your **Git** plugin has been installed in Jenkins:

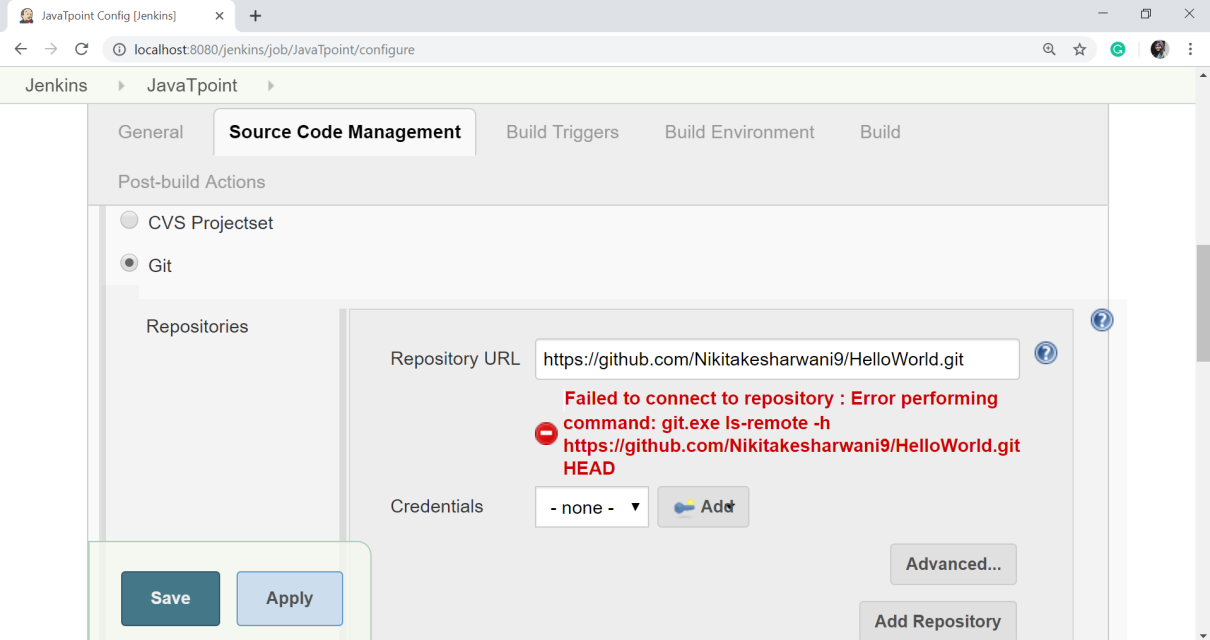


#### Note: if the Git option does not appear, try to reinstall the plugins, followed by a restart into your Jenkins dashboard.

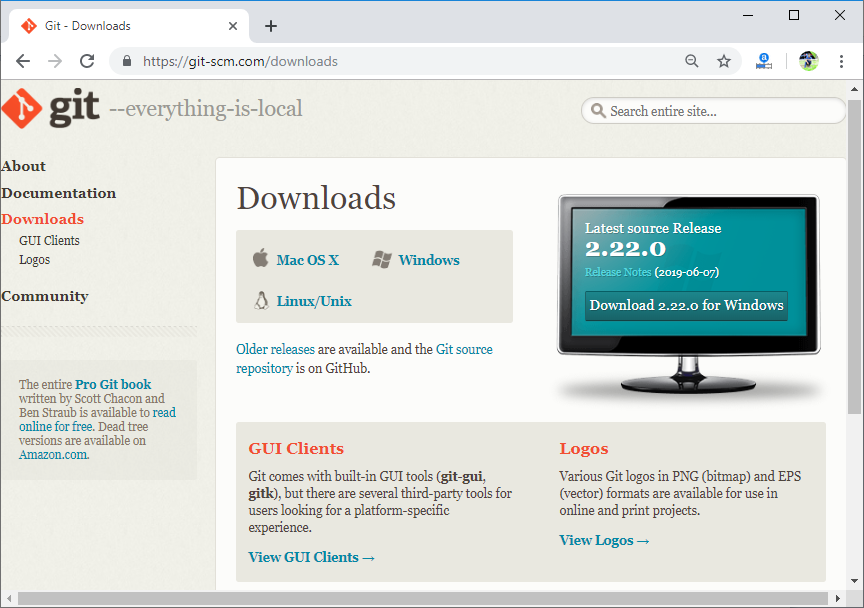
* Enter the Git repository URL on the "Repository URL" option to pull the code from GitHub.



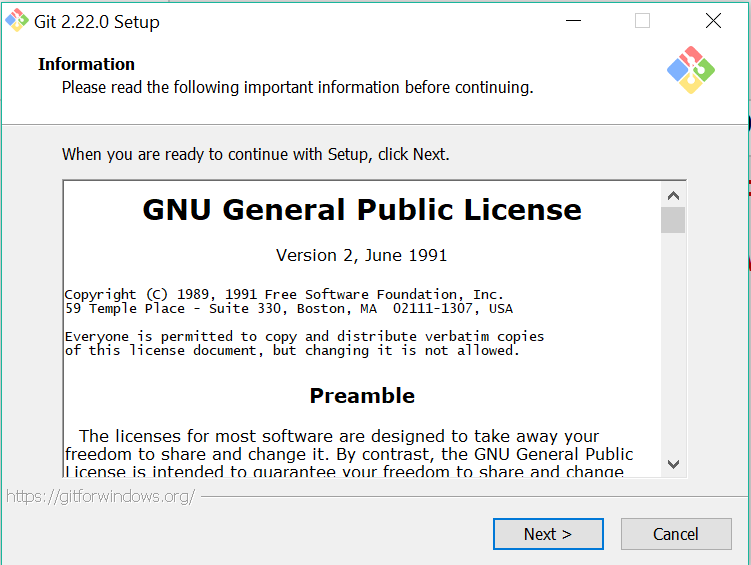
* You might get an error when first time you enter the repository URL. For example:



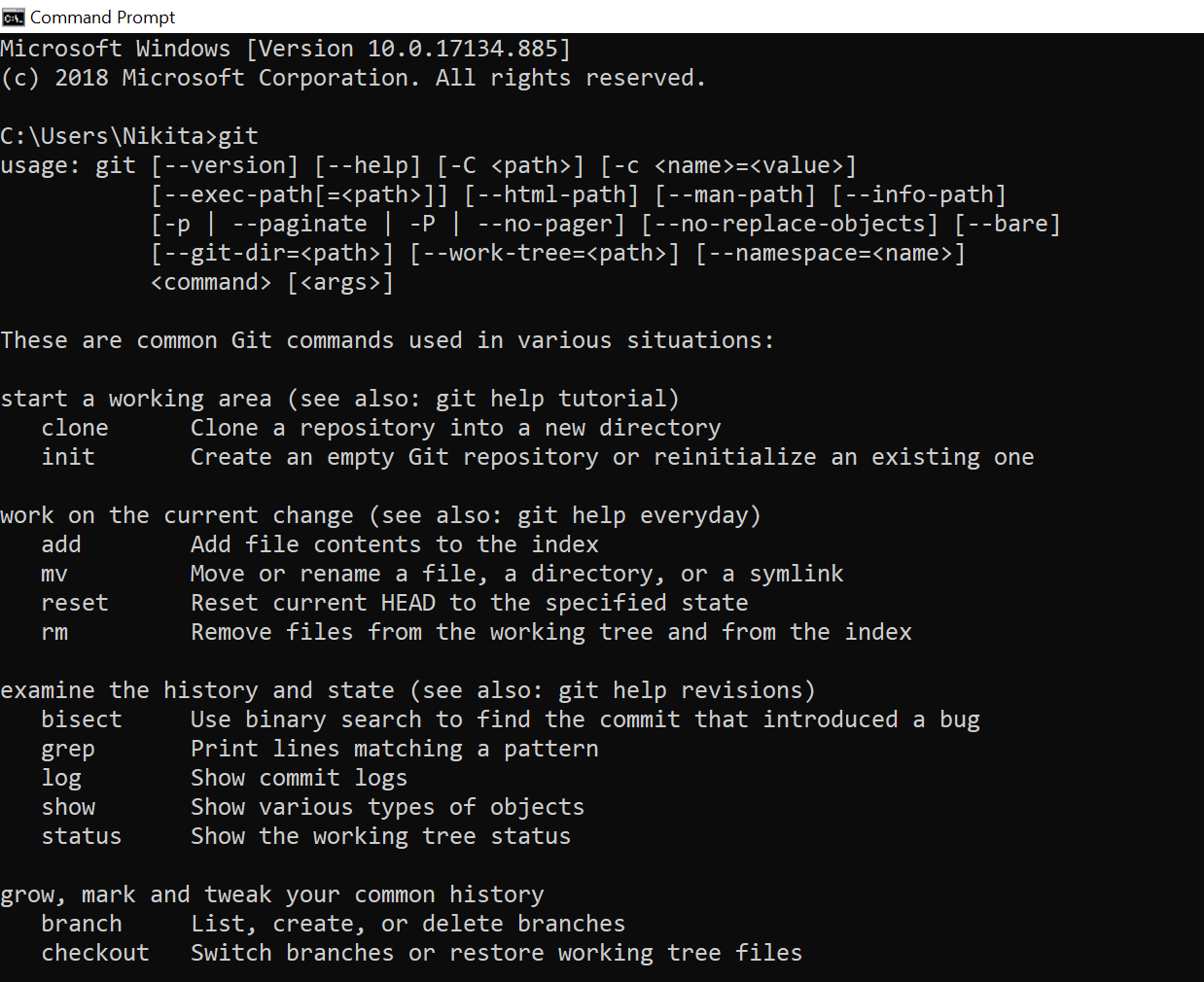
This happens if you don't have Git installed in your system. To install the Git in your system, download the appropriate Git setup according to your operating system. I am installing for windows. Once the download is completed, install the Git.



Complete the following instructions to install the Git:



You can execute Git repositories in your Jenkins once Git has been installed on your system. To check if the Git has been installed on your system, open the command prompt, type Git and press Enter.



In the above screen, you observe that syntax and different options come up for Git. This means that Git has been installed in your machine.

* Now try to add the Git URL into Jenkins.
* Git is now successfully configured on your system.

Maven Setup:

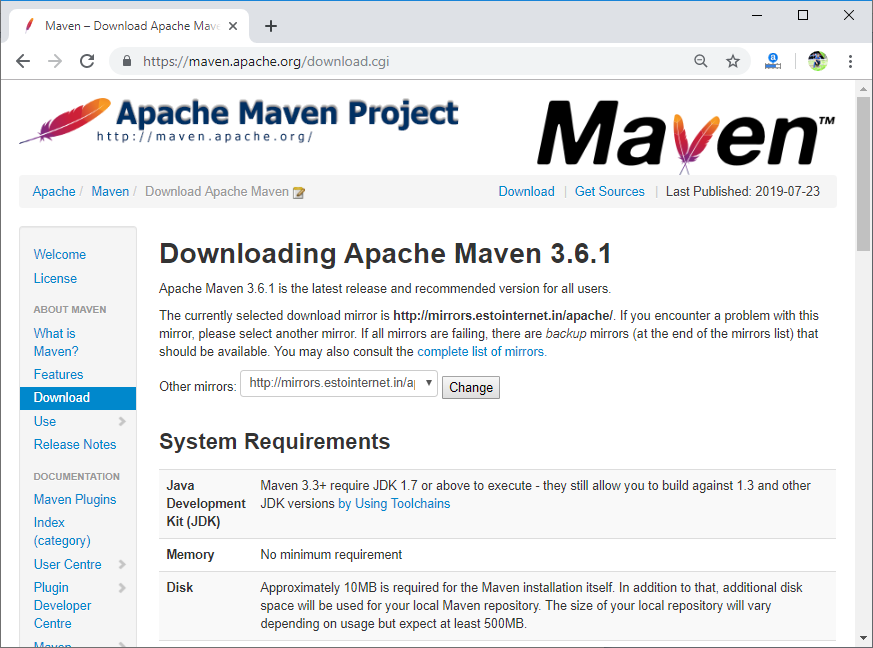
Maven Setup

**Maven** is a powerful project management and comprehension tool that provides complete build life cycle framework to assist developers. It is based on the concept of a POM (Project Object Model) that includes project information and configuration information for Maven such as construction directory, source directory, test source directory, dependency, Goals, plugins etc.

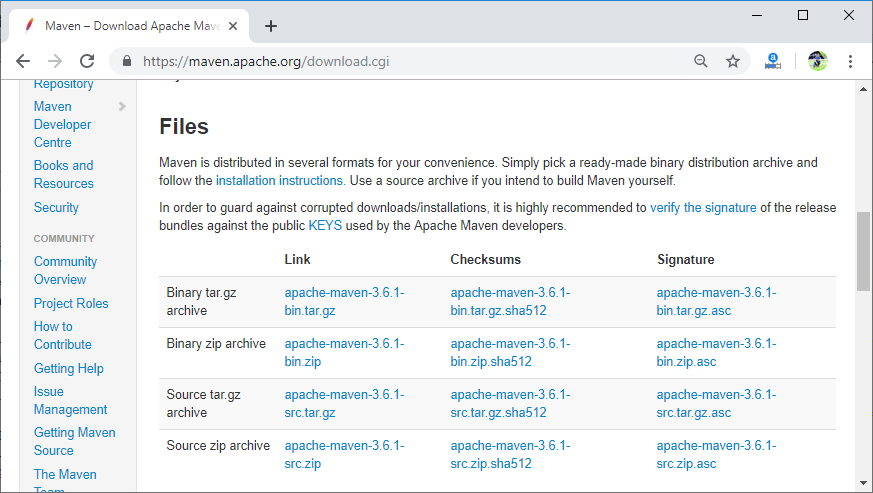
Maven is build automation tool used basically for Java projects, though it can also be used to build and manage projects written in C#, Scala, Ruby, and other languages. Maven addresses two aspects of building software: 1st it describes how software is build and 2nd it describes its dependencies.

Downloading Maven

The official website for Apache Maven is <https://maven.apache.org/download.cgi>. Click on the given link to download the Maven. When you click on the given link, you will get the home page of the official Maven website as given below:



Go to the files section and download the Maven by the given link for Binary zip archive file.



Once the file is downloaded, extract the file into your system.

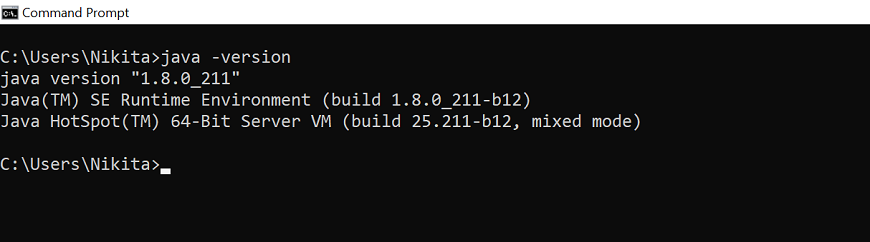
Setting Up Java and Maven in Jenkins

* First of all, you have to set the JAVA\_HOME and MAVEN\_HOME environment variable in your system.

To set the JAVA\_HOME and MAVEN\_HOME path, [**click here**](https://www.mkyong.com/maven/how-to-install-maven-in-windows/).

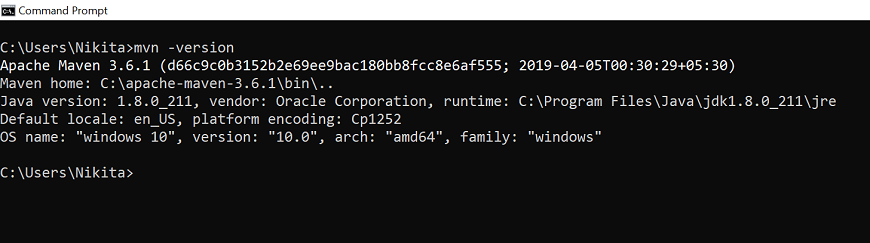
You can verify that the JAVA\_HOME environment variable is properly configured or not by using the following command:

1. C:\java -version

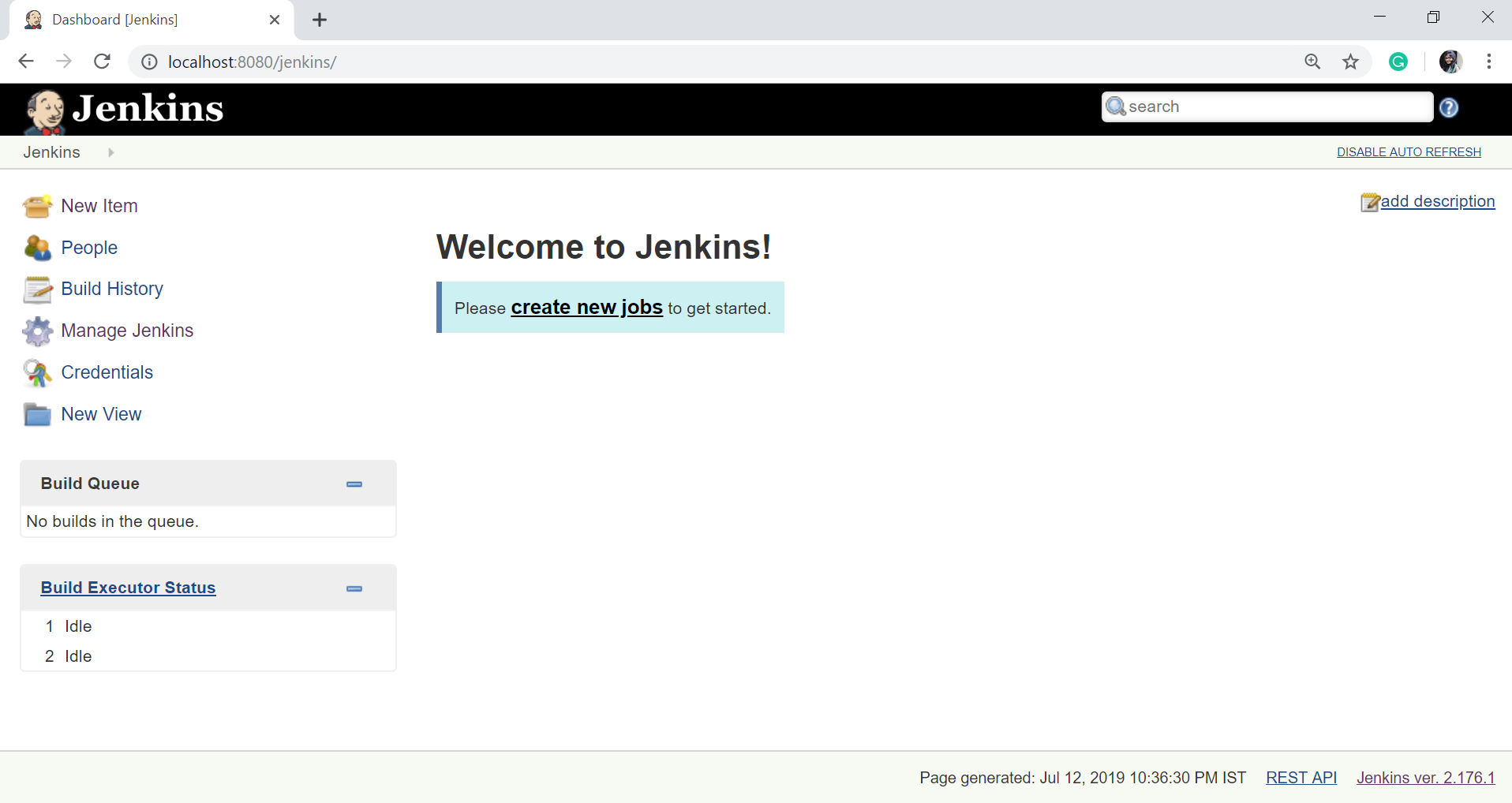


Similarly, you can verify that the JAVA\_HOME environment variable is properly configured or not by using the following command:

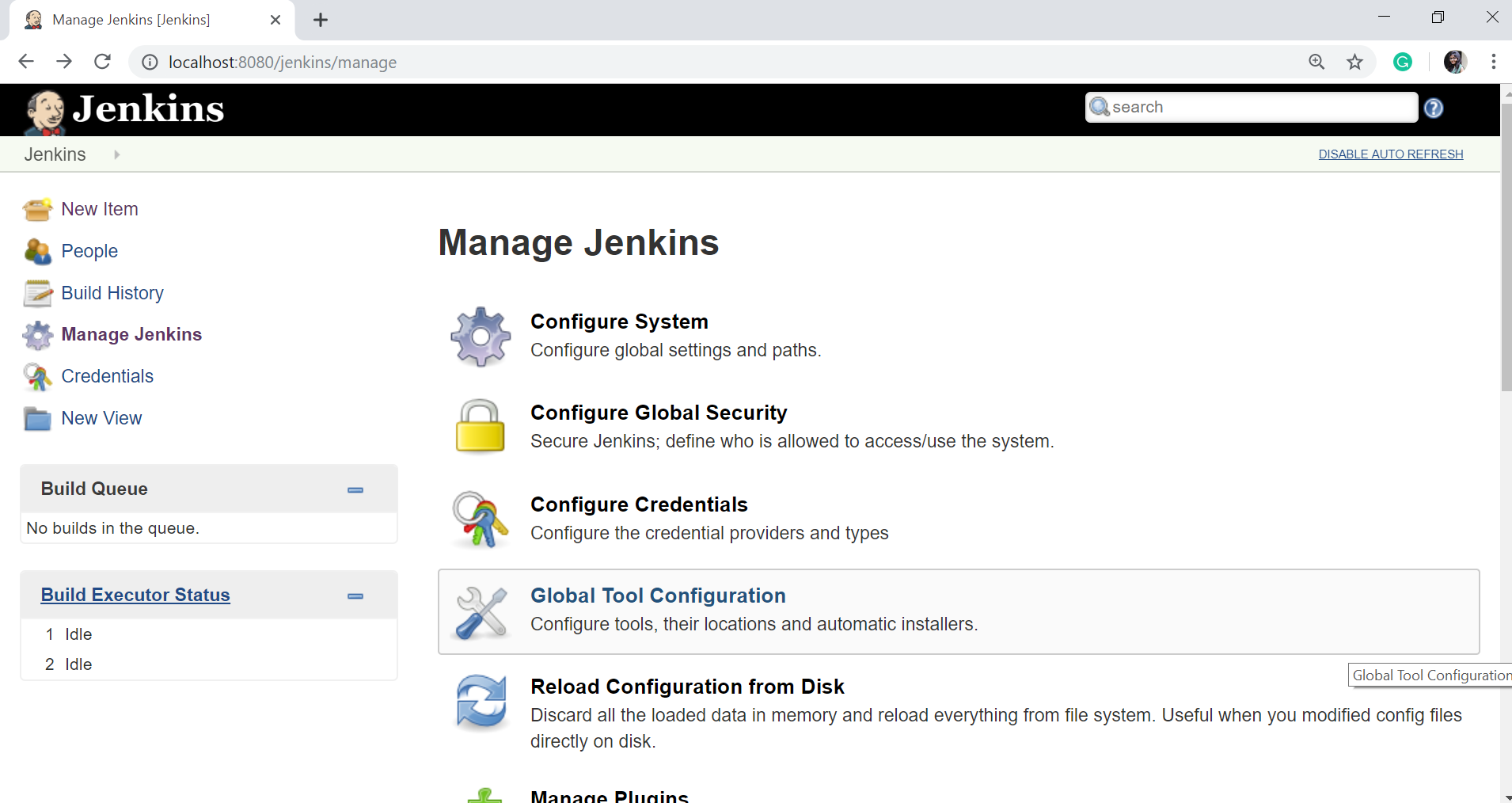
1. C:\ mvn -version



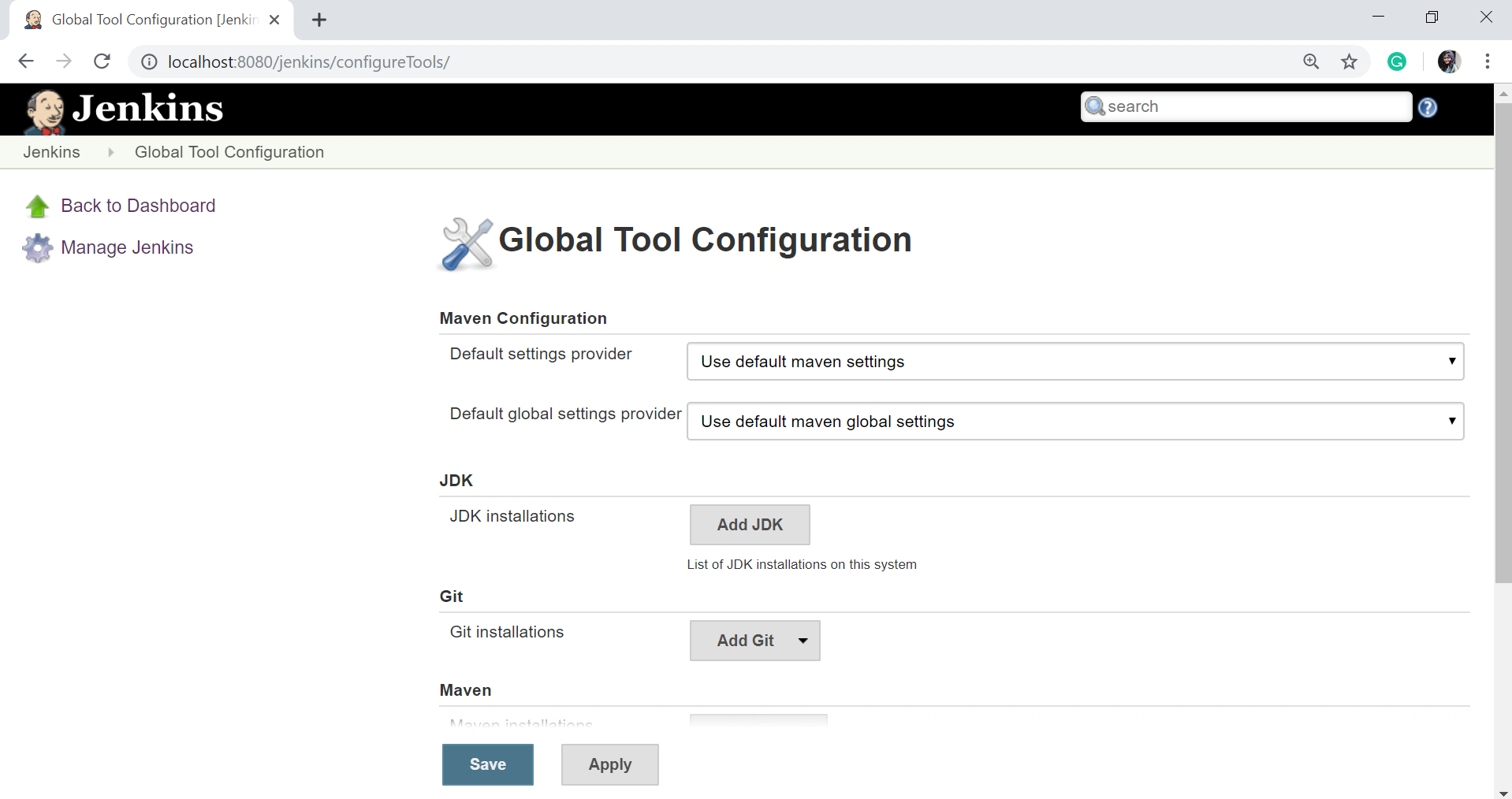
* Now, in the Jenkins dashboard (Home screen) click on manage Jenkins from the left-hand side menu.



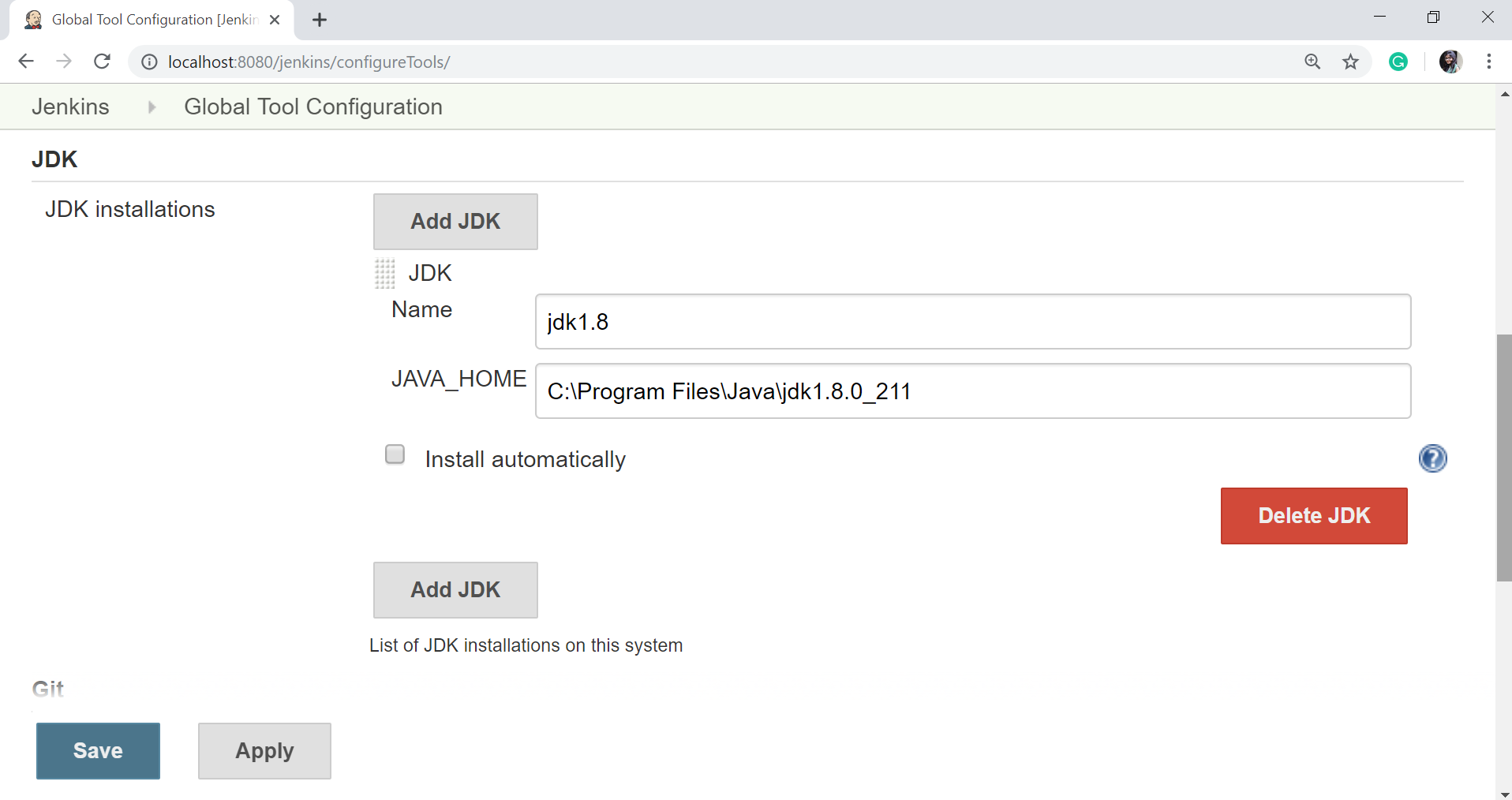
Click on "Global Tool Configuration" option.



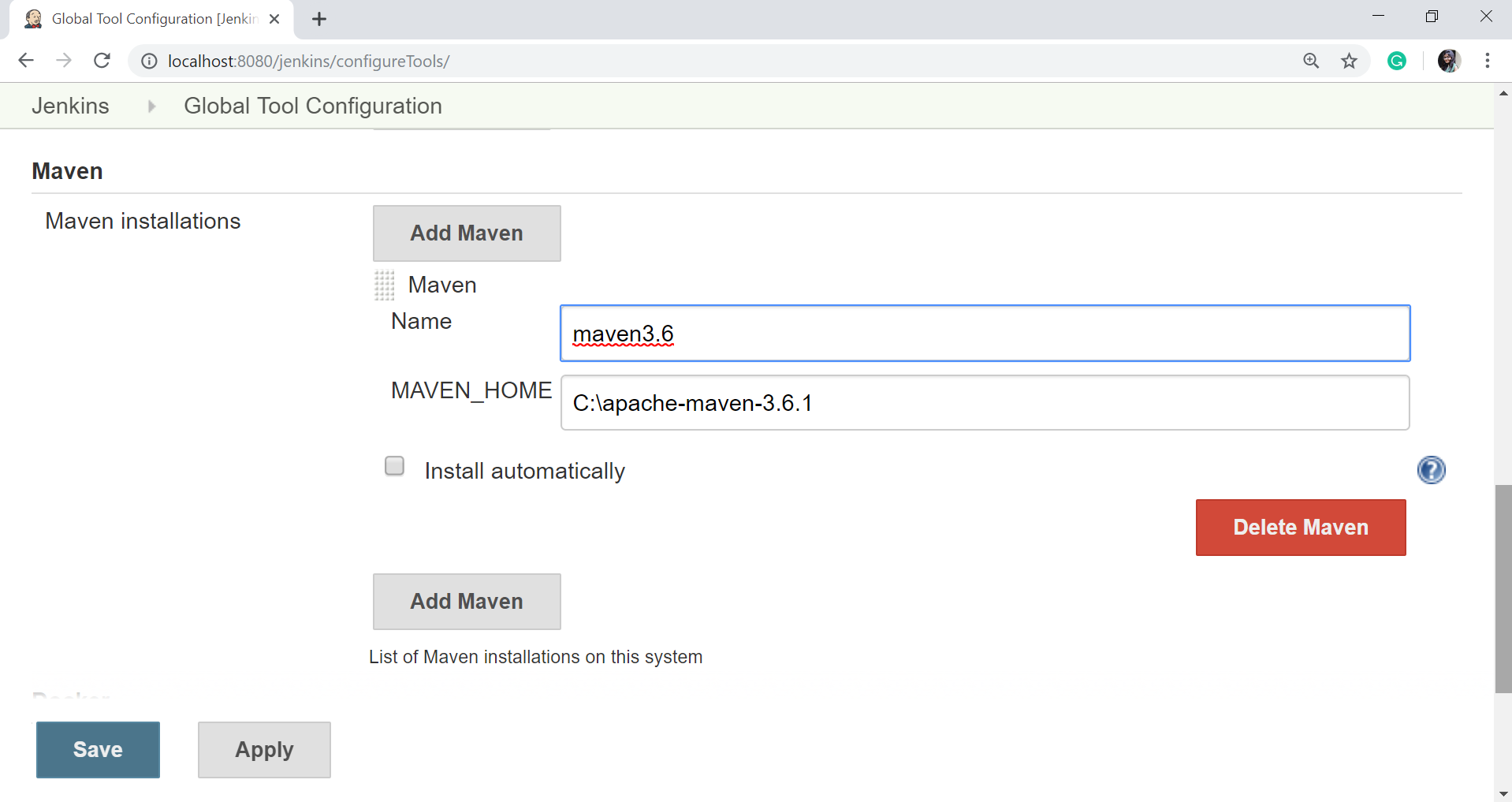
To configure Java, click on "**Add JDK**" button in the JDK section.



Give a **Name** and **JAVA\_HOME** path, or check on **install automatically** checkbox.

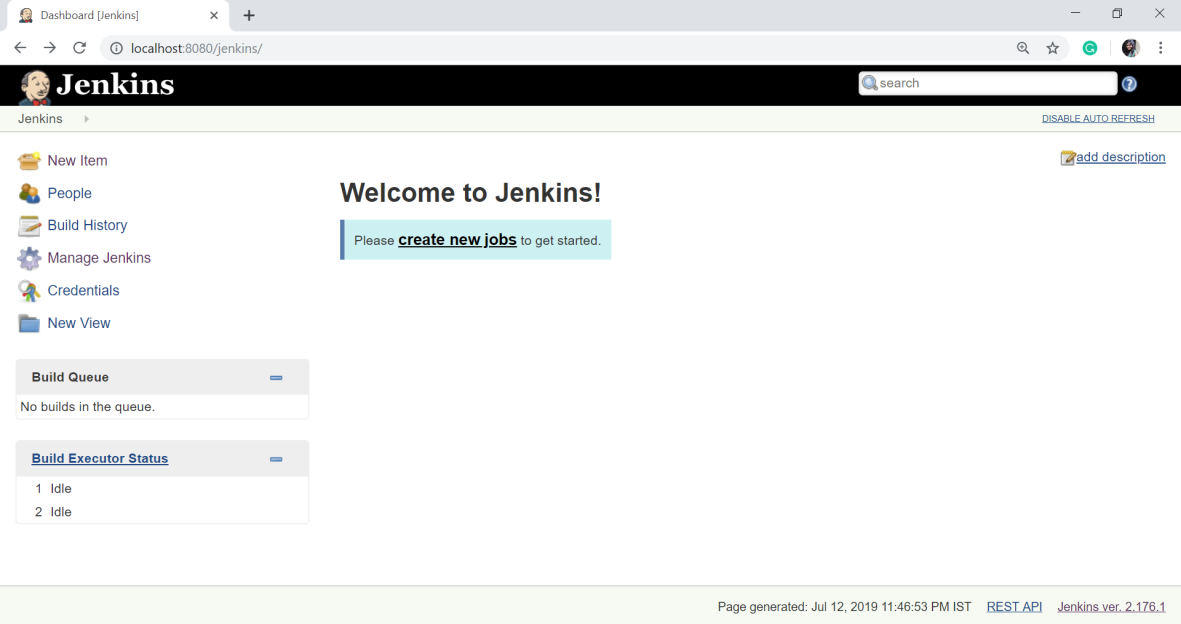


And now, to configure Maven, click on "Add Maven" button in the Maven section, give any **Name** and **MAVEN\_HOME** path or check to install automatically checkbox.

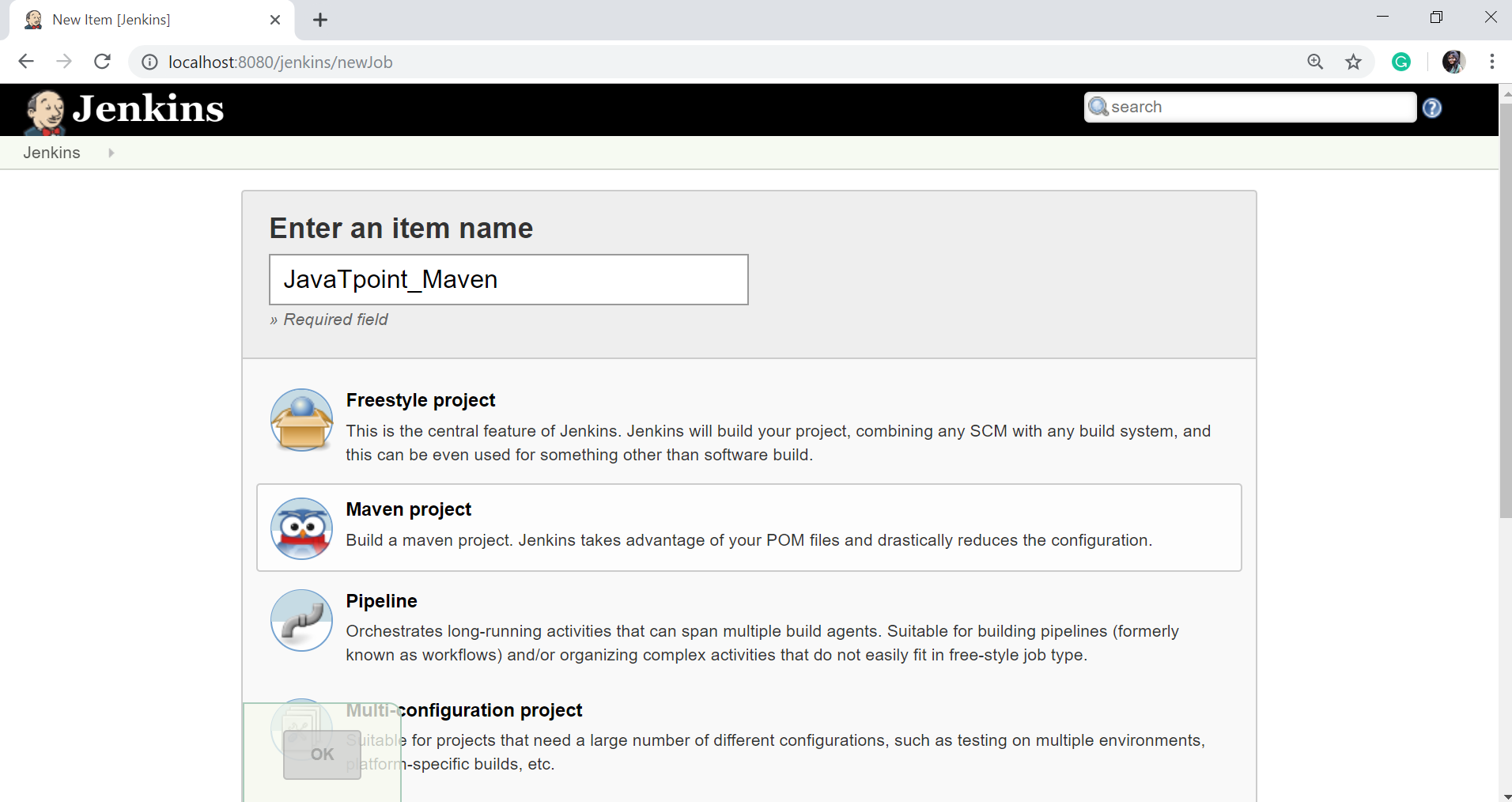


Then, click on the "**Save**" button at the end of the screen.

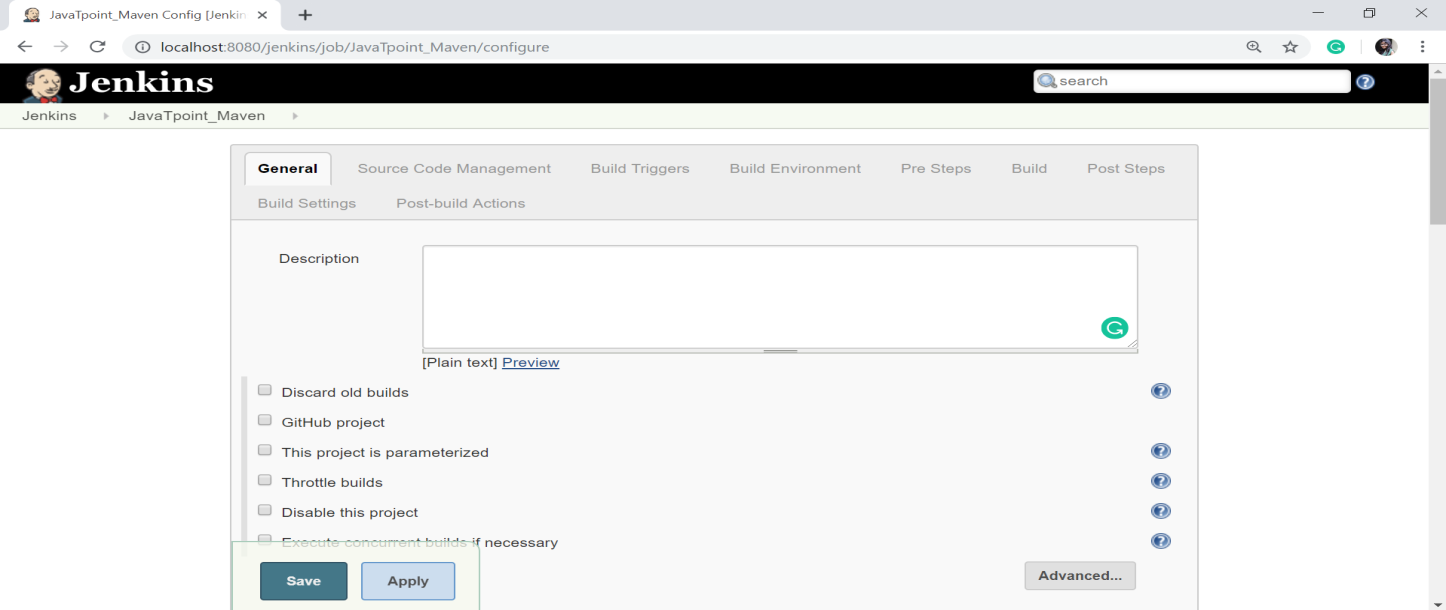
Now, you can create a job with the Maven project. To do that, click on the **New Item** option or **create a new job** option.



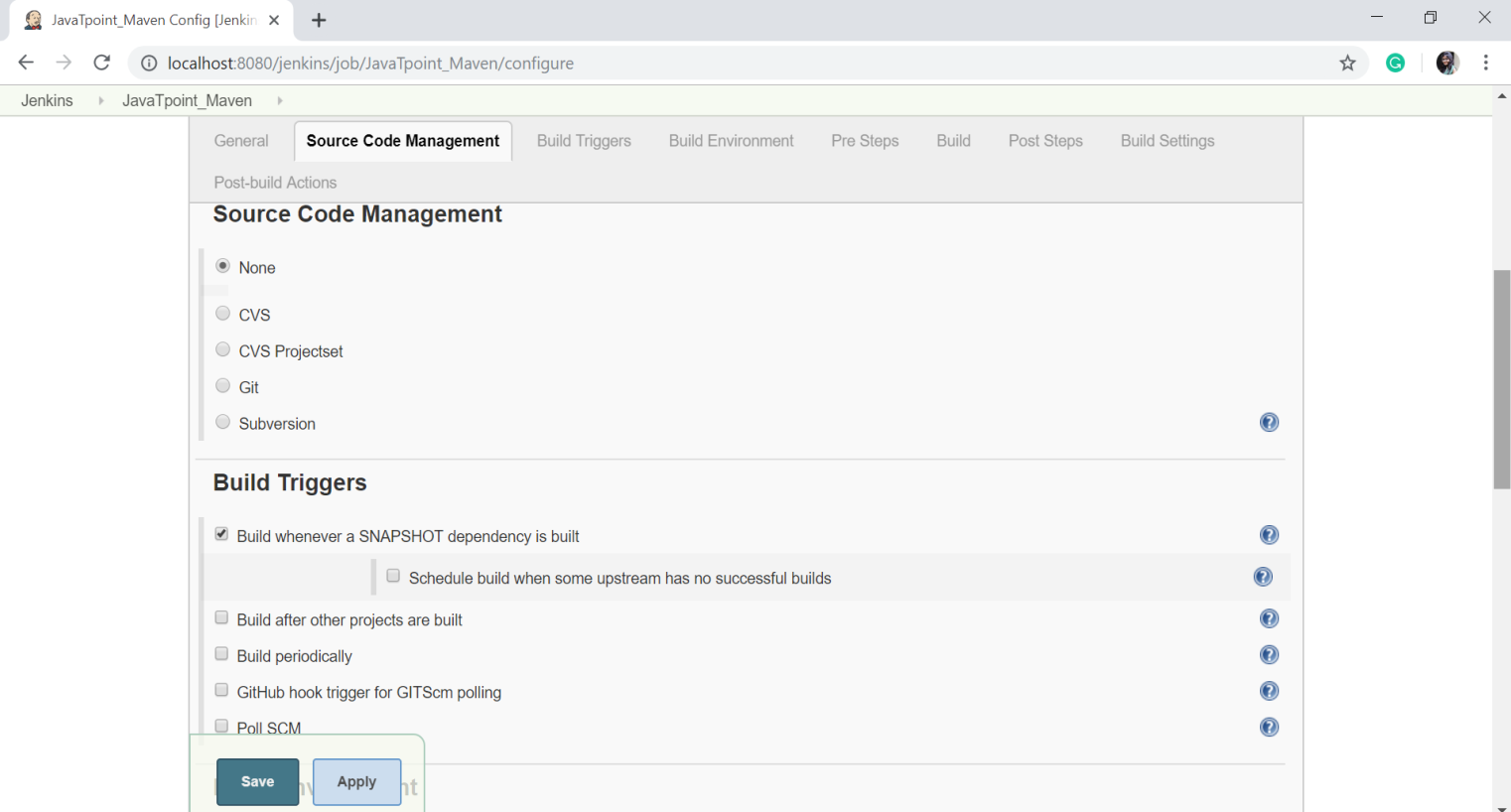
Enter the **Item Name** and select the **Maven Project**.



Click OK.

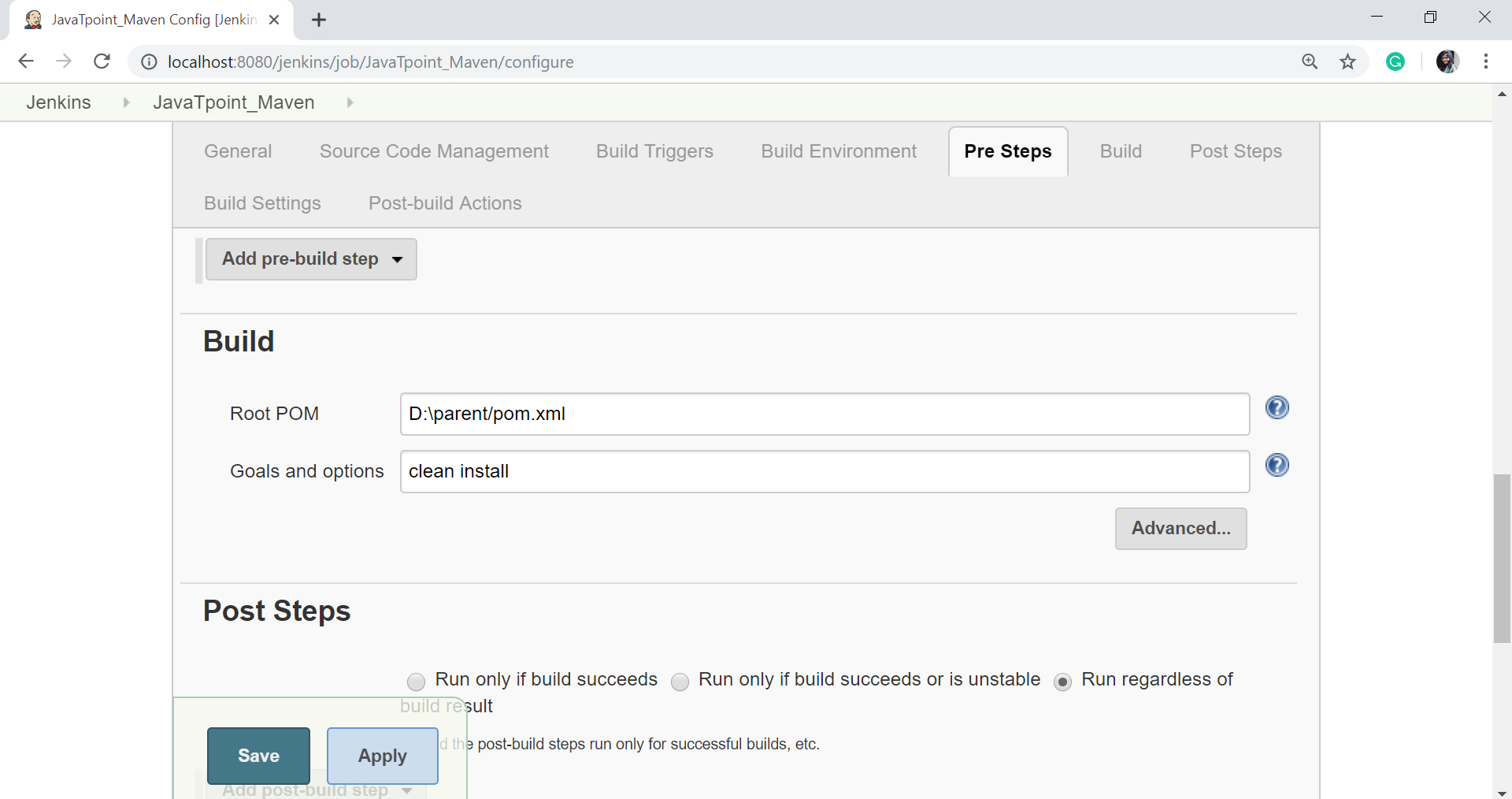


Now configure the job. Give the description and in the **Source Code Management** section, select the required option.



In the Build Triggers section, there are multiple options, select the required one.

Add the pom.xml file's path in the **Root POM** option.



Configure the other fields as per your requirement and then click on the **Save** button.

s

s