

Lab 3.3: Continuous Integration with Jenkins, Git, and Maven

This section will guide you to:

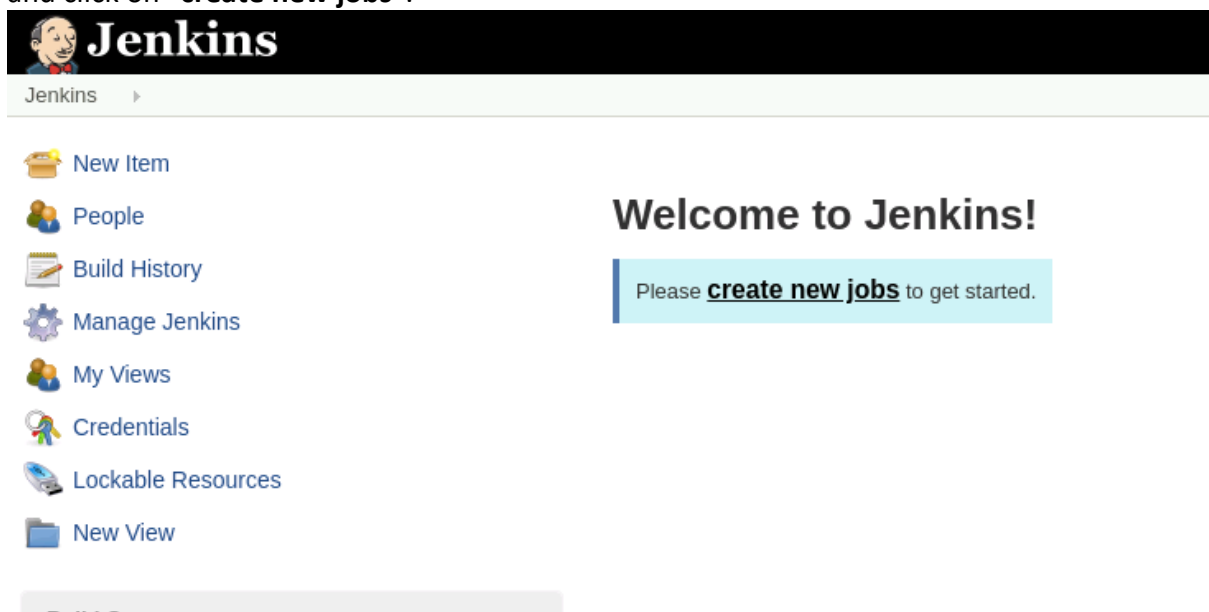
- Create Jenkins job for Maven project
- Poll Git for commits
- Build trigger using Push mechanism

This lab has 3 subsections:

- 3.3.1 Create first Jenkins job.
- 3.3.2 Install and configure Maven.
- 3.3.3 Configure Jenkins to work with Java, Git, and Maven.
- 3.3.4 Create a Jenkins job for your Maven project, and run the project.
- 3.3.5 Poll Git for commits and automatically trigger the build.

Step 3.3.1 : Create first Jenkins job

Login to Jenkins on <http://x.x.x.x:8080> using credentials: **admin/password**, and click on “**create new jobs**”.



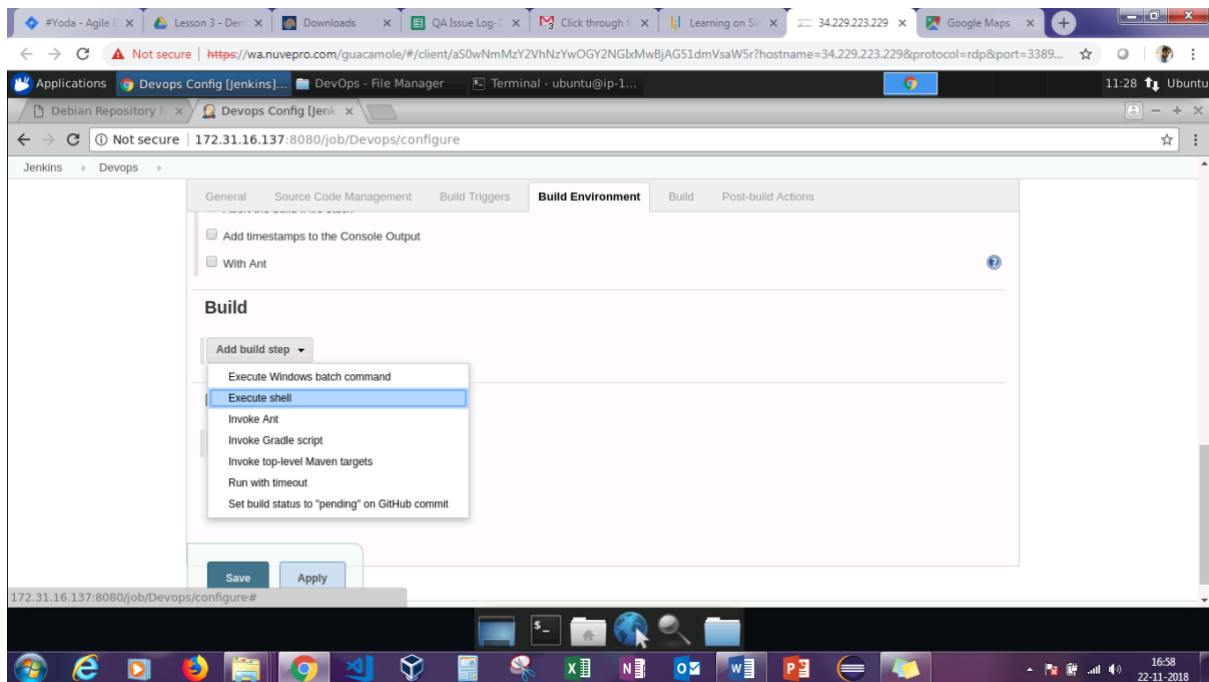
Provide a name without any space, and select **Freestyle project**.

Click OK.

Type in the description for the job.

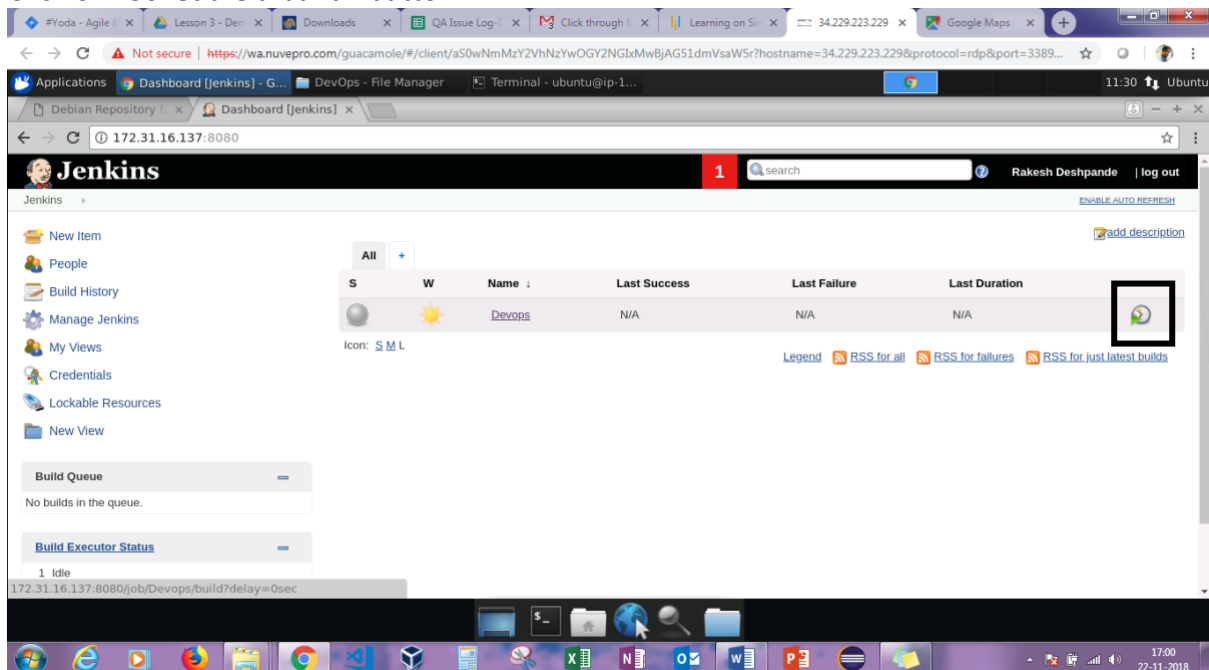
Under **Build** section, click on **Execute shell**, and type out the command:

echo "hello Jenkins"



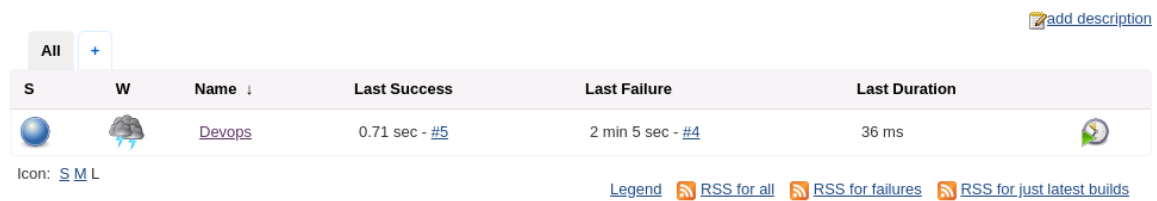
Click **Save**, and go back to the landing page to see the new job appear there.

Click on **"Schedule a build"** button.





Click on the job name to look at the Build history.

The blue ball indicates success. Click on the blue ball to look at the console output as shown in the screenshot.



The screenshot shows the Jenkins build history for a job named 'Devops'. At the top right is a link 'add description'. Below it is a table with columns: 'S' (Status), 'W' (Weather icon), 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The table contains one row for build #5, which is successful (blue ball icon). Below the table are links for 'Icon: S M L' and 'Legend', and three RSS feeds: 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'.

| S | W | Name | Last Success | Last Failure | Last Duration |
|---|---|------------------------|-------------------------------|----------------------------------|---------------|
|  |  | Devops | 0.71 sec - #5 | 2 min 5 sec - #4 | 36 ms |

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

You can see the echo command successfully run.

Under “**Last Success**” column, there’s the link to the build section.

“**W**” column indicates sunshine which means that from the last few commands everything has been successfully executed.

Step 3.3.2 : Install and configure Maven

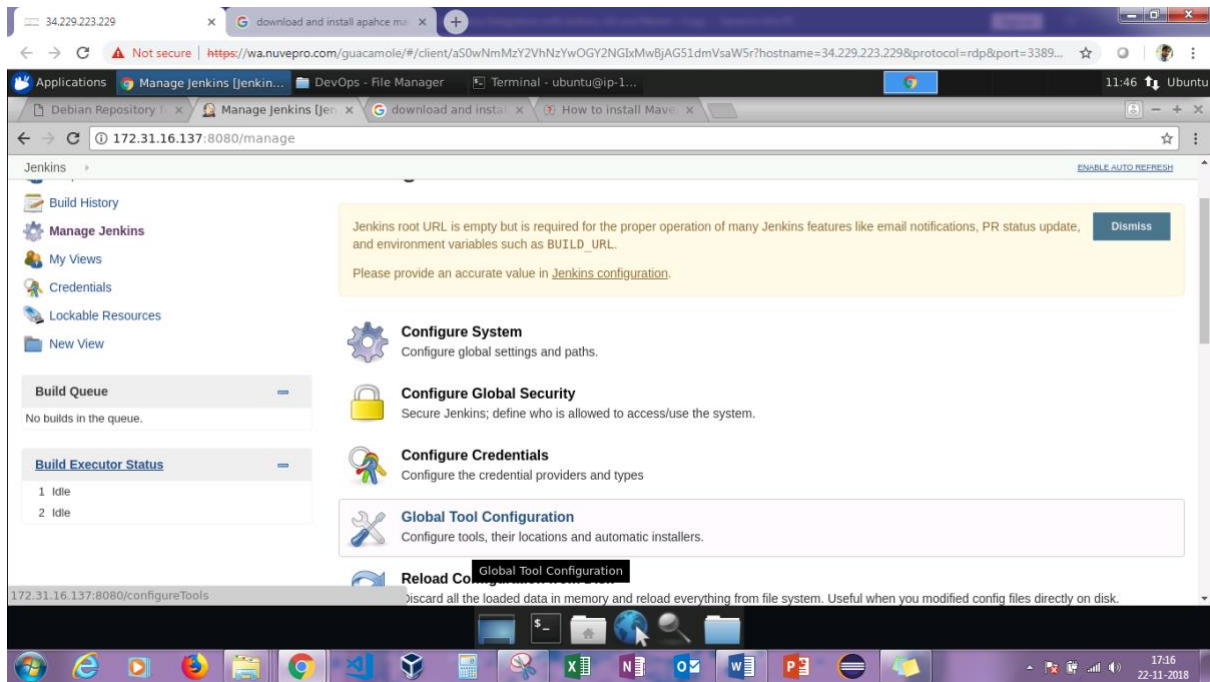
Run these commands terminal to install Maven:

```
sudo apt-get policy maven
sudo apt-get install maven
mvn --version
```

Step 3.3.3 : Configure Jenkins to work with Java, Git, and Maven

Go to Jenkins landing page → Manage Jenkins → Global Tool Configuration

This loads Global Jenkins configuration.



Now, JDK, Git, and Maven configurations need to be added.
Complete the JDK configuration. Click on **"Add JDK"** button. Specify the name as "localJDK".

If you already have JDK installed, uncheck the checkbox **"Install automatically"**. Under **"JAVA HOME"**, set the **"JAVA HOME"** path. You can find JAVA_HOME PATH by using the following command:

```
echo $JAVA_HOME
```

Add the Git settings. Specify the name as "localGit", and you can find path to Git executable using the following command:

```
which git
```

Add the Maven settings. Specify the name as "localMaven", and find MAVEN_HOME by using the following command:

```
mvn -v
```

```
ubuntu@ip-172-31-8-13:~$ mvn -v
Apache Maven 3.3.9
Maven home: /usr/share/maven
Java version: 1.8.0_181, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.4.0-1072-aws", arch: "amd64", family: "unix"
```

Step 3.3.4 : Create a Jenkins job for your Maven project, and run the project.

You need to first create a repository in GitHub. Click on import and type in the url shown in the screenshot below.

Then click on “Import repository” and copy the url. You will require it later.

Create a Jenkins job for your Maven project.

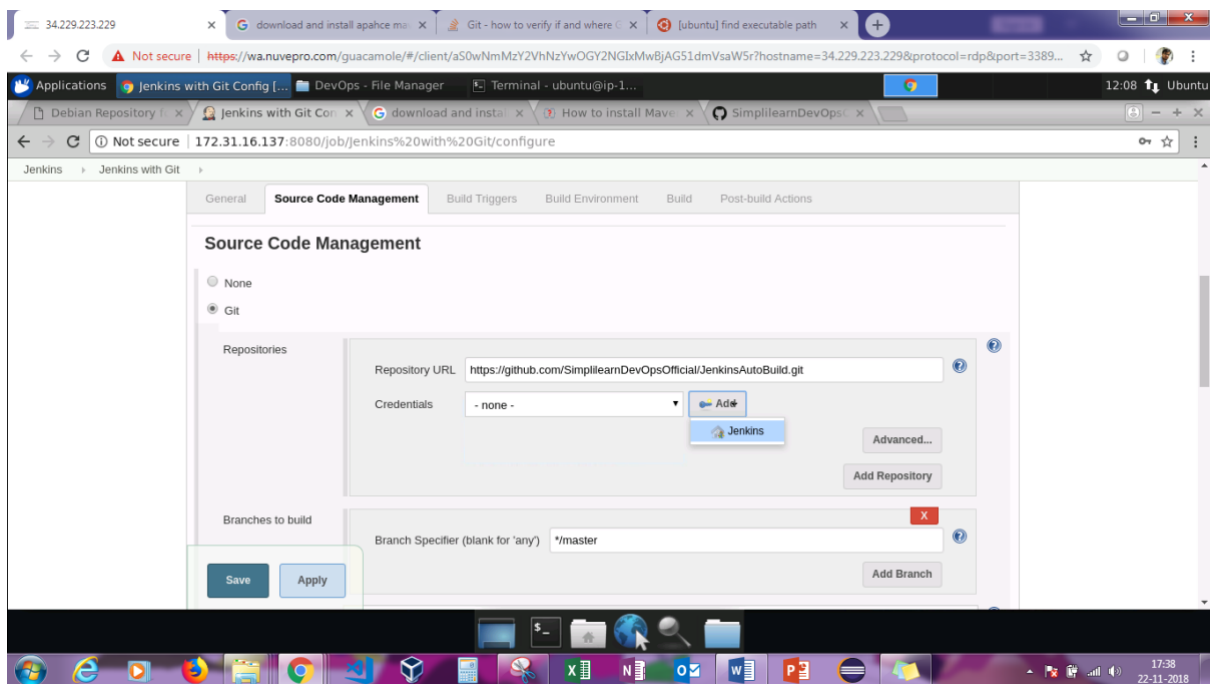
Go to Jenkins landing page, and click on “New Item”.

Set the name as “maven-project” of type “Freestyle project” and click “OK”.

Set the description as “first maven project”.

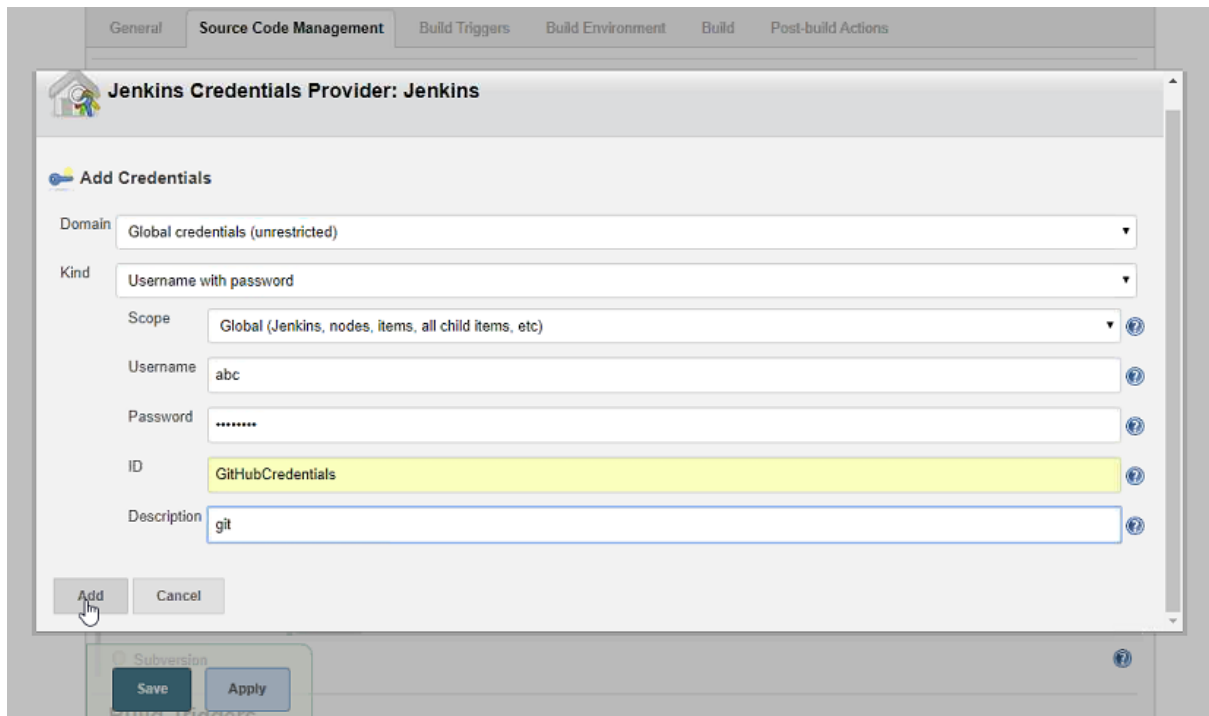
Under source code management, set the details.

Fill in your repository url which you can get by clicking on “**clone**” (done at the beginning of step 3.3.4).



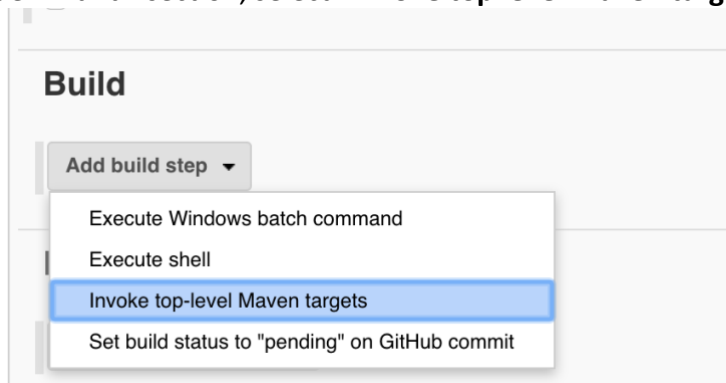
Under “**Credentials**”, click on Jenkins and then enter the **username** and **password** of the Git account. Then click on “**Add**”.

After adding it, select it from the dropdown box.

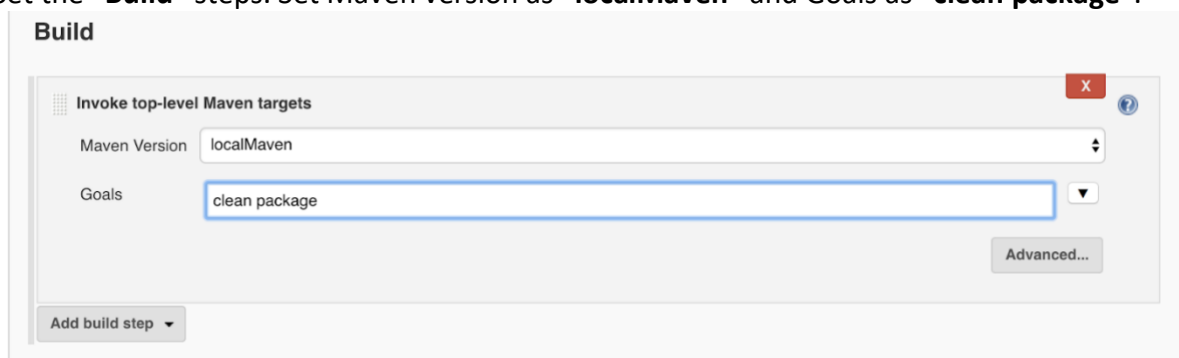


The image shows the 'Jenkins Credentials Provider: Jenkins' dialog box. It has tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'Add Credentials' section is active. It contains fields for 'Domain' (Global credentials (unrestricted)), 'Kind' (Username with password), 'Scope' (Global (Jenkins, nodes, items, all child items, etc)), 'Username' (abc), 'Password' (masked with dots), 'ID' (GitHubCredentials), and 'Description' (git). At the bottom, there are 'Add' and 'Cancel' buttons. A mouse cursor is pointing at the 'Add' button.

Under **“Build”** section, select **“Invoke top level Maven target”** in **“Add a Build step”**.



Set the **“Build”** steps. Set Maven version as **“localMaven”** and Goals as **“clean package”**.










The image shows the 'Build' section of the Jenkins configuration page. A build step titled 'Invoke top-level Maven targets' is added. It has a 'Maven Version' dropdown set to 'localMaven' and a 'Goals' text field containing 'clean package'. There is an 'Advanced...' button at the bottom right of the step configuration area. An 'Add build step' button is visible at the bottom left.

Click Save.

Run the project.

Go to the dashboard, select the job, and click on **“Build Now”**.

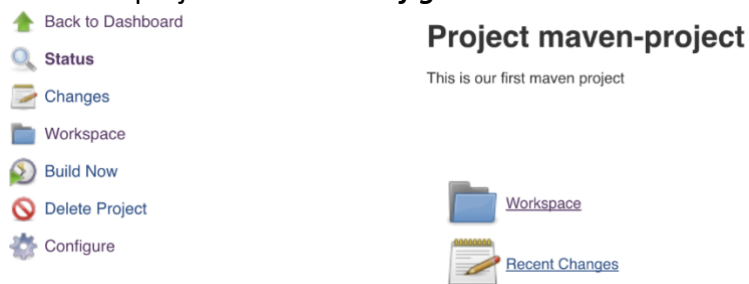
-  [Back to Dashboard](#)
-  [Status](#)
-  [Changes](#)
-  [Workspace](#)
-  [Build Now](#)
-  [Delete Project](#)
-  [Configure](#)

Wait till you see the blue ball as a success sign.
Click on the **"Build"** icon when it is complete, and observe the *console output*.

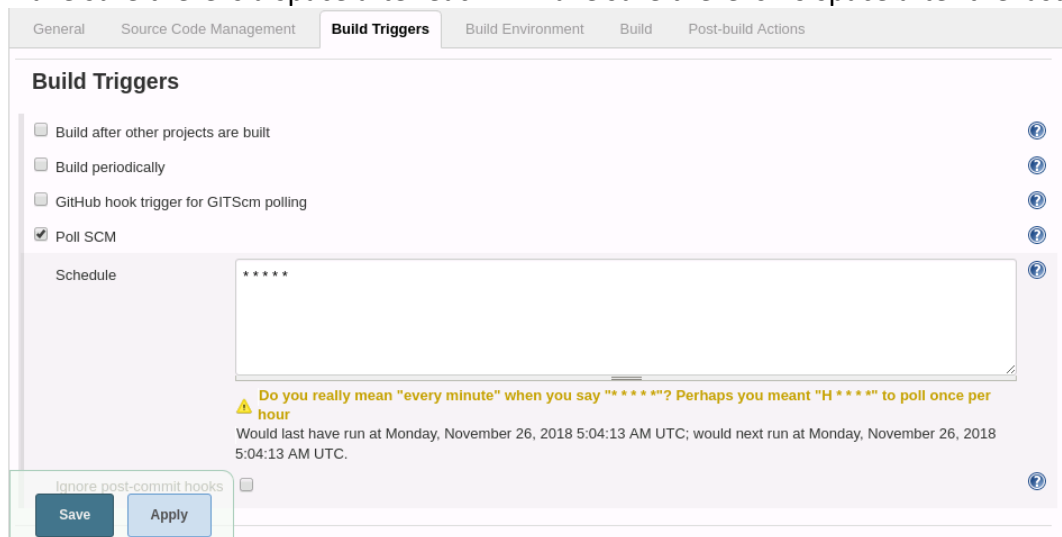
Step 3.3.5 : Poll Git for commits and automatically trigger the build

Poll Git for commits:

Go to the project and click *configure*.











Scroll to the **"Build Trigger"** section, and select Poll SCM. Set the schedule as **"* * * * *"**.
Note: Make sure there is a space after each *. Make sure there is no space after the last *.



Click Save.

You end up with an extra item in the side menu: **"Git Polling Log"**.

-  [Back to Dashboard](#)
-  [Status](#)
-  [Changes](#)
-  [Workspace](#)
-  [Build Now](#)
-  [Delete Project](#)
-  [Configure](#)
-  [Git Polling Log](#)

Click on ***Git Polling Log*** after a minute to see that Jenkins has already polled Git for more commits.

Automatically trigger a new “**Build**”.

On your system, clone your Maven project from GitHub using the “**clone**” command.

If it asks for a password, enter Git password.

```
$ git log  
$ vim README.md
```

Make a change, and save the file.

```
$ git add .  
$ git commit -m “another commit”  
$ git push <Repository_URL>
```

Now go back to Git to check if there is another commit. Go back to Jenkins (Git Polling Log), and check if the new build has been done.