# Introduction

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

### What Is Continuous Integration.

Developers will push their code several times in a day to a central repository, every time there is code change it should be pulled, built, tested and notify to the user. **No deployment involved here**

### Types of Environments

1. **Development**

2. **QA** -only Functional testing of the system

3. **Integration** **Testing** -Tests the system from end to end

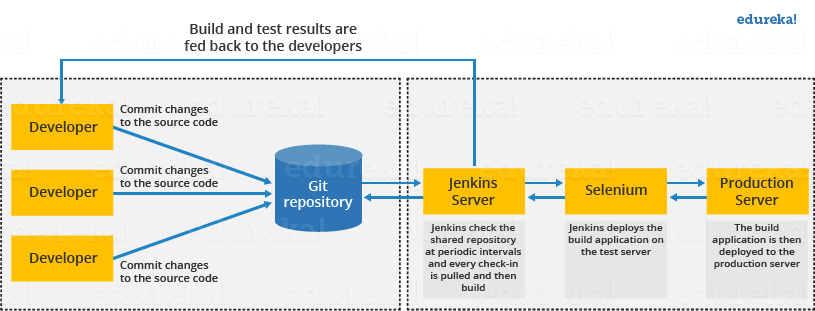
4. **User Acceptance Testing(UAT**) -user will validate the functionality over time.

5. **CERT** -CERT is Certification environment! It’s just where you certify your product so that it can move to production

6. **Production** - Production

7. **Production Parallel** -A parallel of production to replicate production issues

## how Jenkins works

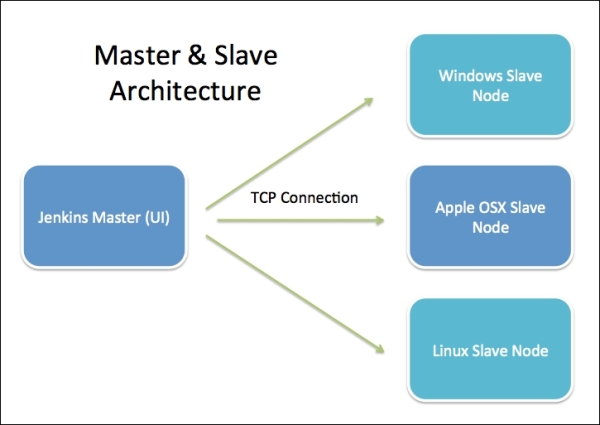


* developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start 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.

## Jenkins Architecture

Jenkins architecture has two components:

* Jenkins Master/Server
* Jenkins Slave/Node



## Jenkins Installation

### Windows

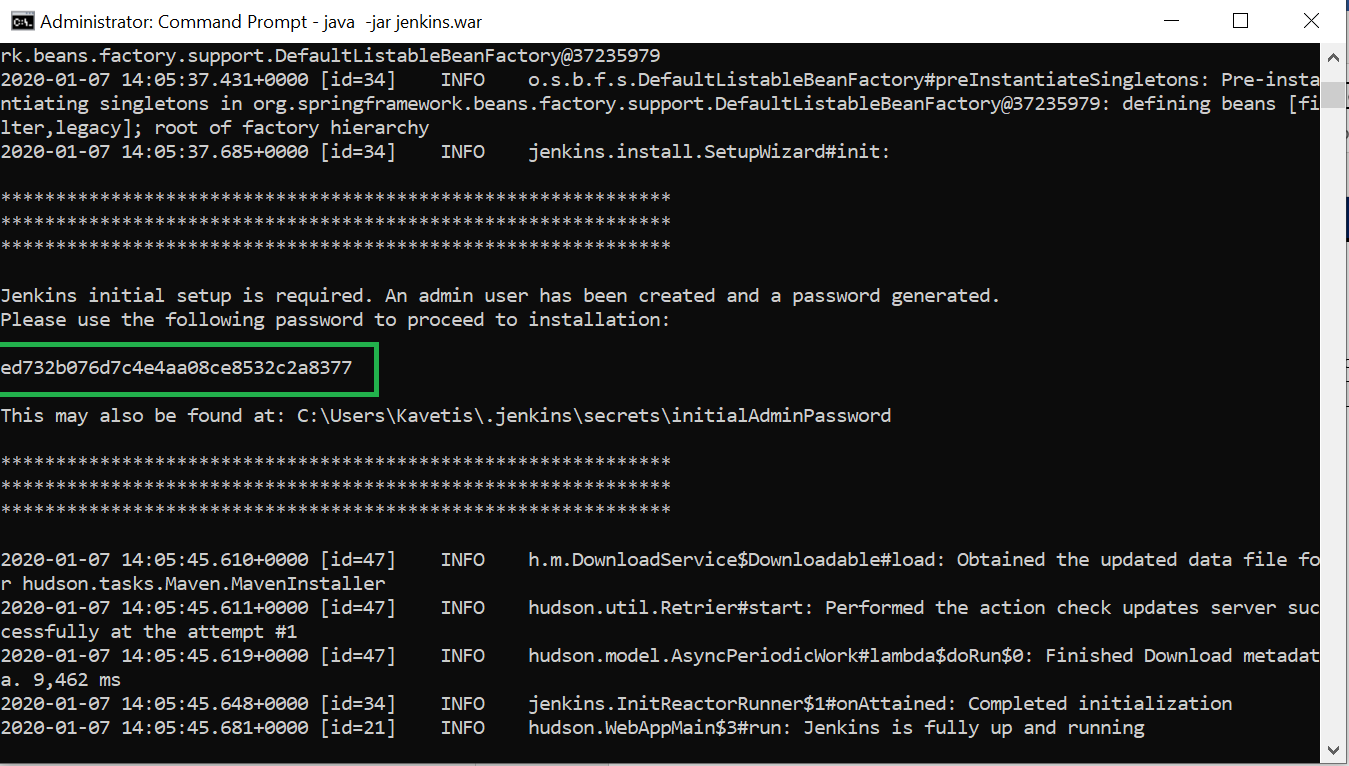
1.[Download & Install Java](https://www.oracle.com/technetwork/java/javase/downloads/index.html)

[2. Download Jenkins war File](https://jenkins.io/download/)

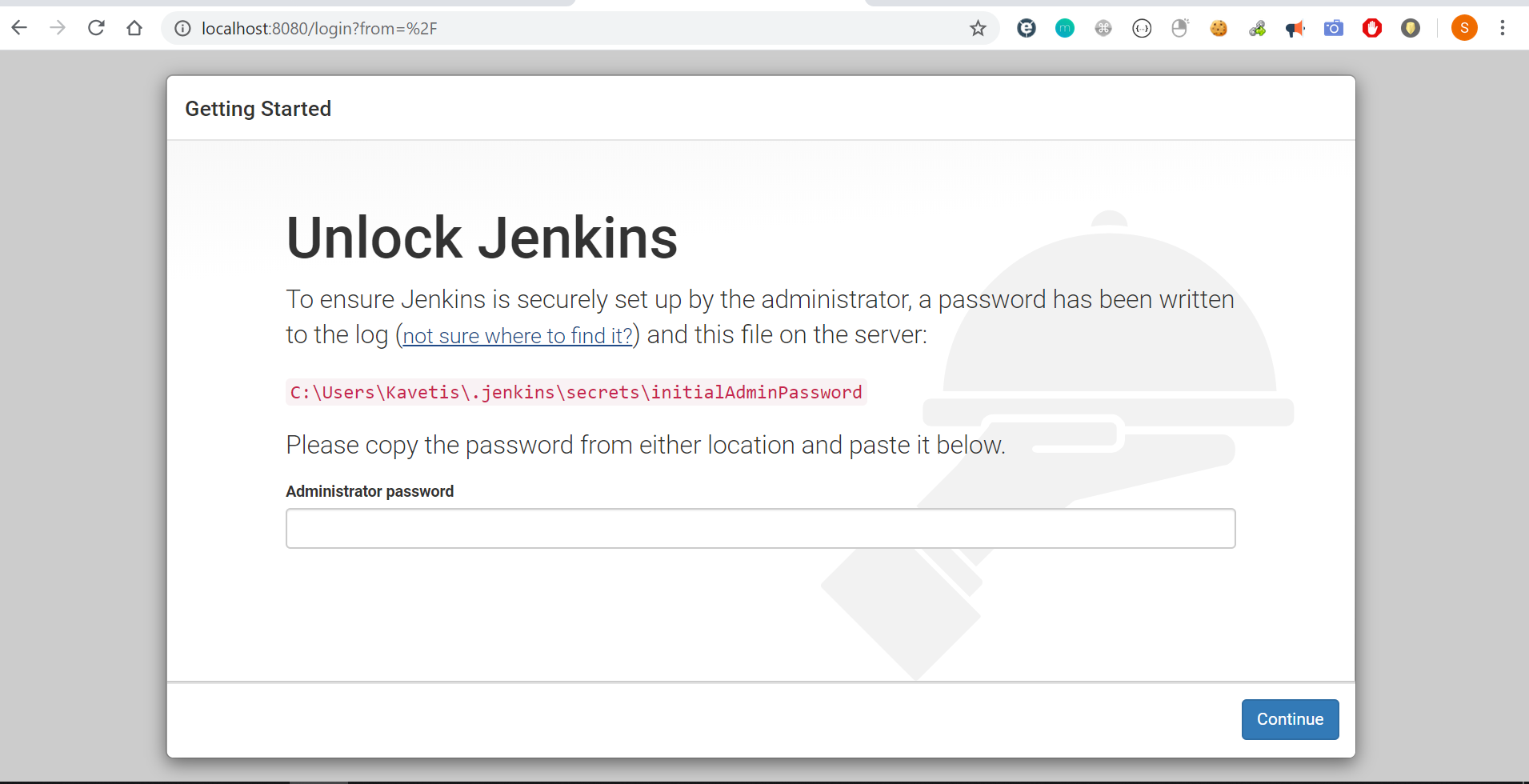
3. Open Command Line as **Administrator** & Run below cmd by navigate to downloaded location

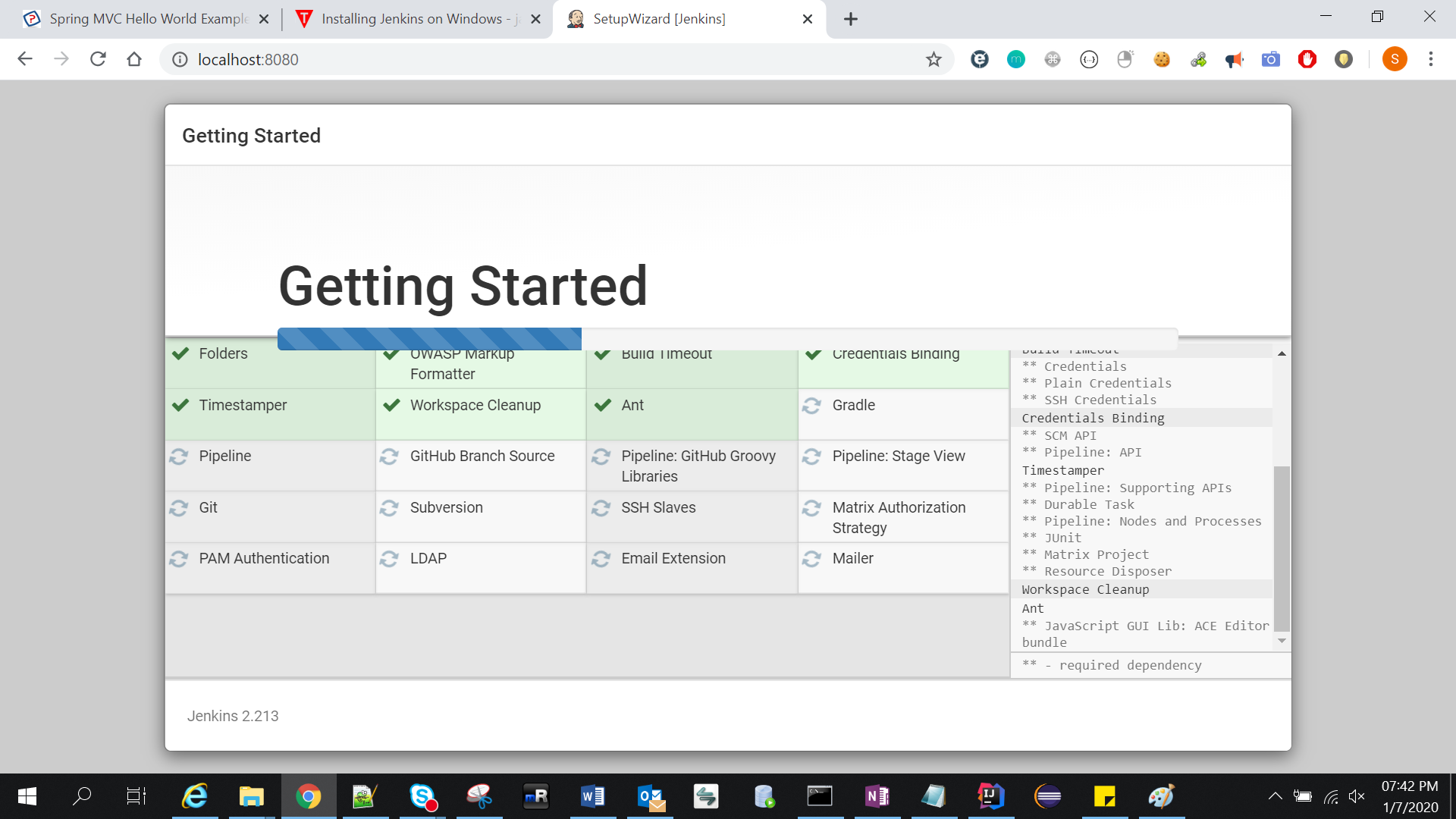
java -jar Jenkins.war

java -jar -httpPort=9999 //to change default port

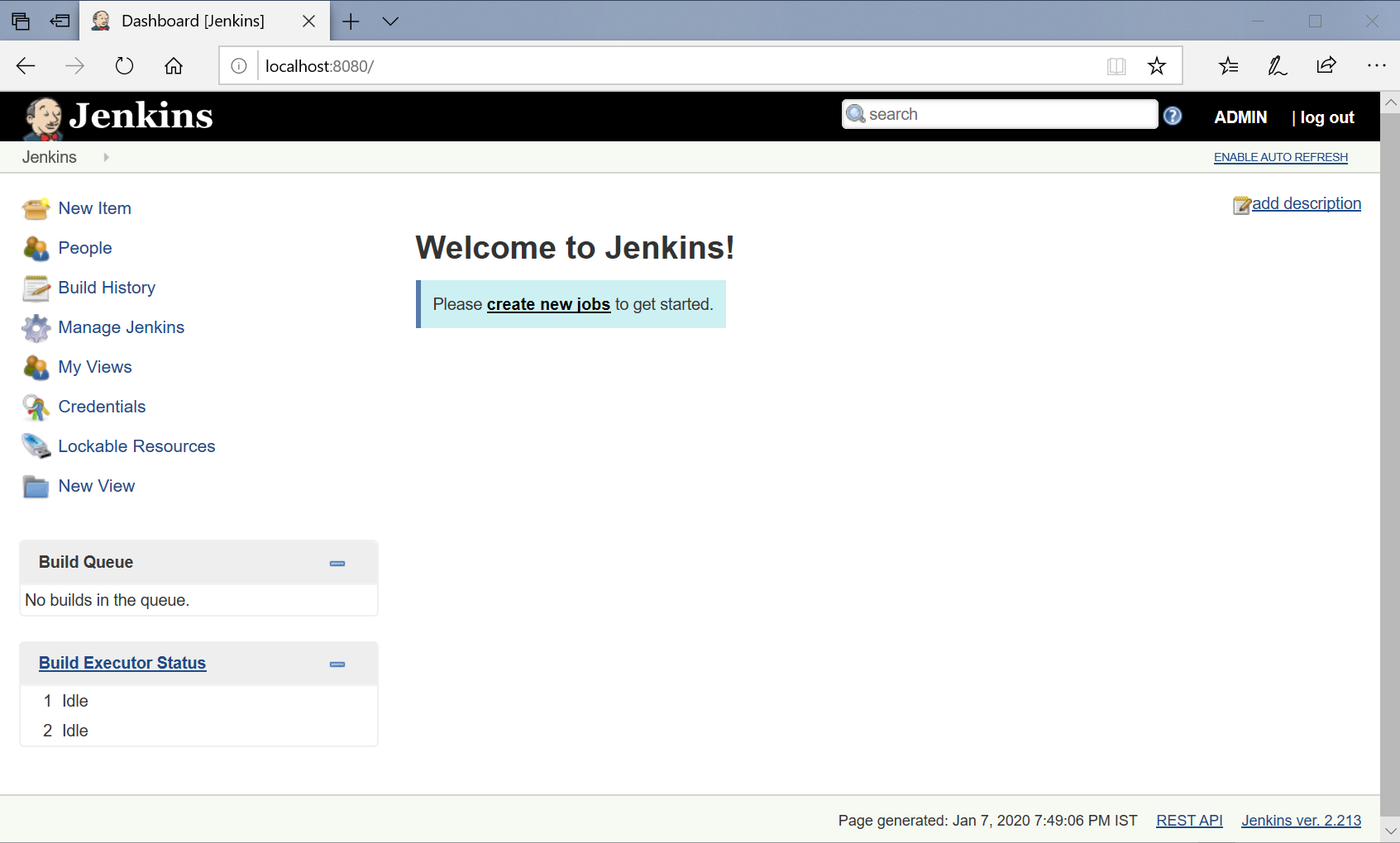


Copy the password from console

4.Accessing Jenkins by <http://localhost:8080> , it will ask for the Administrator password – Paste above copied password 

5.Next, select plug-ins to install. 

6.Create admin user 

7.After that’s just click finish, I will navigate to Dashboard page. 

### Ubuntu -vagrant

**Installing Java**

* Open a terminal window on your Ubuntu server.
* Issue the command below & Allow the installation to complete.

sudo add-apt-repository ppa:webupd8team/java

sudo -E add-apt-repository ppa:openjdk-r/ppa

sudo apt-get update

sudo apt-get install openjdk-8-jdk

To verify Java has been installed, issue the command:

java --version

**Jenkins Install**

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install Jenkins -y

**Check Status**

service jenkins status

systemctl status jenkins

**jenkins restart**

sudo /etc/init.d/jenkins restart

Usage: /etc/init.d/jenkins {start|stop|status|restart|force-reload}

Jenkins Install location

/etc/init.d/jenkins

Port Change

edit the /etc/default/jenkins to replace HTTP\_PORT=8081

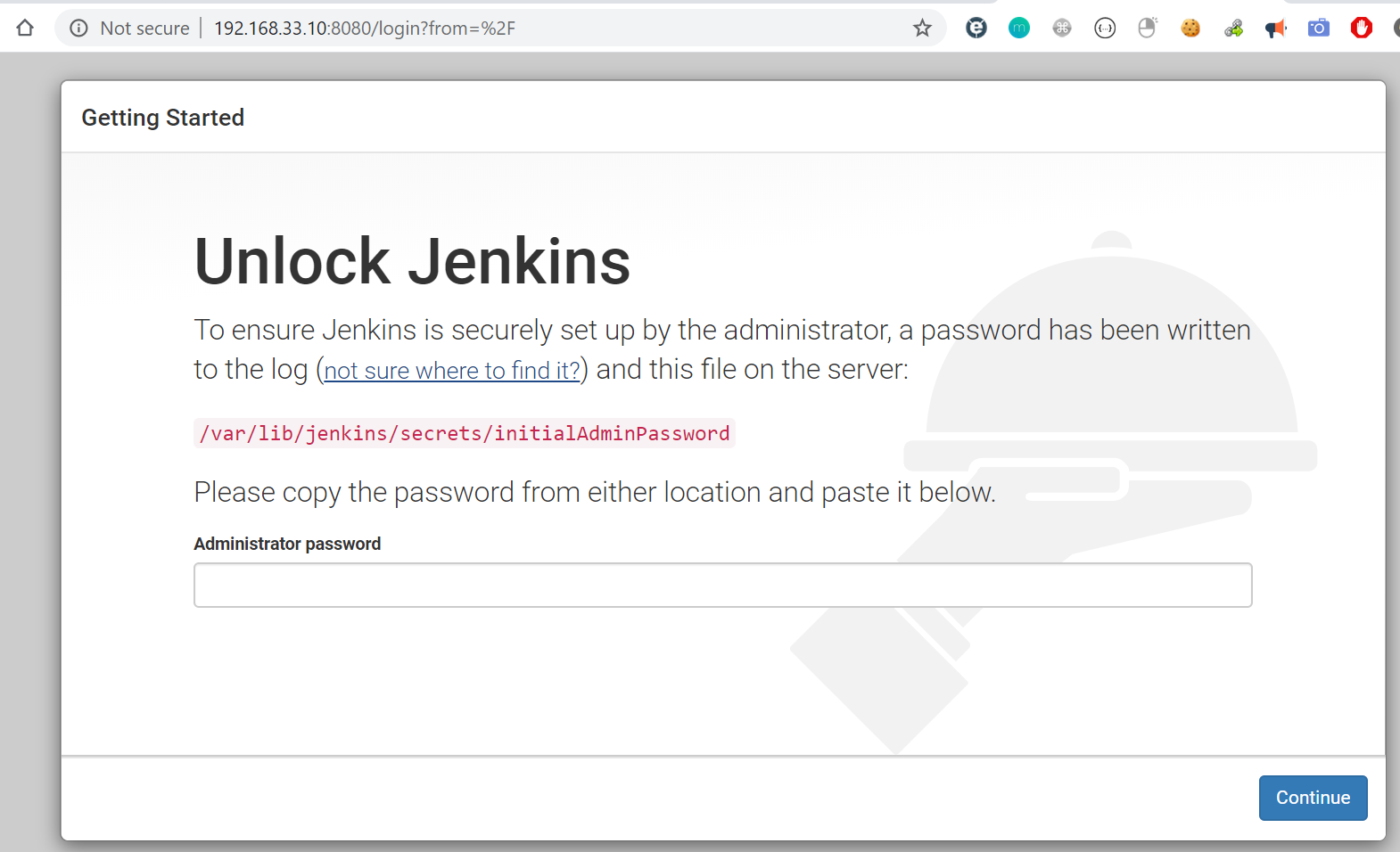
**To Access Jenkins VIA Browser**

* Open the vagrant file (should be in the directory where you specified to create a new vagrant machine).
* Search for config.vm.network.

config.vm.network "private\_network", ip: "192.168.33.10"

* Here ip address (192.168.33.10) can be any ip address you want.
* Now logout from the vagrant machine and reload your vagrant machine by this command vagrant reload.

Access Jenkins : <http://192.168.33.10:8080/login?from=%2F>



In the terminal window, we’ll use the **cat** command to display the password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Follow same steps as windows Installation from now on wards.

### CentOS – Vagarnt

Installing Java

sudo yum install java-1.8.0-openjdk-devel

To verify Java has been installed, issue the command:

java --version

**Install Jenkins**

To enable Jenkins repository, import the GPG key using the following curl command:

curl --silent --location http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo | sudo tee /etc/yum.repos.d/jenkins.repo

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key

install the latest stable version of Jenkins

sudo yum install jenkins

Start Jenkins

sudo systemctl start jenkins

Check Status

systemctl status Jenkins

service jenkins status

Finally enable the Jenkins service to start on system boot.

sudo systemctl enable jenkins

Update Vagrant file with Private IP to able to access with host system browser

config.vm.network "private\_network", ip: "192.168.33.11"

Reload vagrant CentOS box

vagrant reload

Access Jenkins <http://192.168.33.11:8080> , for the first time enter password by doing

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

O/p : 576f5d4d5e6a45b68061ba22c08eb27a

Select Plugins to install > Done!!!

<https://linuxize.com/post/how-to-install-jenkins-on-centos-7/>

## Manage Jenkins

**Configure System**

Configure global settings and paths.

* **Home directory** -where Jenkins stores all of its data in file system.
* **Maven Configuration** - Installation location, MAVEN\_HOME, .m2 location,
* **Jenkins URL** - Custom Jenkins URL
* **SonarQube**  - Servers Details
* **E-mail Notification** - SMTP servers
* **Artifactory servers**
* **Subversion**

**Configure Global Security**

Secure Jenkins; define who can access/use the system.

**Configure Credentials**

Configure the credential providers and types – LDAP servers.

**Global Tool Configuration**

Configure tools, their locations and automatic installers.

* Maven installations
* JDK installations
* Ant installations
* SonarQube Scanner installations

**Reload Configuration from Disk**

Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

**Manage Plugins**

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

**System Information**

Displays various environmental information to assist trouble-shooting.

**System Log**

System log captures output from java.util.logging output related to Jenkins.

**Load Statistics**

Check your resource utilization and see if you need more computers for your builds.

**Jenkins CLI**

Access/manage Jenkins from your shell, or from your script.

**Script Console**

Executes arbitrary script for administration/trouble-shooting/diagnostics.

**Manage Nodes and Clouds**

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

**Manage Users**

Create/delete/modify users that can log in to this Jenkins

**Prepare for Shutdown**

Stops executing new builds, so that the system can be eventually shut down safely.

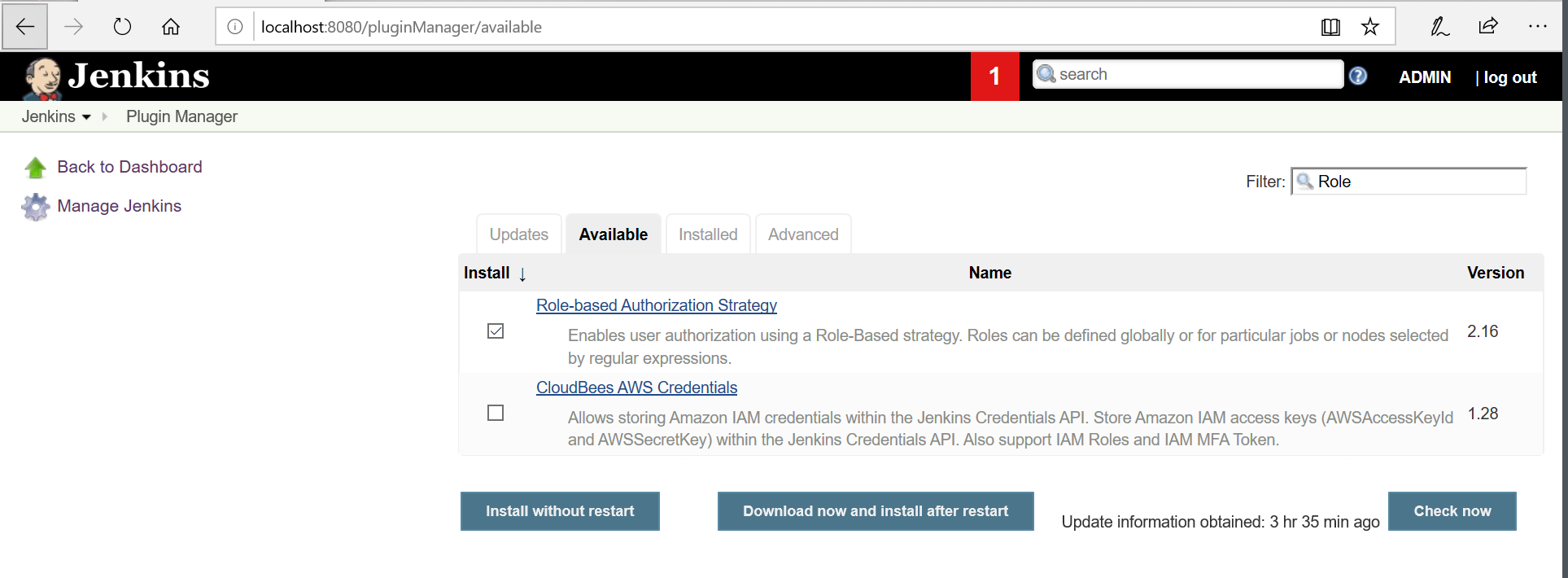
# Jenkins – Manage users

We can manage users & their access in 3 ways

1. Users & Role Management – Using Jenkins plugin
2. Using LDAP
3. Using SSO

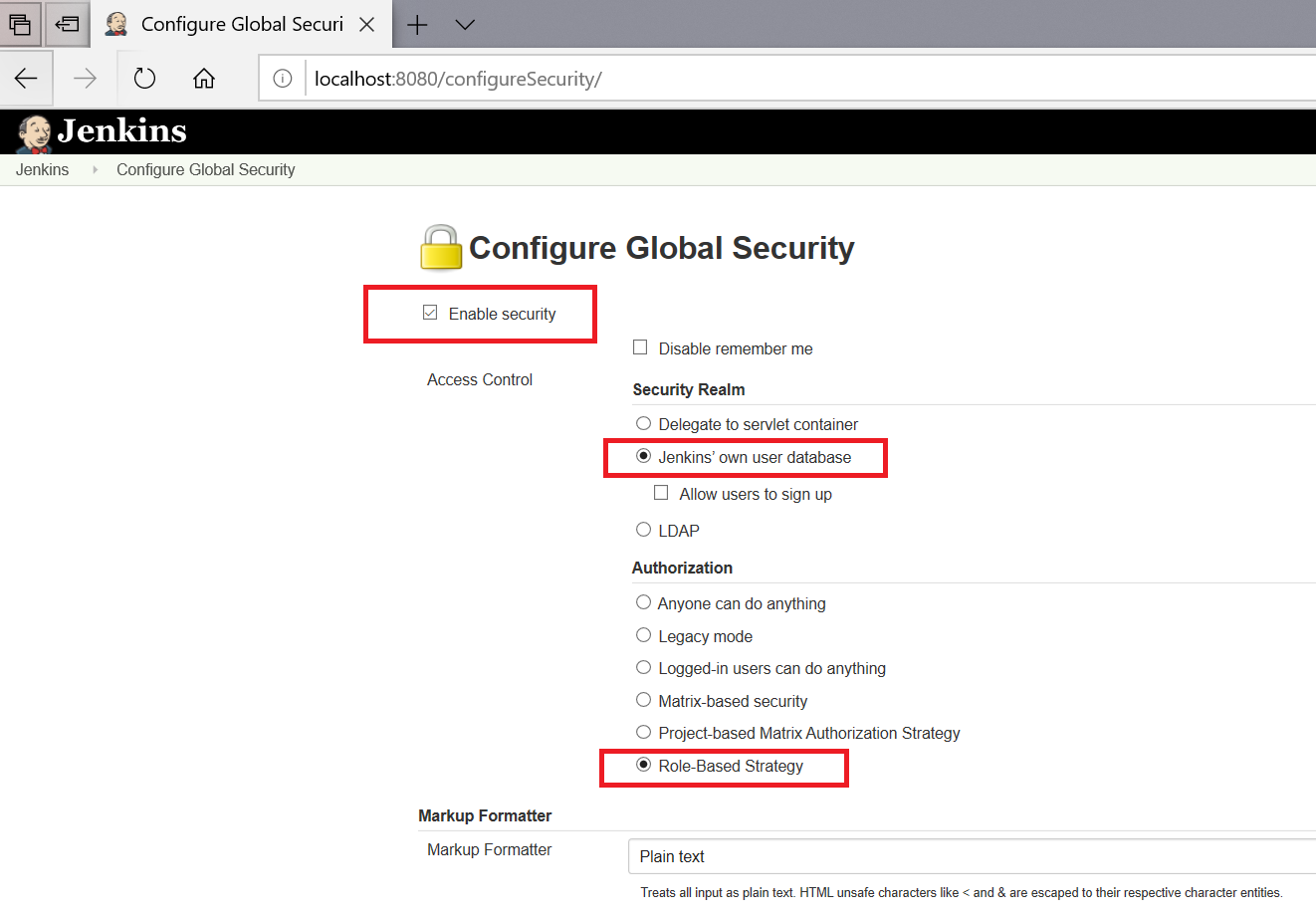
## Users & Role Management

By default, when you create a user in Jenkins, it can access almost everything. In this, you can create multiple users and assign different roles and privileges to different users. For doing that we need to install **"Role-based Authorization Strategy** ".

**Jenkins >Manage Jenkins> Available tab**>filter - search "**Role-based Authorization Strategy**" , install the plugin. 

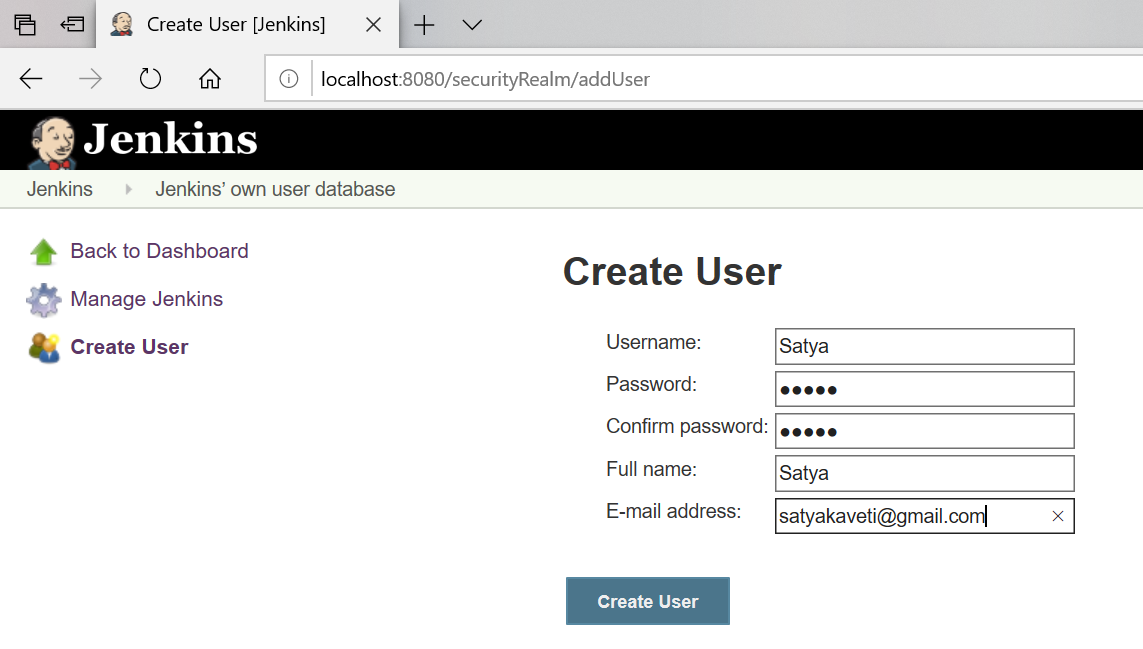
After Plugin installation, go to the **'Manage Jenkins**' and then click on **'Configure Global Security'**.

* Check on **Enable security** option.
* On the Security Realm section, select **'jenkins' own user database**'.
* On Authorization section, select **'Role-Based Strategy'**.

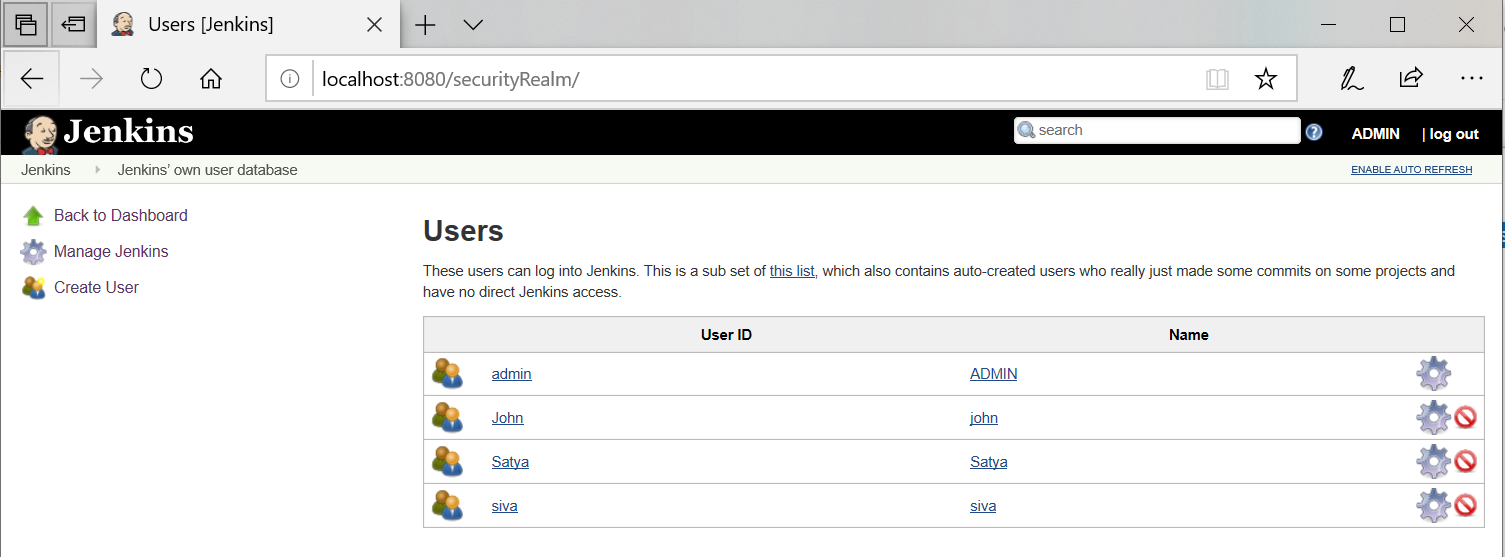


### Now add users

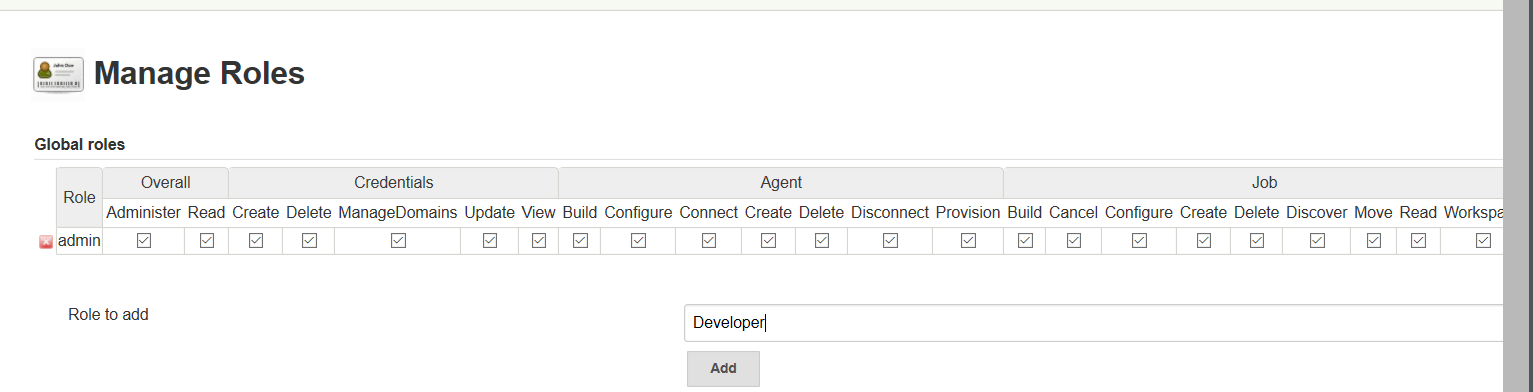
**Dashboard> Manage Jenkins > Manage Users > Create User**

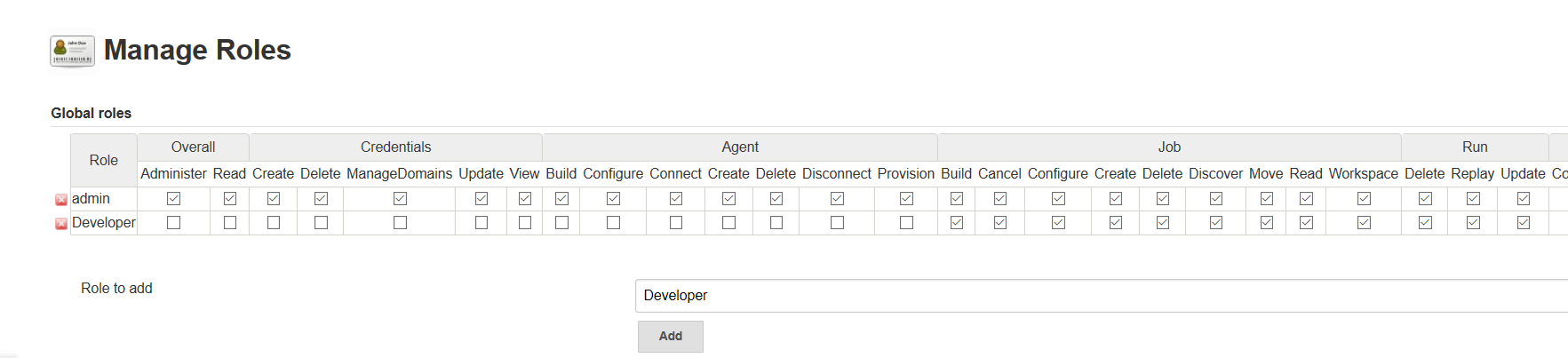


Add required no. of users



### Adding Roles

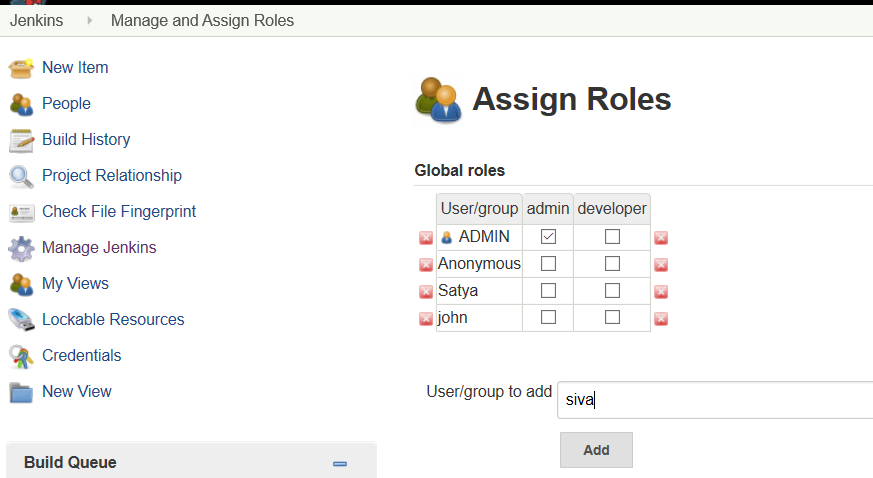
**Manage Jenkins> Manage and Assign Roles> Manage Roles>** **Role to add** - Provide details& Save 

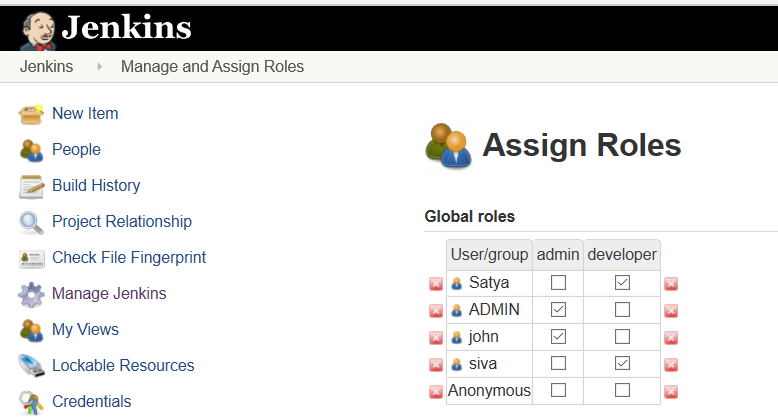
Provide Access levels for that role. Ex: only Build & configure jobs 

### Assign Roles to Users

Manage Jenkins> Manage and Assign Roles> Manage Roles> Assign Roles

* Add the User name on **User/group to add** option.
* Click on **Add** button.



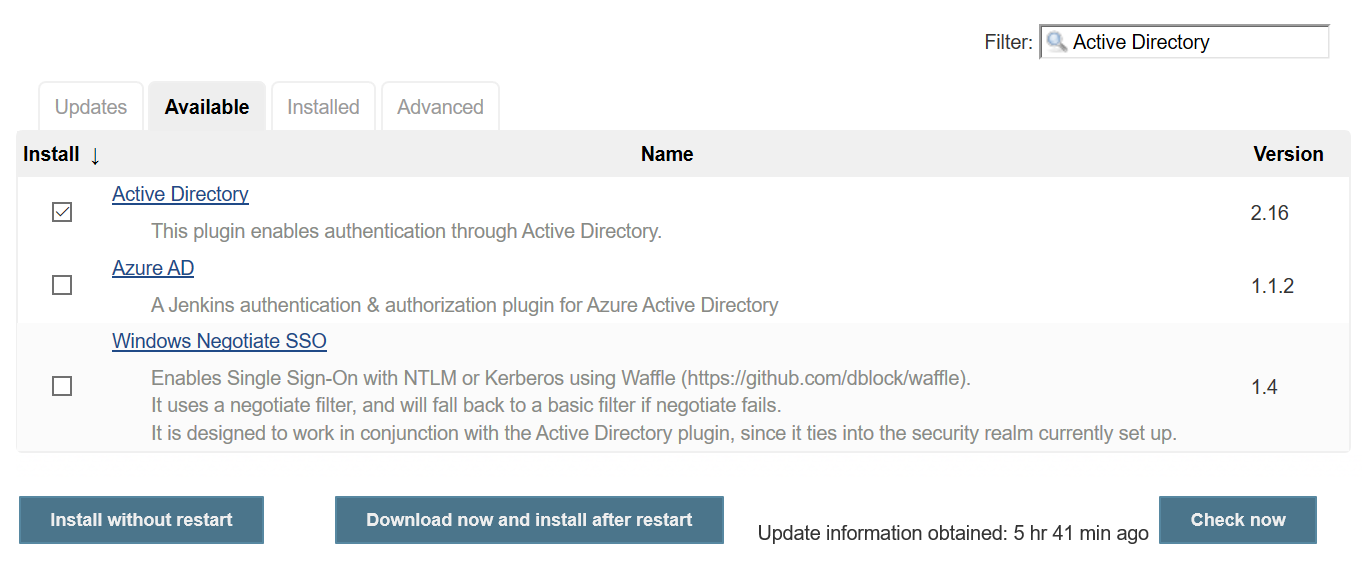


## LDAP Configuration

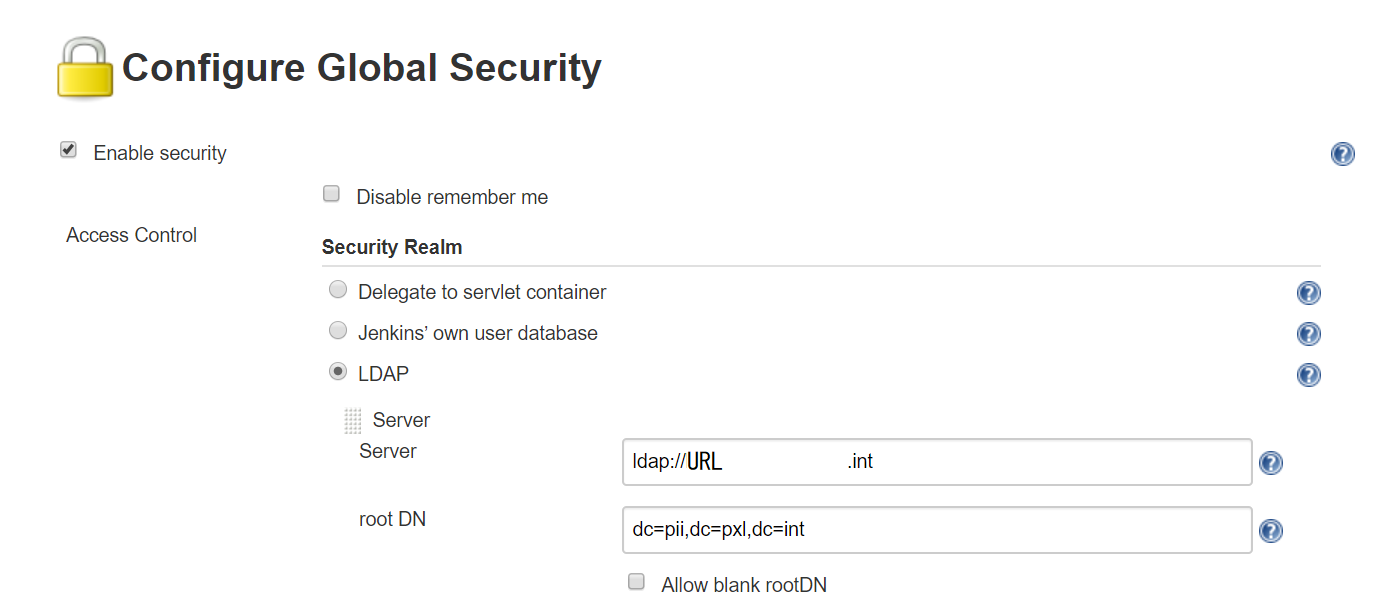
If we configure Jenkins with LDAP Integration, it will allow users to login with their Domain accounts, and manage permissions using the **Role Based Security plugin**.

For doing this we need to install below two plug-ins

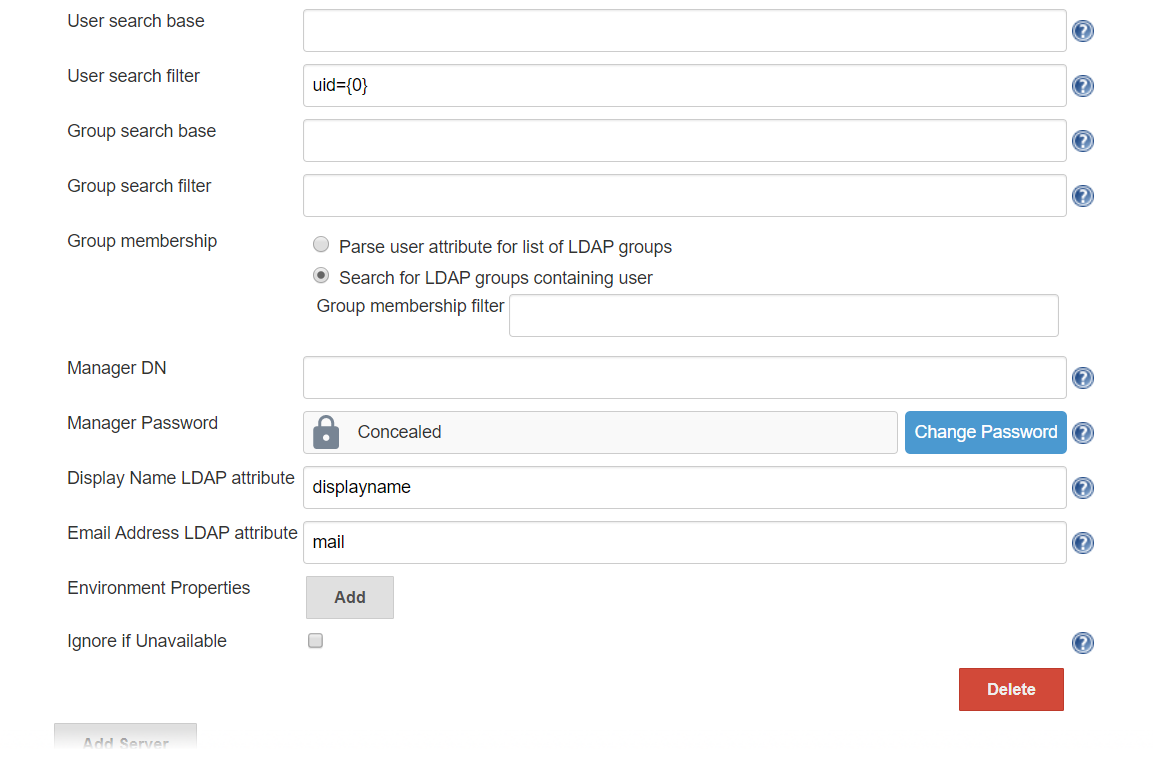
* [**The Active Directory Plugin**](https://wiki.jenkins.io/display/JENKINS/Active+Directory+Plugin)**(**only for AD accounts integration**)**
* [**The Role-based Authorization Strategy Plugin**](https://wiki.jenkins.io/display/JENKINS/Role+Strategy+Plugin)

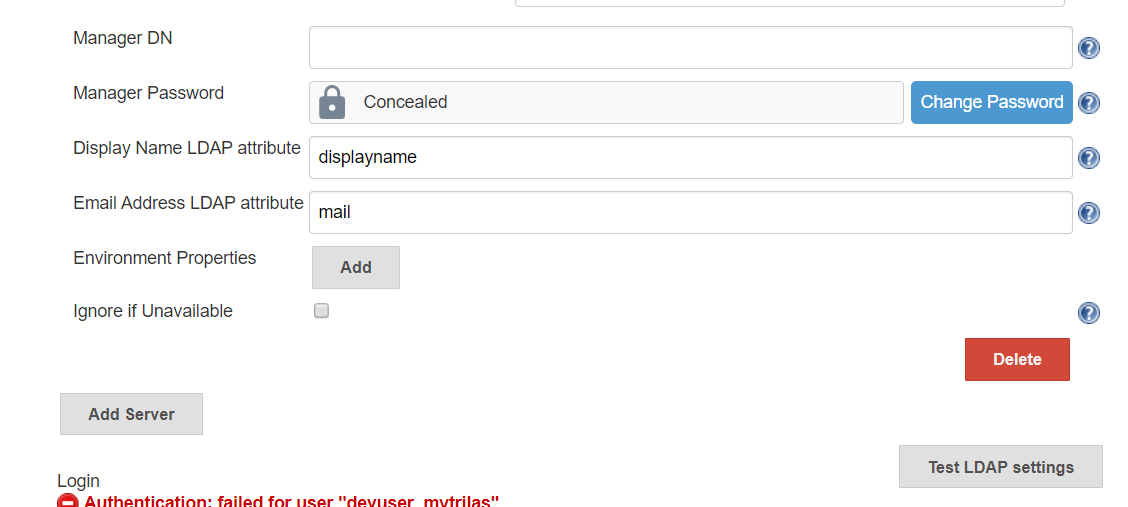


Once above plugins got install follow below steps to move forward.

**Manage Jenkins > Configure Global Security >**Check "Enable Security" checkbox is ticked. 

Under **Security Realm > select LDAP**  radio button

Provide required details & save 

To Test the LDAP Configuration, click on Test LADP settings

## SSO Configuration

To provide SSO Services in our environment, we do use [Keycloak](http://www.keycloak.org/) as the central service. To use this in [Jenkins](http://www.jenkins.io/), we do use the [OpenId Connect Plugin](https://github.com/jenkinsci/oic-auth-plugin).

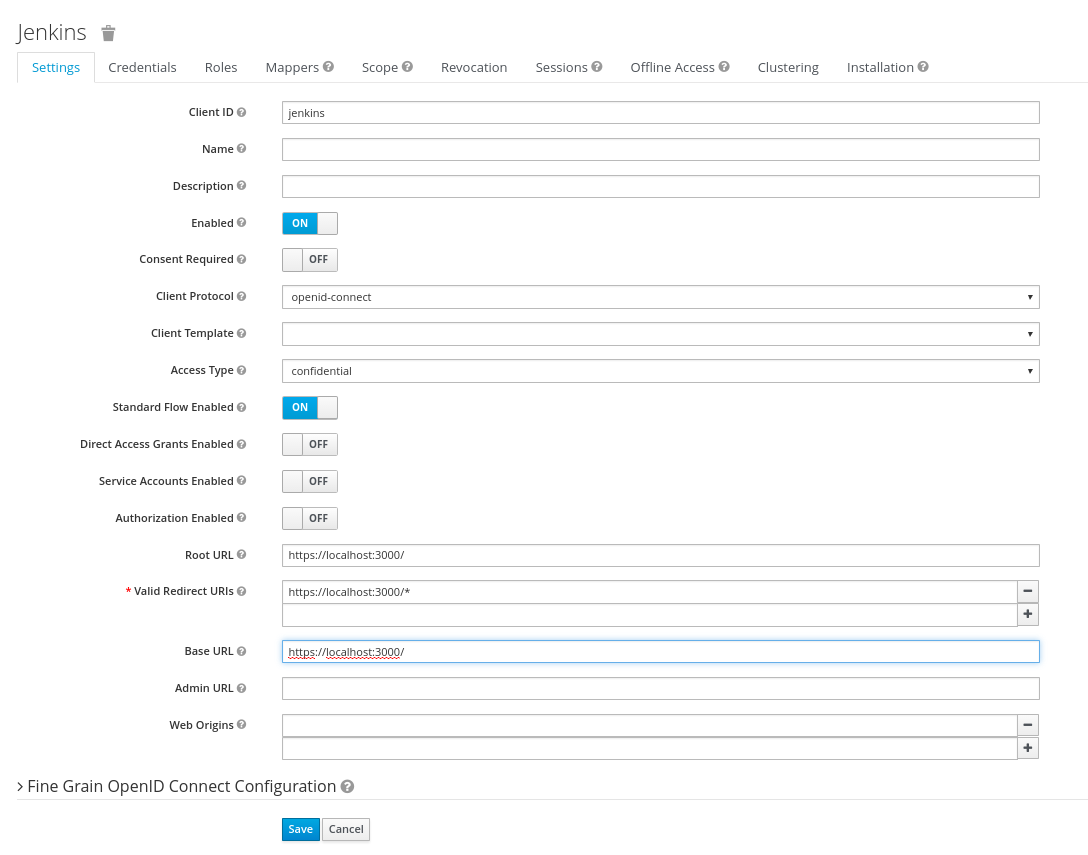
For this to work, a new client has to be created in the Keycloak System, and a couple of endpoints have to be configured in the Jenkins Security Settings.

**Keycloak Settings**

All Settings done in the Keycloak Server are described in this section.

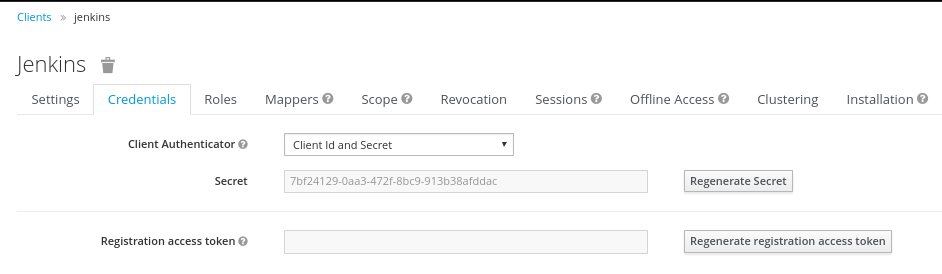
**Keycloak Client**

The Keycloak client needs to get configured in the following way. The Client needs to have a unique name (in this case 'jenkins') and the 'Access Type' needs to get defined as 'confidential'. Please note especially the 'Valid Redirect URIs', which needs to get set to the URL of the Jenkins System (http://loccalhost:3000 in the screenshot).



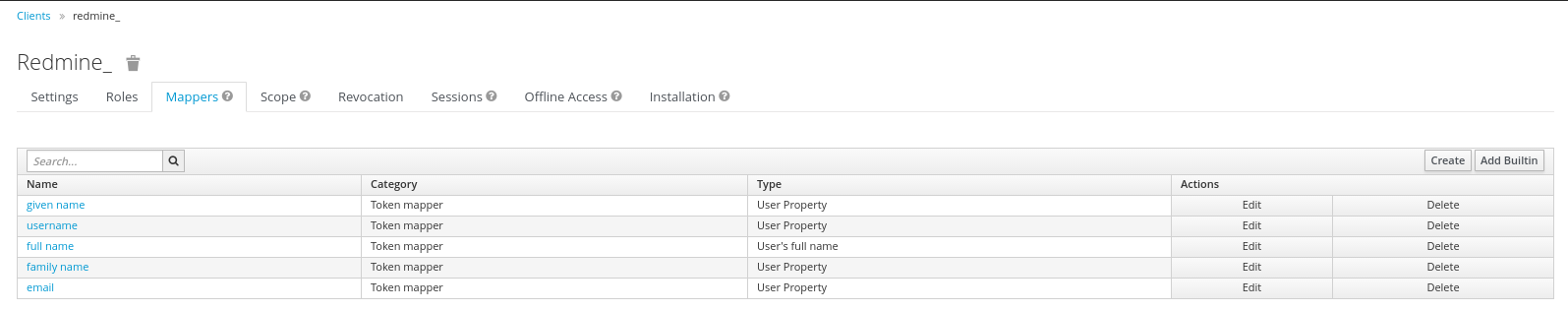
**Keycloak Credentials**

Because we have set the 'Access Type' to 'confidential' we do get offered the Credentials Tab. On this tab, we get a secret, which needs to get put into the corresponding Setting of the Jenkins OpenId Connect plugin. This secret allows to initiate a secure connection between the Keycloak System and your client application (Jenkins in this case).



**Keycloak Mappers**

In order to provide some necessary user information to the Redmine System, the standard Keycloak Mappers have to be adopted and a new Mapper has to get created.



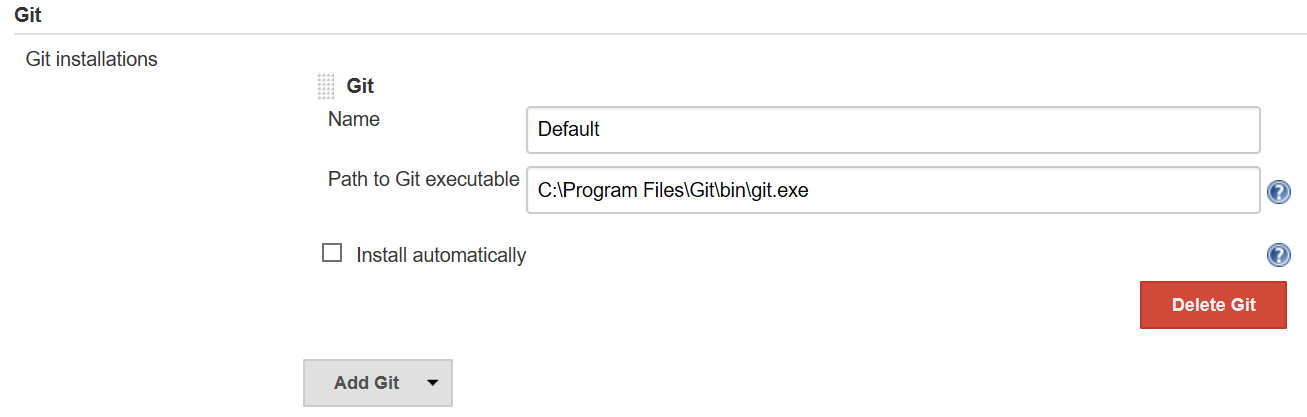
**Jenkins OpenId Connect Settings**

* The Settings done in the OpenId Connect Plugin Settings page are described in here.
* The most relevant settings are the 'Client ID', the 'Client Secret' and the 'Urls'.
* The 'Client ID' is the name of the client in your Keycloak System (jenkins in our case). By Specification, this should be a URI, but a plain name is working as well, and seems to be easier IMHO.
* The 'Urls' have to be set to the Keycloak Server with the path /auth/realms/. Please adopt the REALMNAME (devopskube in the picture), to your own needs.
* The 'Client Secret' is the secret found in the Keycloak Credentials page (see above), and should be copied from there.

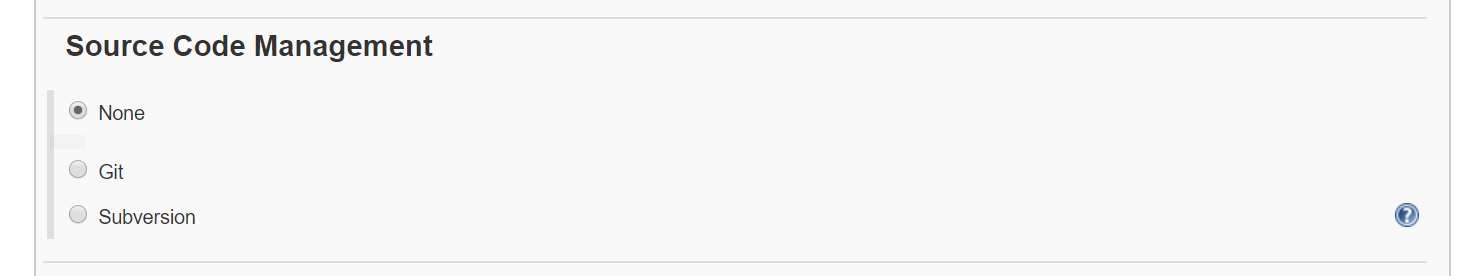


# Jenkins – Common Configurations

### Install & Configure Git

Jenkins dashboard > Manage Jenkins > Global Tool Configuration > Git : Provide valid git.exe path 

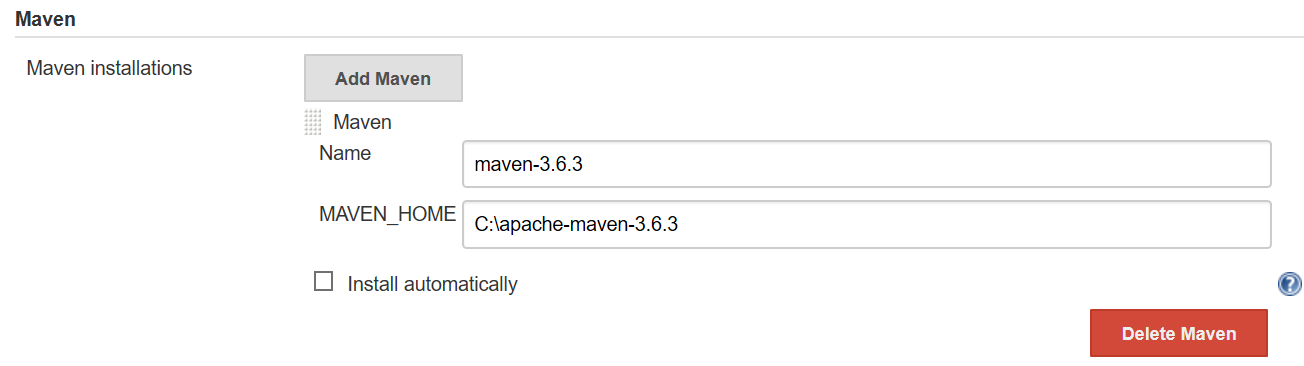
Jenkins Dashboard > Manage Jenkins > Manage Plugins >Filter: **Git plugin** > Install without restart

After Jenkins is restarted, Git will be available as an option whilst configuring jobs

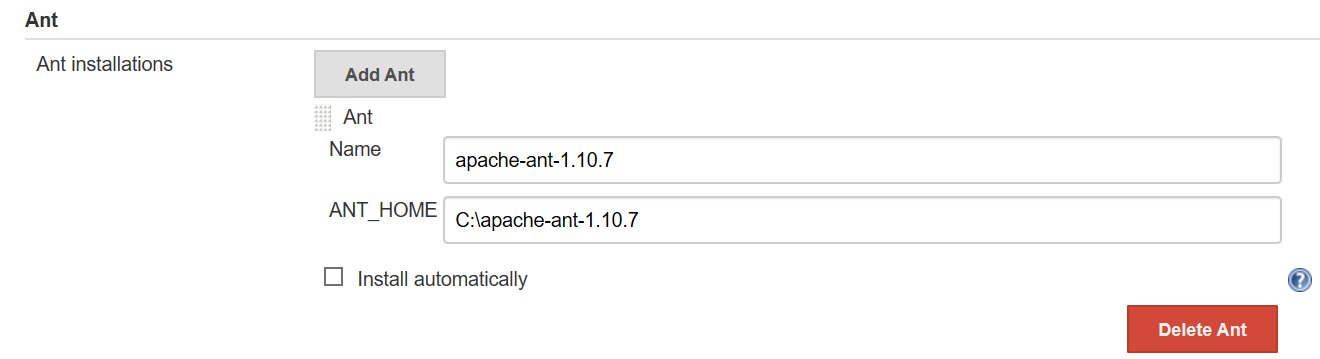
### Install & Configure Java

Jenkins dashboard > Manage Jenkins > Global Tool Configuration >Java section :"Add JDK" 

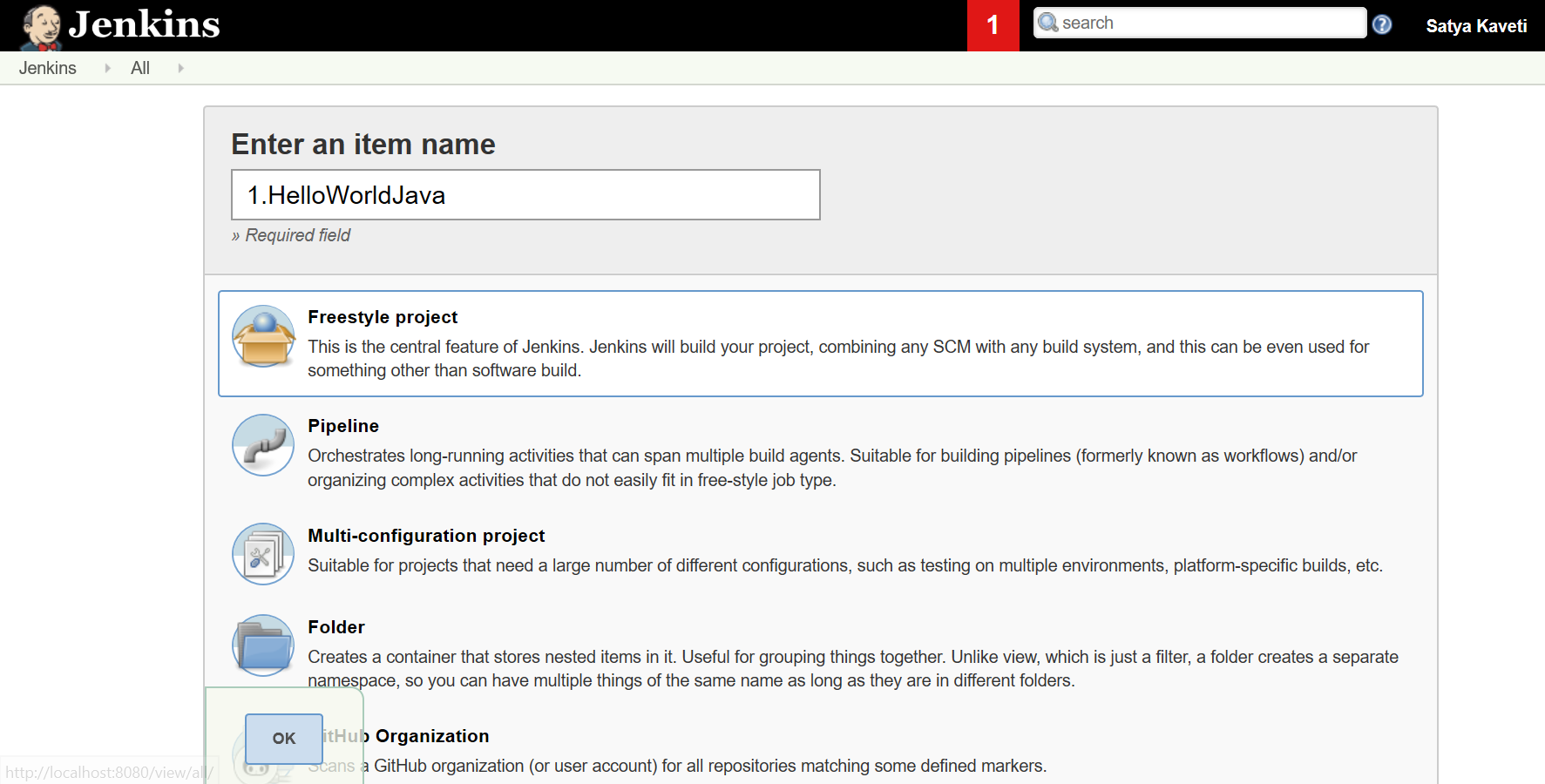
### Install & Configure Maven

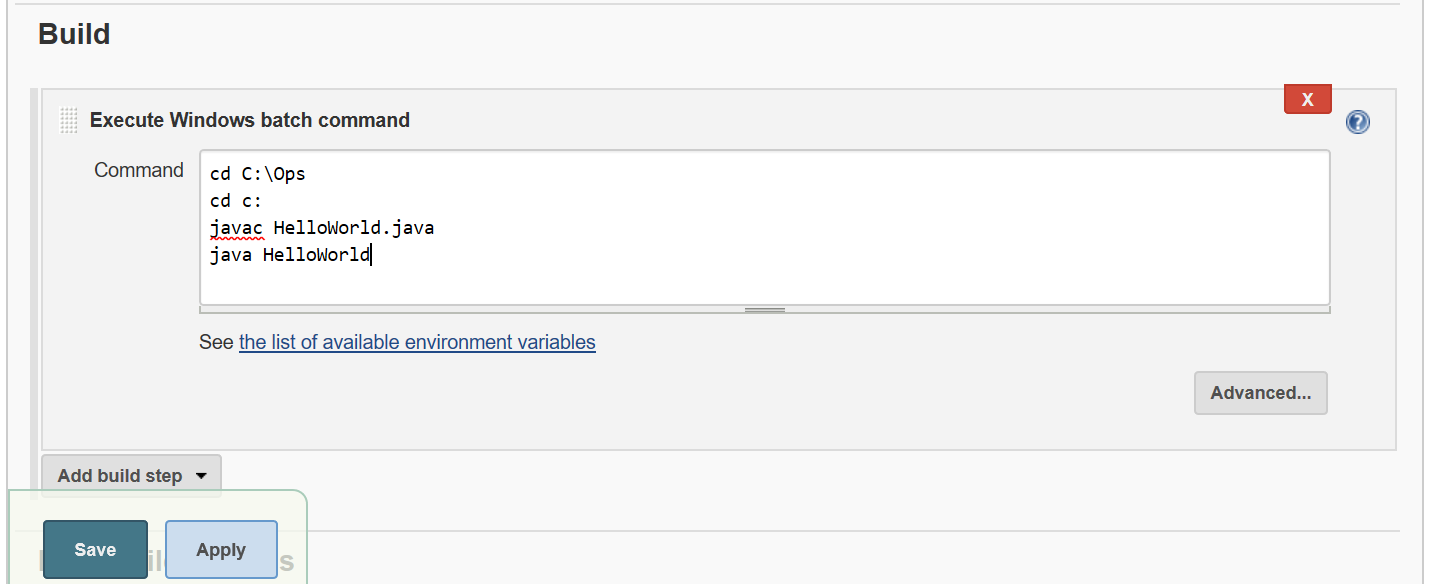
Jenkins dashboard > Manage Jenkins > Global Tool Configuration >Maven section :"Add MAVEN" 

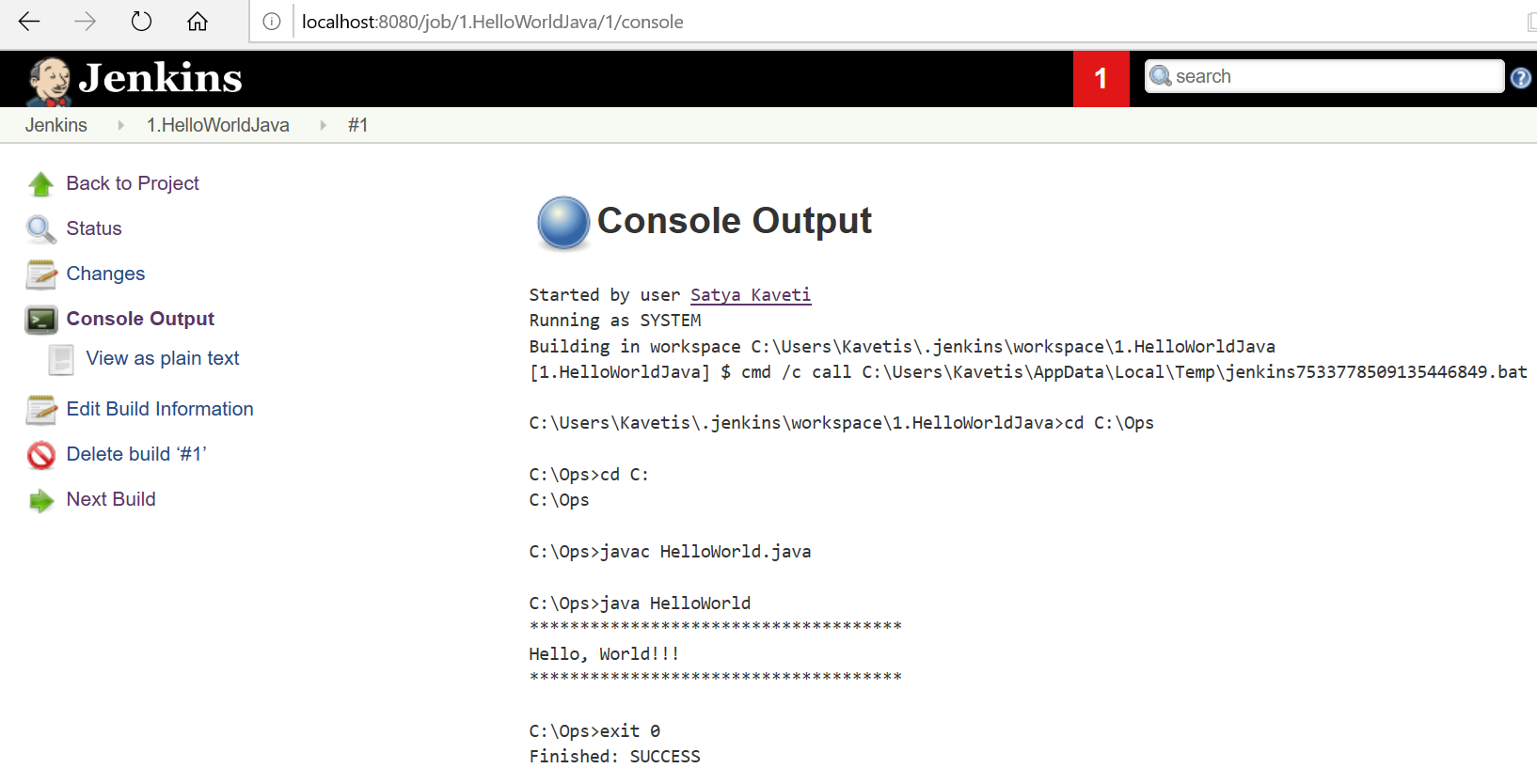
### Install & Configure Ant

Jenkins dashboard > Manage Jenkins > Global Tool Configuration >Ant section :"Add ANT" 

# Jenkins – Java Hello world example

Jenkins Dashboard > New Item > Freestyle Project

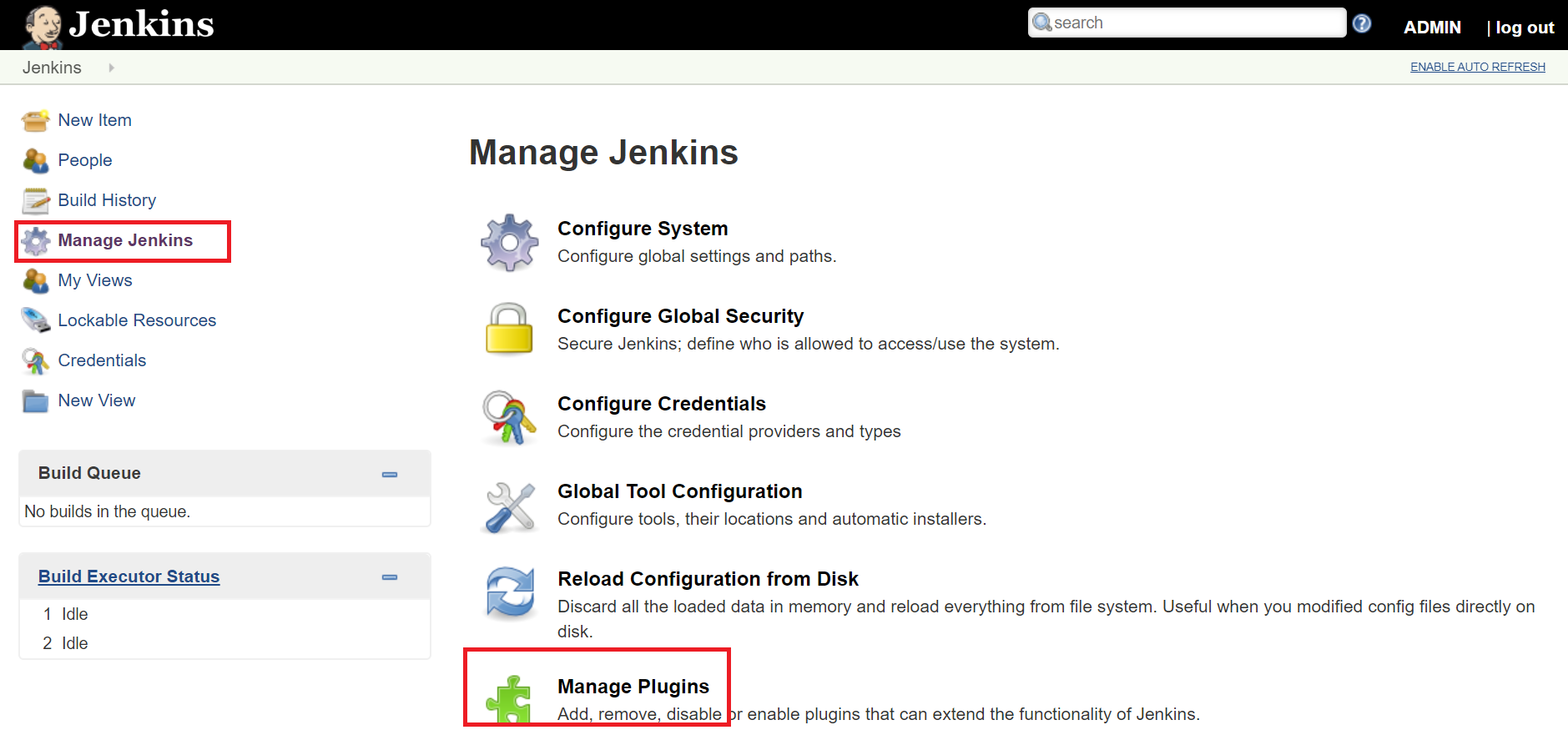
Build Tab > Just add below Windows batch commands to build & run local java files

Click on Build Now & check Console output

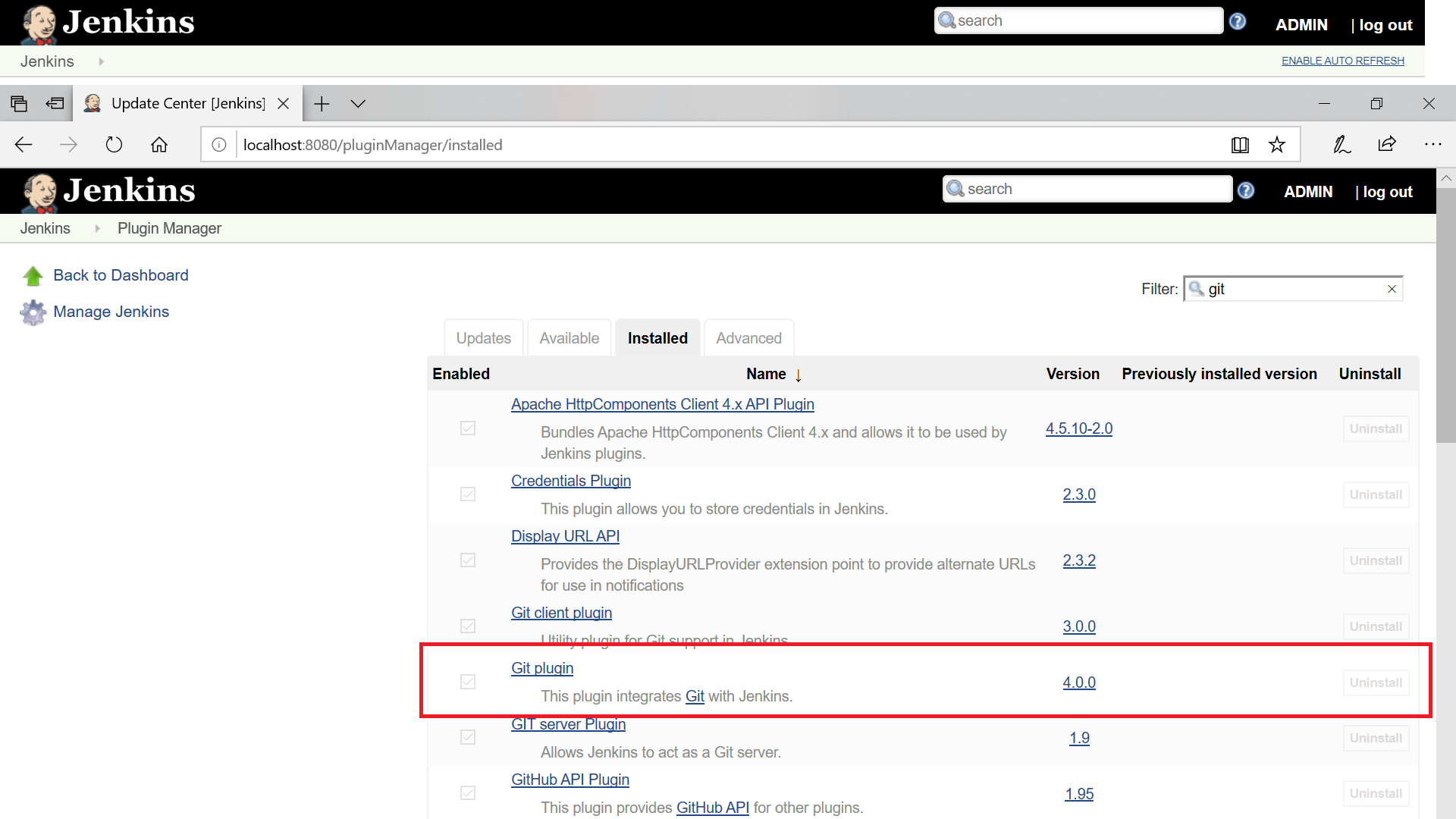
# Jenkins – GitHub Integration

Jenkins is a Continuous Integration Tool; it needs to check out source code from a repository and build code.

### Install GitHub Plugin

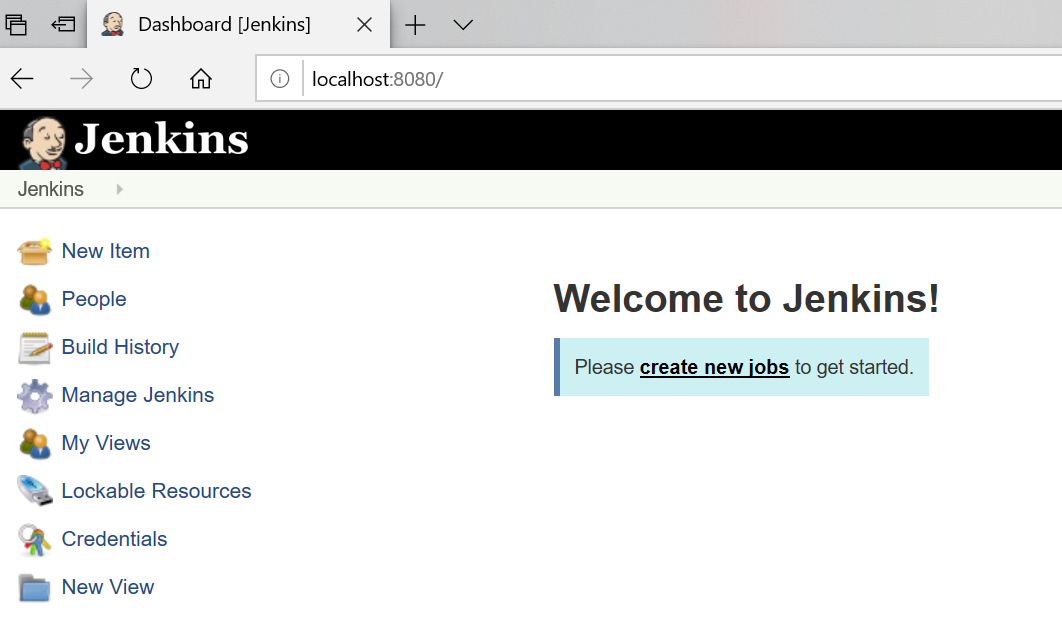
Jenkins Dashboard> Manage Jenkins> Manage Plugins. 

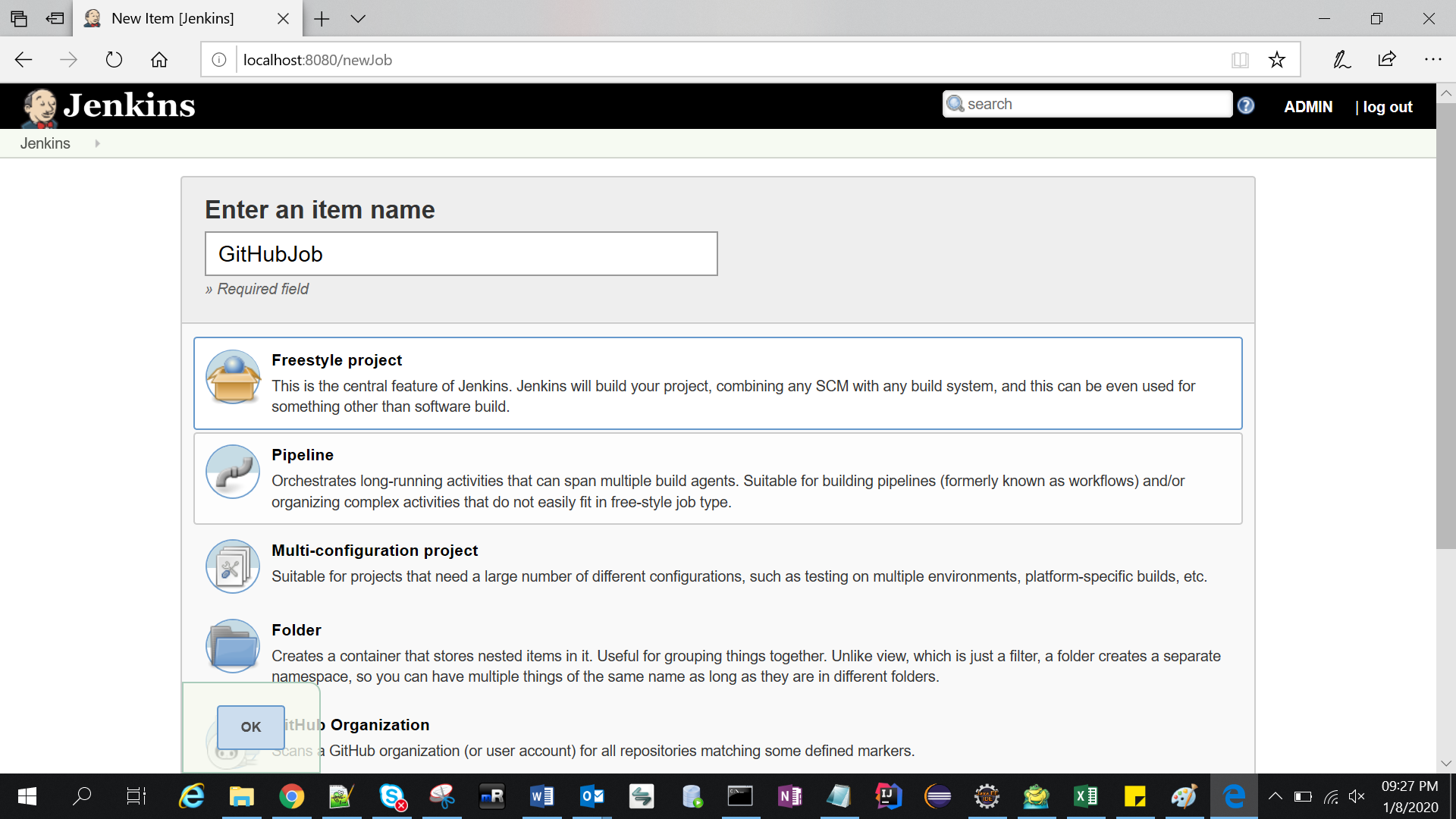
Next page – click on the "Available tab”. Search for "Git Plugin", Install it.

Installed plug-ins can be shown in "installed" tab. 

### Integrating Jenkins with GitHub

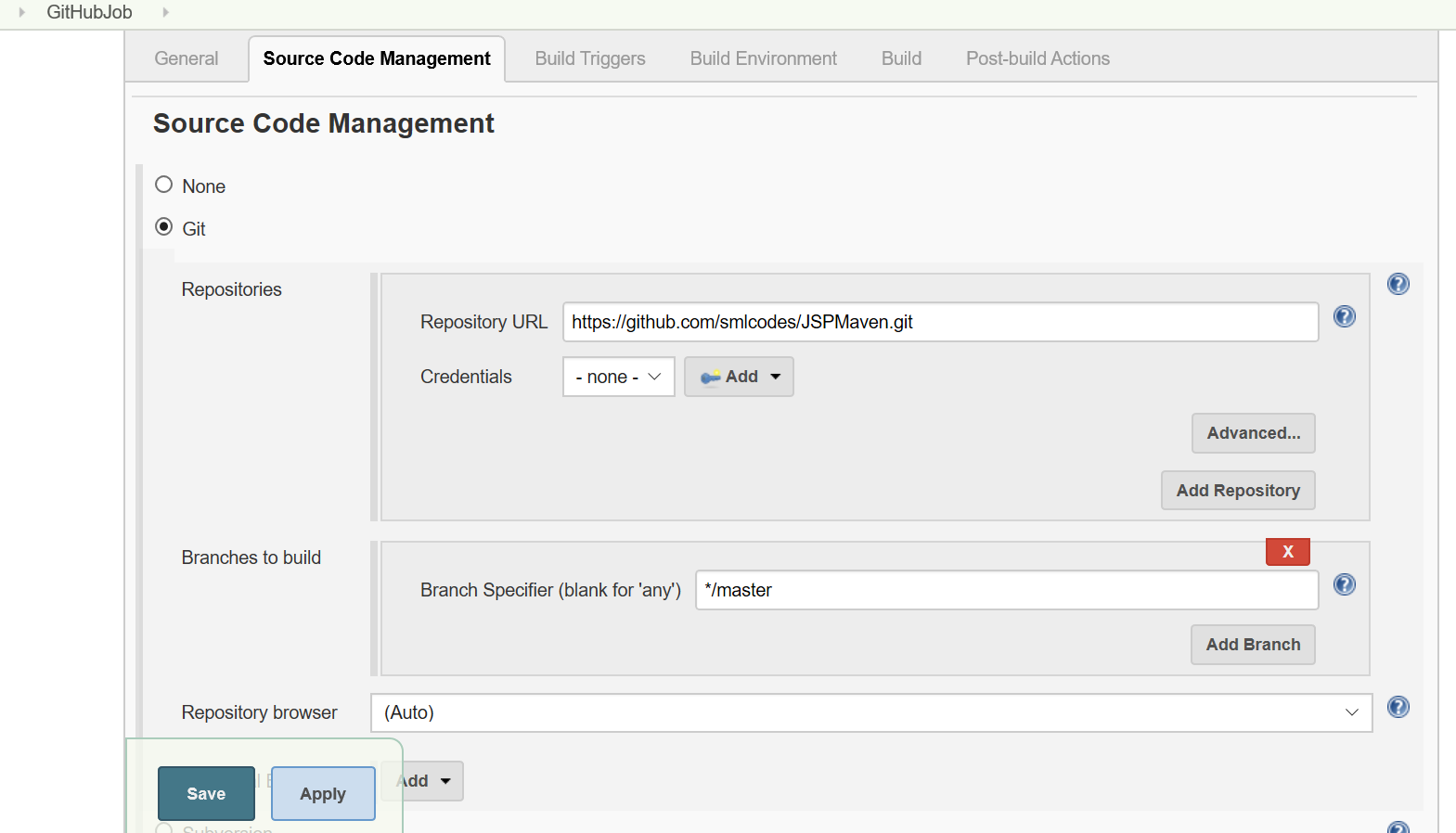
We must have Git installed in your system

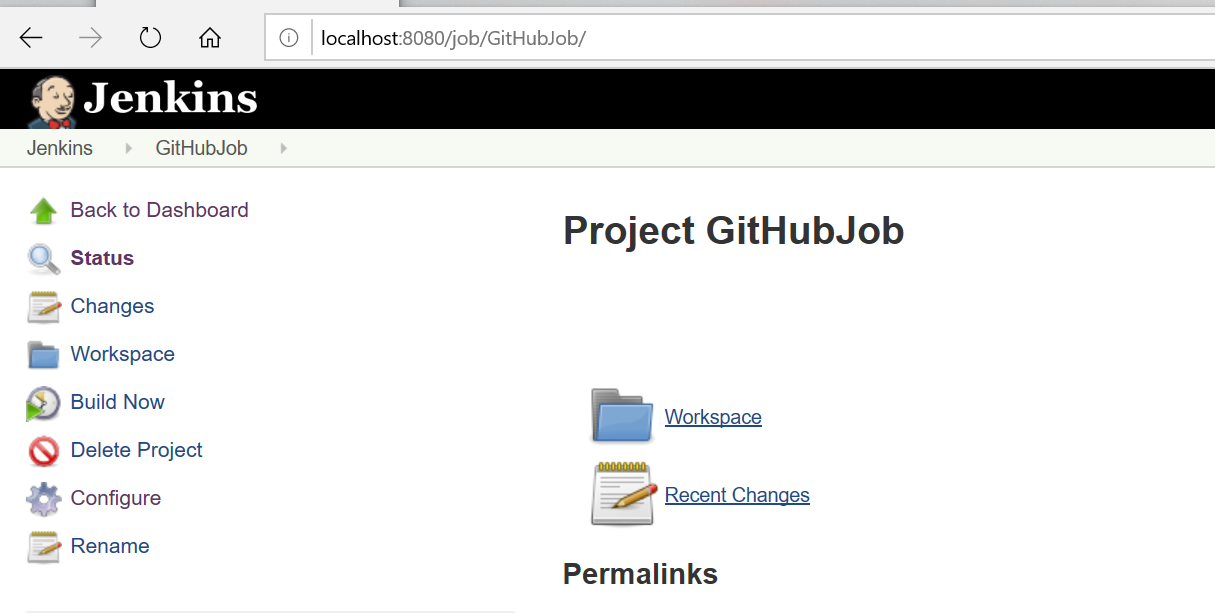
Jenkins Dashboard > Create new job

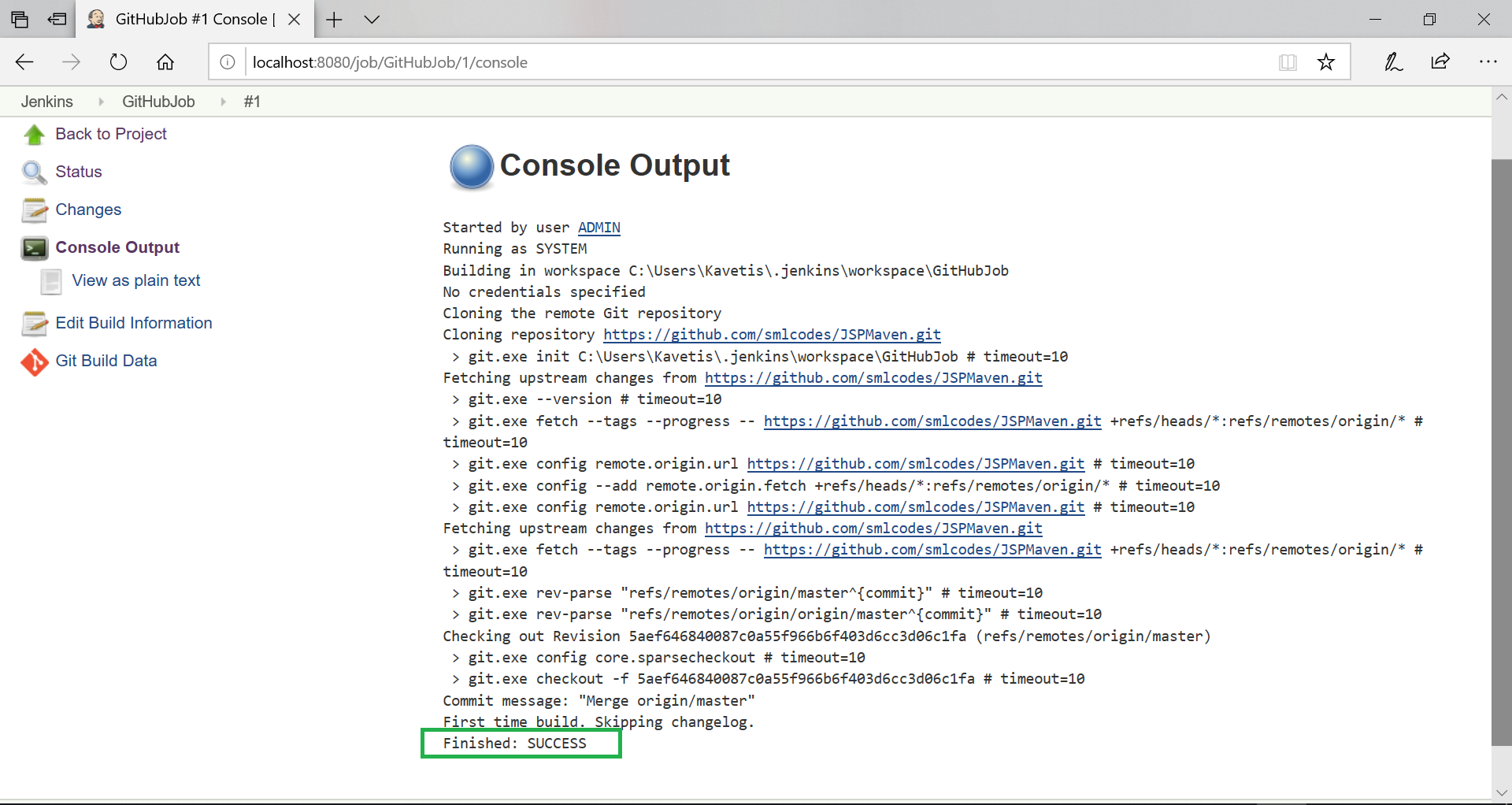
Now enter the item name and select the job type. For example, item name is "GithubJob" and job type is "Freestyle project". Then OK

Go to “Source Code Management” Tab > Select Git > Provide Github Repo URL >Save / Apply

Example : <https://github.com/smlcodes/JSPMaven.git>



To Check Integration just Build it. 

If build Success, it is integrated propetly. 

# Jenkins – Maven Project Integration

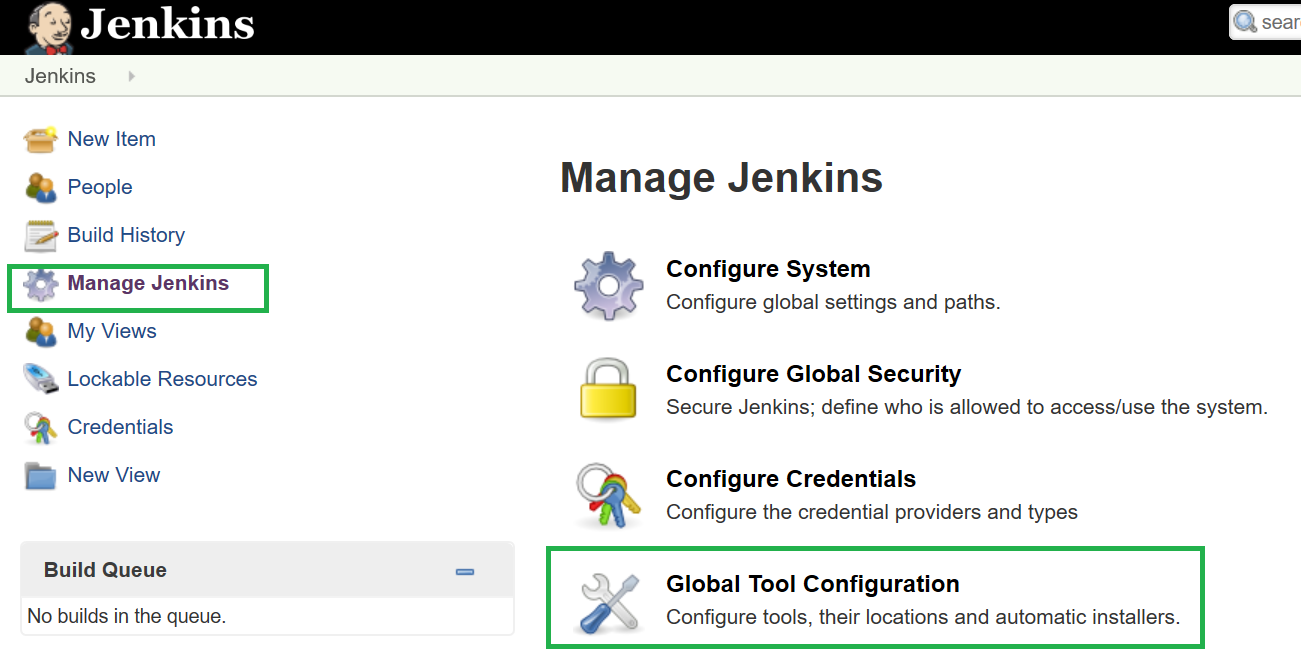
Make sure Maven is installed in your system

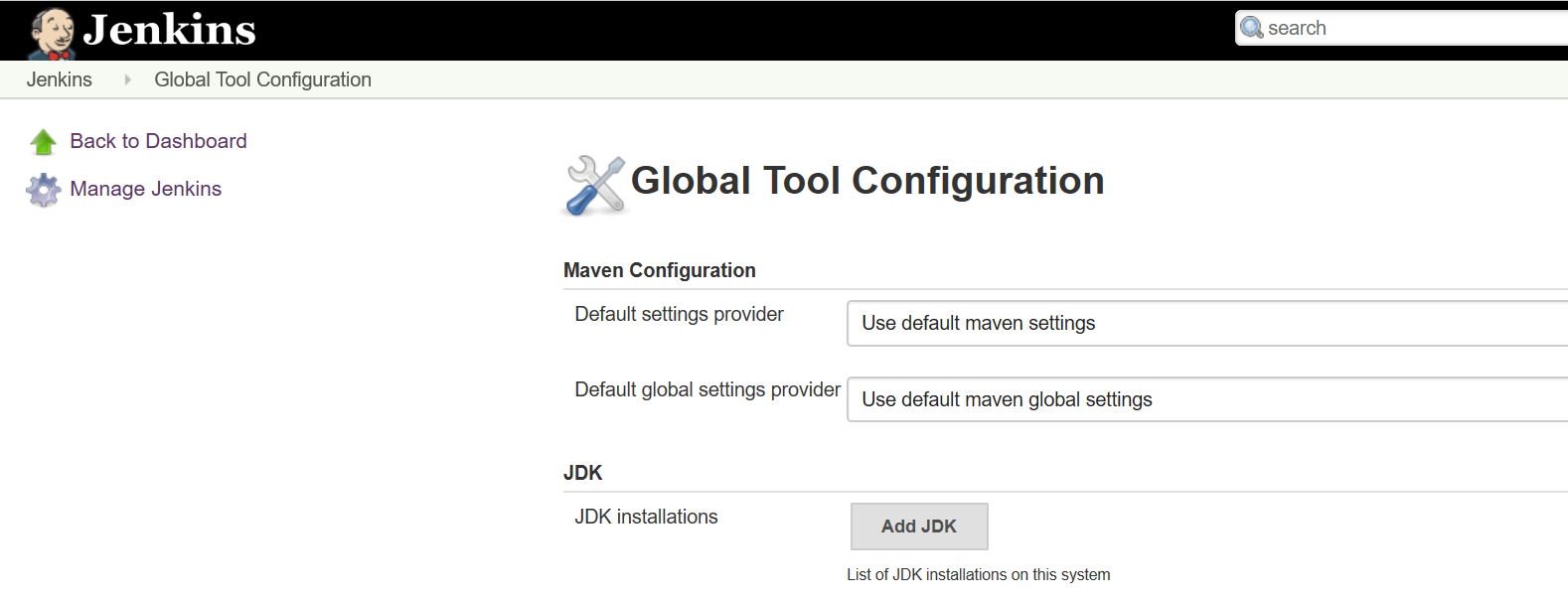
* Downloading Maven - <https://maven.apache.org/download.cgi>
* Add a MAVEN\_HOME system variables and point it to the Maven folder.
* Check Maven Installation by Running "**mvn –version**" on command line

### Configure Maven in Jenkins

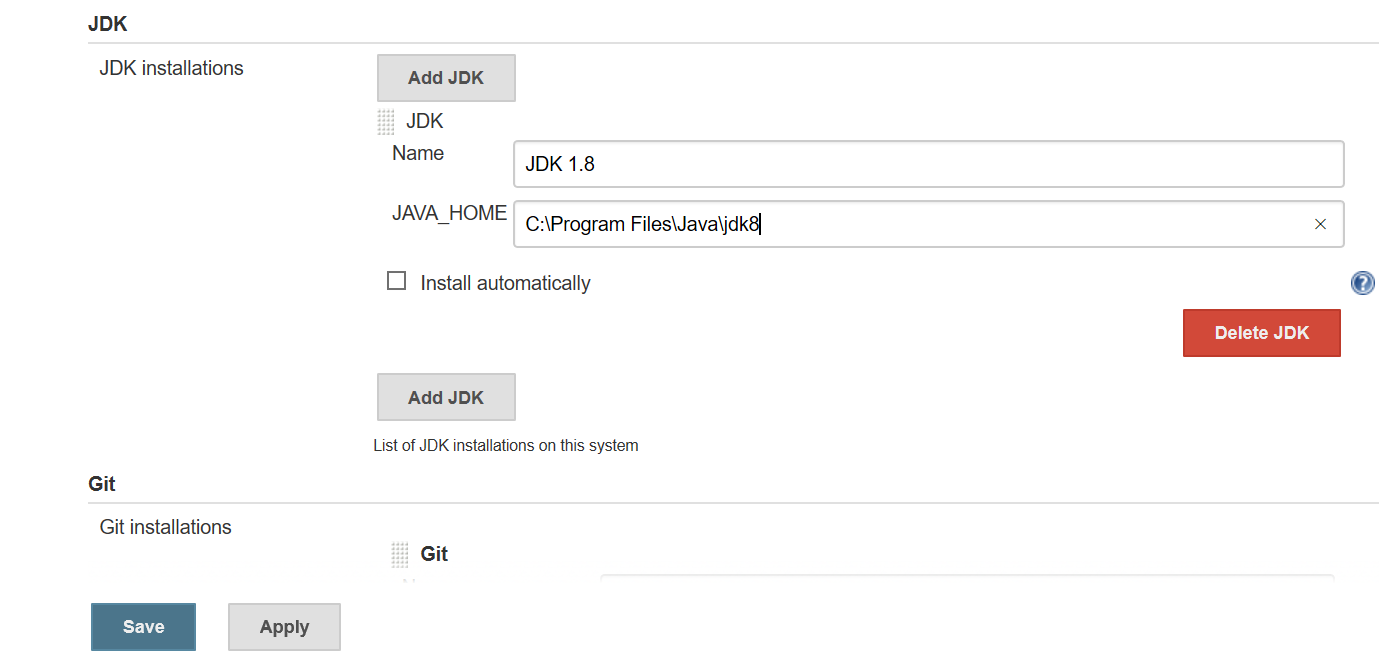
Install below plug-ins in Jenkins

* Maven Integration
* Maven SNAPSHOT Check

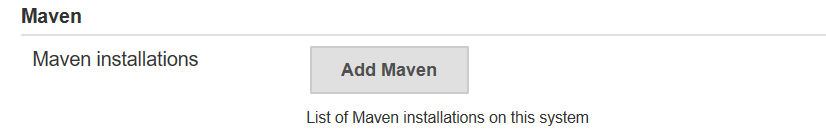
Jenkins Dashboard > Manage > Global Tool Configuration

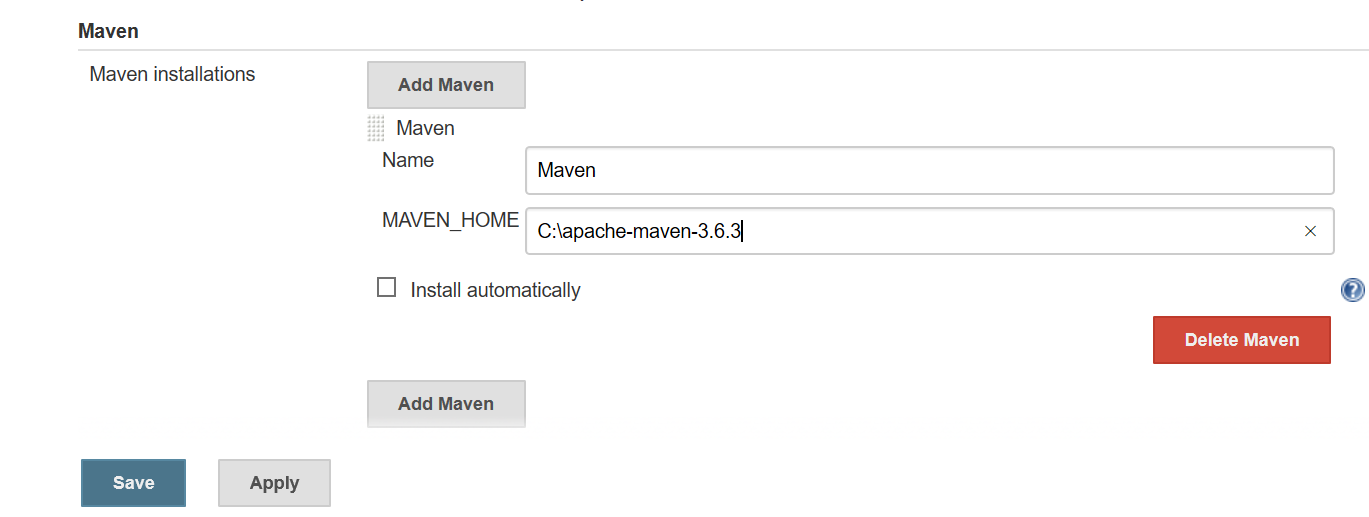
To configure Java, click on "Add JDK" 

Provide Name, JAVA\_HOME - Provide Java Home Path – save



To configure Maven, click on "Add Maven" , MAVEN\_HOME- Provide Maven Home Path

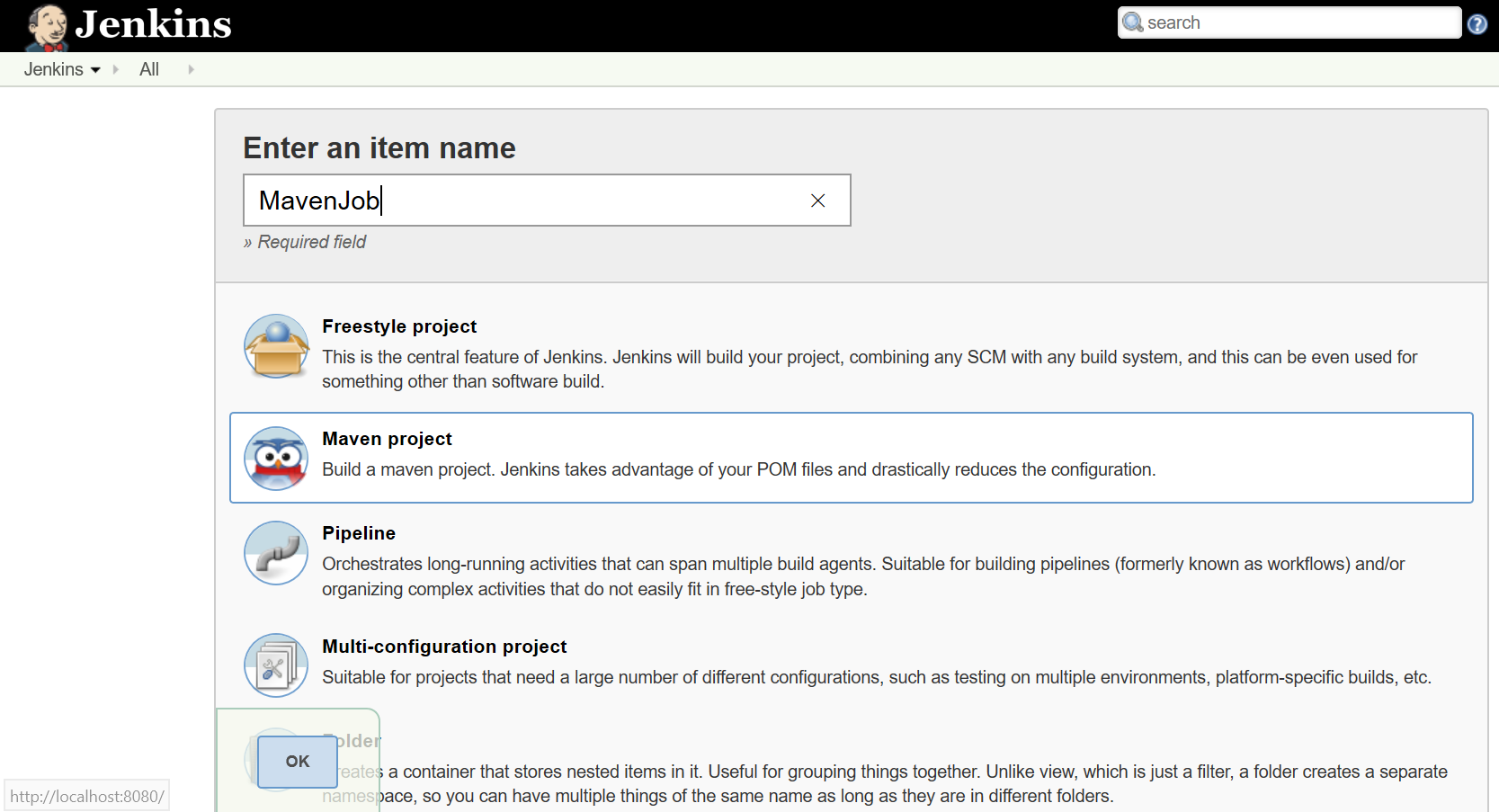




### Jenkins –Creating Maven Job

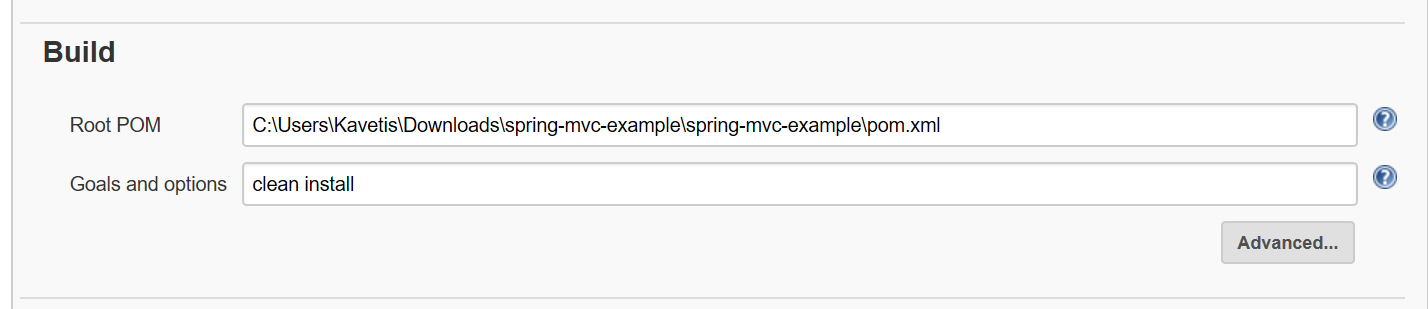
I have a sample maven project in my local system at **C:\Users\Kavetis\Downloads\springmvcexample**”. To build this project with Jenkins follow below steps.

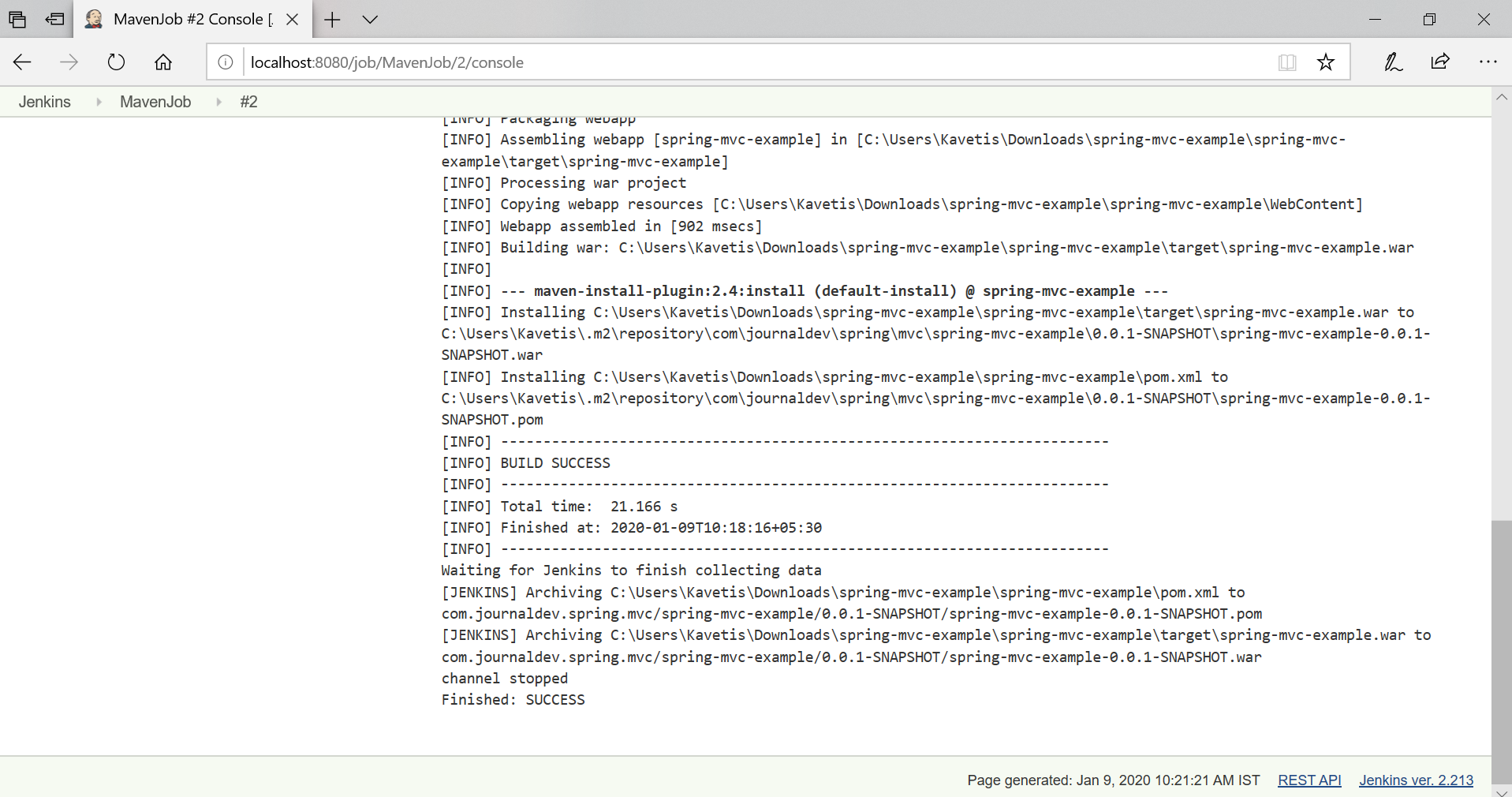
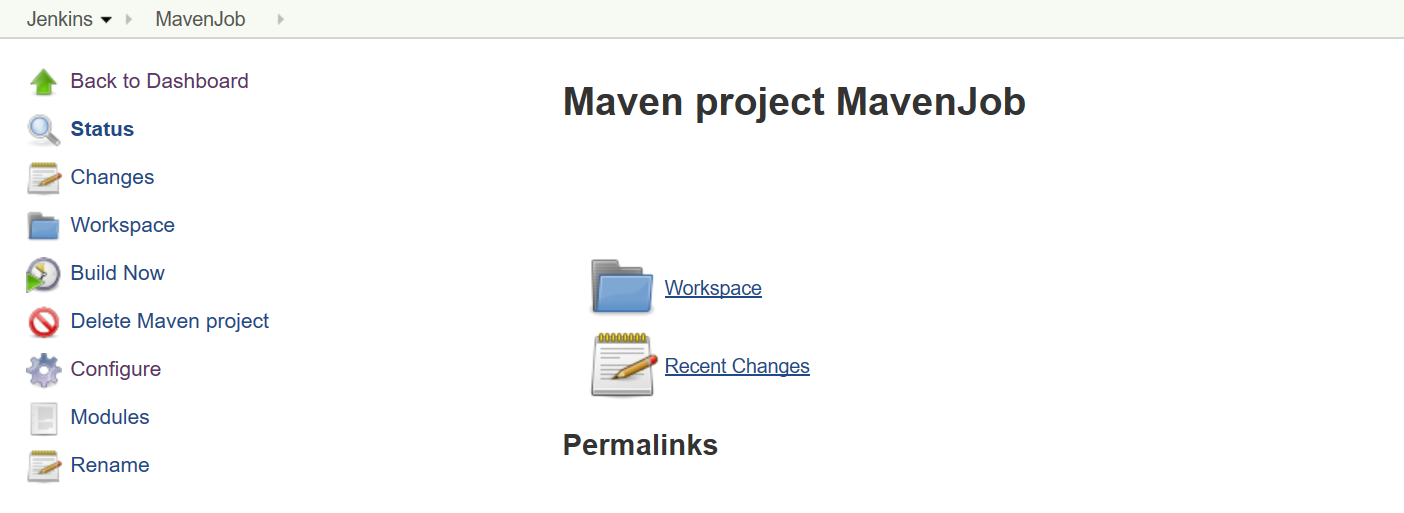
Make sure **Maven Integration** plugin is installed.

Click on the New Item link to create a CI job. & Select the Maven project radio button

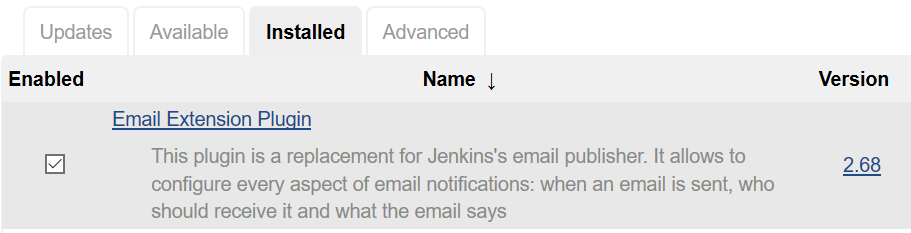
Go to the Build section of new job.

* In the Root POM textbox, enter full path to pom.xml
* In Goals and options section, enter **clean install** (***without mvn***)& Save/Apply



click on the **Build Now** link. 

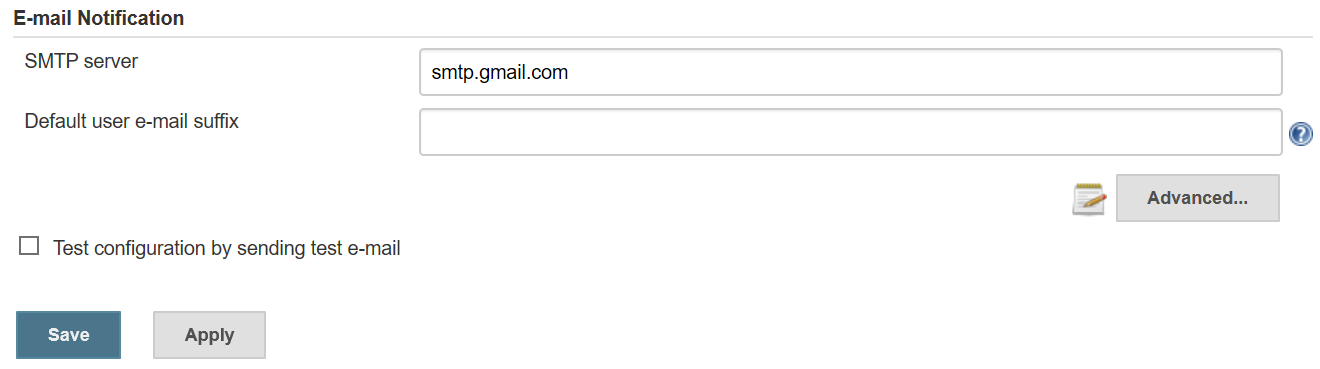
# Jenkins -Email Notification

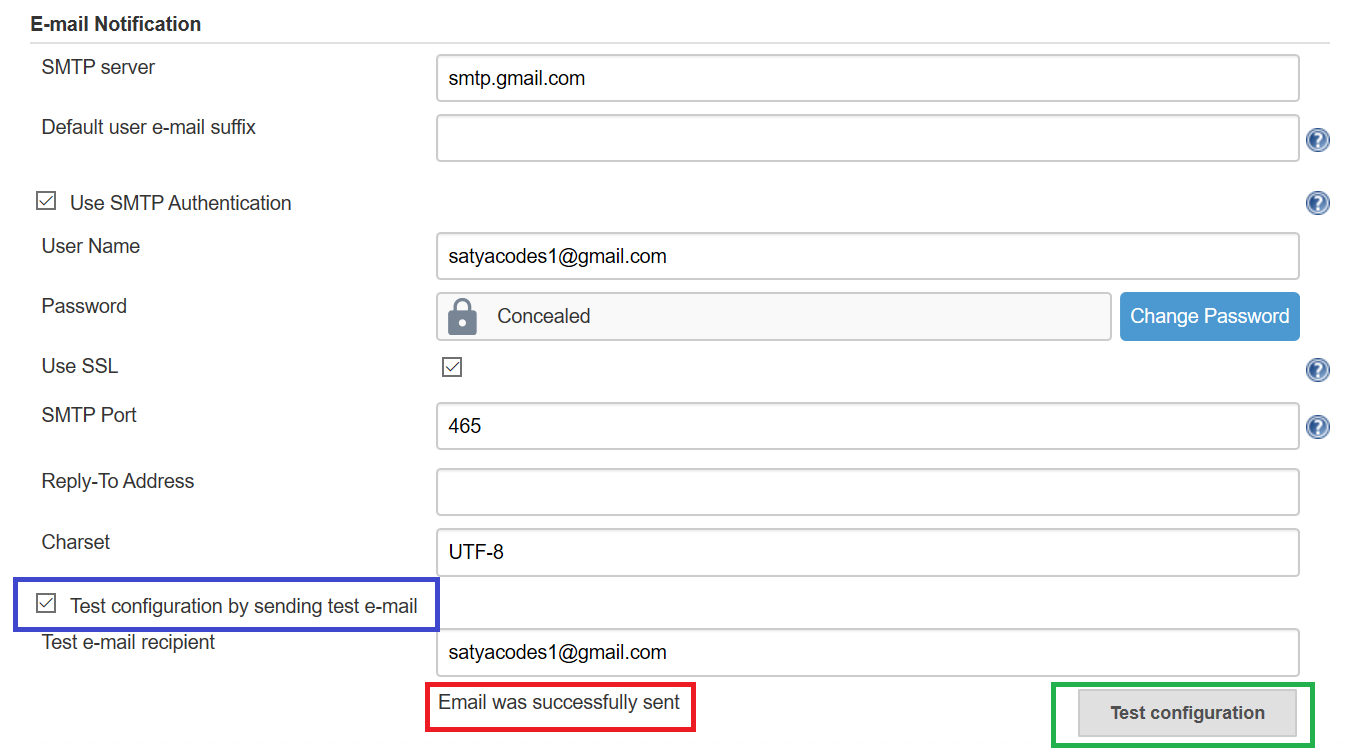
Manage Jenkins > Manage Plugins > install "[**Email Extension Plugin**](https://plugins.jenkins.io/email-ext)" and "Email Extension template Plugin". 

**Manage Jenkins > Configure System > E-mail Notification**

Provide SMTP Server Details. for Example, here I’m using Gmail, for that configure below things

Gmail SMTP server name: smtp.gmail.com



Check [ ] - **Use SMTP Authentication** & provide below details 

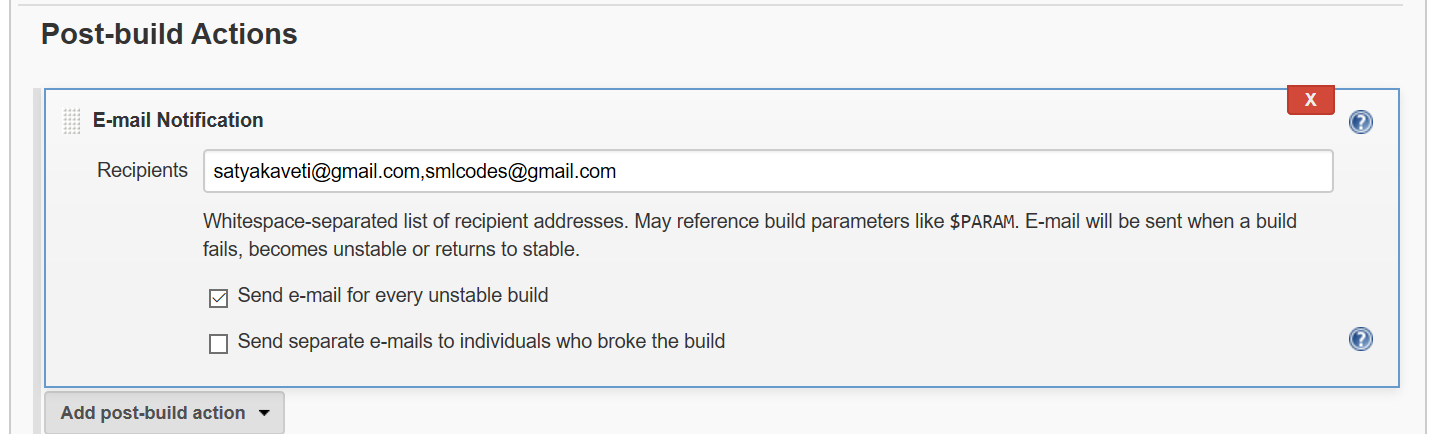
Save & Test Email

**For any Issues, please do below things**

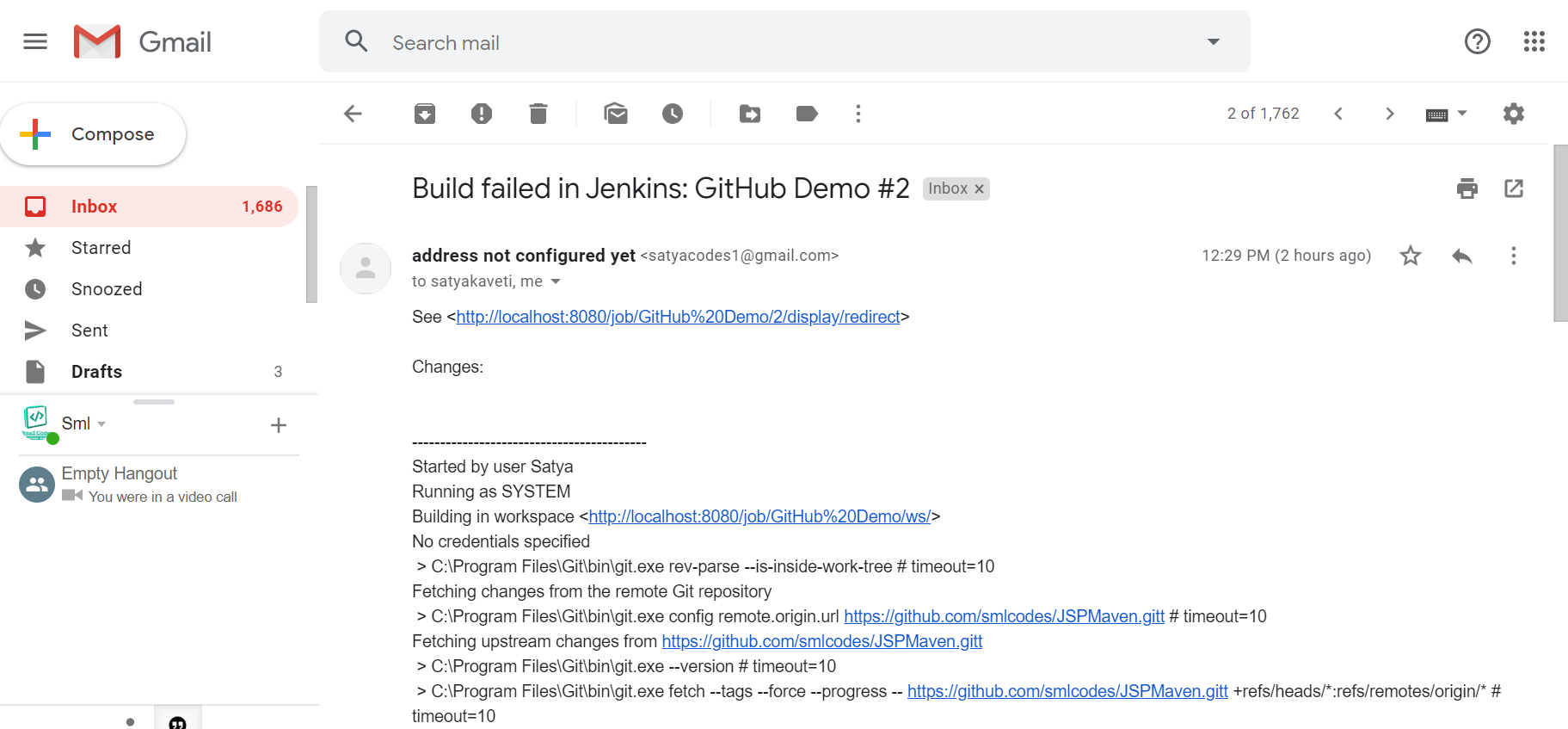
* Update mailer plugin to 1.1 in Jenkins. Update jenkins to latest version(optional )
* In Google Account Settings : Allow less secure app access: click <https://www.google.com/settings/security/lesssecureapps> and select Turn on.
* Unlock captcha: click <https://accounts.google.com/DisplayUnlockCaptcha> and select continue

### Configure mail notification on Build failure/Success

Open already Created Job > **configure option**.

**Post-build actions > E-mail Notification** section. Enter the receipt email id and select **'send e-mail for every unstable build**' option. 

Try to fail the Build. See Console output & Check mentioned emails 

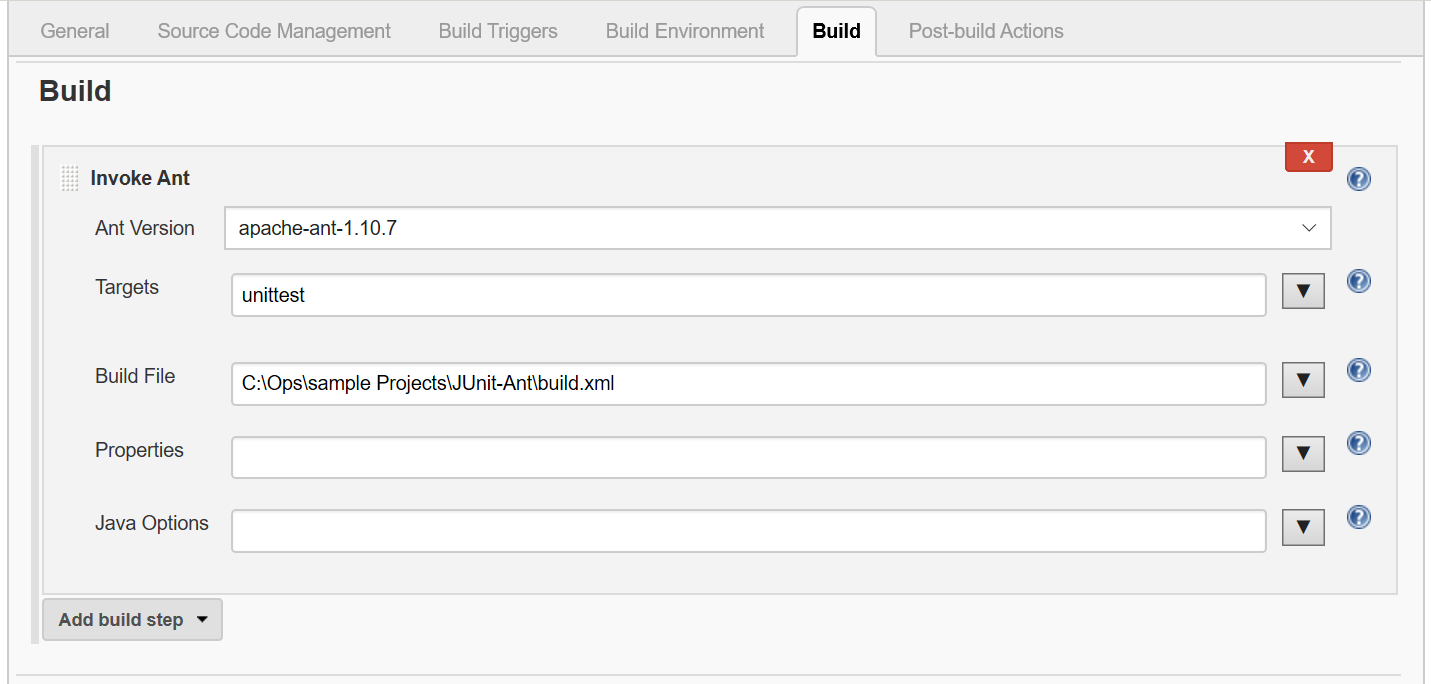


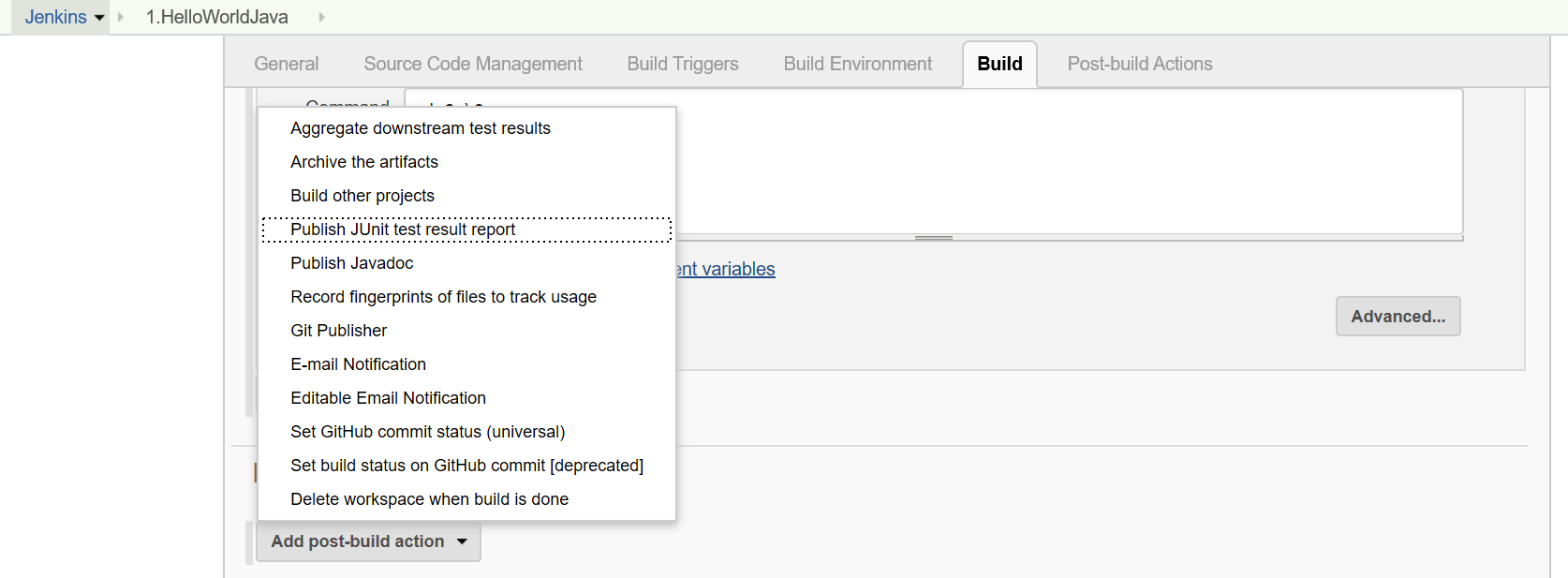
# Jenkins Testing ###

## Jenkins – Junit Reporting using ANT

Create Job > Free Style

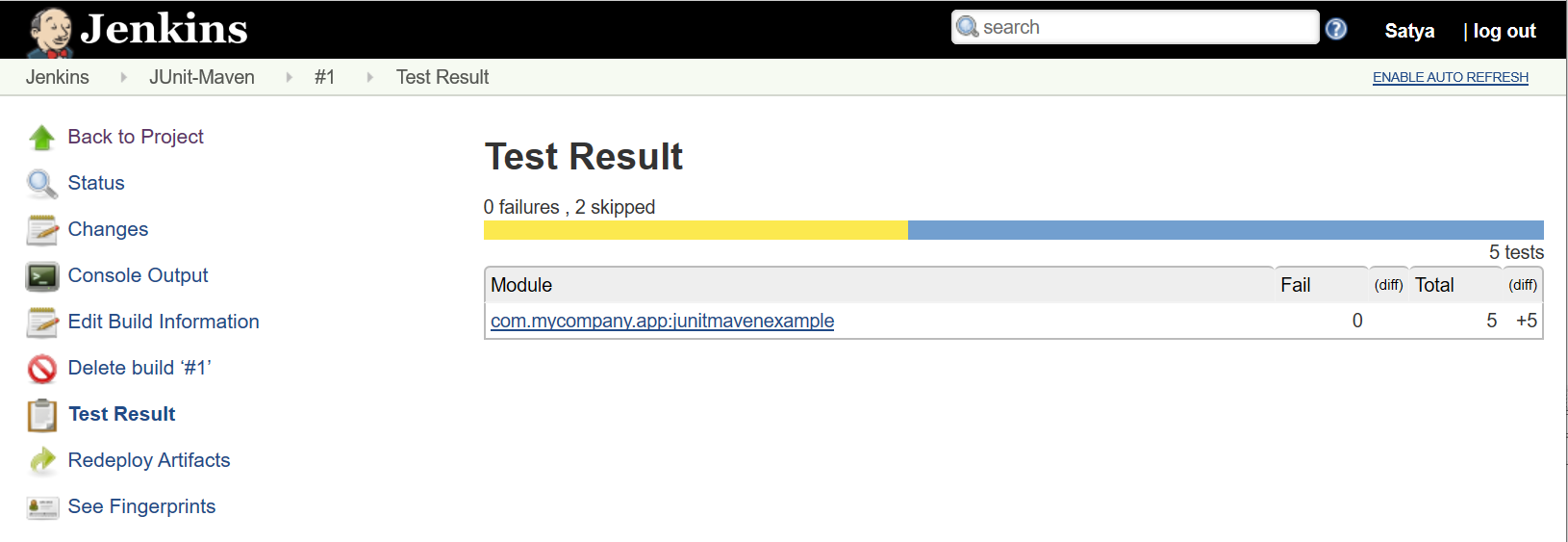
**Build >Add Build Step> Invoke Ant :** provide below details



Next, **Post-build Step > choose “Publish Junit test result report”** 

Test report XMLs - location of reports xmls



* For maven projects it will automatically generates the JUnit Reports. 
* For selenium, we need to add **“Hudson Selenium Plugin”**

## Jenkins – Selenium Integration

* Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass.
* Jenkins can schedule your tests to run at specific time.
* You can save the execution history and Test Reports.
* Jenkins supports Maven for building and Testing a project in continuous integration

<https://www.guru99.com/maven-jenkins-with-selenium-complete-tutorial.html>

Dashboard > Manage Jenkins > Manage Plugins - Install "**Hudson Selenium Plugin**"

Manage Jenkins > Configure system > Selenium > Provide "selenium-server-standalone-xxx.jar" PATH

Add build step and choose the optin of “SeleniumHQ htmlSuite Run”

# Jenkins – Master & Slave

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

**Jenkins Master**

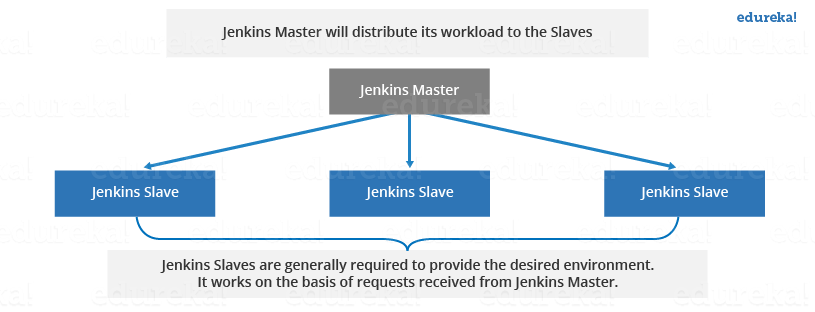
Your main Jenkins server is the Master. The Master’s job is to handle:

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

**Jenkins Slave**

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

* It hears requests from the Jenkins Master instance.
* Slaves can run on a variety of operating systems.
* The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
* You can configure a project to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.



Jenkins master/slave architecture is used for distributed build environments, where the workload of building projects is distributed to multiple agent nodes, and we can use different environments for each build.

Jenkins master node will be used for scheduling jobs, monitoring slave nodes, dispatching builds to slave nodes, recording and representing the build result, and also executing build jobs directly.

Jenkins slave nodes can run on a variety operating system like Windows and Linux, and there is no need to install full Jenkins packages on it.

Master -Windows 192.168.33.1

Slave1 -Ubuntu 192.168.33.10

Slave2 -CentOS 192.168.33.11

There are two ways of authentication for setting up the Jenkins slaves. [**Ref**](https://devopscube.com/setup-slaves-on-jenkins-2/).

1. Using username and password
2. Using ssh keys.

## Using Username & password

We will go for this scenario where

* Master – Windows
* Slavers – Linux Systems

### In Slave Nodes

1.Install Java on Node machines

**2.Create user**

We need to create a user on Slave Machine, Master node will use the credentials to make connection with slave/agent nodes. Here The new user will be called jenkins with new directory /var/lib/jenkins as home directory

# Create new Directory for Jenkins user

mkdir /var/lib/jenkins

# Change Permiisnons

sudo chmod -R 777 /var/lib/jenkins

# Create Jenkins user

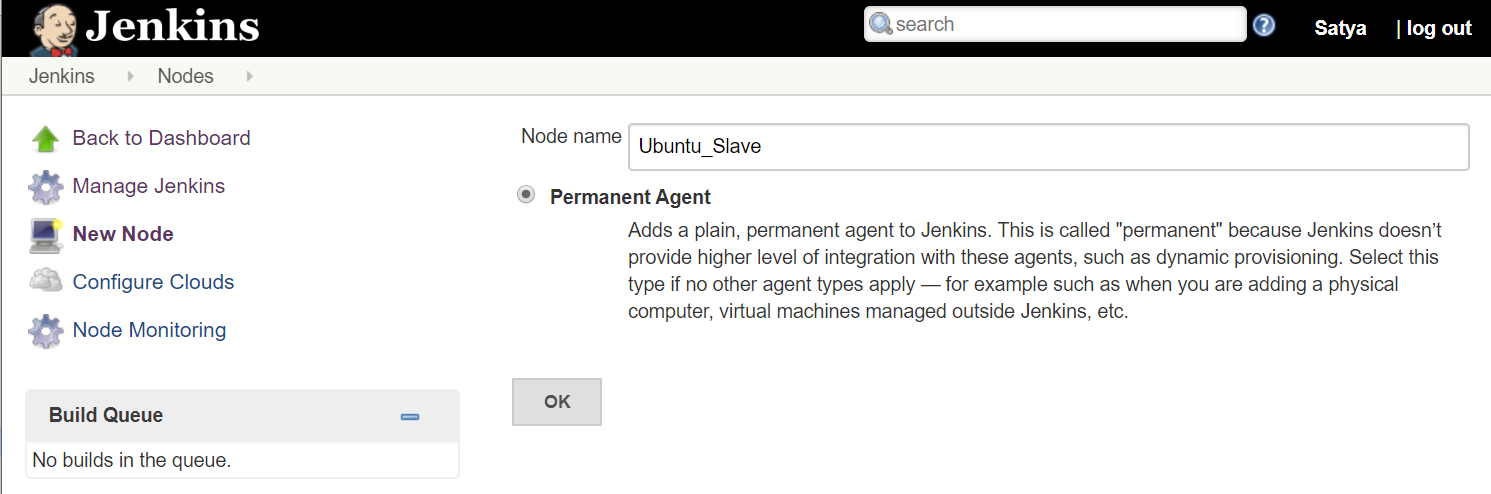
sudo useradd -d /var/lib/jenkins jenkins

passwd jenkins

If user already exist , delete all instances with that user

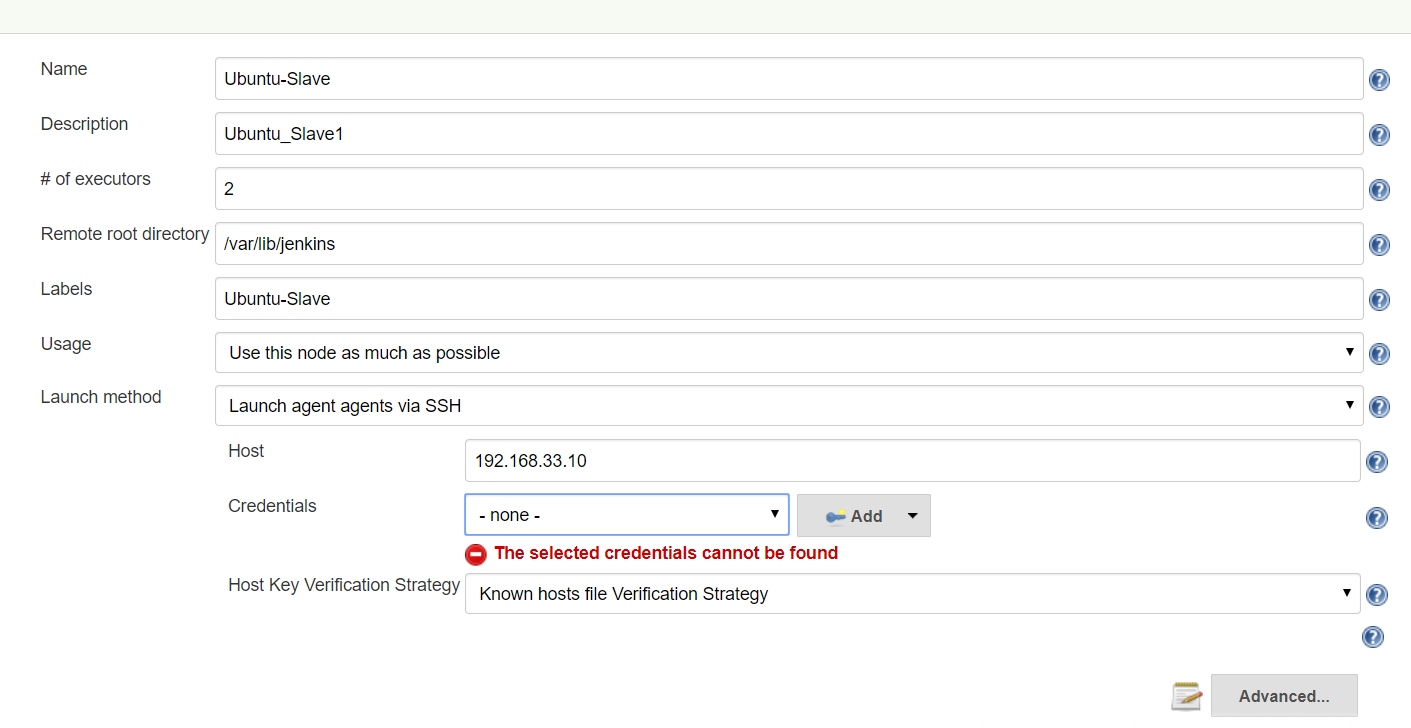
sudo userdel -r jenkins

### In Jenkins Master Node

Go to Dashboard > Manage Jenkins > Manage Nodes > New Node - tick: Permanent Agent

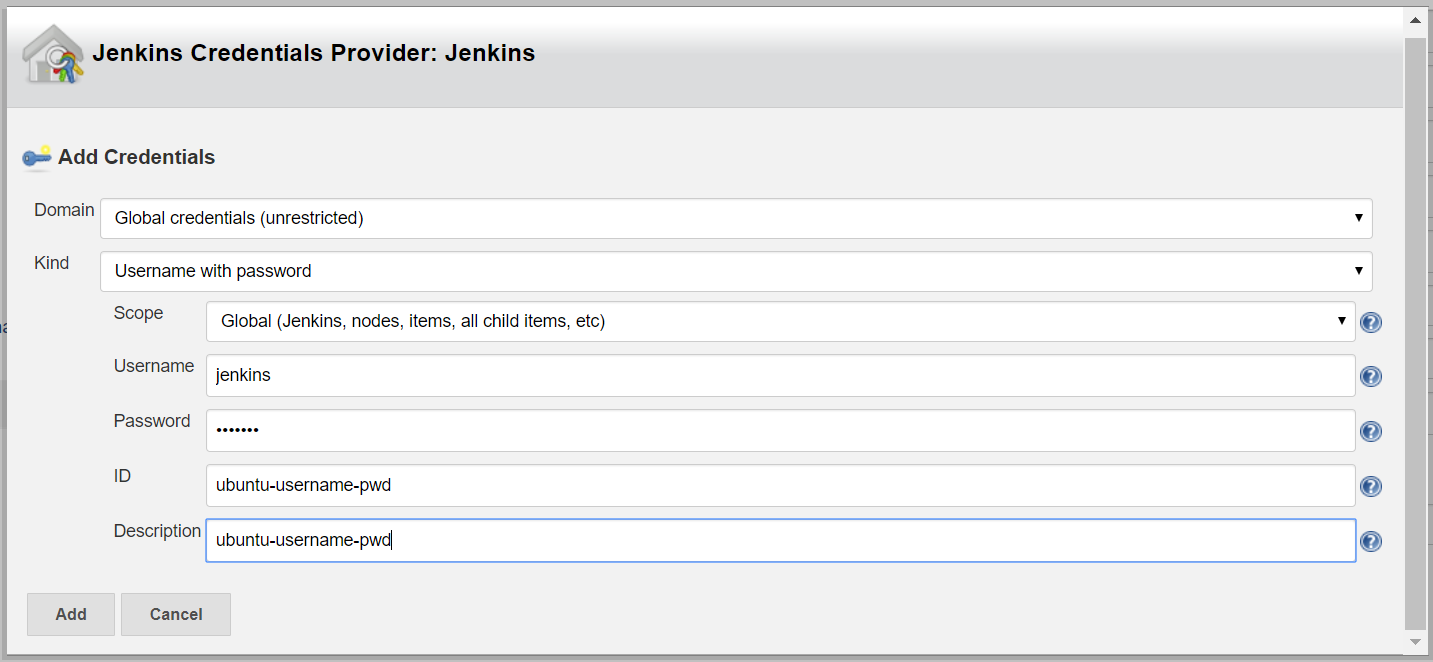
On next screen, fill below details

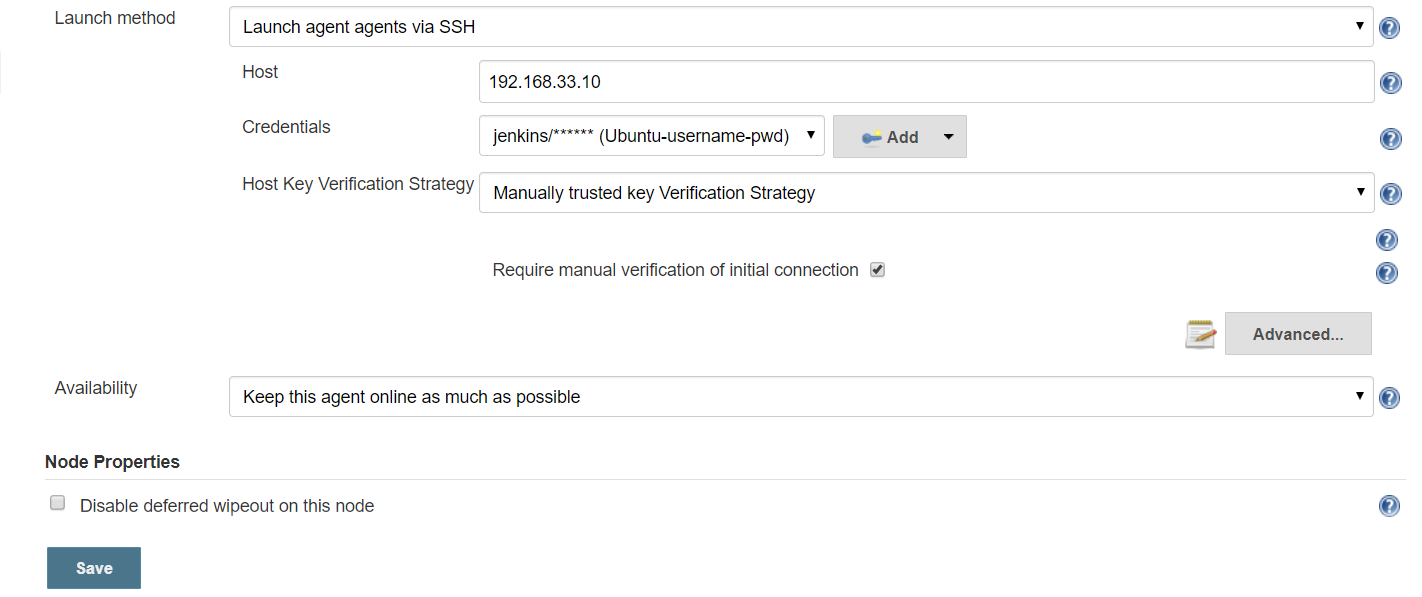
* # of executors - No. of jobs run at a time.
* Remote root directory - folder to store build artifacts/ output files on slave node

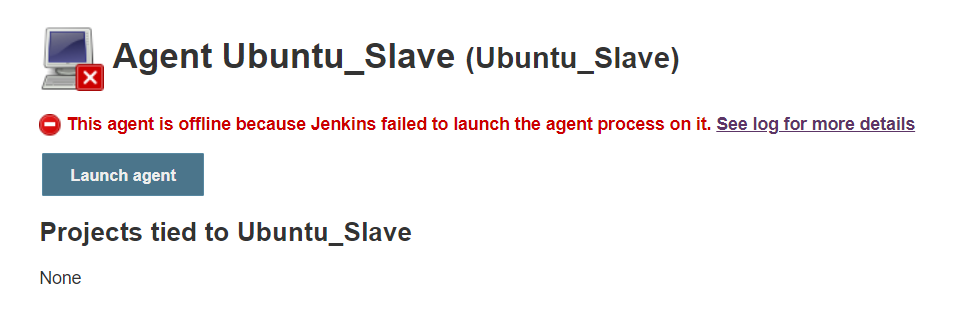


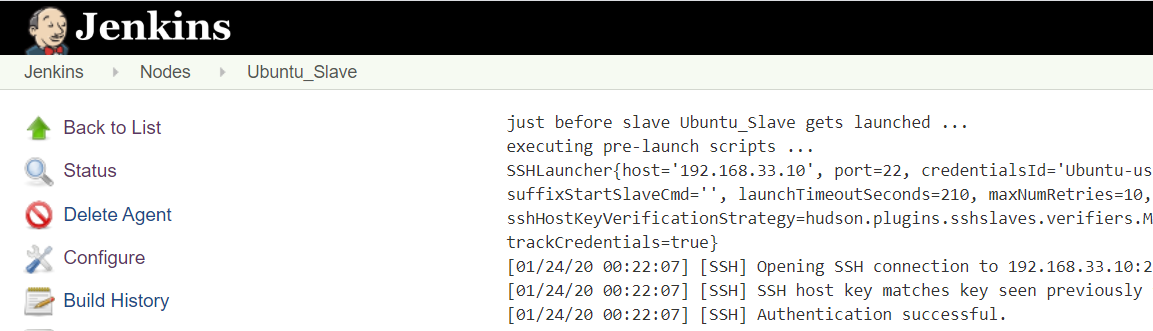
Click on **credentials > Add button : Jenkins** – it will open a pop-up window. Fill with below details

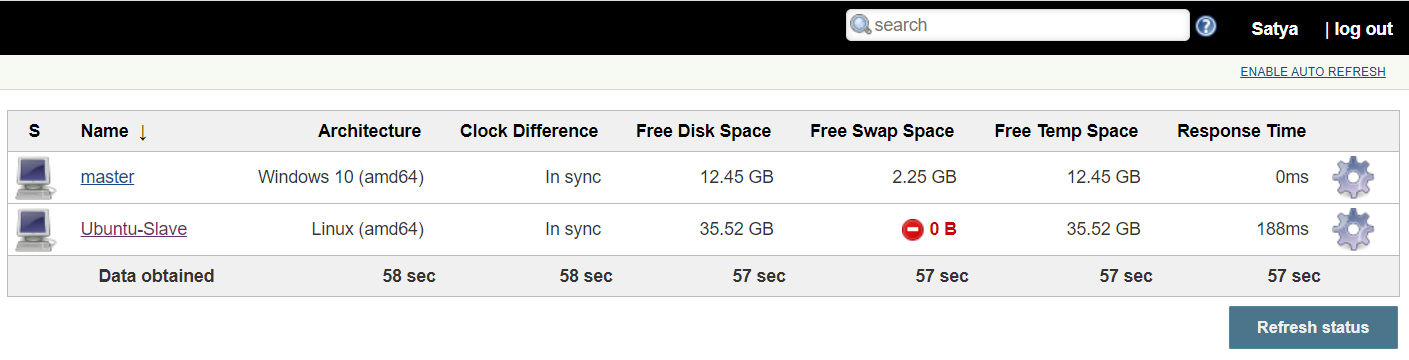
* Domain : Global
* Kind : **Username/password**
* Username **: Created agent user**



Selected created Credentials & select Host Key Verification Strategy : **Manually trusted key Verification Strategy > SAVE** 

Open created Slave & do Luanch Agent

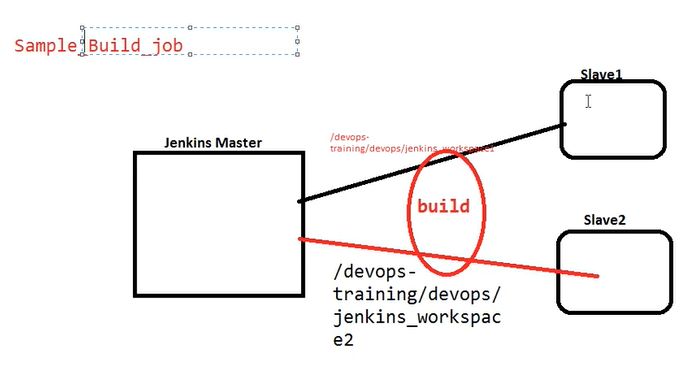




## Using SSH

We will go for this scenario where

* Master – Linux
* Slavers – Linux/Windows Systems



we have Jenkins workspace in Master Node, where all Job-related details & created artifacts will store here.we can find this location by opening Jenkins Master URL

**Dashboard > Configure System > Home directory** : /var/lib/jenkins

In the same way, if we trigger build in Slave node - it has to store all jobs related data in somewhere.For that we need to create a Directory in Slave machine

mkdir /var/lib/sshjenkins

sudo chmod -R 777 /var/lib/sshjenkins

Provide this location as 'Remote Directory laction' in master-slave configuration.

Create user

sudo useradd -d /var/lib/sshjenkins sshjenkins

passwd sshjenkins

Go to above created folder. Create private and public SSH key for that user

private key : id\_rsa

public key : id\_rsa.pub

Siwtch to #sshuser

su - sshjenkins

cd /var/lib/sshjenkins

ssh-keygen -t rsa -m PEM -C "Jenkins agent key" -f "id\_rsa"

Add the public SSH key to the list of authorized keys on the agent machine

cat id\_rsa.pub >> ~/.ssh/authorized\_keys

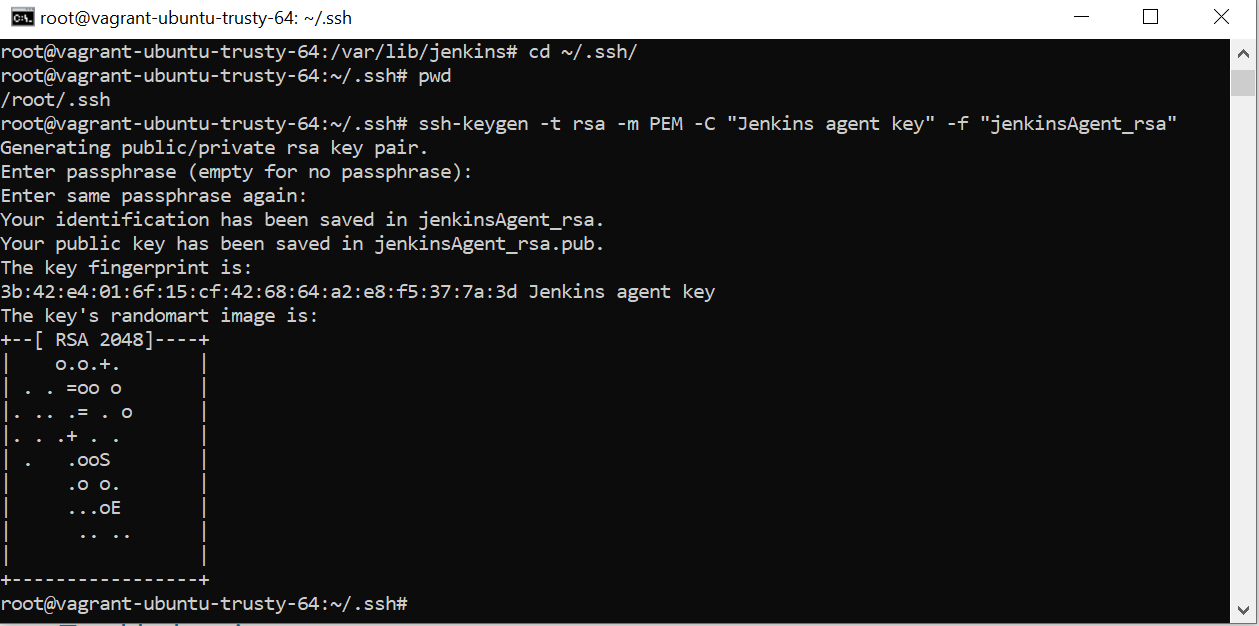
chmod 600 ~/.ssh/authorized\_keys

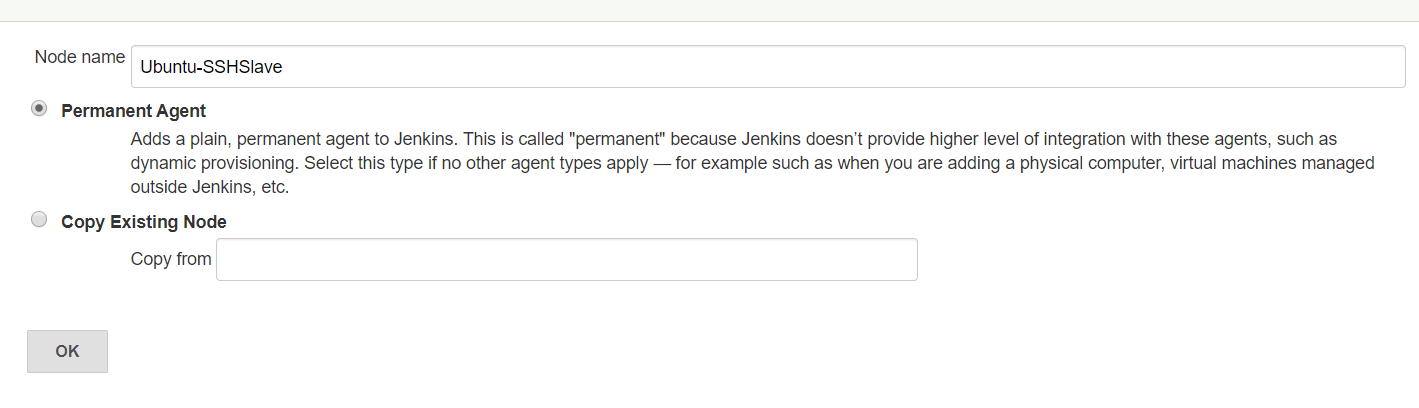
Copy the private SSH key (/var/lib/sshjenkins/id\_rsa) from the agent machine to your OS clipboard

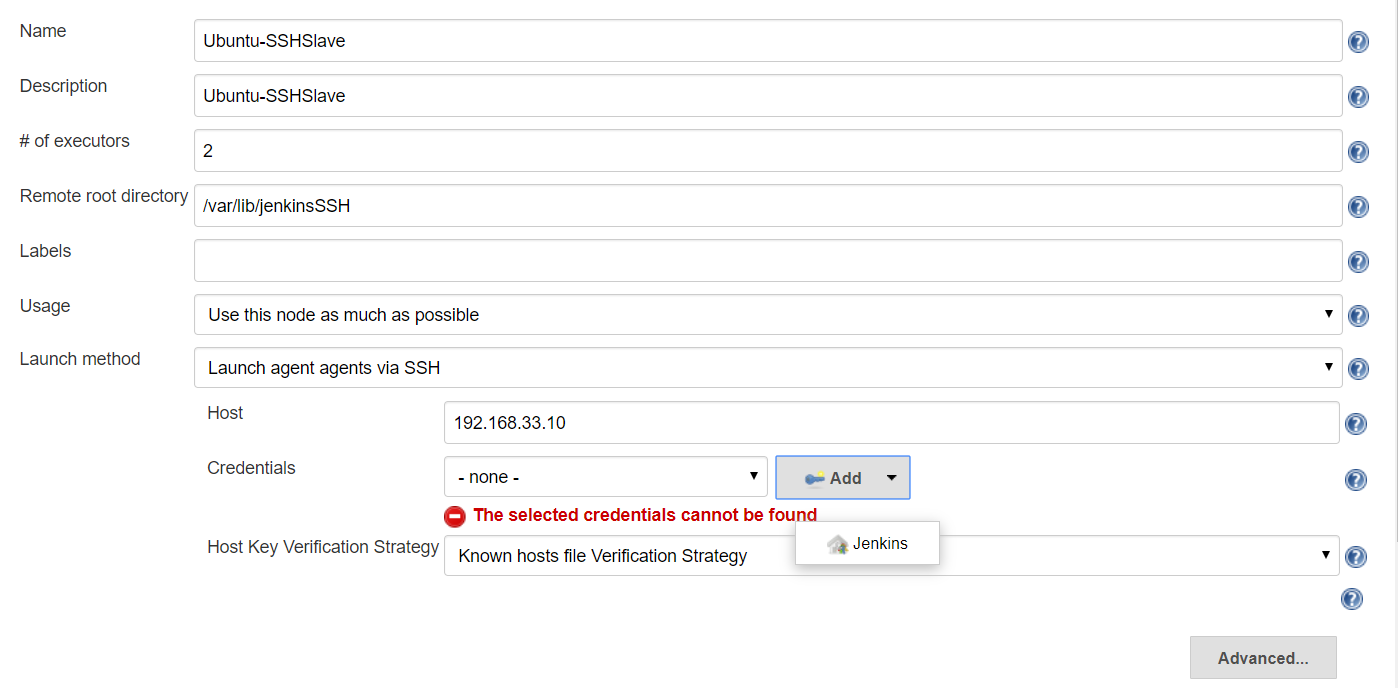
cat id\_rsa

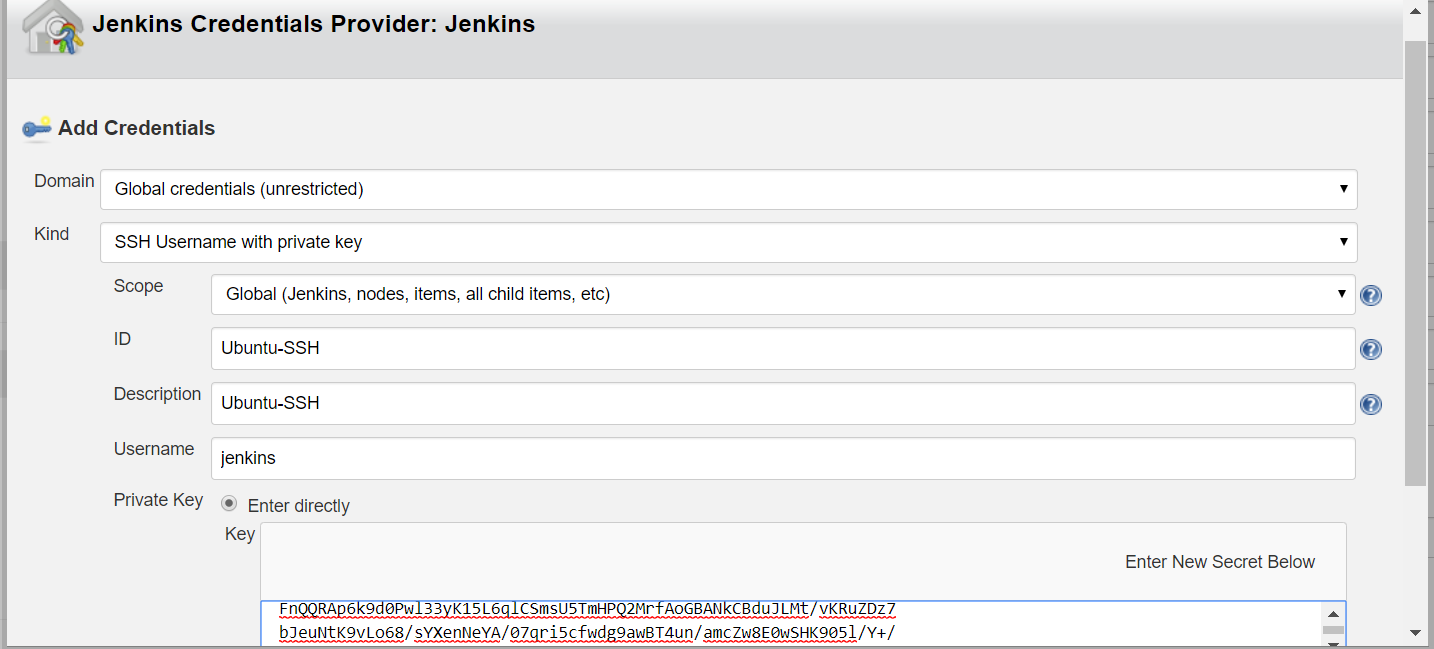
-----BEGIN RSA PRIVATE KEY-----

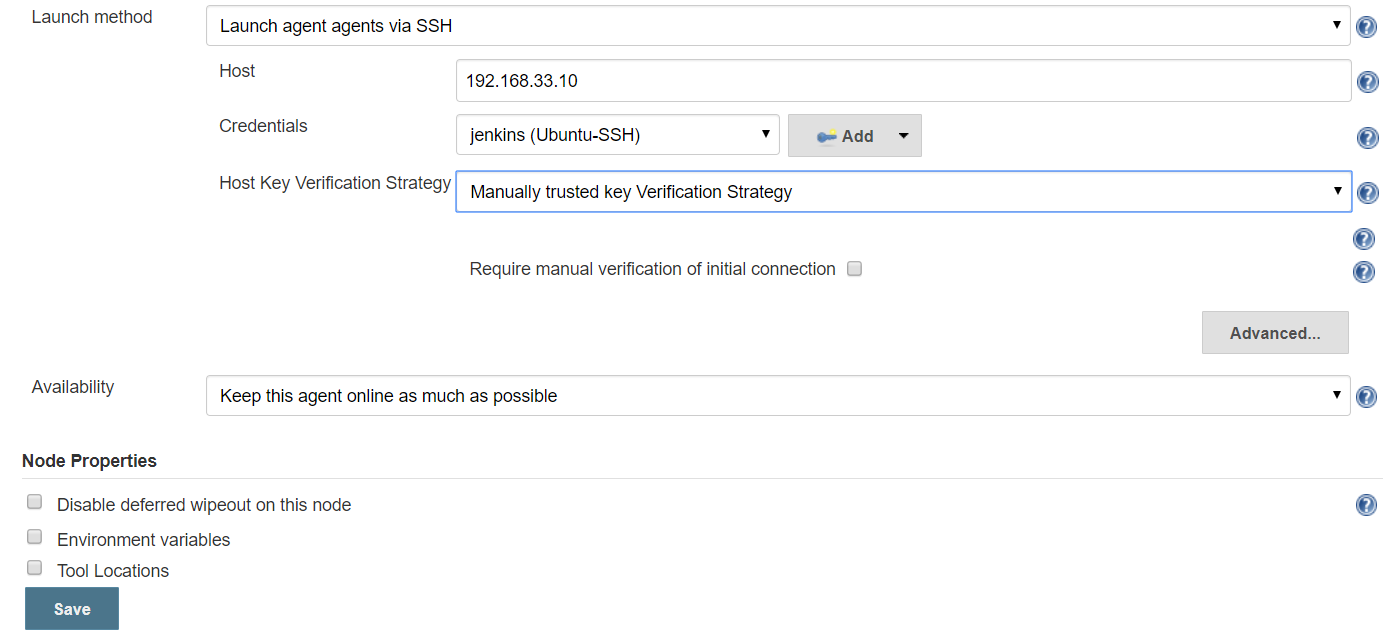
-----END RSA PRIVATE KEY-----











## Jenkins – Master Slave Job Configuration

### Slave – Configuration

Get JDK, MAVEN, GIT installation paths of Slave node

Add Java, Maven paths to Slave nodes.

JAVA\_HOME

/usr/lib/jvm/java-8-openjdk-amd64

M2

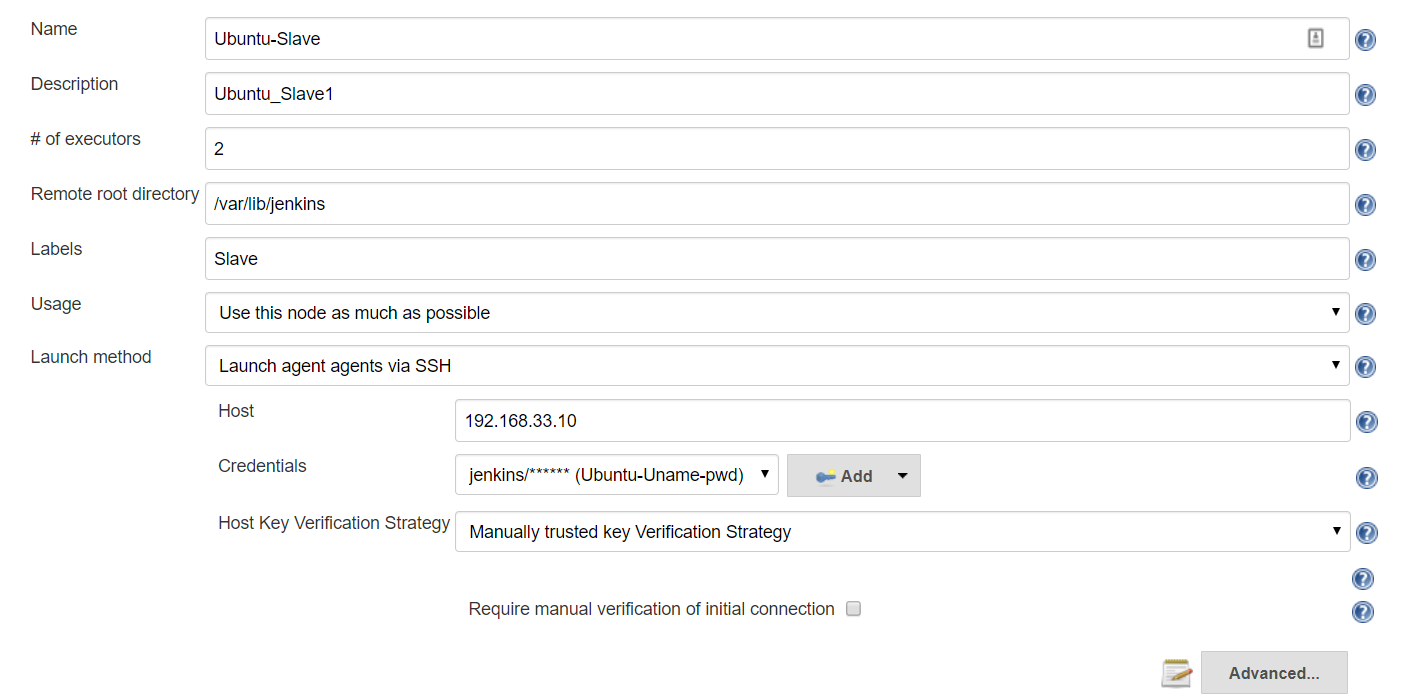
/opt/apache-maven-3.6.3/bin

M2\_HOME

/opt/apache-maven-3.6.3

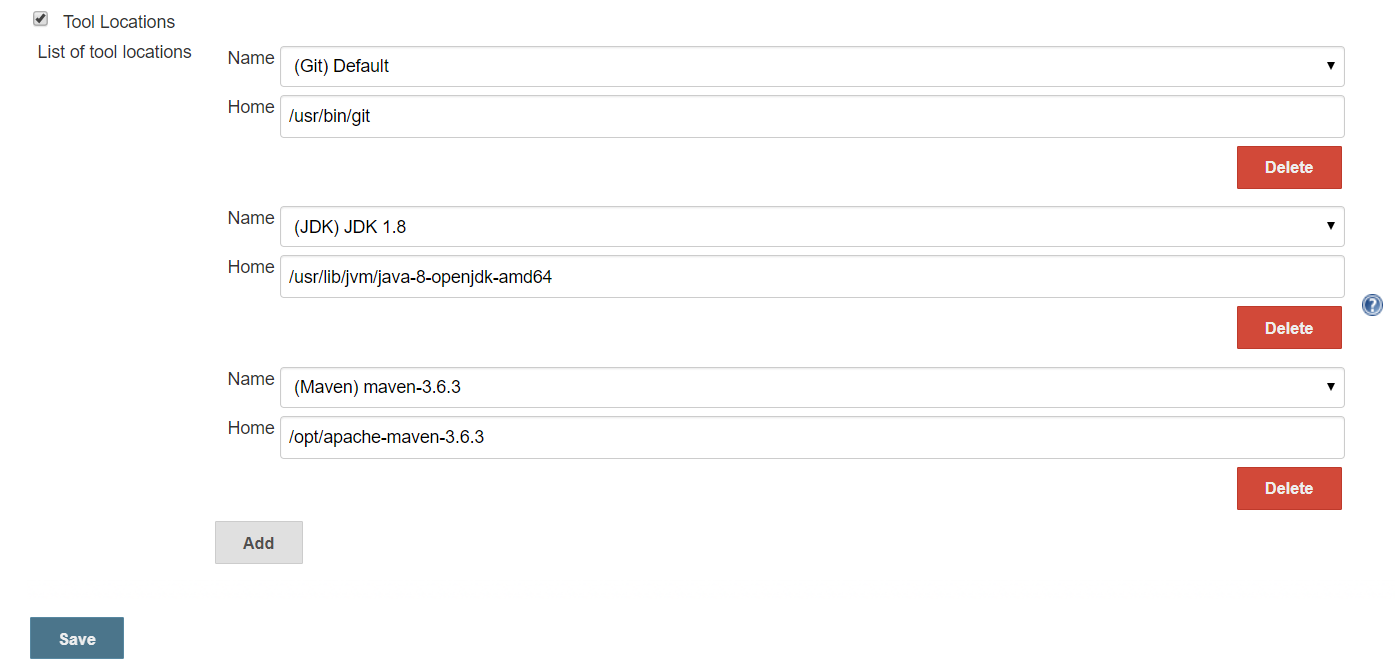
Manage Jenkins >Manage Nodes > Select Node : Ubuntu-Slave > Configure

# Provide Slave Host, Credentials etc



#1 Node Properties > Environment variables 

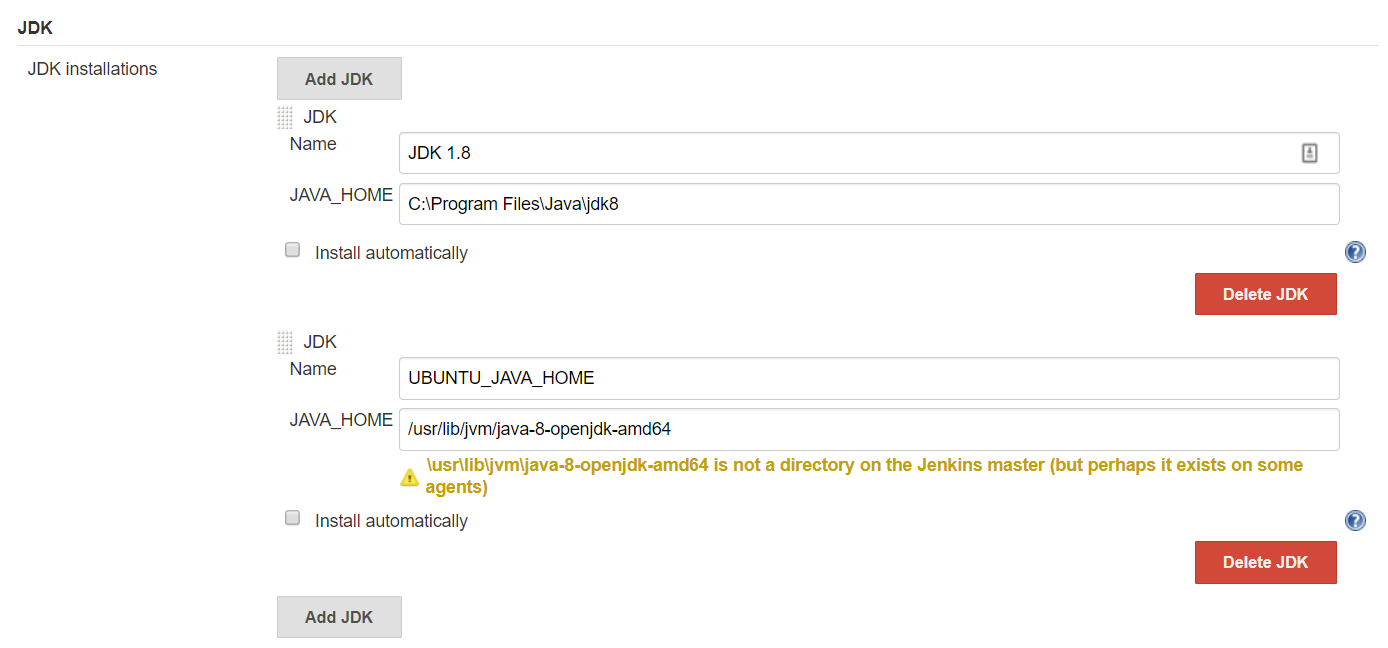
#2 Node Properties > Tool Locations



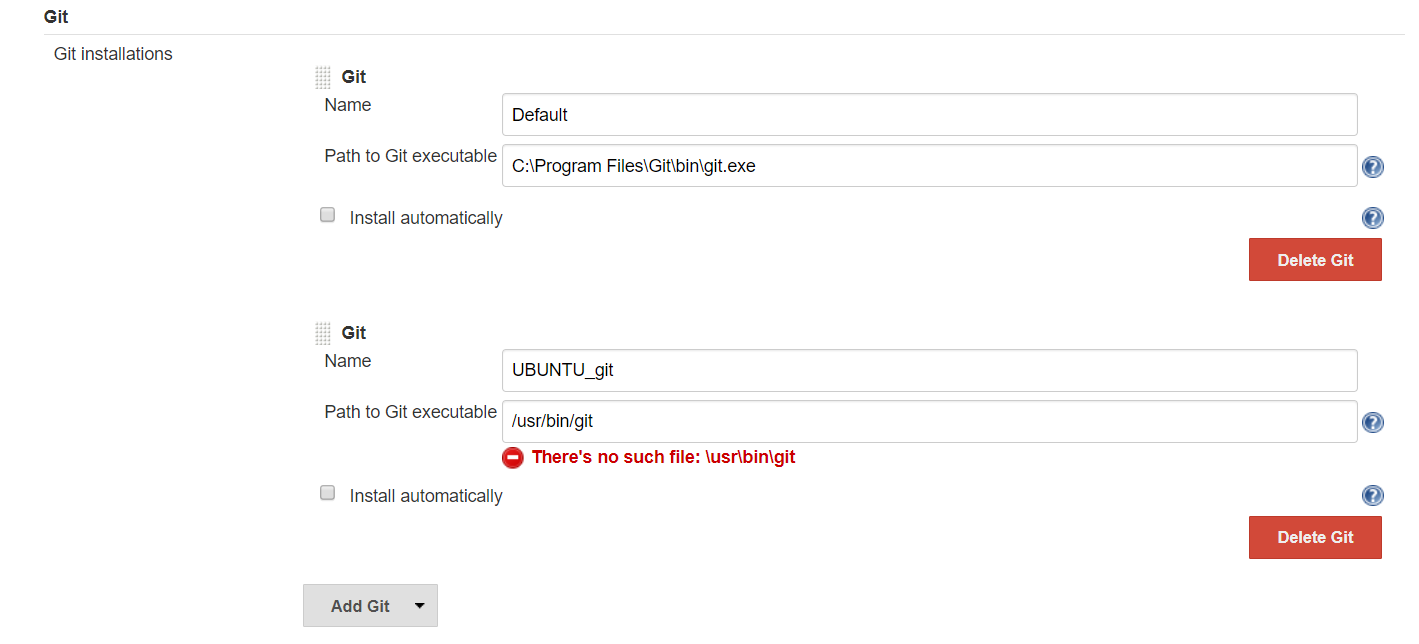
### Master – Configuration

Master must know the Tool Configuration like JDK, Maven used by Nodes.

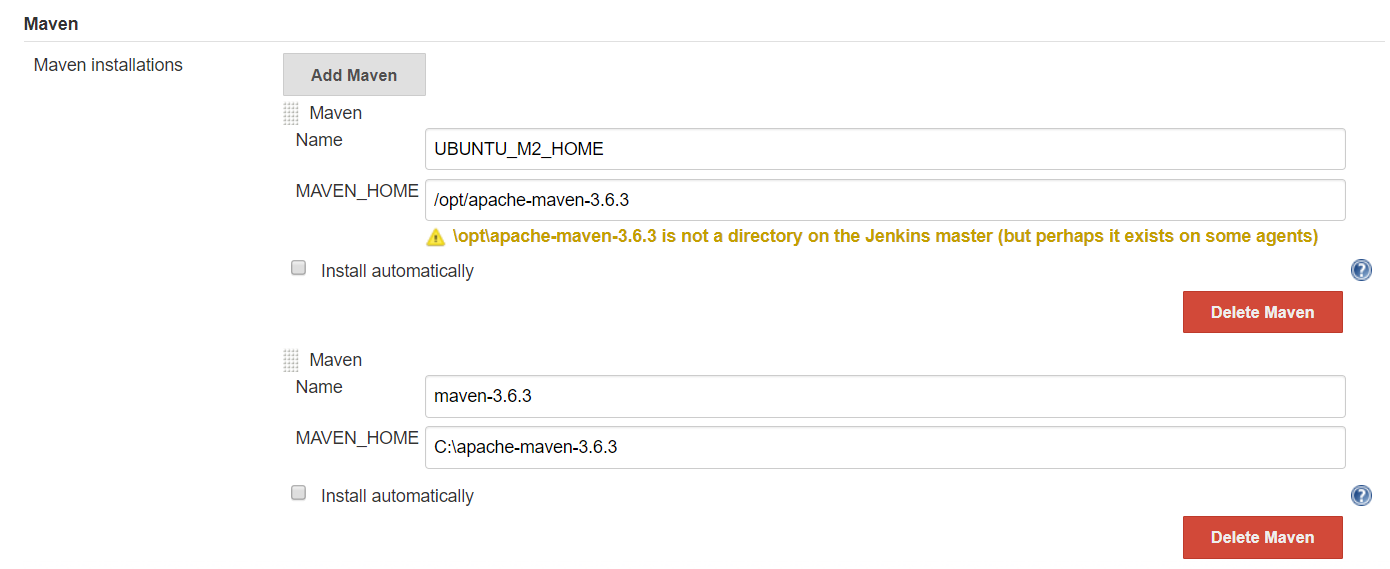
**Configure JDK installations**

Manage Jenkins > Global Tool Configuration > Configure JDK installations used by Slave 

**Git installations**



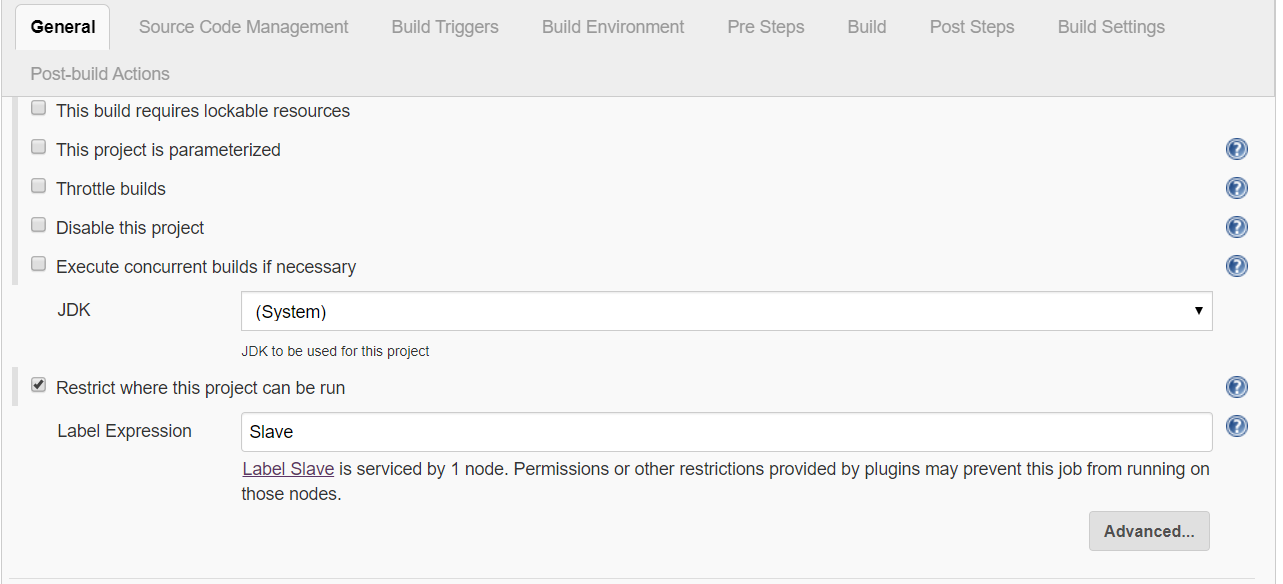
**Maven installations**



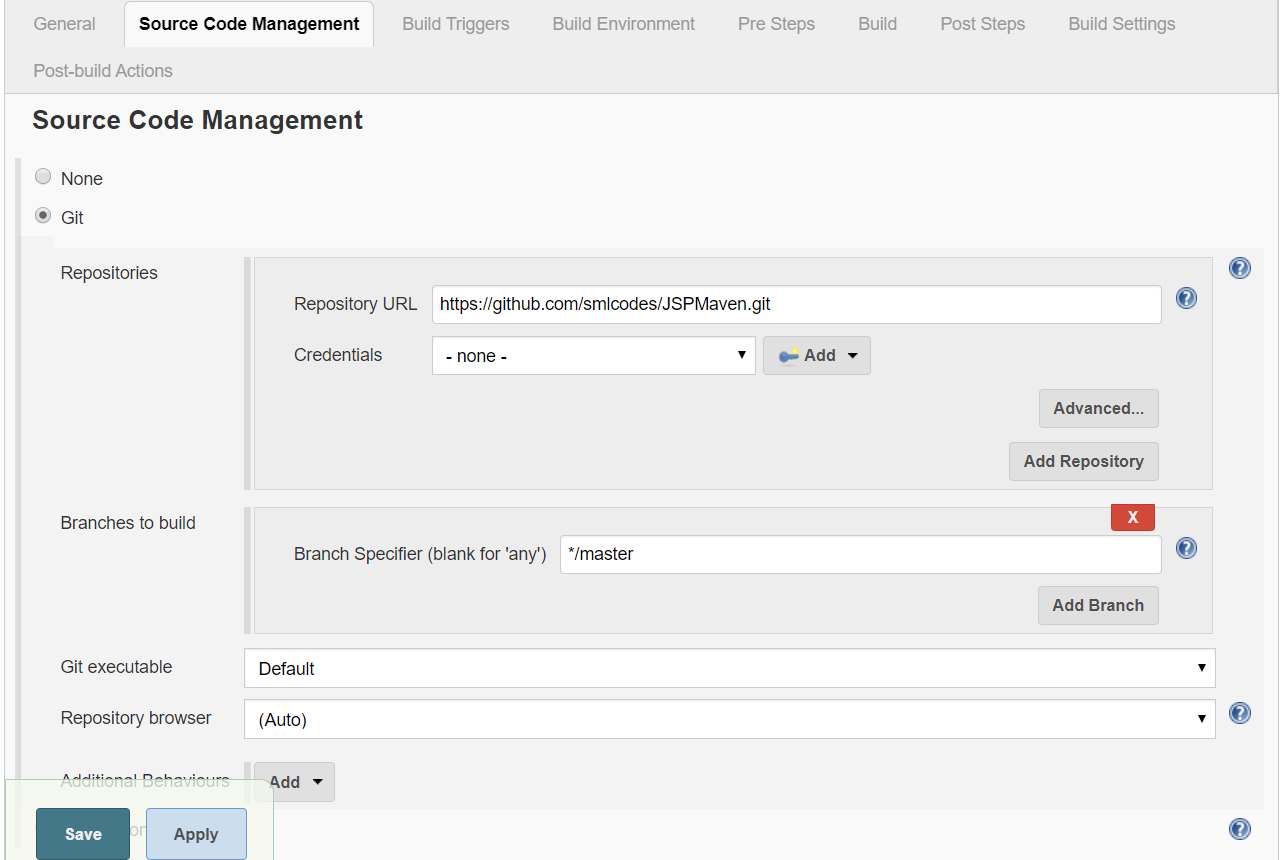
### Master – Configure Jenkins Job

**Open any Job > Configure > General >**

* [Tick] Restrict where this project can be run
* Label : provide Slave label mentioned at the time of Slave creation

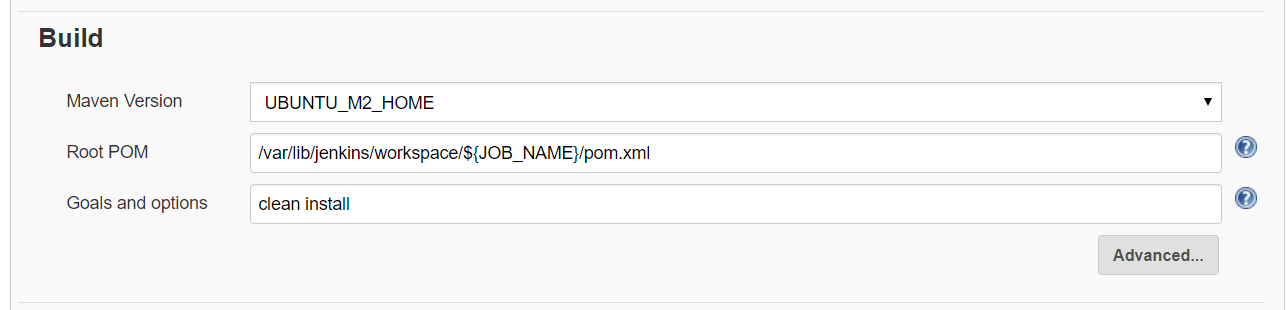


**Source Code Management**



**Build Step :**

location of Build data stored in slave system : /var/lib/jenkins/workspace/${JOB\_NAME}/pom.xml



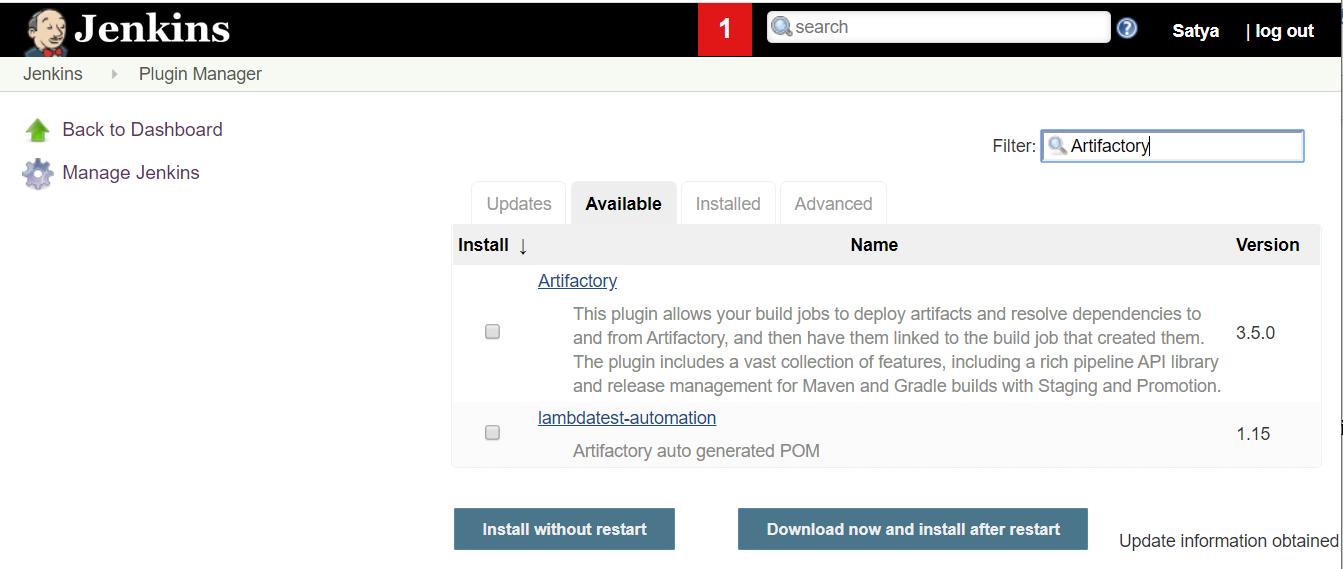
**Save & Build Now**



# Jenkins – JFrog Integration

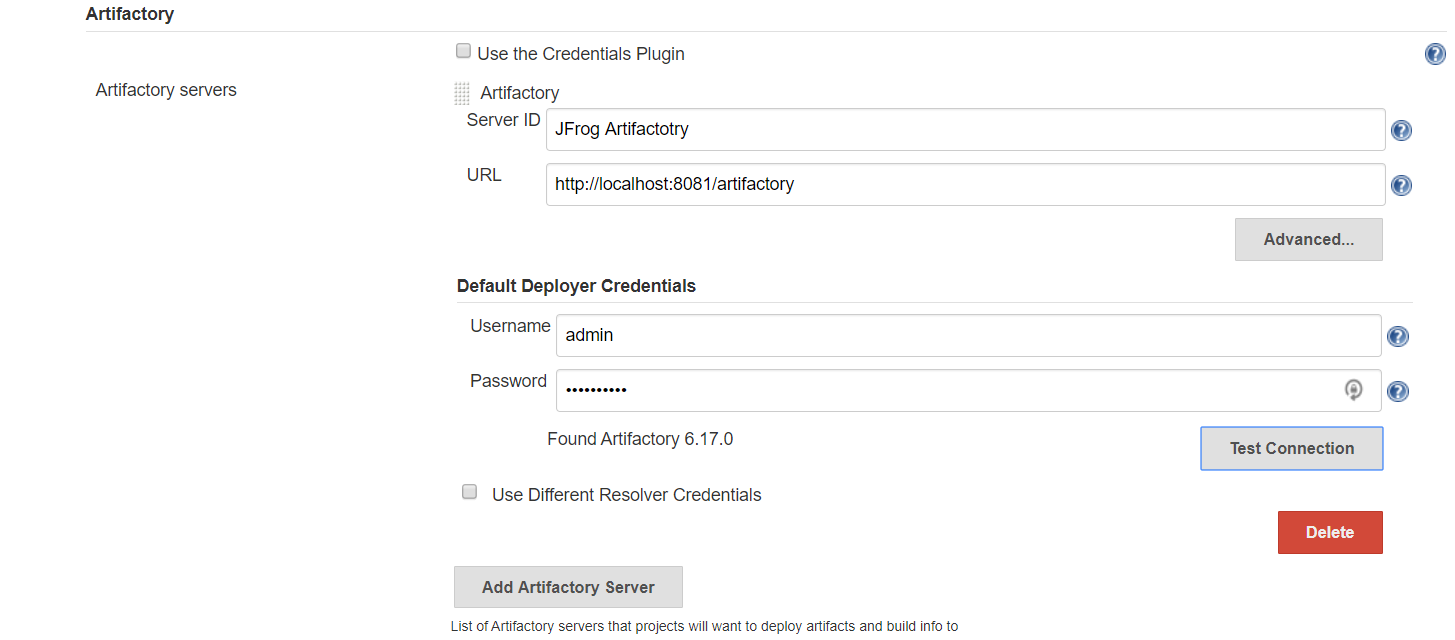
**Add Artifactory Plugin to Jenkins**

Go to Jenkins dashboard -> Manage Jenkins -> Manage Plugins -> Available -> **Artifactory** -> Install without restart.

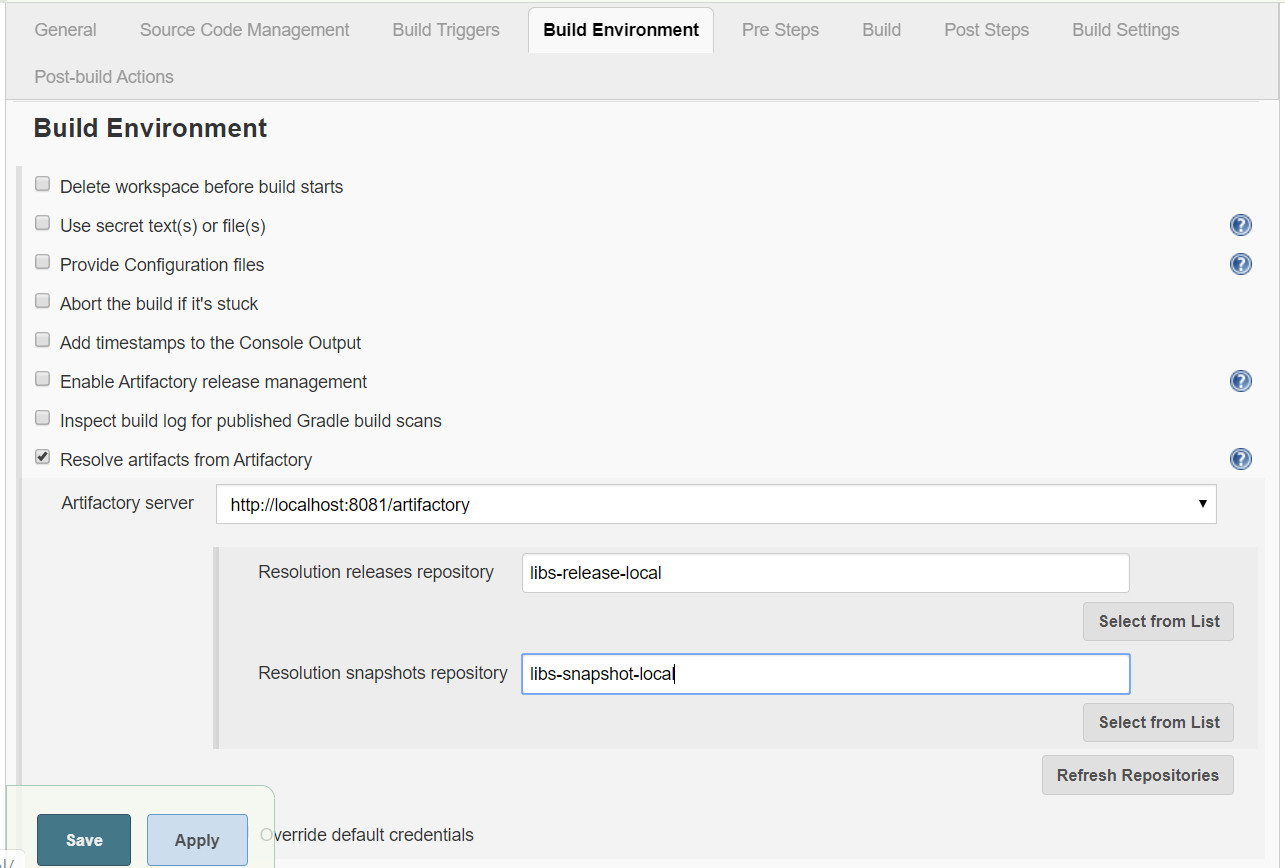


**Configure Artifactory-related settings in Jenkins**

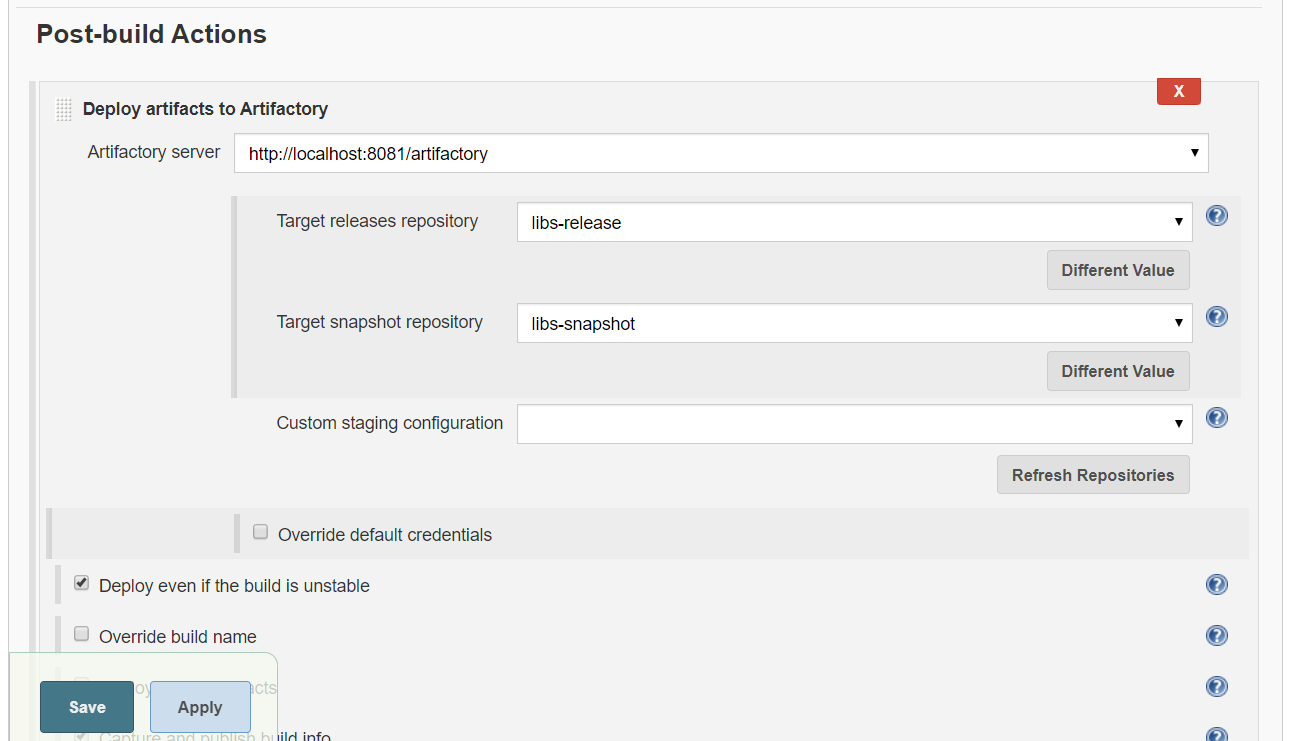
Go to Jenkins dashboard -> Configure System ->Artifactory section ->Add artifactory server -> provide the details -> Test the connection ->apply & save



**Configure Project** : to creates package file after compiling all of the source files.

Go to Build Environment section -> Resolve artifacts from artifactory -> Click on refresh Repositories ->select the repository in release and snapshot field from the lists. 

**post-build section**

Go to Add post-build section ->select **deploy artifacts to artifactory** -> click on refresh -> choose the target releases and snapshot repository (repositories created earlier) ->save 

**Click on Build now.**

Jar files are resolved from the local repository or Artifactory.

**check the package**

Once the package is created, it is stored in artifactory too. Go in the artifactory and check the package.

# Jenkins - Automated Deployment

# Jenkins - Metrics and Trends

# Jenkins – Pipeline

<https://www.edureka.co/blog/jenkins-pipeline-tutorial-continuous-delivery>

# Errors & Solutions

## ERROR: Server rejected the 1 private key(s) for jenkins

I solve this issue following below step:

**From the target slave node's console**

1. Switch to the “root” user.

sudo su

1. Add a jenkins user with the home “/var/lib/jenkins”. {Note : I am keeping my home directory in /var/lib/jenkins} :

useradd -d /var/lib/jenkins Jenkins

**From the Jenkins Master**

Copy the id\_rsa.pub key from the **Jenkins** user on the master.

cat /var/lib/jenkins/.ssh/id\_rsa.pub

**From the target slave node's console**

Create an authorized\_keys file for the Jenkins user.

mkdir /var/lib/jenkins/.ssh

vi /var/lib/jenkins/.ssh/authorized\_keys

Paste the key from the Jenkins master into the file vim. Save with “:wq!”.

# Ref.

<https://javatpoint.com/jenkins>

<https://www.edureka.co/blog/what-is-jenkins/>

Master Slave

* <https://www.howtoforge.com/tutorial/ubuntu-jenkins-master-slave/>
* <https://hostadvice.com/how-to/how-to-setup-jenkins-master-and-slave-on-ubuntu-18-04-lts/>

JFROG : artifactory

<https://c4clouds.com/>