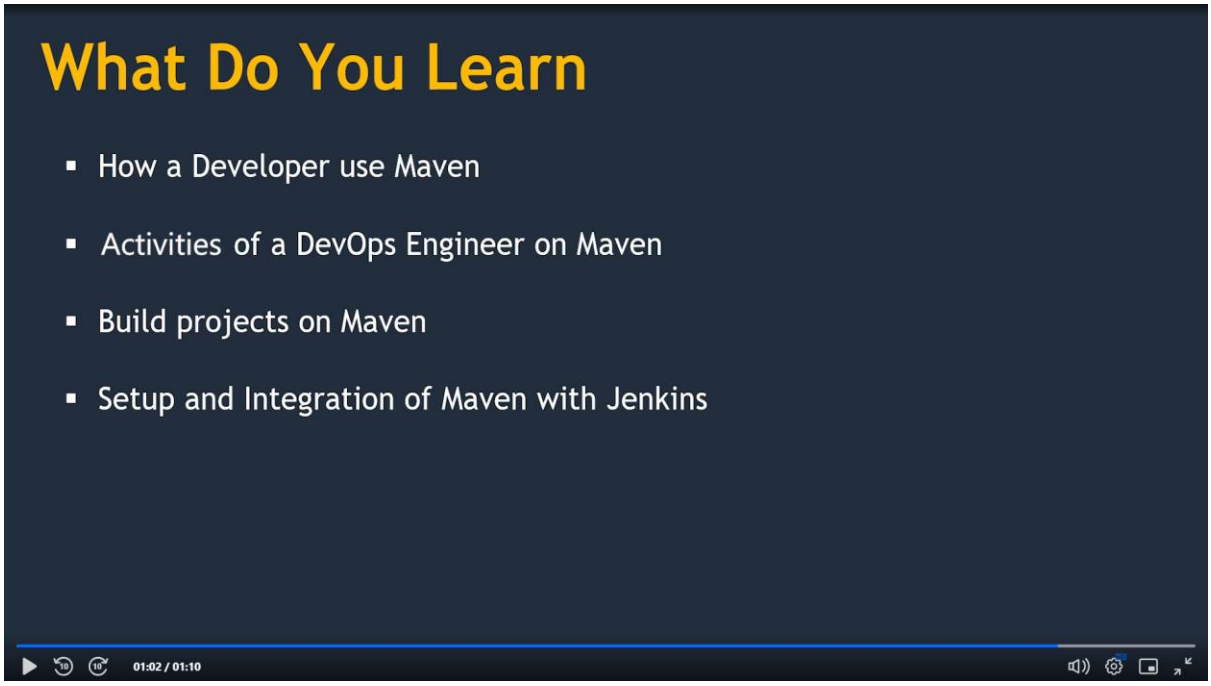
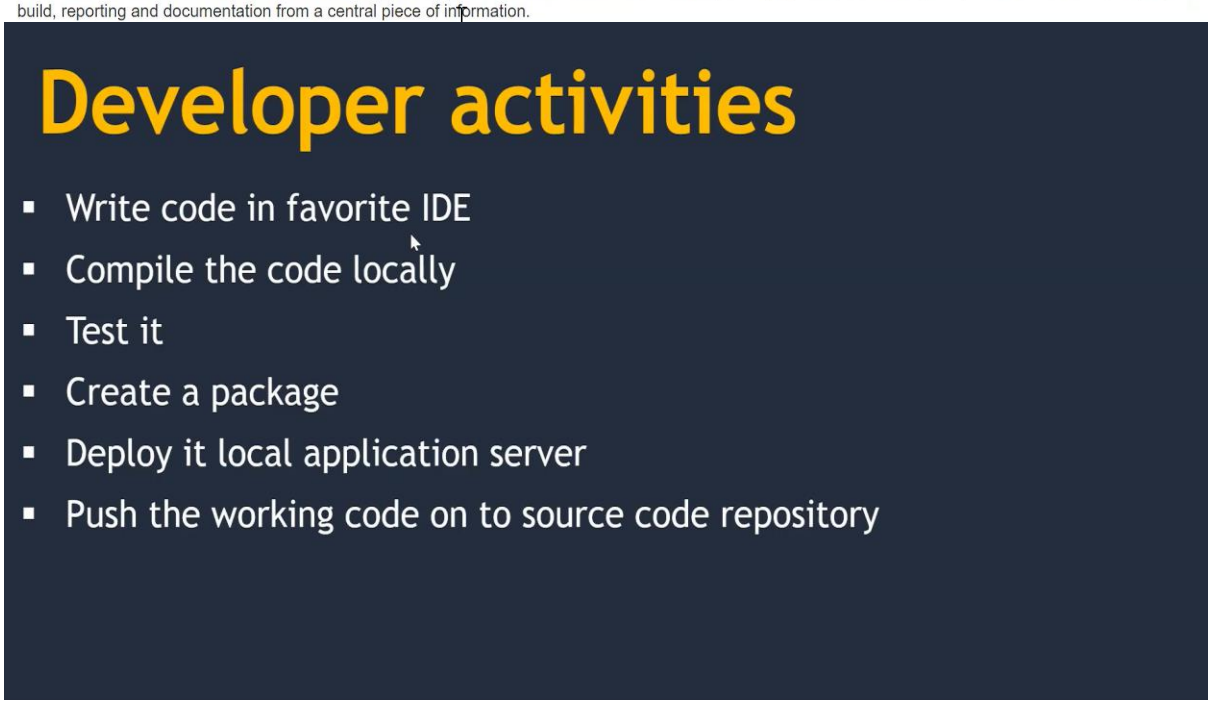
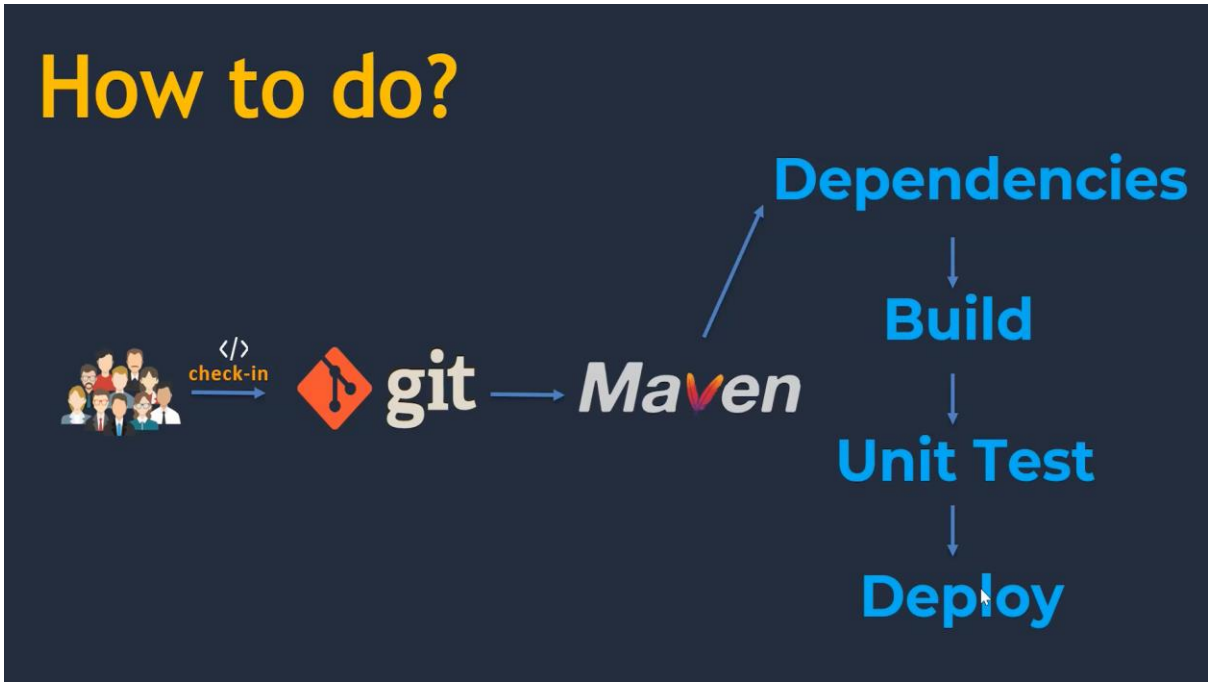


Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

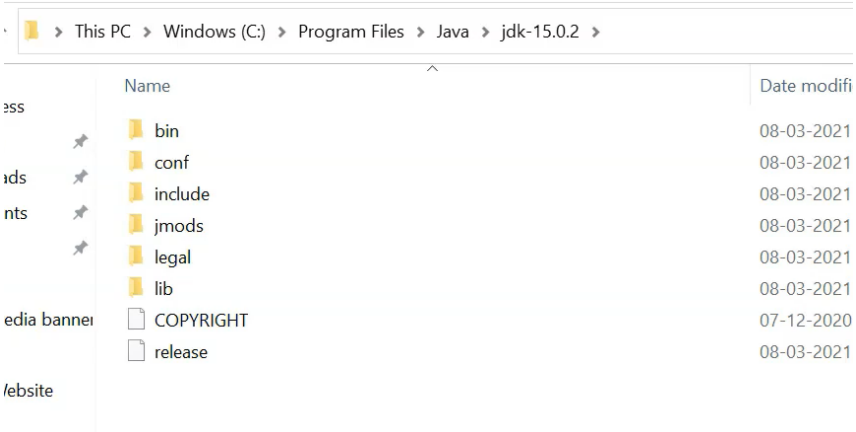


Maven downloads third party dependencies we just need to mention in maven file . Maven will build the application with all dependencies .

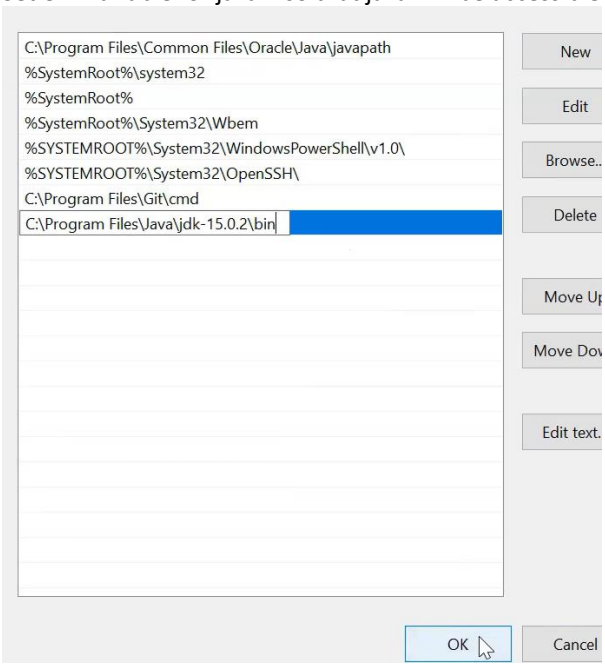


Maven is used to download dependency , build , test ,deploy .

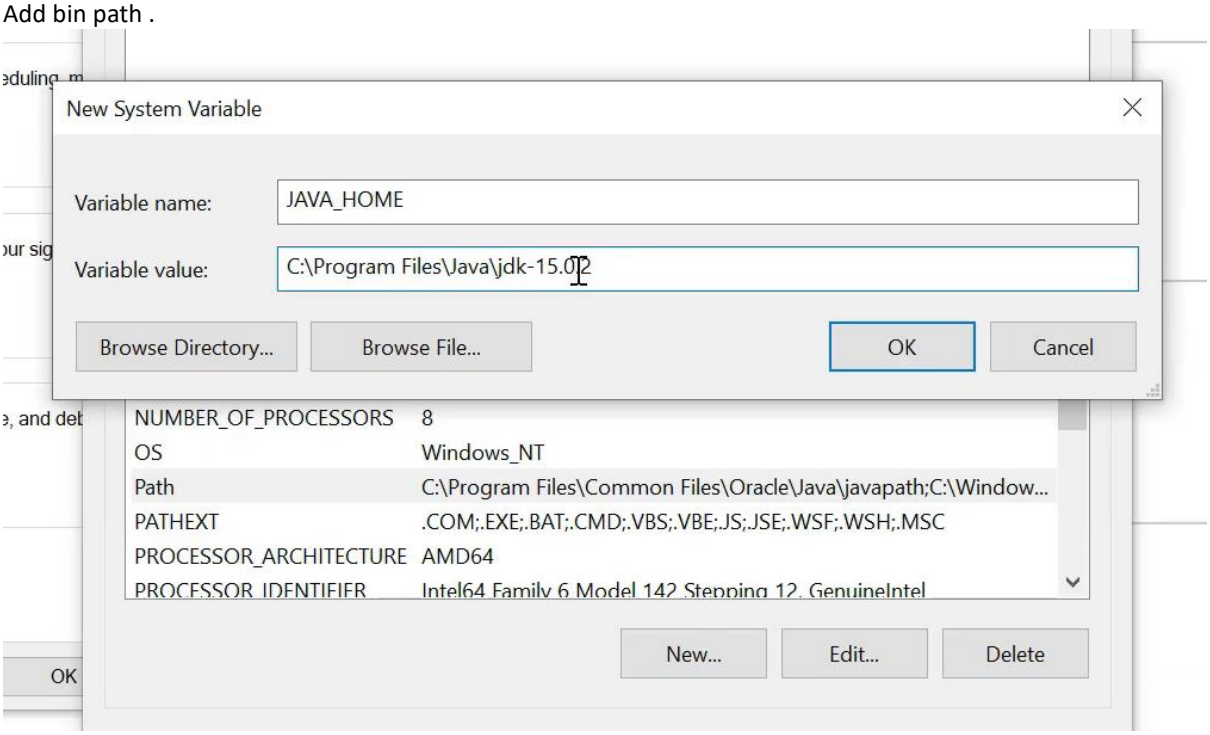
Java jdk install > oracle.com – jdk download .



Set env variable for java – so that java will be accessible from over all system .



Add bin path .

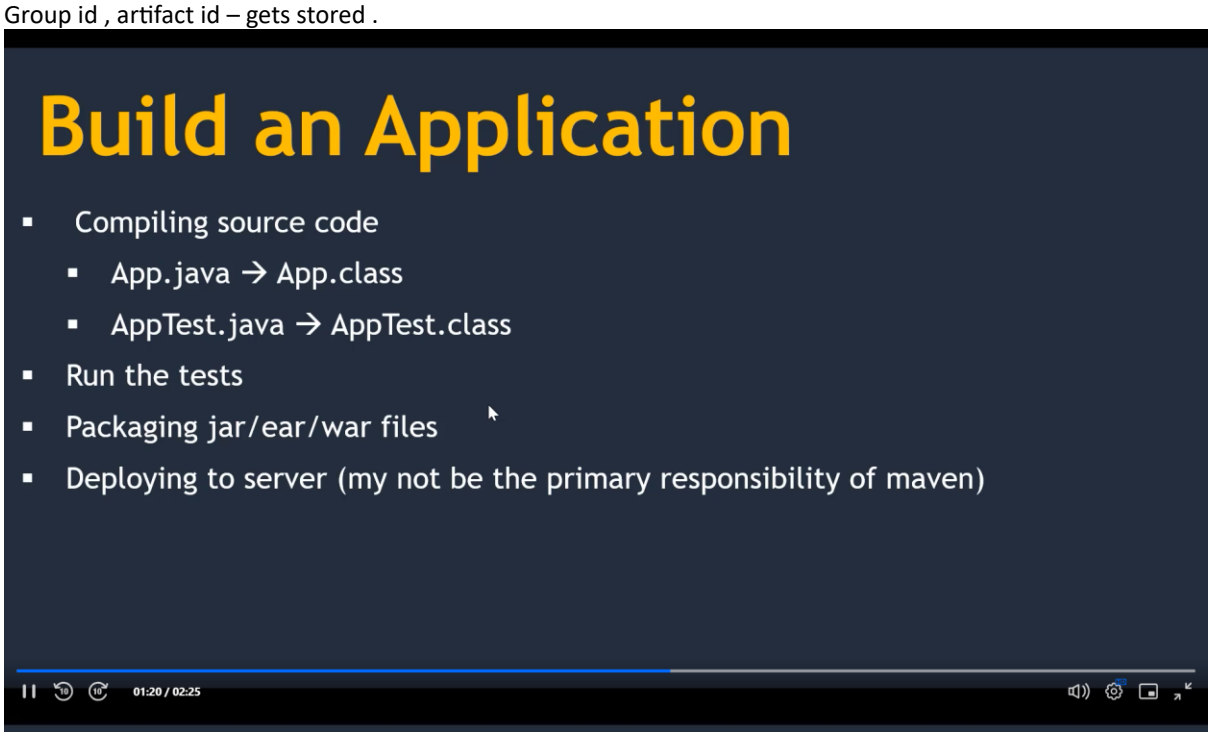


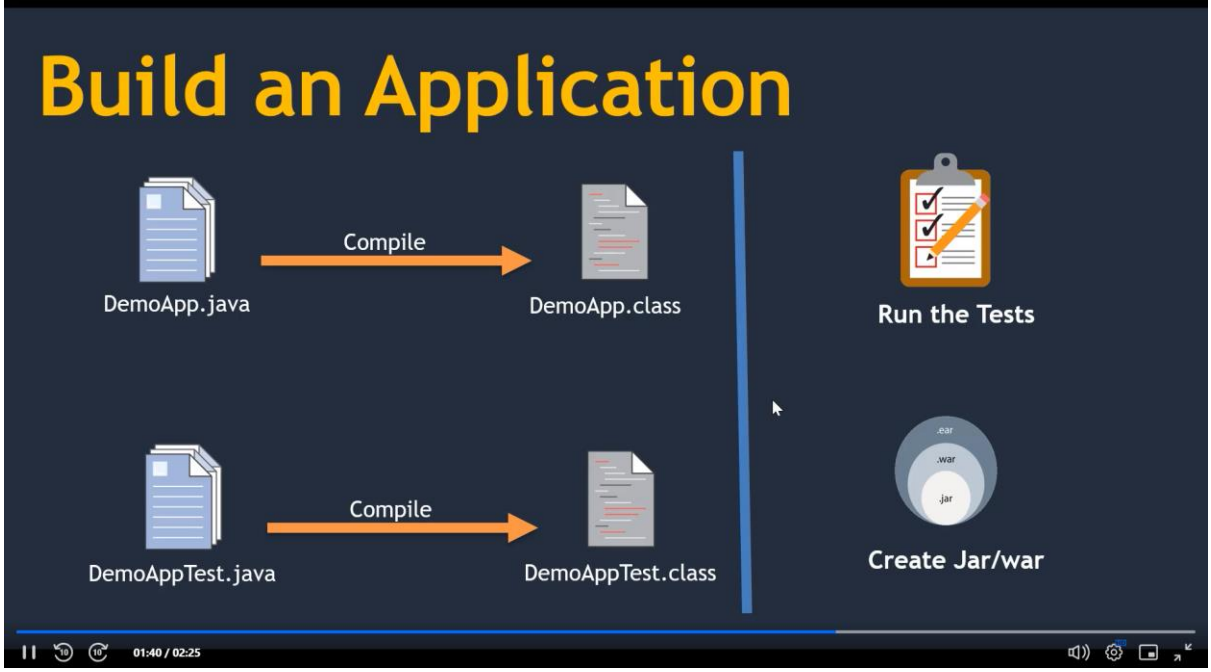
Add system variable .
Java –version : to check java version .
Javac – compilation of java program .

Maven coordinates :
/src/main/java – actual production code .
/src/test/java – test cases .

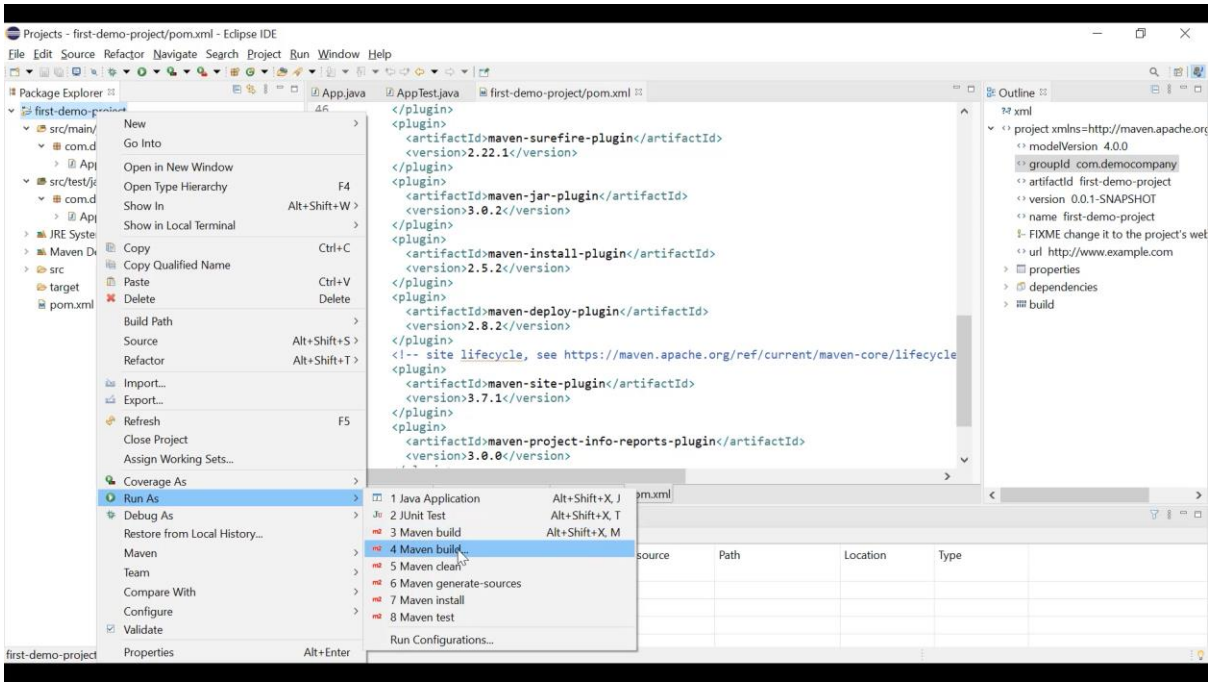
Pom.xml – dependency for our project .

Group id , artifact id – gets stored .





To build application .



We can do maven build .

To build maven project we need goals .

There are three built-in build lifecycles: default, clean and site. The `default` lifecycle handles your project deployment, the `clean` lifecycle handles project cleaning, while the `site` lifecycle handles the creation of your project's site documentation.

A Build Lifecycle is Made Up of Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

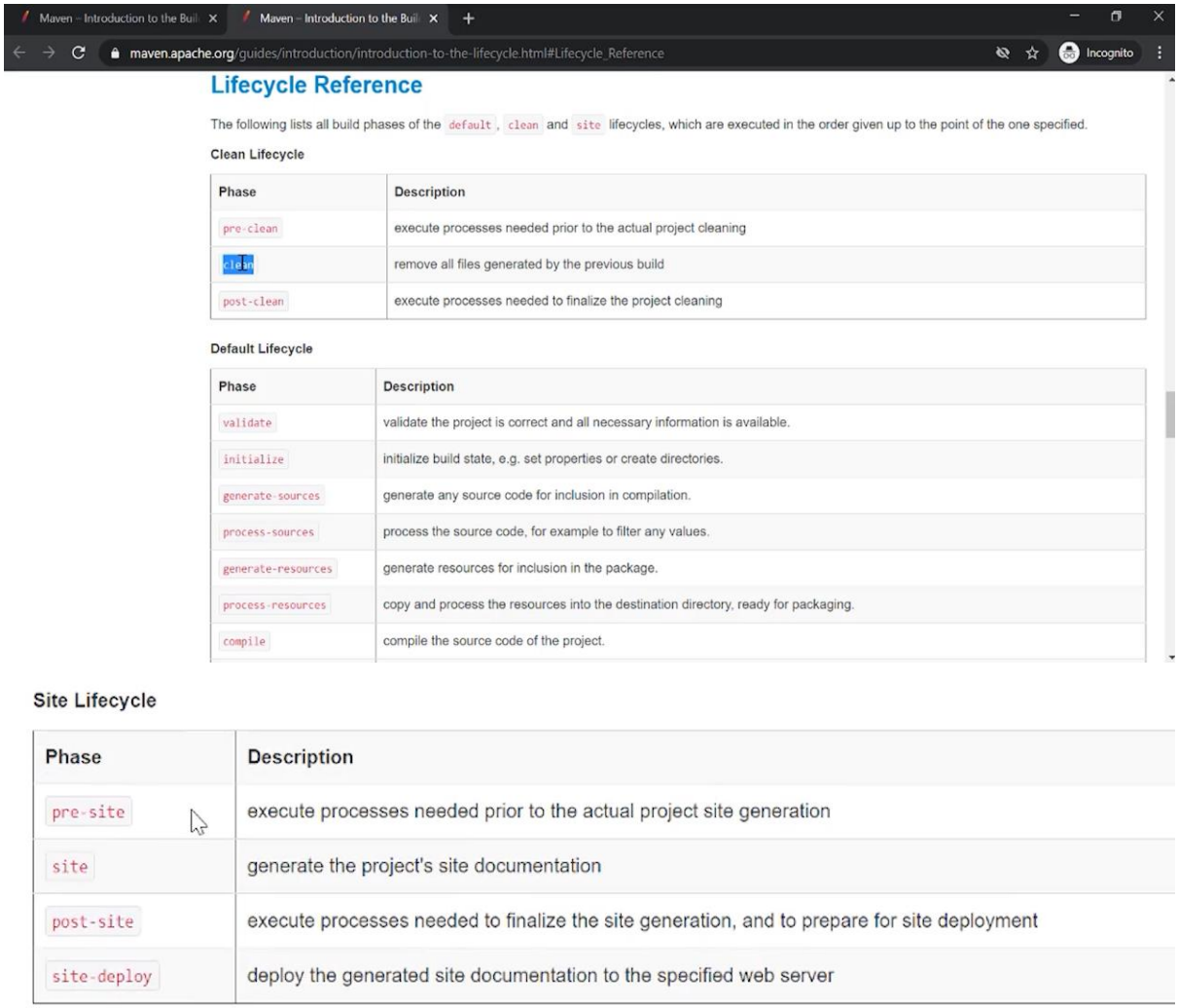
For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

- `validate` - validate the project is correct and all necessary information is available
- `compile` - compile the source code of the project
- `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- `package` - take the compiled code and package it in its distributable format, such as a JAR.
- `verify` - run any checks on results of integration tests to ensure quality criteria are met
- `install` - install the package into the local repository, for use as a dependency in other projects locally
- `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

Maven Goals

clean	-	remove all files generated by the previous build
validate	-	validate the project is correct and all necessary information is available.
compile	-	compile the source code of the project.
test-compile	-	compile the test source code into the test destination directory
test	-	run tests using a suitable unit testing framework.
package	-	take the compiled code and package it in its distributable format, such as a JAR.
verify	-	run any checks to verify the package is valid and meets quality criteria.
install	-	install the package into the local repository, for use as a dependency in other projects locally.
deploy	-	copies the final package to the remote repository for sharing with other projects

Maven goals are executed in sequential manner . If we run any goal then previous goal will automatically be run .



Lifecycle Reference

The following lists all build phases of the `default` , `clean` and `site` lifecycles, which are executed in the order given up to the point of the one specified.

Clean Lifecycle

Phase	Description
`pre-clean`	execute processes needed prior to the actual project cleaning
`clean`	remove all files generated by the previous build
`post-clean`	execute processes needed to finalize the project cleaning

Default Lifecycle

Phase	Description
`validate`	validate the project is correct and all necessary information is available.
`initialize`	initialize build state, e.g. set properties or create directories.
`generate-sources`	generate any source code for inclusion in compilation.
`process-sources`	process the source code, for example to filter any values.
`generate-resources`	generate resources for inclusion in the package.
`process-resources`	copy and process the resources into the destination directory, ready for packaging.
`compile`	compile the source code of the project.

Site Lifecycle

Phase	Description
`pre-site`	execute processes needed prior to the actual project site generation
`site`	generate the project's site documentation
`post-site`	execute processes needed to finalize the site generation, and to prepare for site deployment
`site-deploy`	deploy the generated site documentation to the specified web server

If we want to create documentation of our project then we can use site lifecycle .
We can compile our source code , once compilation done our target directory will contain all class files .

If we run goal test-compile – it will validate , compile , and test , Whenever we run package we should clean as default goal if not done so it will use olders files for package .

Once package goal is successful – it will package the jar file . Package will get stored in target . artifact id is used while we package our jar file .

What is a POM?

A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is `target` ; the source directory, which is `src/main/java` ; the test source directory, which is `src/test/java` ; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.

Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

```
<groupId>com.democompany</groupId>
<artifactId>first-demo-project</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

If we don’t specify any package parameter then it will treat it as jar .

```
</dependencies>

<build>
  <pluginManagement><!-- lock down plugins versions to avoi
    <plugins>
      <!-- clean lifecycle, see https://maven.apache.org/re
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
      <!-- default lifecycle, jar packaging: see https://ma
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
```

```
<name>first-demo-project</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</properties>
```

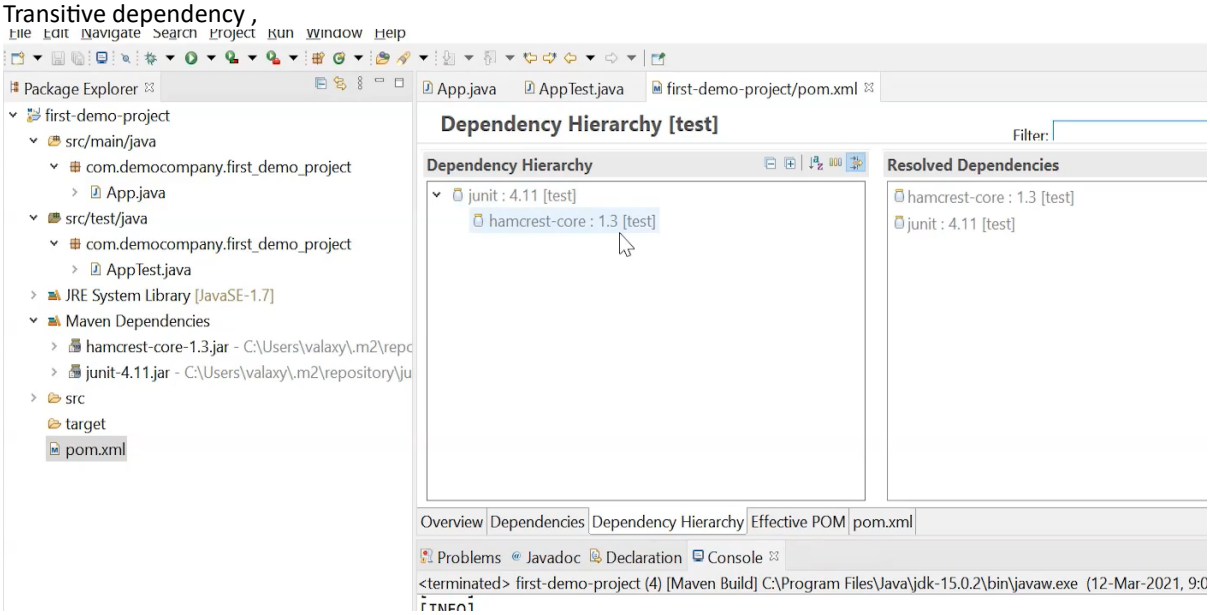
In above we have given name of project , compiler to be used in project .

We need to specify dependency . What all package required to build this application .
It will download junit package as dependency using maven .
We can also add third party dependencies that can be added .

```
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <version>2.4.3</version>
    <scope>test</scope>
</dependency>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

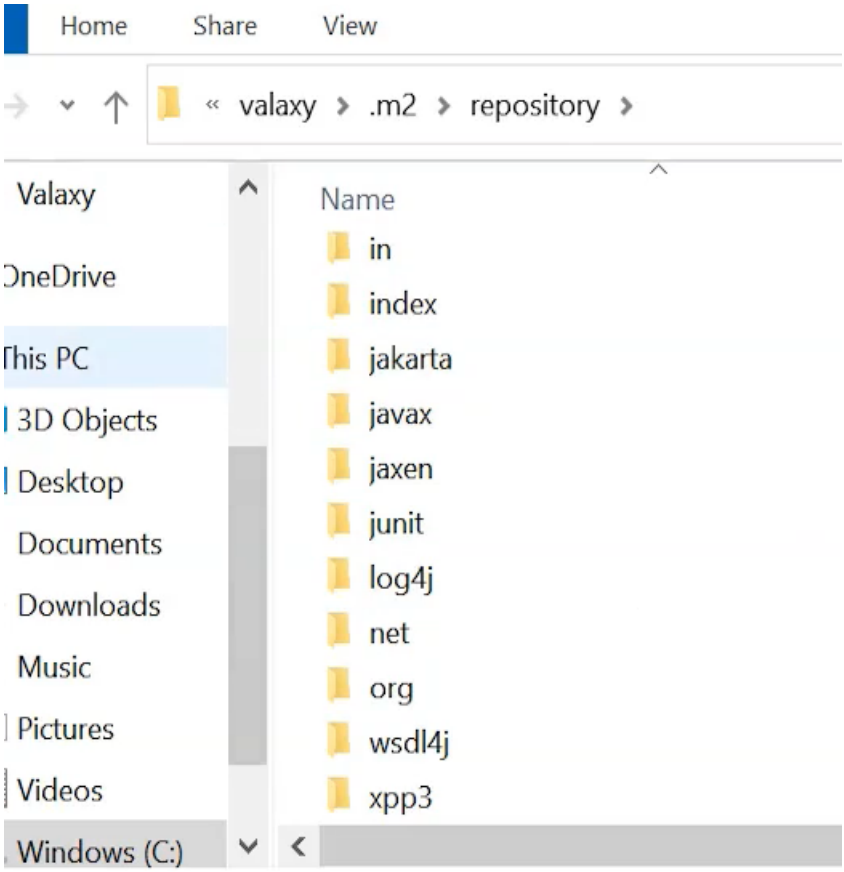
Transitive dependency ,



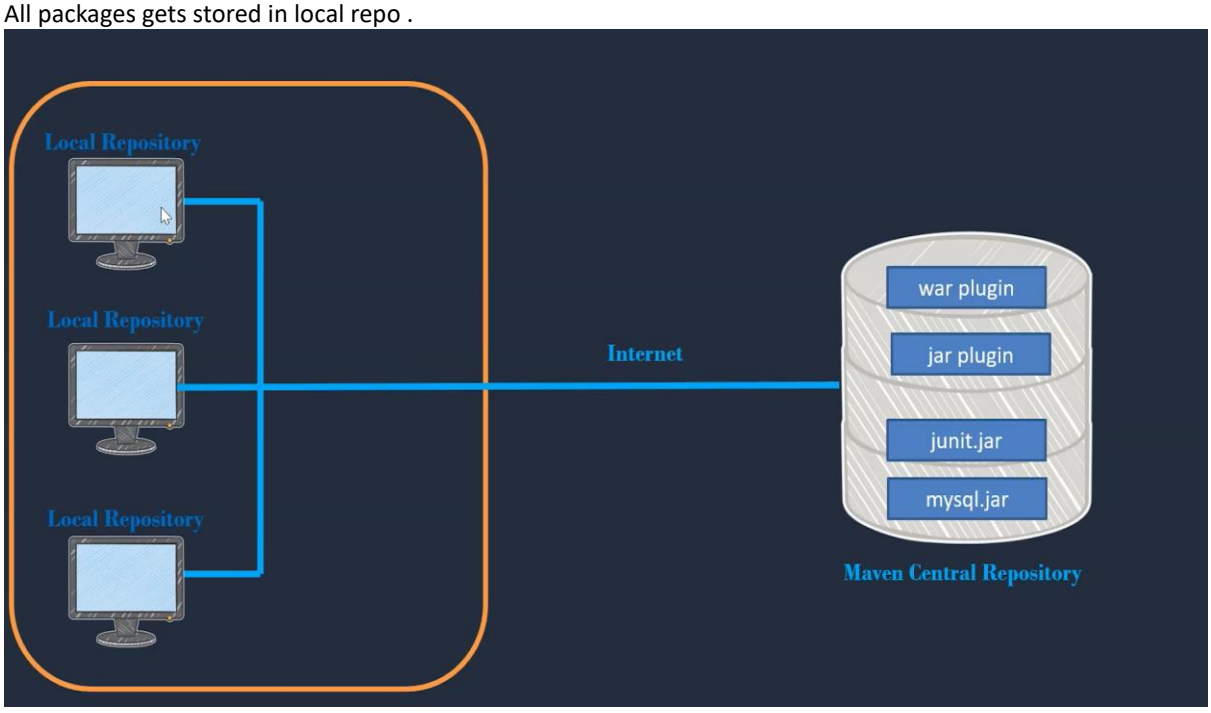
Some dependency requires other dependency this is known as transitive dependency .

Local repo and remote repo in maven :

There is local repo in our system . Whenevr we run maven project local repo gets created with .m2 .

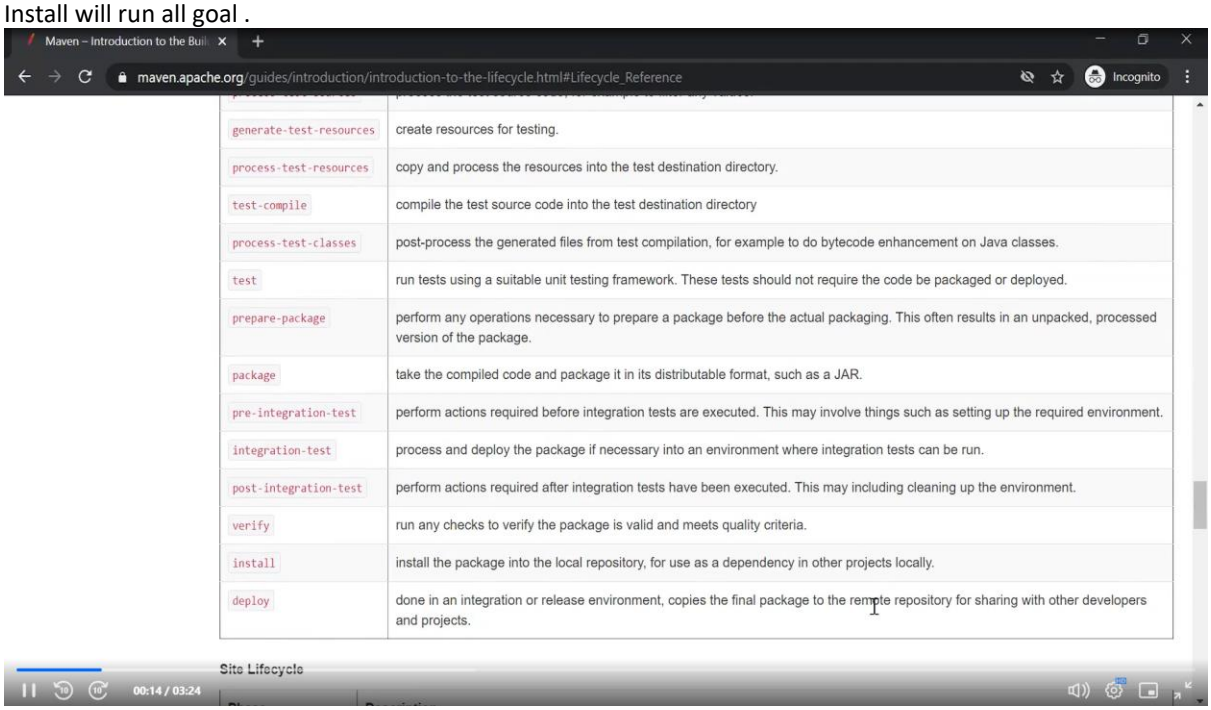


All packages gets stored in local repo .



When we use clean goal – target repo will not be visible .

Install will run all goal .



Deploy goal – copy
Src and pom.xml are sufficient to build the code .

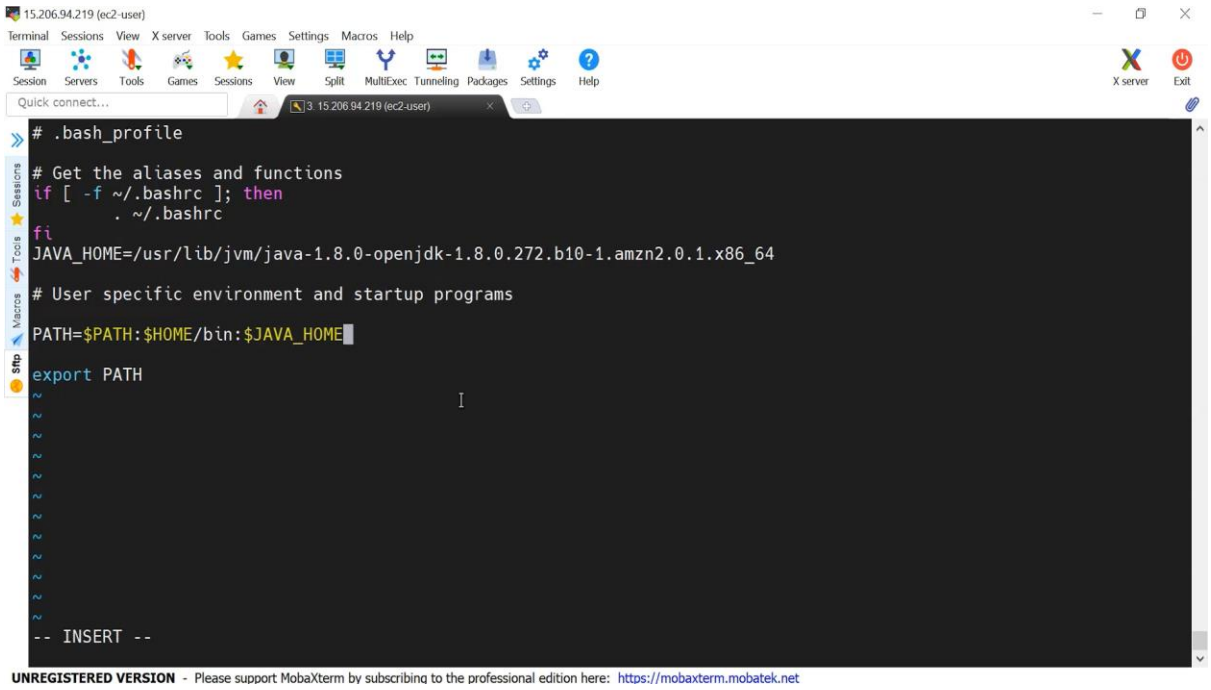
Setting up maven server :
Install jdk in linux server .
Maven.appache.org – we will get url for maven tar package to download in server .
Yum install java-1.8*
Java -version

Set java home path :
Check home path for java : whereis java .

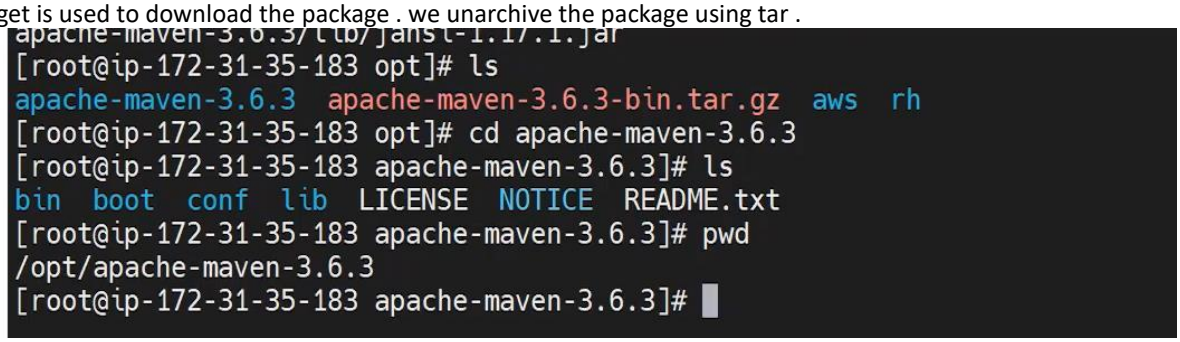
```
[root@ip-172-31-35-183 ~]# whereis java
java: /usr/bin/java /usr/lib/java /etc/java /usr/share/java /usr/share/man/man1/java.1.gz
[root@ip-172-31-35-183 ~]# find /usr/lib/jvm/java-1.8* | head -n 3
/usr/lib/jvm/java-1.8.0
/usr/lib/jvm/java-1.8.0-openjdk
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64
[root@ip-172-31-35-183 ~]#
```

Set java home path in bashrc profile .

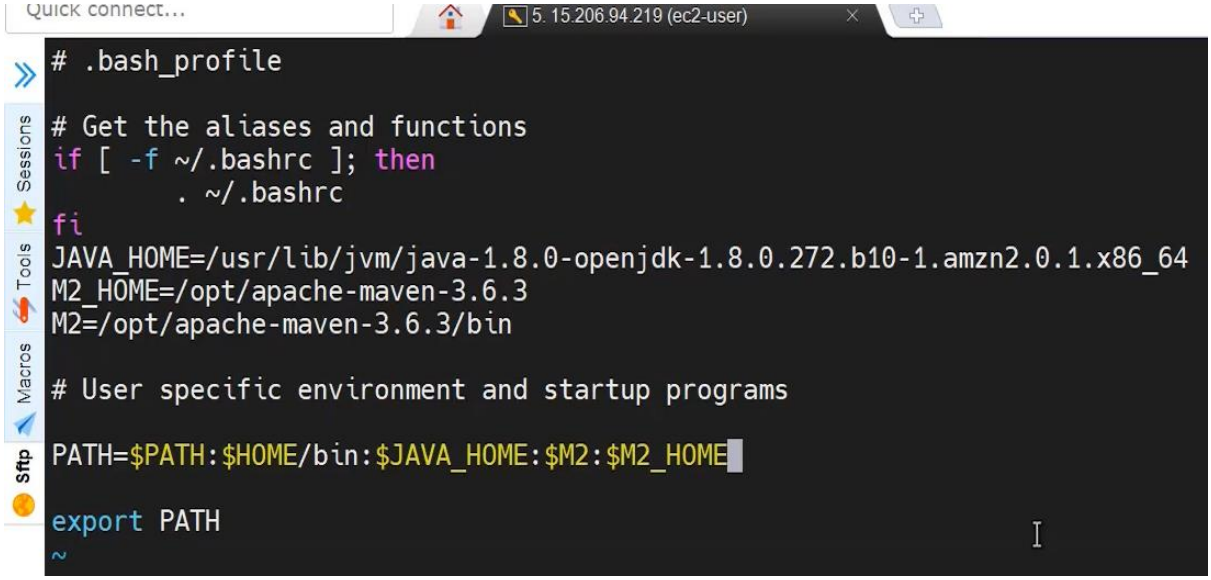
```
[root@ip-172-31-35-183 ~]# vi ~/.bash_profile
[root@ip-172-31-35-183 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```



Wget is used to download the package . we unarchive the package using tar .

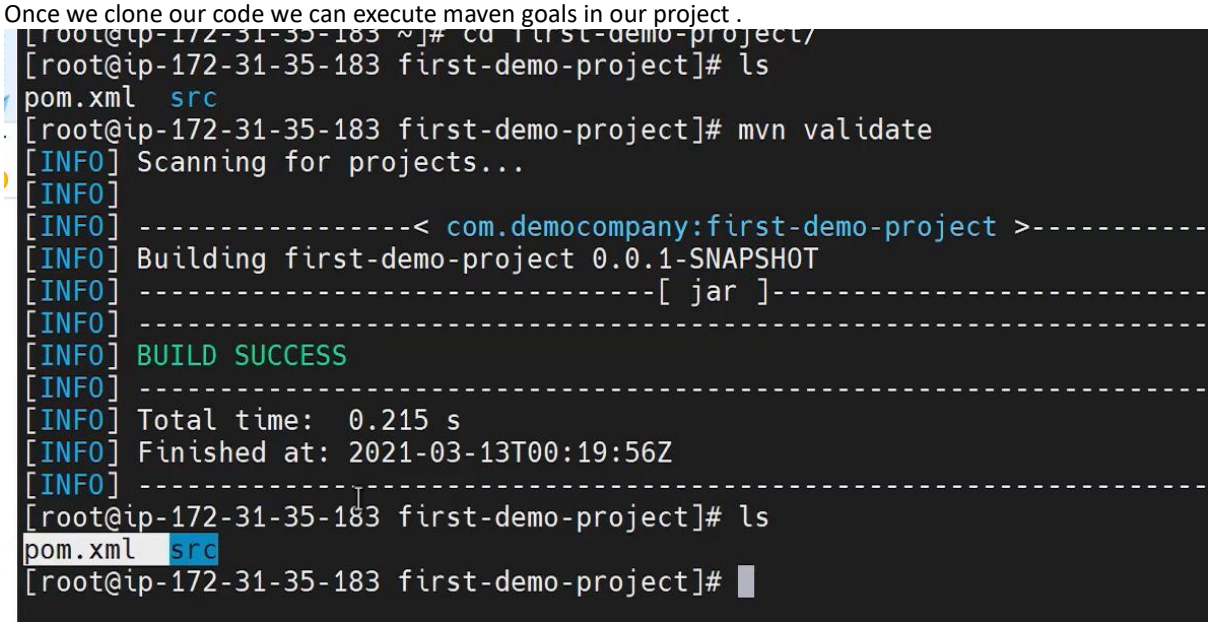


Now we have to set maven home path also .

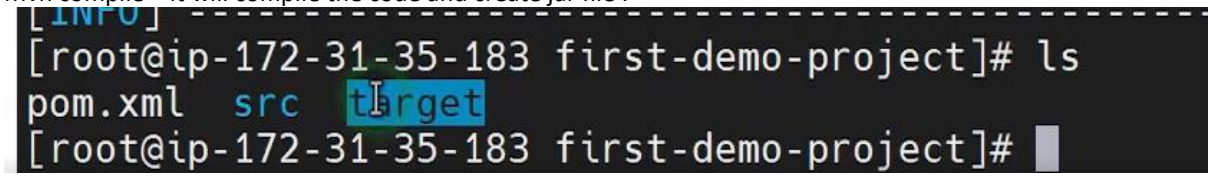


Edit in bash_profile .

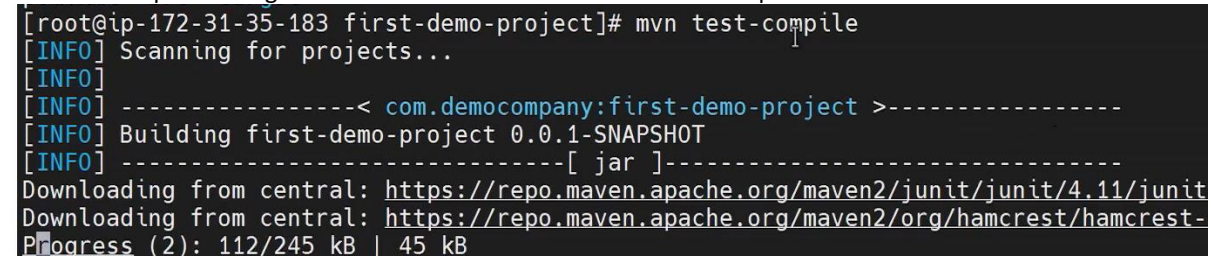
Mvn –version : to check maven version .
Install git in linux server : yum install git .
Once we clone our code we can execute maven goals in our project .



Mvn compile – it will compile the code and create jar file .



Target gets created when we compile the code .
Mvn test-compile – along with source code test cases will also be compiled .



```
[root@ip-172-31-35-183 target]# tree test-classes
test-classes
└── com
    └── democompany
        └── first_demo_project
            └── AppTest.class

3 directories, 1 file
```

To view directory structure : tree

mvn test : to test source code .
mvn package : to package our artifact .
mvn install install package in local repo .
lets find where our local repo for maven is present .

```
[root@ip-172-31-35-183 first-demo-project]# find / -name .m2
/root/.m2
[root@ip-172-31-35-183 first-demo-project]#
```

The command "find / -name .m2" searches for a directory named ".m2" starting from the root directory ("/") in a Linux or Unix-like operating system. This command is commonly used to locate the ".m2" directory, which is a default directory used by Apache Maven for storing local repository data.

mvn install – install packages in local repo in com directory .

```
pom.xml  src  target
[root@ip-172-31-35-183 first-demo-project]# mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------< com.democompany:first-demo-project >----------
[INFO] Building first-demo-project 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------
Downloading from central: https://repo.maven.apache.org/maven2/org/apach
```

```
first-demo-project  second-demo-project
[root@ip-172-31-35-183 ~]# cd second-demo-project/
[root@ip-172-31-35-183 second-demo-project]# ls
pom.xml  src
[root@ip-172-31-35-183 second-demo-project]# tree .
.
├── pom.xml
└── src
    └── main
        └── webapp
            ├── index.jsp
            └── WEB-INF
                └── web.xml

4 directories, 3 files
[root@ip-172-31-35-183 second-demo-project]#
```

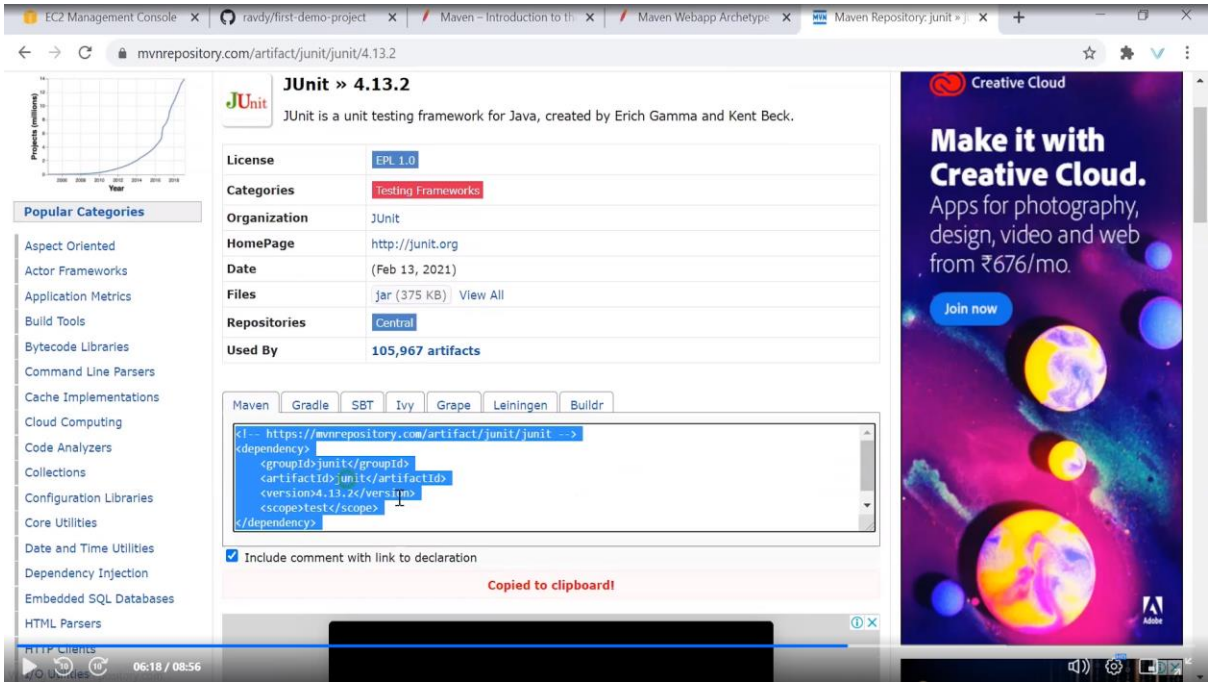
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

Pom.xml version that is 4.0.0 ,

```
<groupId>com.democompaly</groupId>
<artifactId>second-demo-project</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>

<name>second-demo-project Maven Webapp</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>
```

Name and url of project .

Build plugin – which goal we will execute all plugins are mentioned .
Junit dependency is necessary only if we have test cases in our code .
If test cases are not mentioned then we can remove dependency .



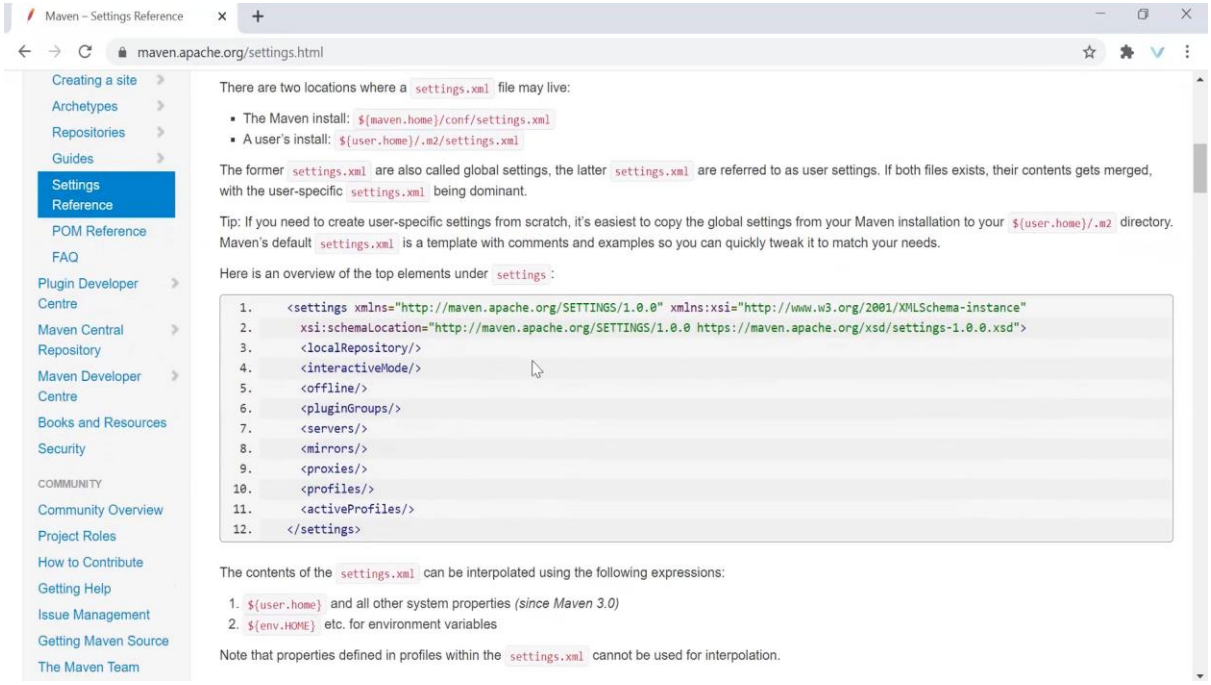
Junit dependency .

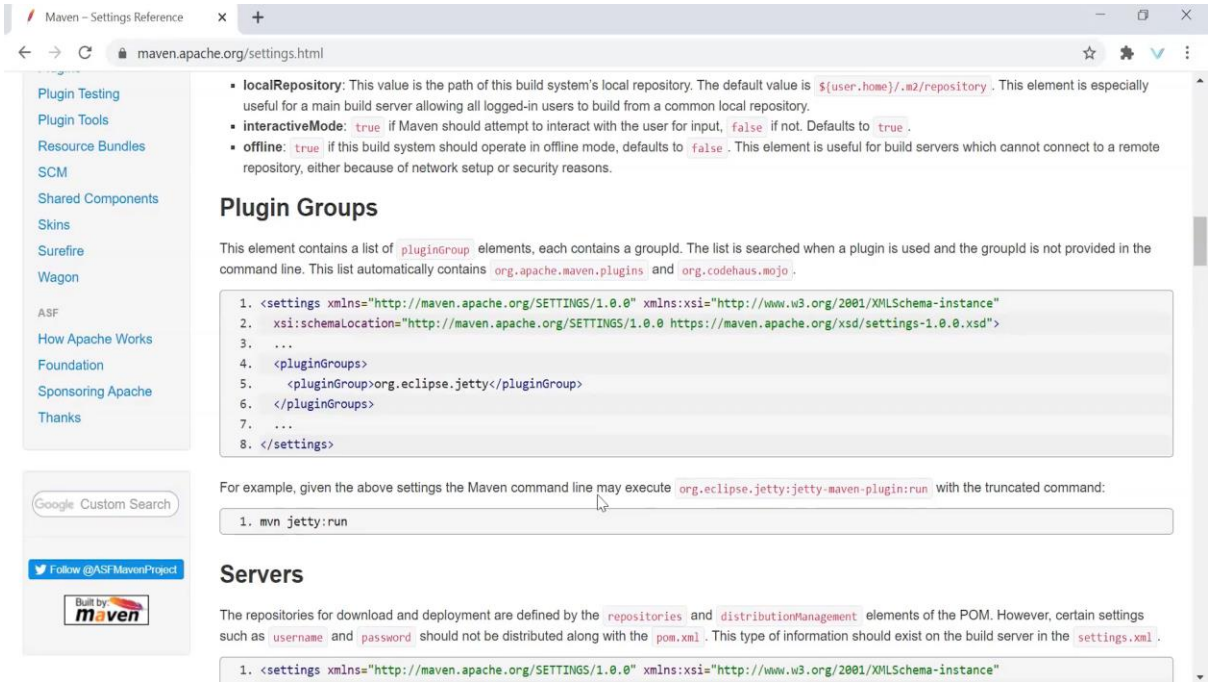
Setting.xml in maven :
Pom.xml is associated to specific project . setting.xml is global configuration .

The `settings` element in the `settings.xml` file contains elements used to define values which configure Maven execution in various ways, like the `pom.xml`, but should not be bundled to any specific project, or distributed to an audience. These include values such as the local repository location, alternate remote repository servers, and authentication information.

There are two locations where a `settings.xml` file may live:

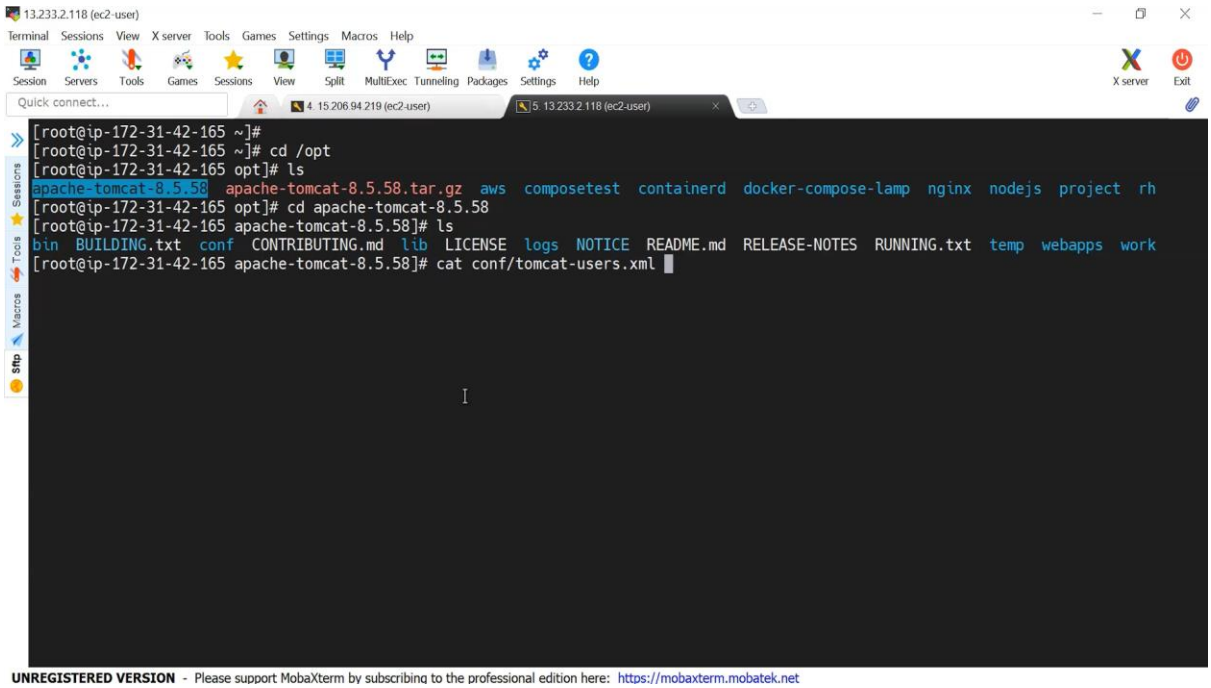
- The Maven install: `${maven.home}/conf/settings.xml`
- A user's install: `${user.home}/.m2/settings.xml`





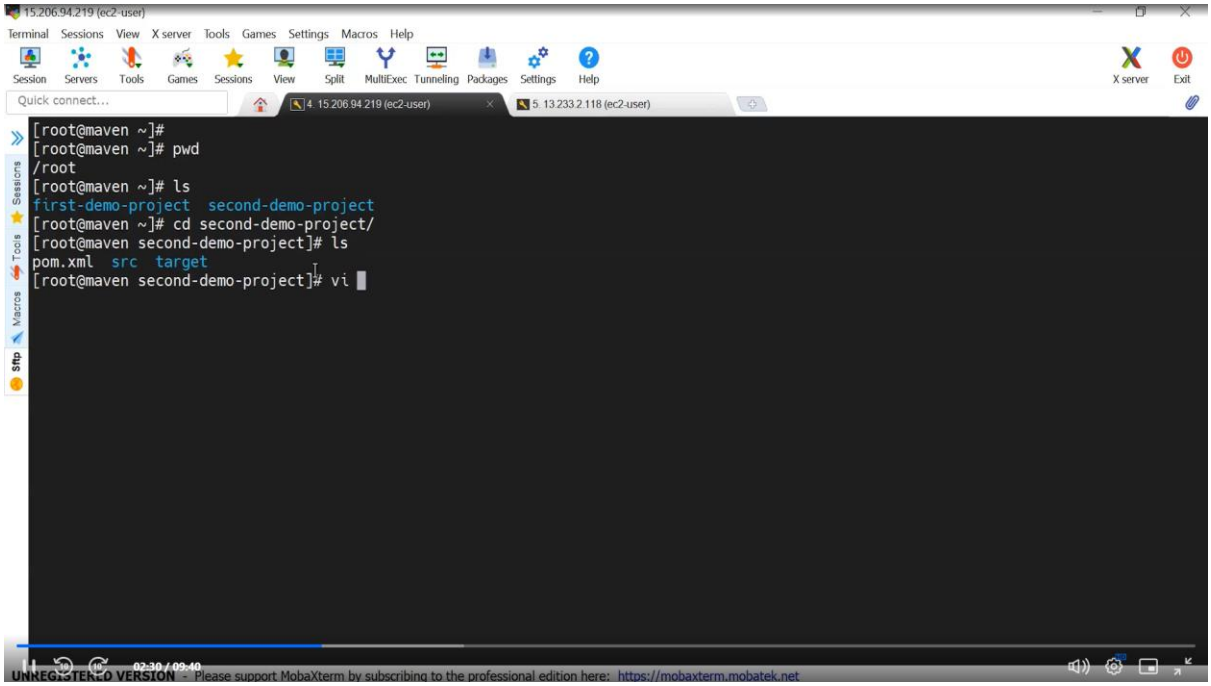
We will deploy application from maven to tomcat .

Deploy an application on tomcat using maven .
https://github.com/ravdy/maven/blob/master/tomcat_installation.MD

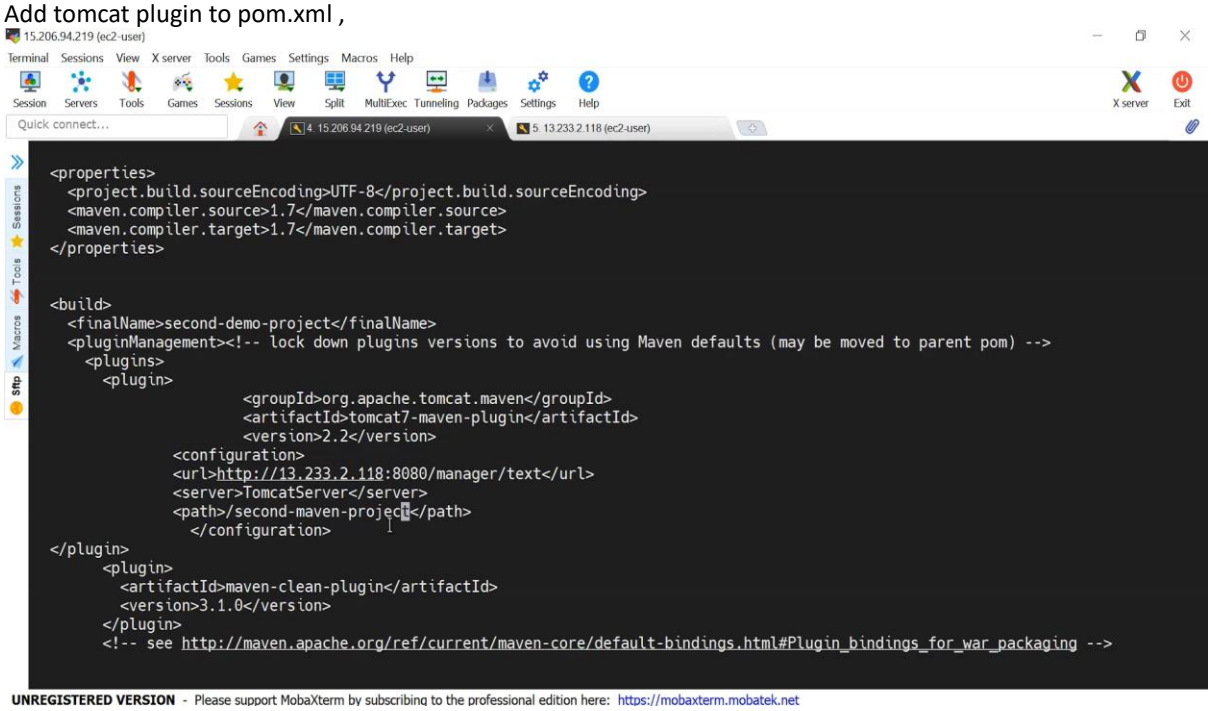


Lets deploy our application to /opt/tomcat/webapp/.
https://github.com/ravdy/maven/blob/master/deploy_warfile_on_tomcat_server_using_maven.md

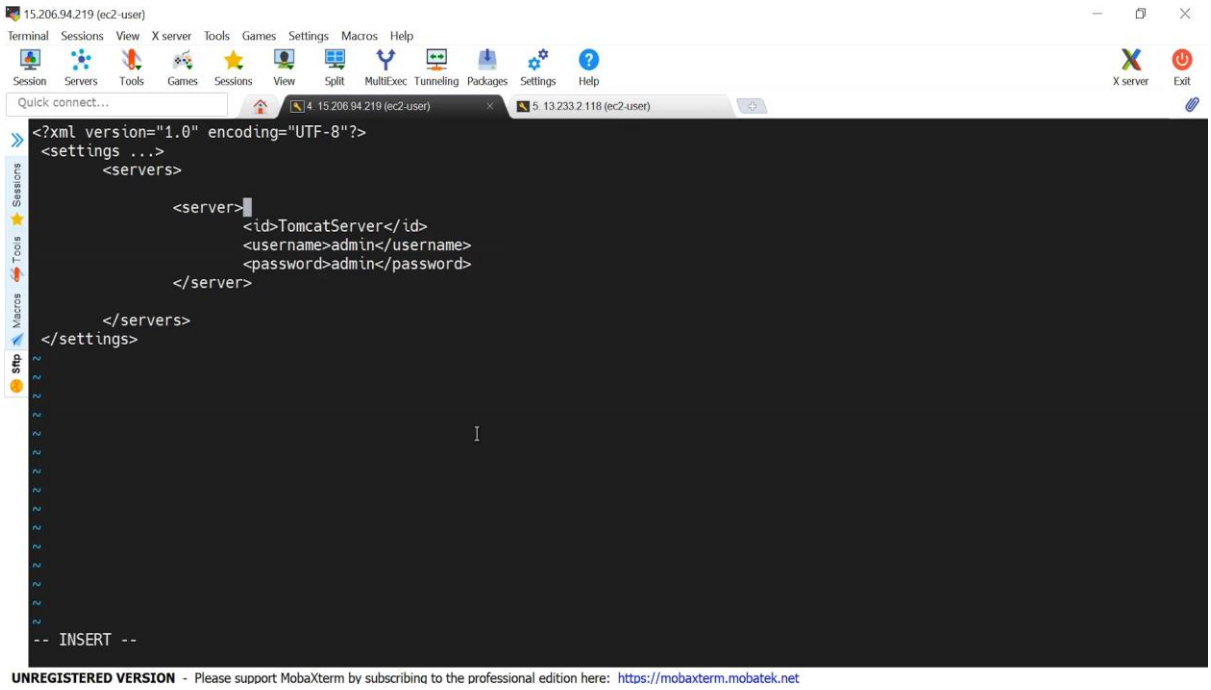
We will deploy second app



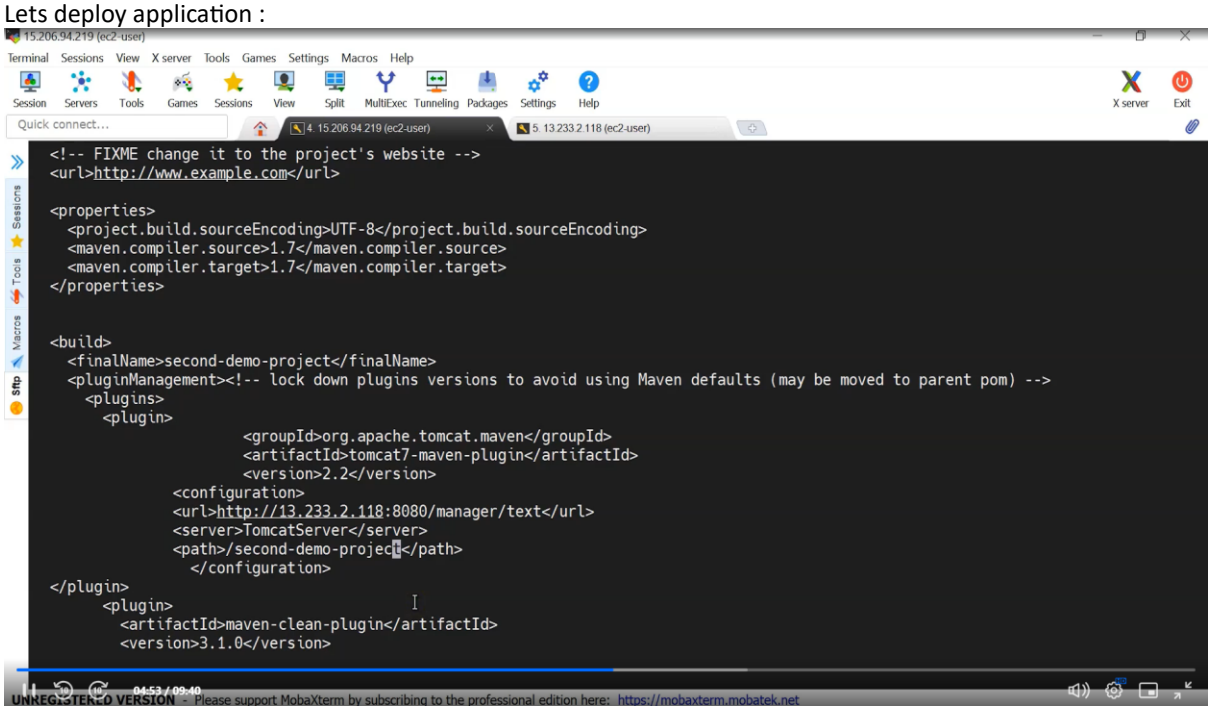
Add tomcat plugin to pom.xml ,



Lets create settings.xml ,



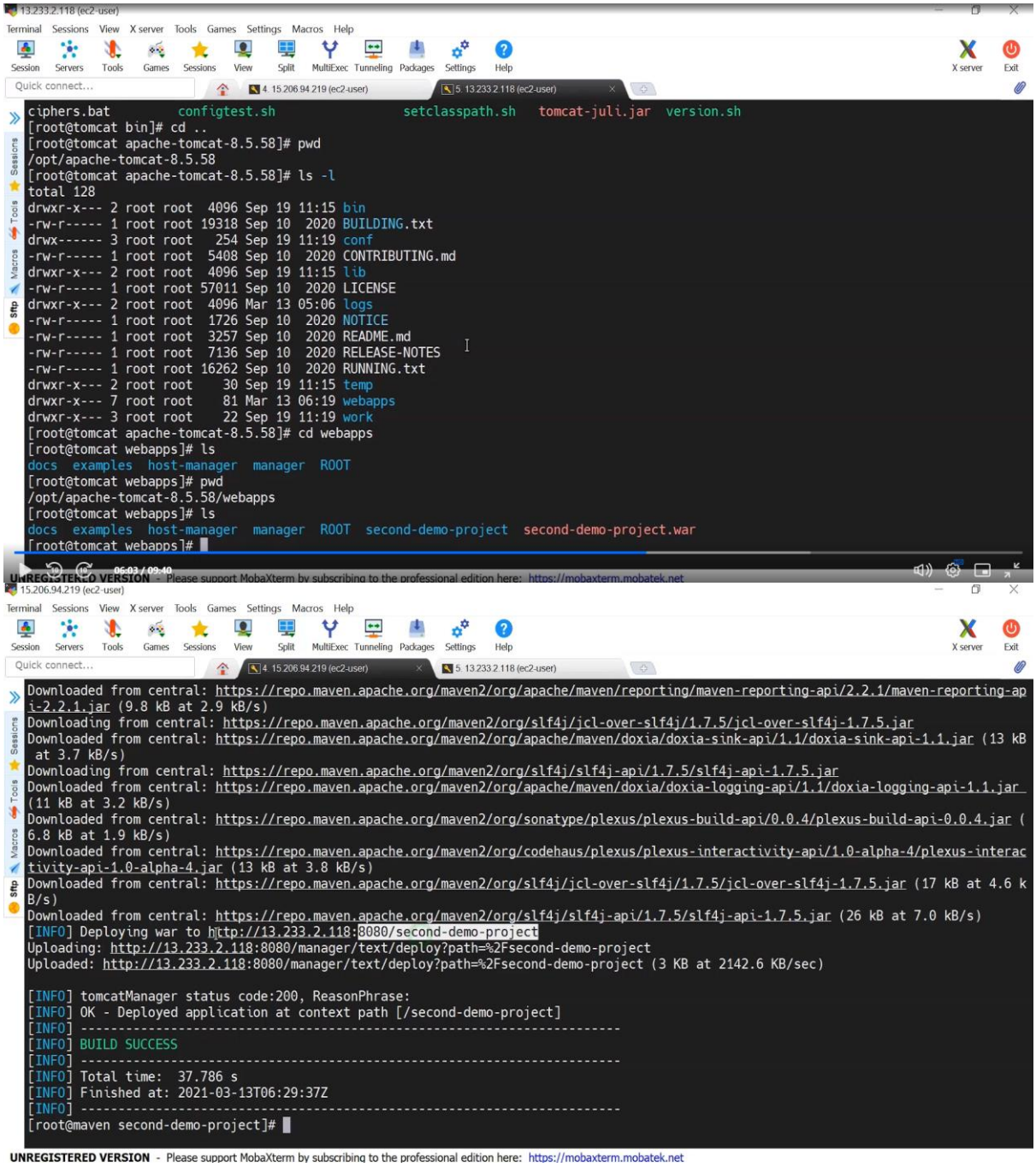
Lets deploy application :



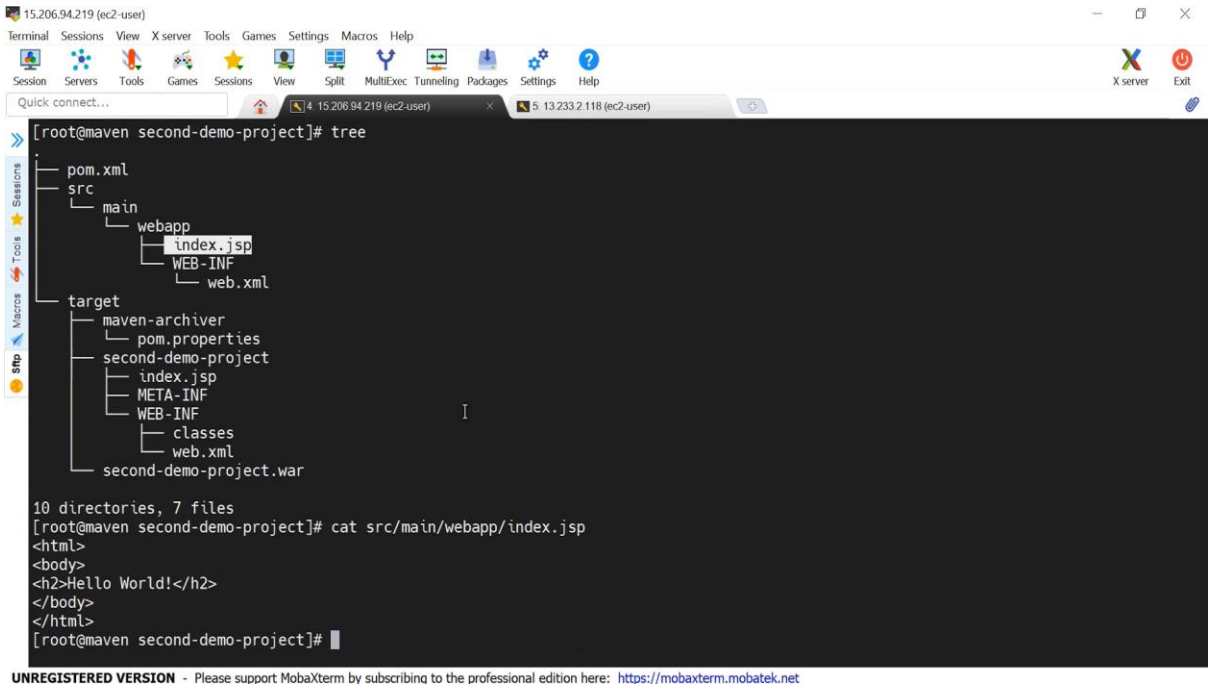
Path should be same as our project name .

Mvn tomcat7:deploy to deploy our tomcat application .

Once deploy done we would be able to see our application deployed in webapps .

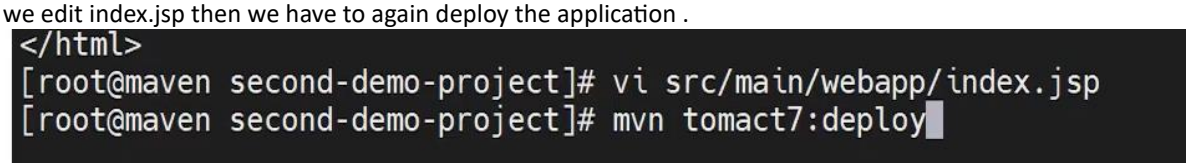


In maven server :

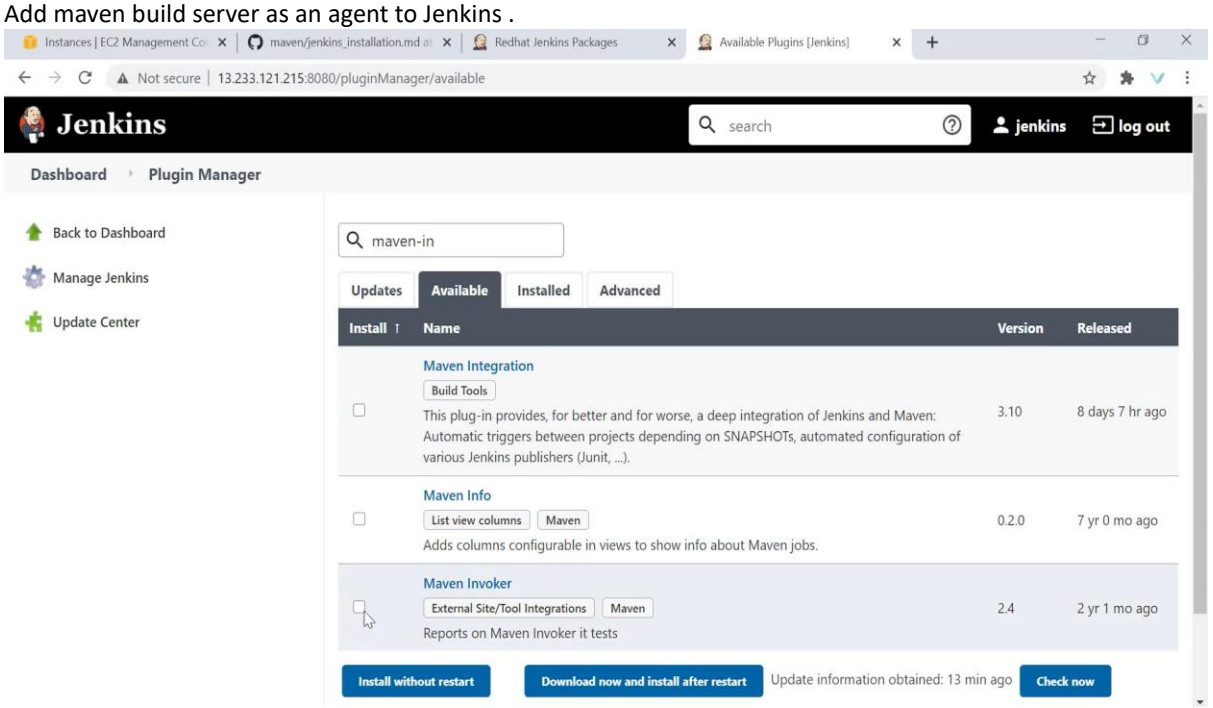


Same application is deployed in index.jsp .

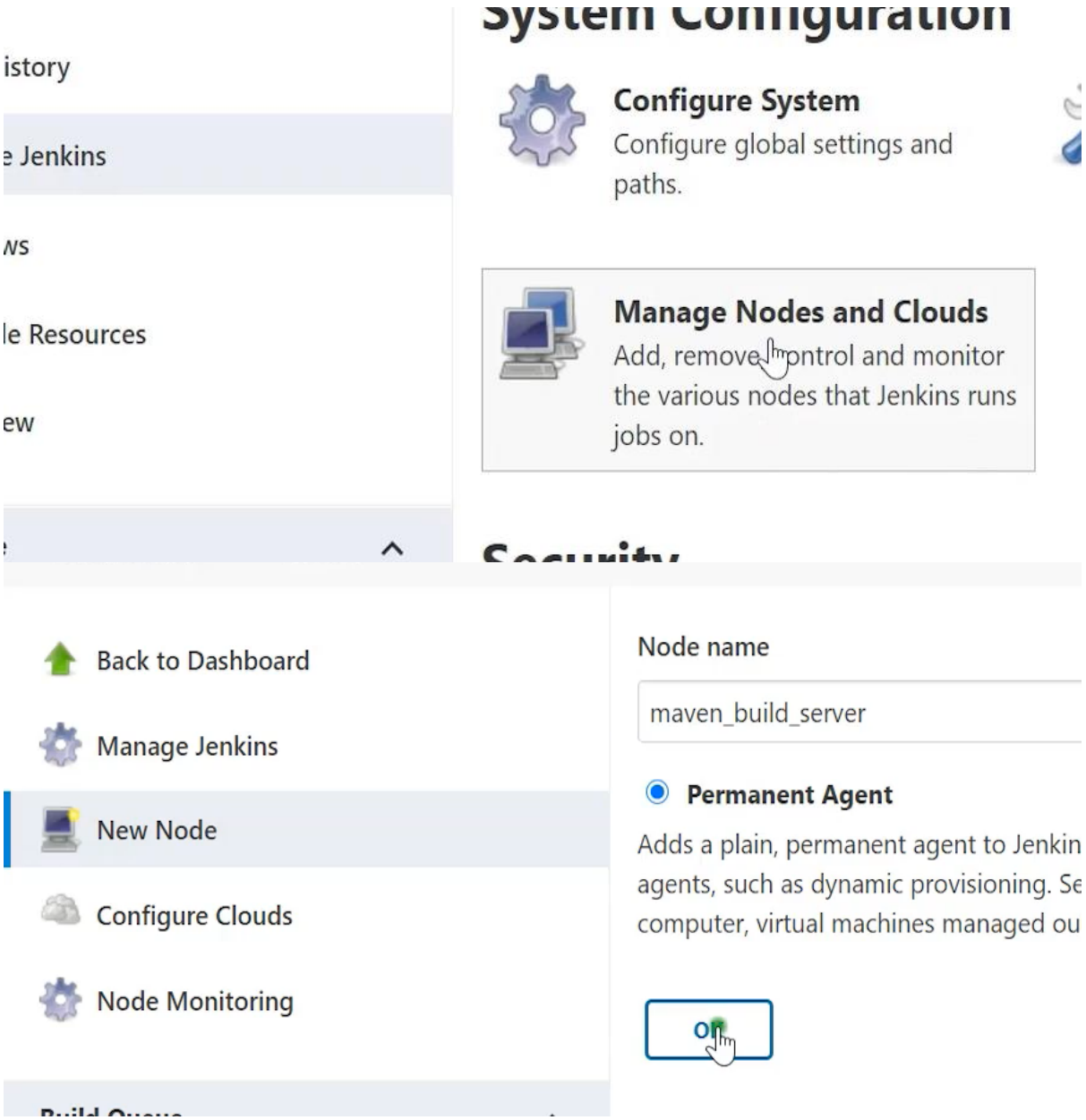
If we edit index.jsp then we have to again deploy the application .

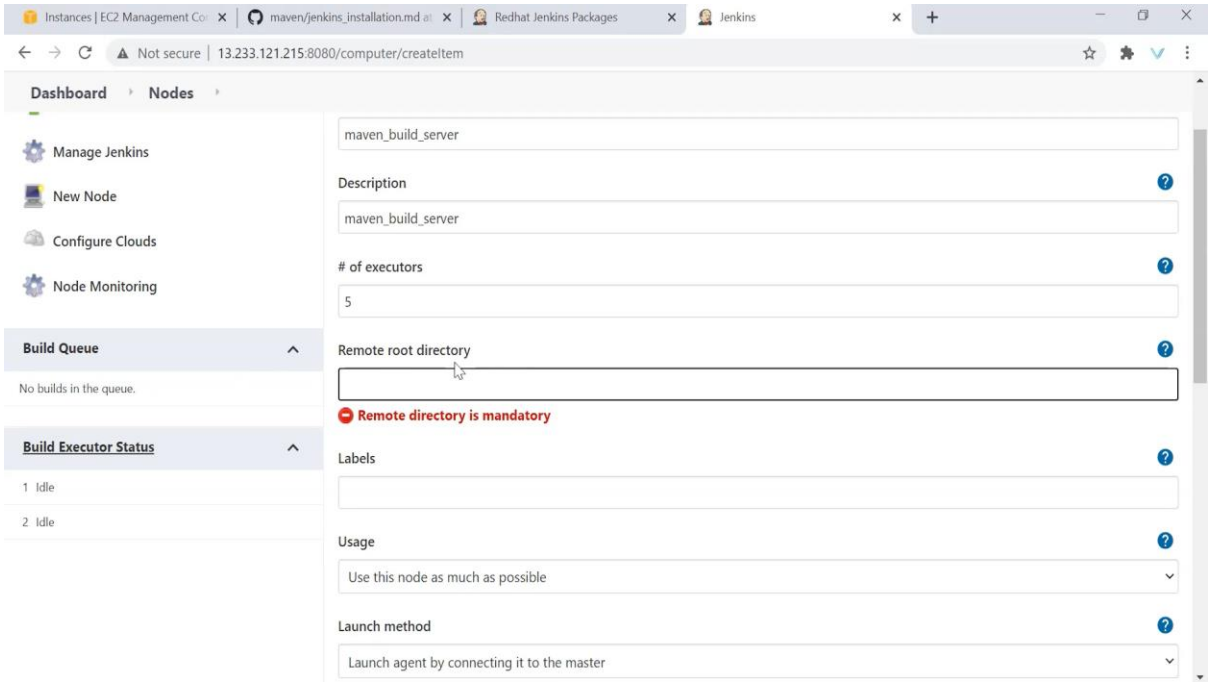


Add maven build server as an agent to Jenkins .



install maven invoker and maven integration plugin .
We are adding maven build server as a slave to Jenkins .


istory
e Jenkins
ns
le Resources
ew
System Configuration
Configure System
Configure global settings and paths.
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Security
Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring
Node name
maven_build_server
Permanent Agent
Adds a plain, permanent agent to Jenkin
agents, such as dynamic provisioning. Se
computer, virtual machines managed ou
OK



Executor – number of parallel build it can run .

For remote root directory as a best practice create new user on maven system .

```
[root@maven second-demo-project]# useradd jenkins
[root@maven second-demo-project]# passwd jenkins
Changing password for user jenkins.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@maven second-demo-project]#
```

Also add this user to sudo file .

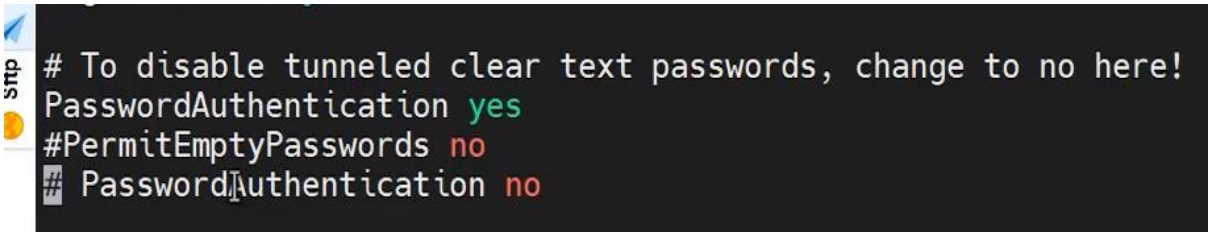
Visudo – added Jenkins as sudo user .

```
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
jenkins ALL=(ALL)       NOPASSWD: ALL
## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL

## Same thing without a password
# %wheel        ALL=(ALL)       NOPASSWD: ALL
## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now
```

By adding Jenkins as above we are telling while executing sudo don't ask for password .

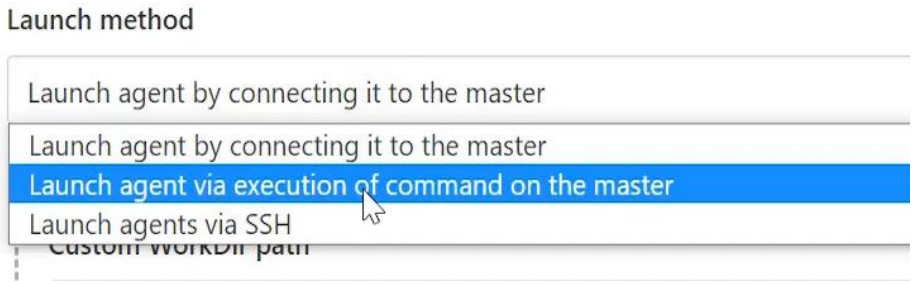
We need to enable password based connection for user .

Edit /etc/ssh/sshd_config

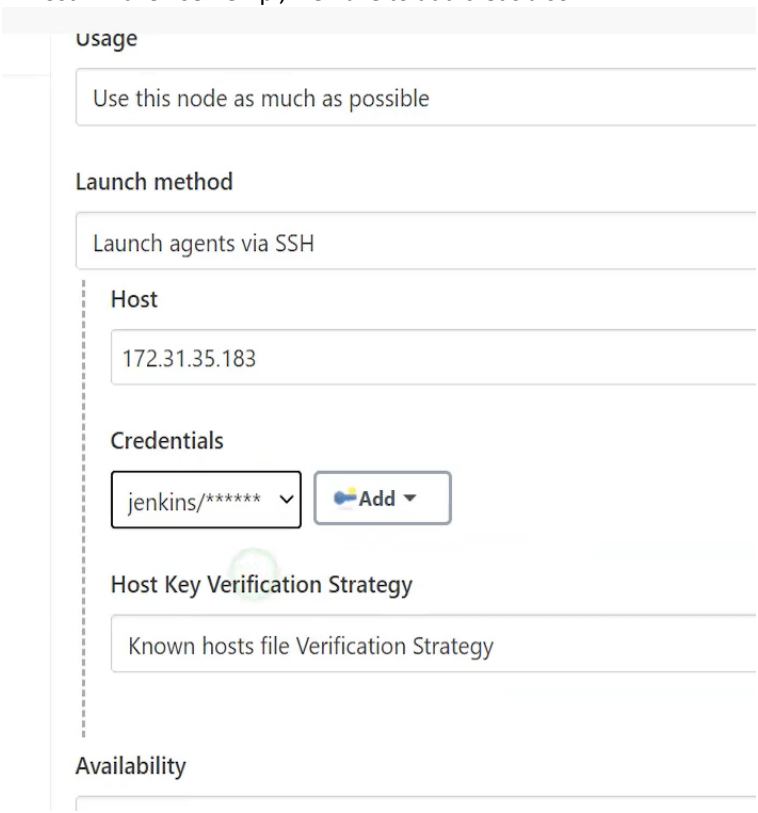


Once password authentication enabled , reload the server – service sshd reload

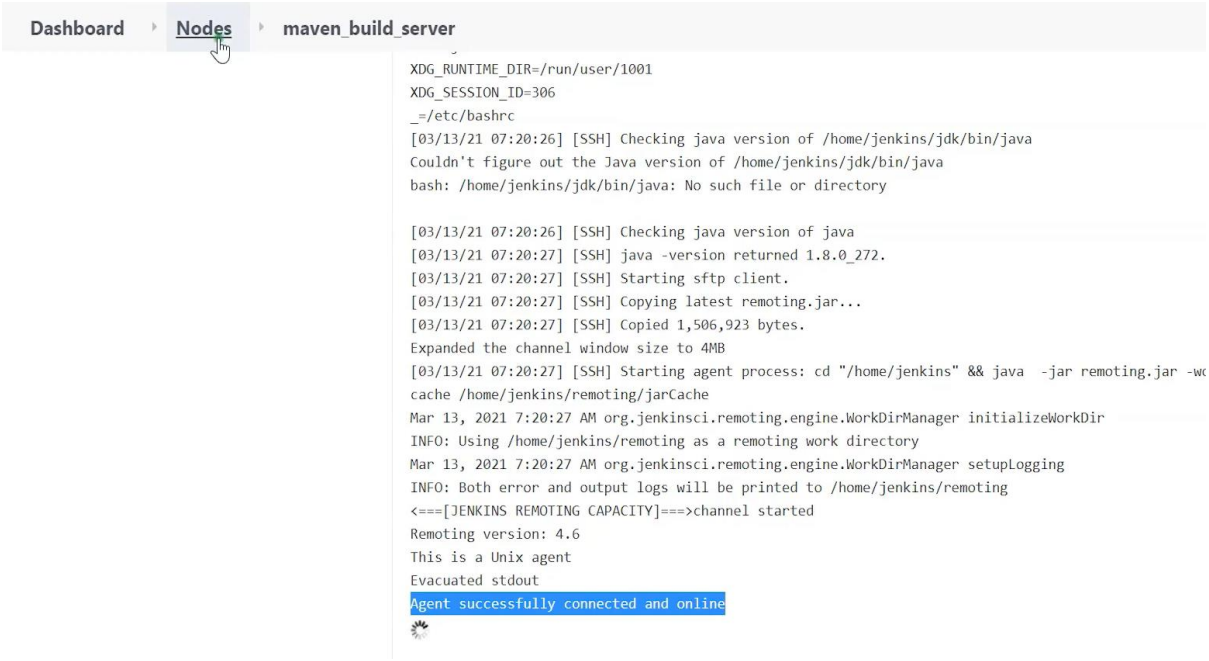
We have enabled password based authentication on maven server . Now use Jenkins to connect to maven server .
Adding maven as slave to Jenkins .
Remote root directory - /home/Jenkins



In host – maven server ip , we have to add creds also .

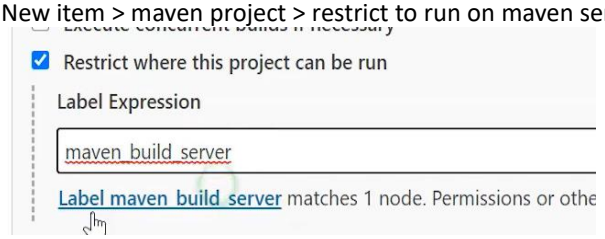


Verification strategy – select non verifying

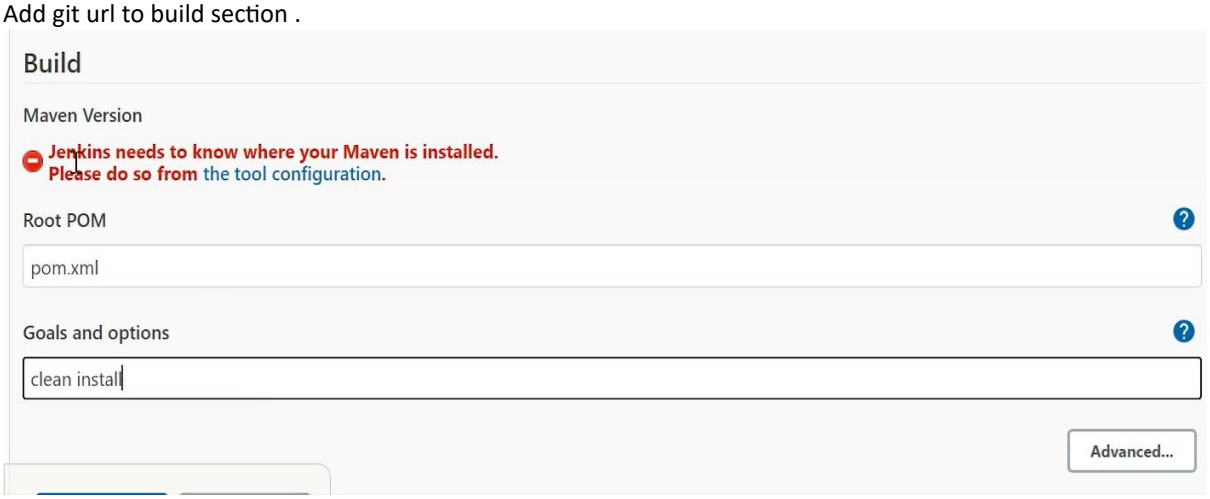


Now agent will connect to maven server .

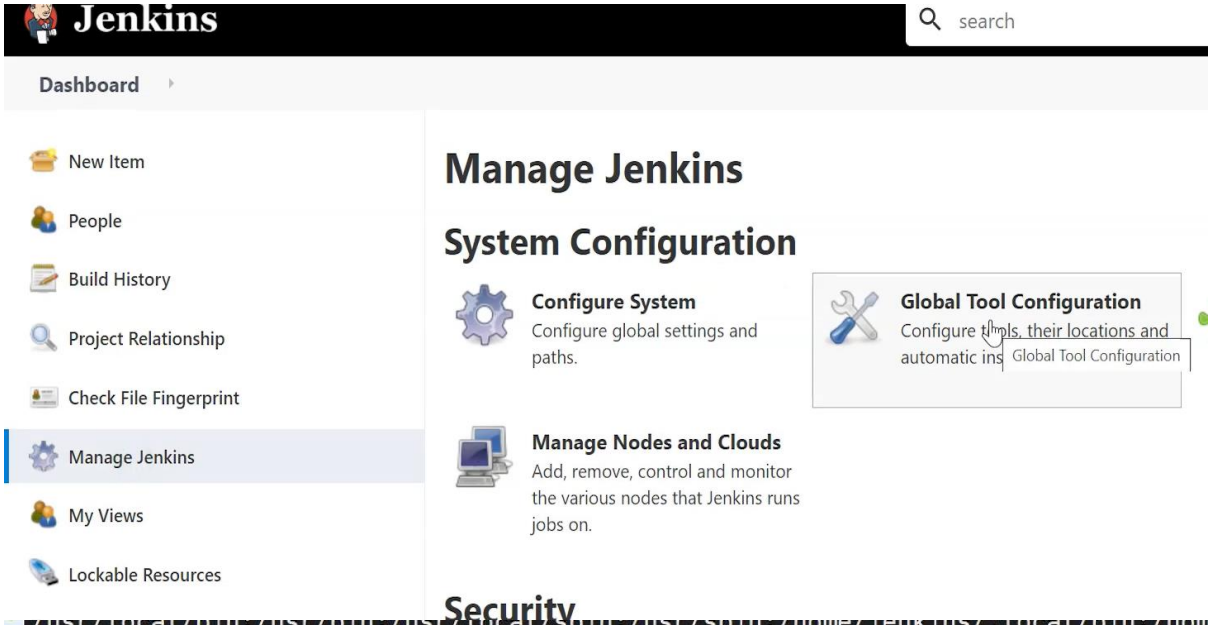
New item > maven project > restrict to run on maven server only .



Add git url to build section .

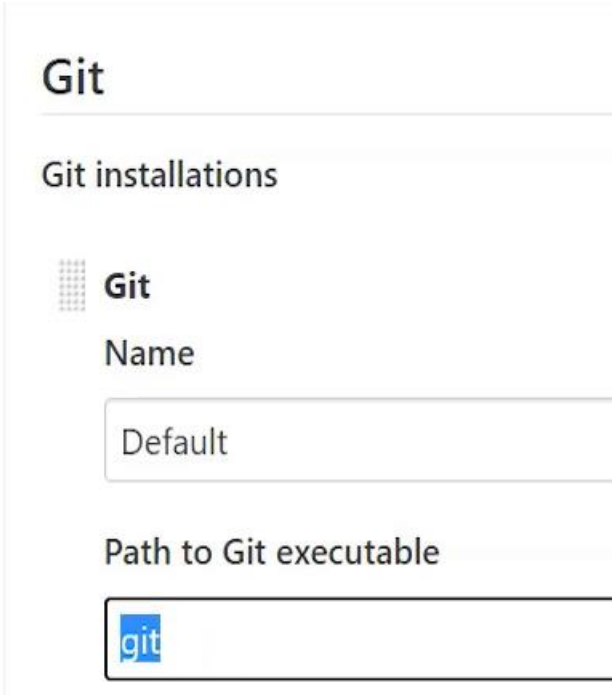


For above error we have to add in global tool configuration .




```
[jenkins@maven ~]$ sudo cat /root/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.272.b10-1.amzn2.0.1.x86_64
M2_HOME=/opt/apache-maven-3.6.3
M2=/opt/apache-maven-3.6.3/bin

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:$JAVA_HOME:$M2:$M2_HOME

export PATH
[jenkins@maven ~]$
```

Add JAVA_HOME path




```
export PATH
[jenkins@maven ~]$ whereis git
git: /usr/bin/git /usr/share/man/man1/git.1.gz
[jenkins@maven ~]$ cd /opt
[jenkins@maven opt]$ ls
apache-maven-3.6.3  apache-maven-3.6.3-bin.tar.gz  aws  jenkins  rh  slave-agent.jnl
[jenkins@maven opt]$ cd apache-maven-3.6.3
[jenkins@maven apache-maven-3.6.3]$ pwd
/opt/apache-maven-3.6.3
[jenkins@maven apache-maven-3.6.3]$
```

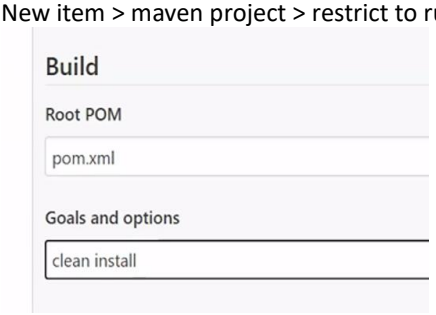
Add maven path also to Jenkins .

```
first-demo-project
[jenkins@maven workspace]$ cd first-demo-project
[jenkins@maven first-demo-project]$ ls
pom.xml  src  target
[jenkins@maven first-demo-project]$
```

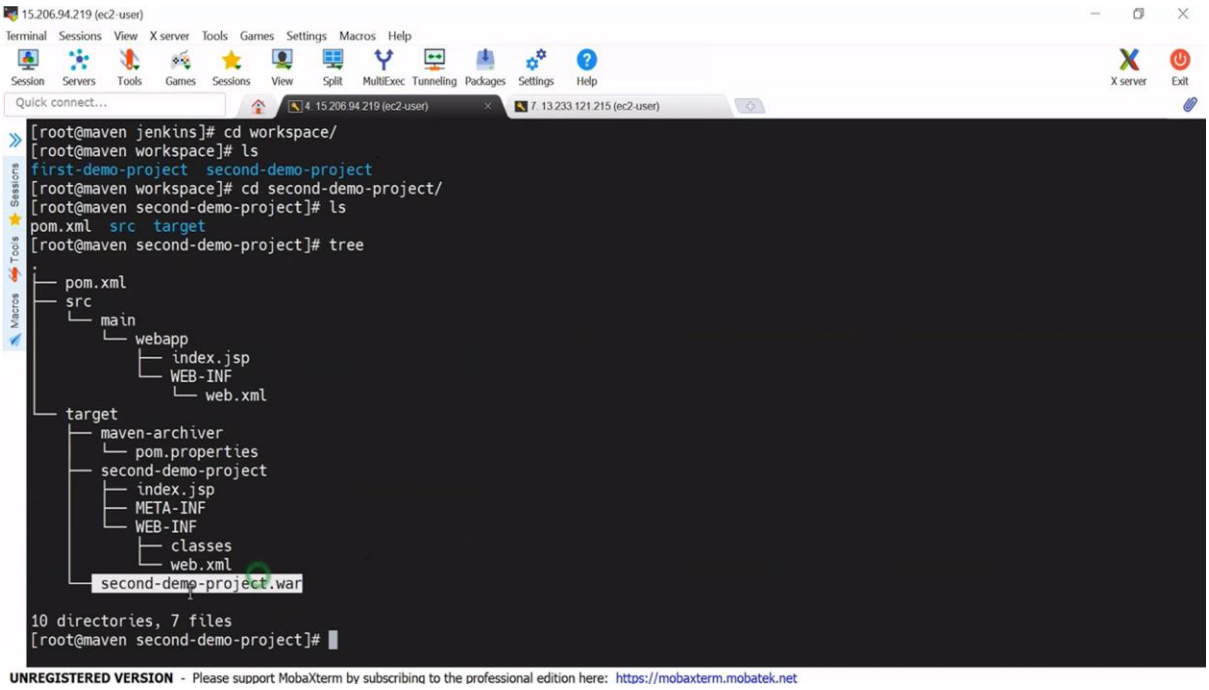
Once we run pipeline our project will be run in workspace folder and target folder will be created as compile goal is also run .

Build war file on our maven build server node .
https://github.com/ravdy/second-demo-project.git

New item > maven project > restrict to run in maven build server > add git repo >



Build the pipeline .



Now the war file is deployed for second project .

Sonarqube installation :
https://github.com/ravdy/DevOps/tree/master/sonarqube

1. A small-scale (individual or small team) instance of the SonarQube server requires at least 2GB of RAM to run efficiently and 1GB of free RAM for the OS. If you are installing an instance for a large teams or Enterprise, please consider the additional recommendations below.

SonarQube is an open-source static testing analysis software, it is used by developers to manage source code quality and consistency.

Prerequisites

Source: https://docs.sonarqube.org/latest/requirements/requirements/

1. An EC2 instance with a minimum of 2 GB RAM (t2.small)
2. Java 11 installation

```
amazon-linux-extras list
amazon-linux-extras install java-openjdk11
```

3. SonarQube cannot be run as root on Unix-based systems, so create a dedicated user account for SonarQube if necessary.

Installation steps

1. Download SonarQube latest verions on to EC2 instace

```
cd /opt
wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-x.x.zip
```

2. extract packages

```
unzip /opt/sonarqube-x.x.zip
```

3. Change ownershipt to the user and Switch to Linux binaries directory to start service

```
chown -R <sonar_user>:<sonar_user_group> /opt/sonarqube-x.x
cd /opt/sonarqube-x.x/bin/linux-x86-64
./sonar.sh start
```

```
[root@ip-172-31-2-255 ~]# java -version
-bash: java: command not found
[root@ip-172-31-2-255 ~]# yum list | grep openjdk
java-1.7.0-openjdk.x86_64              1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.7.0-openjdk-accessibility.x86_64
java-1.7.0-openjdk-demo.x86_64         1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.7.0-openjdk-devel.x86_64        1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.7.0-openjdk-headless.x86_64     1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.7.0-openjdk-javadoc.noarch      1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.7.0-openjdk-src.x86_64          1:1.7.0.261-2.6.22.2.amzn2.0.1 amzn
java-1.8.0-openjdk.x86_64              1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-accessibility.x86_64
java-1.8.0-openjdk-accessibility-debug.x86_64
java-1.8.0-openjdk-debug.x86_64        1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-demo.x86_64         1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-demo-debug.x86_64   1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-devel.x86_64        1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-devel-debug.x86_64  1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-headless.x86_64     1:1.8.0.302.b08-0.amzn2.0.1    amzn
java-1.8.0-openjdk-headless-debug.x86_64
```

Install sonarqube binary package ,

```
[root@ip-172-31-2-255 ~]# cd /opt
[root@ip-172-31-2-255 opt]# ll
total 0
drwxr-xr-x 4 root root 33 Aug 24 19:06 aws
drwxr-xr-x 2 root root  6 Aug 16  2018 rh
[root@ip-172-31-2-255 opt]# wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.9.2.46101.zip
--2021-09-07 07:45:32--  https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.9.2.46101.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com)... 91.134.125.245
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|91.134.125.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 276400757 (264M) [application/zip]
Saving to: 'sonarqube-8.9.2.46101.zip'

100%[=======================================================================================>] 276,400,757 11.5MB/s

2021-09-07 07:45:57 (11.2 MB/s) - 'sonarqube-8.9.2.46101.zip' saved [276400757/276400757]

[root@ip-172-31-2-255 opt]# ll
total 269924
drwxr-xr-x 4 root root        33 Aug 24 19:06 aws
drwxr-xr-x 2 root root         6 Aug 16  2018 rh
-rw-r--r-- 1 root root 276400757 Jul 27 07:55 sonarqube-8.9.2.46101.zip
[root@ip-172-31-2-255 opt]#
```

Unzip sonarqube.zip

```
-rw-r--r--  1 root root 276400757 Jul 27 07:55 sonarqube-8.9.2.46101
[root@ip-172-31-2-255 opt]# cd sonarqube-8.9.2.46101/
[root@ip-172-31-2-255 sonarqube-8.9.2.46101]# ll
total 12
drwxr-xr-x 6 root root   94 Jul 27 06:27 bin
drwxr-xr-x 2 root root   50 Jul 27 06:27 conf
-rw-r--r-- 1 root root 7651 Jul 27 06:27 COPYING
drwxr-xr-x 2 root root   24 Jul 27 06:27 data
drwxr-xr-x 7 root root  132 Jul 27 06:27 elasticsearch
drwxr-xr-x 4 root root   40 Jul 27 06:27 extensions
drwxr-xr-x 6 root root  143 Jul 27 06:41 lib
drwxr-xr-x 2 root root   24 Jul 27 06:27 logs
drwxr-xr-x 2 root root   24 Jul 27 06:27 temp
drwxr-xr-x 6 root root 4096 Jul 27 06:41 web
[root@ip-172-31-2-255 sonarqube-8.9.2.46101]# cd conf
[root@ip-172-31-2-255 conf]# l
-bash: l: command not found
[root@ip-172-31-2-255 conf]# ll
total 28
-rw-r--r-- 1 root root 20529 Jul 27 06:27 sonar.properties
-rw-r--r-- 1 root root  3315 Jul 27 06:27 wrapper.conf
[root@ip-172-31-2-255 conf]# cat sonar.propertie
```

We can update sonar.properties as per our requirement .

```
# The schema must be created first.
#sonar.jdbc.username=
#sonar.jdbc.password=
```

To connect to database we need to specify .

Elastic serach work with non root user so we need to start sonaqrube server from sonarqube new user .

```
[root@ip-172-31-2-255 linux-x86-64]# ./sonar.sh start
Starting SonarQube...
Started SonarQube.
[root@ip-172-31-2-255 linux-x86-64]# ./sonar.sh status
SonarQube is running (3605).
[root@ip-172-31-2-255 linux-x86-64]# ./sonar.sh status
SonarQube is not running.
```

```
se.
uncaught exception in thread [main]
java.lang.RuntimeException: can not run elasticsearch as root
        at org.elasticsearch.bootstrap.Bootstrap.initializeNatives(Bootstrap.java:101)
        at org.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:168)
        at org.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:397)
        at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:159)
```

So we need to run sonarqube from non root user .

```
[root@ip-172-31-2-255 logs]# useradd sonaradmin
[root@ip-172-31-2-255 logs]# chown -R sonaradmin:sonaradmin /opt/sonarqube-8.9.2.46101
[root@ip-172-31-2-255 logs]# pwd
/opt/sonarqube-8.9.2.46101/logs
[root@ip-172-31-2-255 logs]# cd ..
[root@ip-172-31-2-255 sonarqube-8.9.2.46101]# ll
total 12
drwxr-xr-x 6 sonaradmin sonaradmin   94 Jul 27 06:27 bin
drwxr-xr-x 2 sonaradmin sonaradmin   50 Jul 27 06:27 conf
-rw-r--r-- 1 sonaradmin sonaradmin 7651 Jul 27 06:27 COPYING
drwxr-xr-x 2 sonaradmin sonaradmin   24 Jul 27 06:27 data
drwxr-xr-x 7 sonaradmin sonaradmin  132 Jul 27 06:27 elasticsearch
drwxr-xr-x 4 sonaradmin sonaradmin   40 Jul 27 06:27 extensions
drwxr-xr-x 6 sonaradmin sonaradmin  143 Jul 27 06:41 lib
drwxr-xr-x 2 sonaradmin sonaradmin   64 Sep  7 07:49 logs
drwxr-xr-x 3 sonaradmin sonaradmin   38 Sep  7 07:48 temp
drwxr-xr-x 6 sonaradmin sonaradmin 4096 Jul 27 06:41 web
[root@ip-172-31-2-255 sonarqube-8.9.2.46101]# su - so
```

Integration of sonarqube with Jenkins .

https://github.com/ravdy/DevOps/blob/master/sonarqube/integrate_sonar_with_jenkins.md

1. a Sonarqube Server Click here to Setup Sonarqube server
2. A Jenkins server Click here to set up a Jenkins server

Integration Steps

On Sonarqube server

1. Generate a sonarqube token to authenticate from Jenkins

On Jenkins server

1. Install Sonarqube plugin
2. Configure Sonarqube credentials
3. Add Sonarqube to jenkins "configure system"
4. Install SonarScanner
5. Run Pipeline job

🧹 CleanUp

Stop sonarqube services and delete the instance Stop Jenkins service and delete the instance

🔗 My Profile

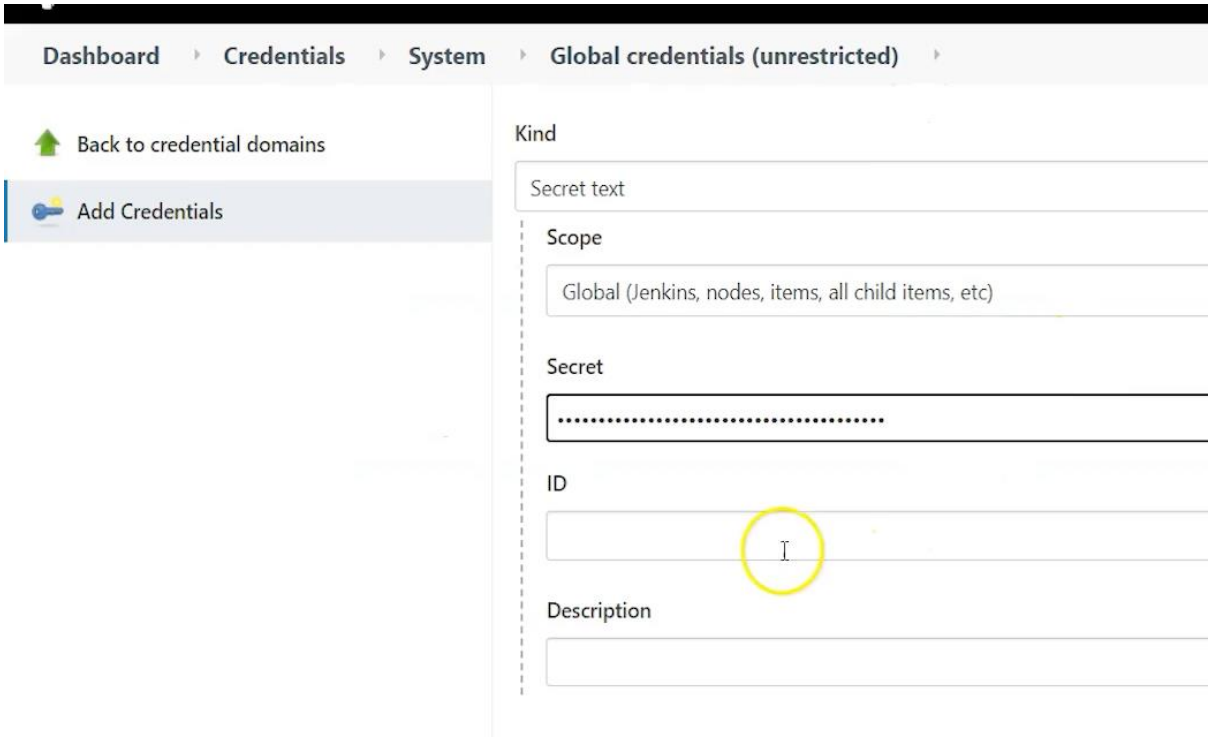
Sonarscanner will run analysis on our project .

Pipeline for sonarqube scan .
https://github.com/ravdy/DevOps/blob/master/sonarqube/Jenkinsfile

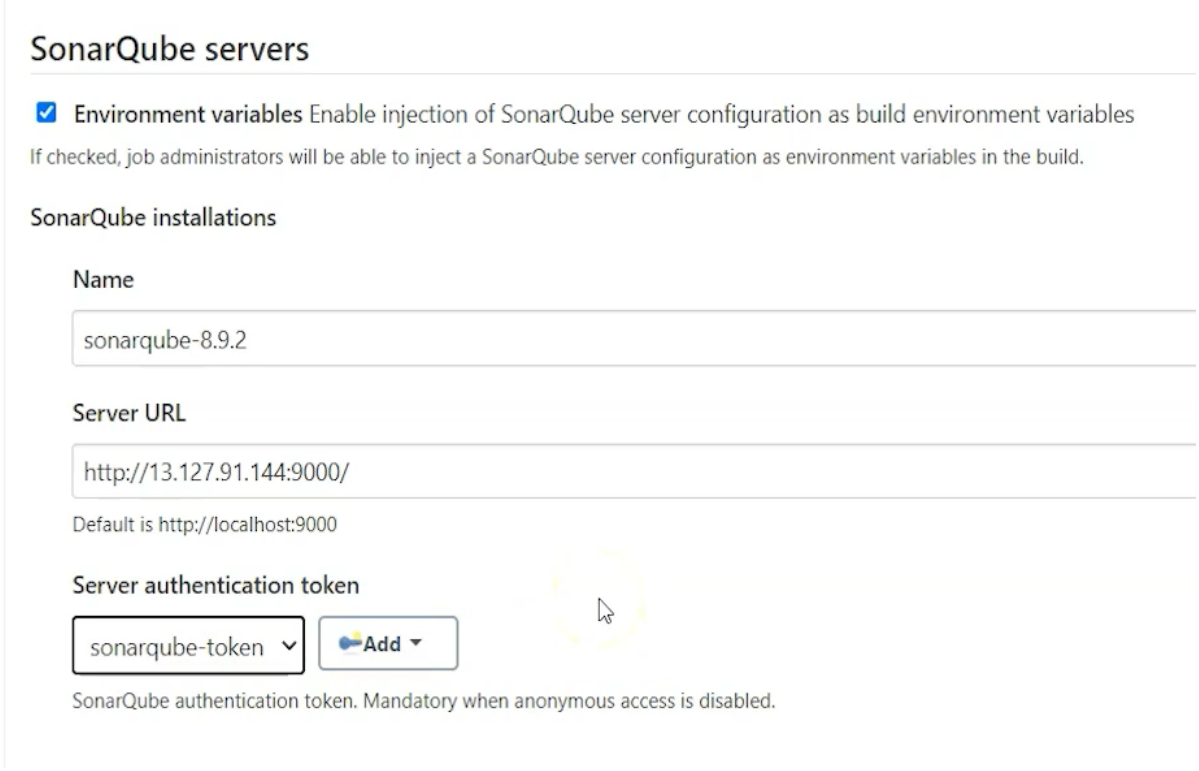
Code | Blame | 29 lines (28 loc) · 721 Bytes

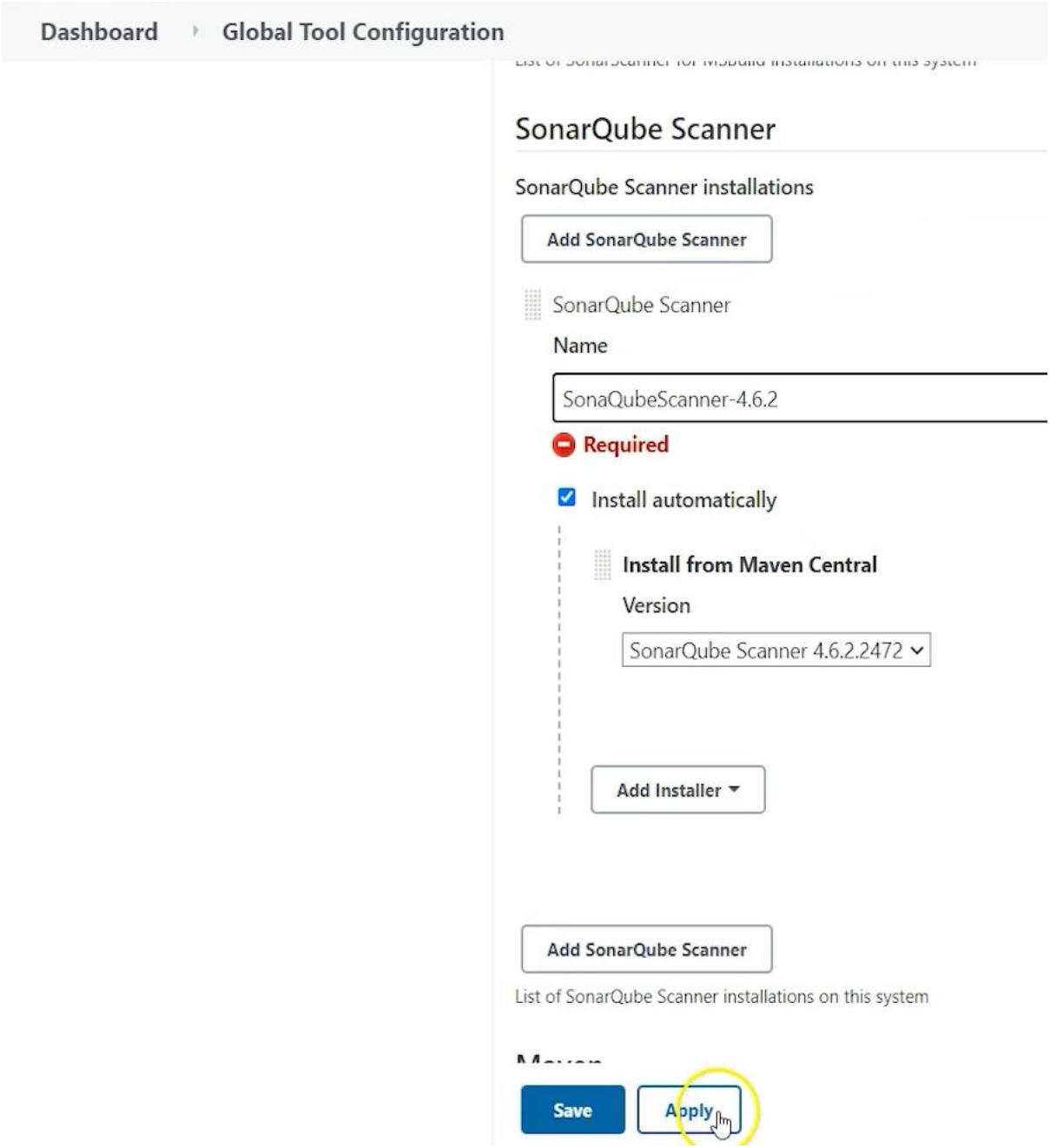
```
pipeline{
    agent any
    environment {
        PATH = "$PATH:/opt/apache-maven-3.8.2/bin"
    }
    stages{
       stage('GetCode'){
            steps{
                git 'https://github.com/ravdy/javaloginapp.git'
            }
        }
       stage('Build'){
            steps{
                sh 'mvn clean package'
            }
        }
        stage('SonarQube analysis') {
//    def scannerHome = tool 'SonarScanner 4.0';
        steps{
        withSonarQubeEnv('sonarqube-8.9') {
        // If you have configured more than one global server connection, you can specify its name
//      sh "${scannerHome}/bin/sonar-scanner"
        sh "mvn sonar:sonar"
    }
        }
        }

    }
}
```

Install sonarqube scanner plugin .



Add token as creds for sonarqube .
Manage Jenkins > configure system > add sonarqube server to Jenkins >



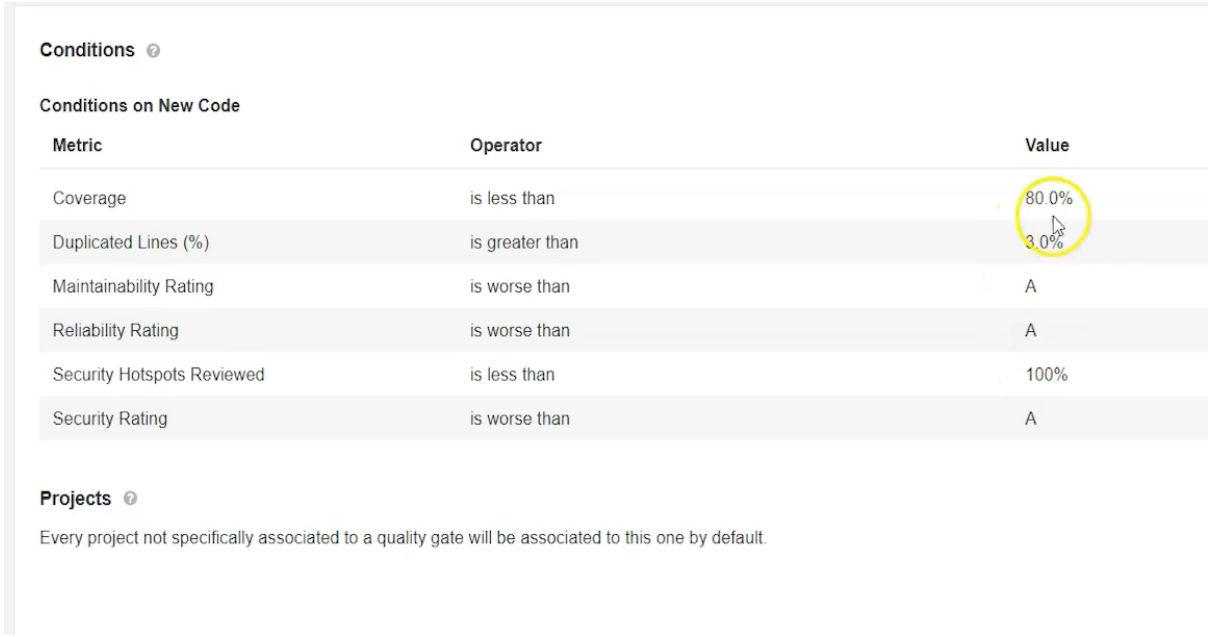


Install sonarqube scanner .
Lets create pipeline for sonarqube scan .
New item > pipeline job > pipeline script – copy script >

In mvn version we can see maven home path :

```
[root@ip-172-31-34-251 ~]# mvn -version
Apache Maven 3.8.2 (ea98e05a04480131370aa0c110b8c54cf726c06f)
Maven home: /opt/apache-maven-3.8.2
Java version: 11.0.12, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-11-openjdk-11.0.12.0.7-0.amzn2.0.2.
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.14.243-185.433.amzn2.x86_64", arch: "amd64", family: "unix"
[root@ip-172-31-34-251 ~]#
```

Copy paste the code and sonarqube scan will be done .
We can create a new quality profile and apply all rules applicable .
Quality gate – here we check bugs velnerabilites based on given value and if it is below the given threshold then our scan will get failed .



Conditions

Conditions on New Code

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Projects

Every project not specifically associated to a quality gate will be associated to this one by default.

Quality gate implementation in sonarqube :
https://jenkinshero.com/sonarqube-quality-gates-in-jenkins-build-pipeline/