UPES

Xebia

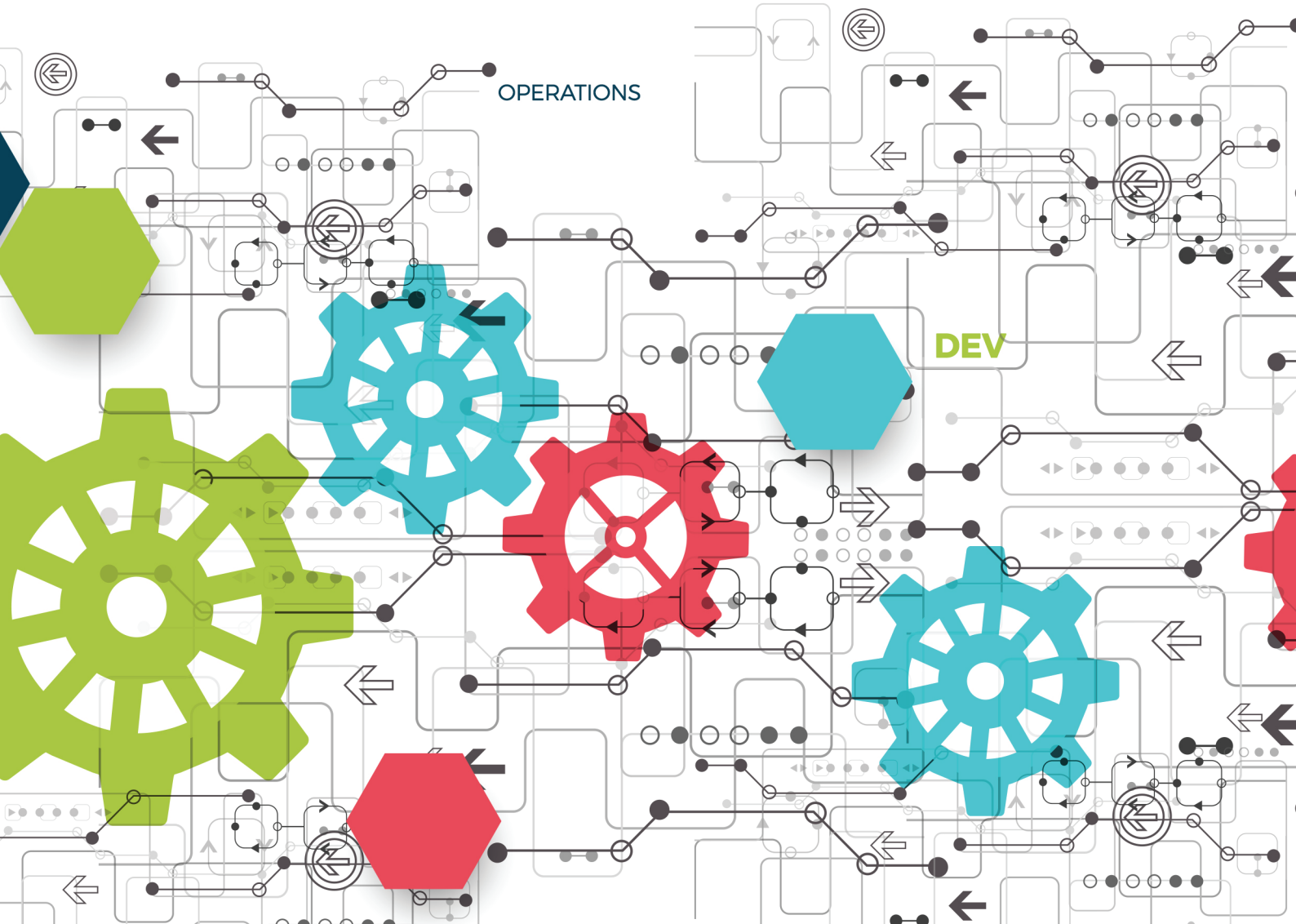
YEAR 3

B.Tech Computer Science and Engineering in DevOps

Continuous Integration and Continuous Delivery

Semester **05** | **Labs**

Release 1.0.0



Copyright & Disclaimer

B. TECH CSE with Specialization in Continuous Integration and Continuous Delivery

Version 1.0.0

Continuous Integration and Continuous Delivery Lab

Syllabus:

1.1 Introduction to Jenkins and setup/configuration

1.2 Jenkins plugins

1.3 Installation and Configuration of git/Java/maven on Build server (Windows)

1.4 Jenkins job, parameters, build, post-build actions and Pipeline

1.5 Jenkins Agent/Slave configuration with Windows/Ubuntu master hosts

1.6 Configuring Jenkins with git plugin

1.7 Creating a Java application (V 1.0)

1.8 Create a new Jenkins pipeline

1.9 Merging local changes to the version control system (Git)

1.10 Installing/Configuring Nexus

1.11 Publishing artifacts to Nexus

1.12 Installing/Configuring Sonarqube

1.13 Publishing Static code analysis to Sonarqube

1.14 Use Jenkins as a Continuous Integration server

1.15 Deploying the application to staging/prod environment

1.16 Merging feature branch code (V 2.0) to existing application created in step 1.6

1.17 Uploading plugins manually in Jenkins

1.18 Backup Management in Jenkins Server

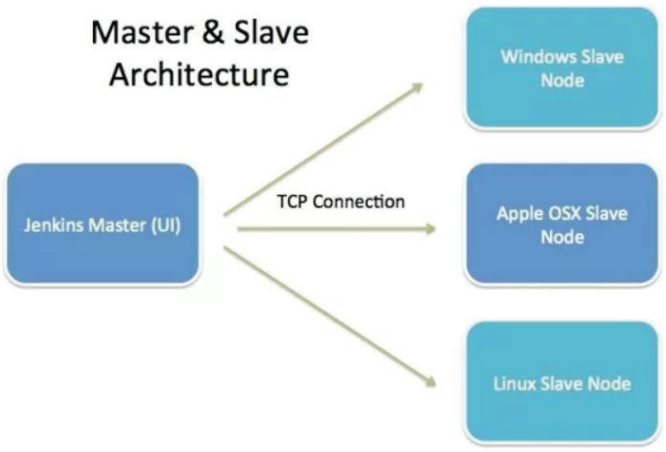
1.1 Introduction to Jenkins and setup/ configuration

Jenkins is the most popular open source CI/CD (continuous integration and continuous delivery) tool available today. Initially it was called Hudson, developed by Sun Microsystems in 2004-2005, and renamed to Jenkins in 2011.

In the Jenkins 'master/slave' architecture, a master represents itself as a basic Jenkins installation, and handles all tasks for build system. 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.

In this section, we will learn how to set up Jenkins master-slave architecture using the latest Windows / Ubuntu 18.04 LTS. We will learn to set up the Jenkins master server, and then add other CentOS/Ubuntu nodes as Jenkins slaves.

1.1.1 Understanding Jenkins master/agent architecture



Jenkins supports the master/agent architecture. In master/agent architecture, we can install Jenkins on master and then utilize other agents for distributing the load.

We should delegate Jenkins jobs to agents for execution. This way, we can support multiple executions using different resources.

There are specific scenarios where master/agent architecture is extremely useful, such as:

- The Jenkins machine has limited capacity; even with the higher capacity, there will be a time where it can't fulfil all requests. By distributing the load on agent nodes, we can free resources available on the system where Jenkins is installed.
- Different jobs require different kinds of resources, and they are restricted to specific machines only. In such a case, we can only utilize that machine.
- Different operating systems are required or some tools work only in specific OS, so we can utilize those tools by making a system agent on which they are installed.
- To avoid a single point of failure.

Prerequisites

Jenkins runs on current versions of:

- Linux
- Windows
- macOS

- Unix/BSD
- Docker containers
- Java 8 or Java 11 is required
- Java 9 and 10 are not supported
- Virtual Box to deploy Jenkins Slaves
- Master – Windows 7/10/2012 / Ubuntu 18.0+
- Slave – Windows/CentOS7
- Deployment Host – CentOS 7/Ubuntu 18.0+

What we will do?

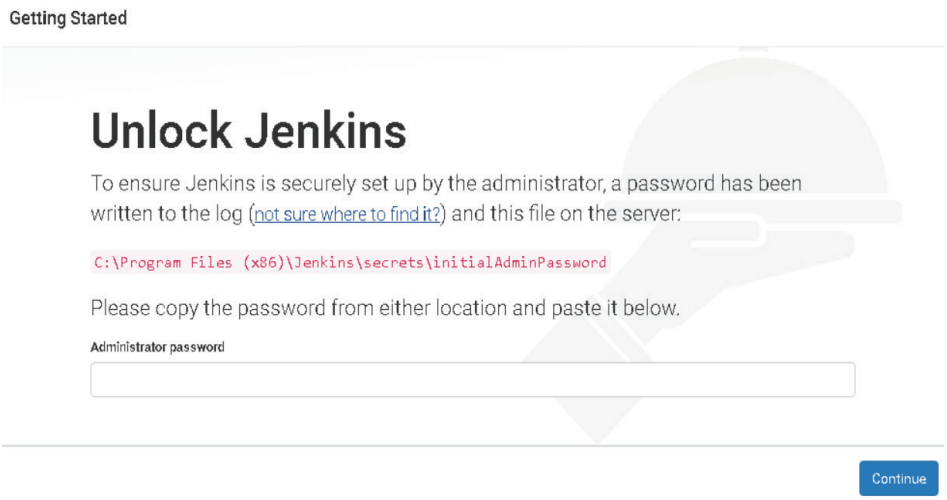
- Install Jenkins on Windows / Ubuntu
- Configure Jenkins Master Credentials
- Configure Slave Agent
- Add New Slave Node
- Configure Slave Agent to Execute Build
- Running a sample Job

Lab:

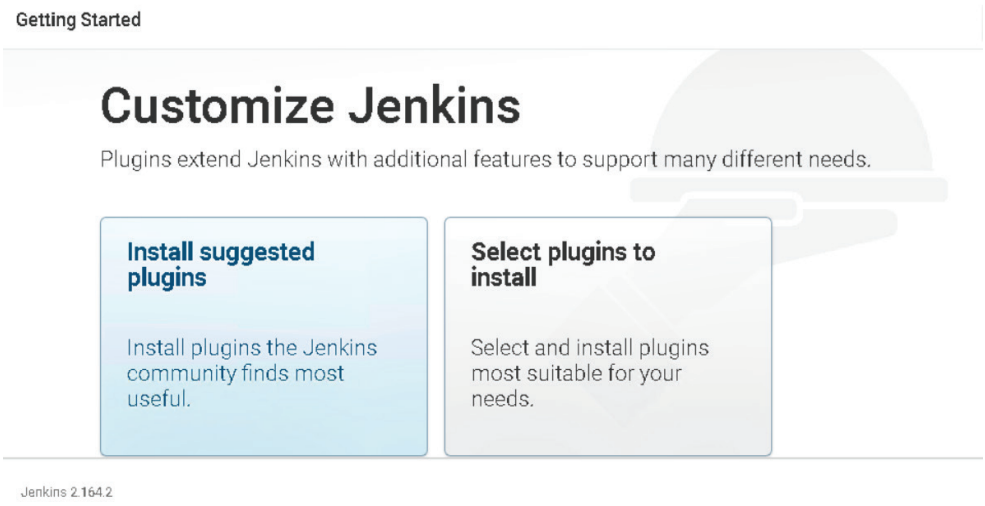
1.1.2 - Install Jenkins Master - Windows

- Navigate to *https://jenkins.io/download/* and select Windows platform, download will start automatically.
- Unzip jenkins-2.164+ zip file and run Jenkins.msi
- Select default settings and install Jenkins

Unlock Jenkins by following the instructions:



Choose 'Install Suggested Plugins', click on it.



Create username, Password and login to Jenkins portal. Additional users can be created after logging into the portal

After the plugins installation is complete, we need to create a new admin password. Type in your admin username, password, email etc. and click on 'Save and Continue'.

Now click the 'Start using Jenkins' button.

If you want to continue using Windows machine as Jenkins master then you can ignore the below step (1.13) and go to 1.2 section of this lab.

1.1.3 Install Jenkins Master – Ubuntu (Optional Step)

In this section, we will go through the installation of, Java, maven and git installation and configuration of all the tools to run a build.

- Install Java
- Install Maven
- Install Git bash
- Install Jenkins
- Configure UFW Firewall

- Configure Jenkins
- Jenkins Security
- Testing

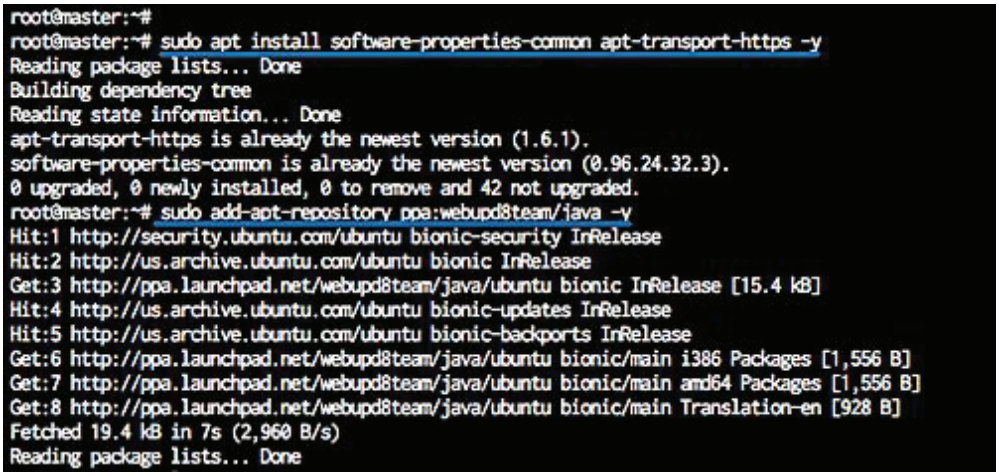
1.1.4 Install Java

Jenkins is a Java-based application, so we need to install Java OpenJDK on the server. In this step, we will install Java 8 from a PPA repository which we will add first.

Install software-properties-common packages, then add the java OpenJDK PPA

repository.

```
sudo add-apt-repository ppa:openjdk-r/ppa -y
```



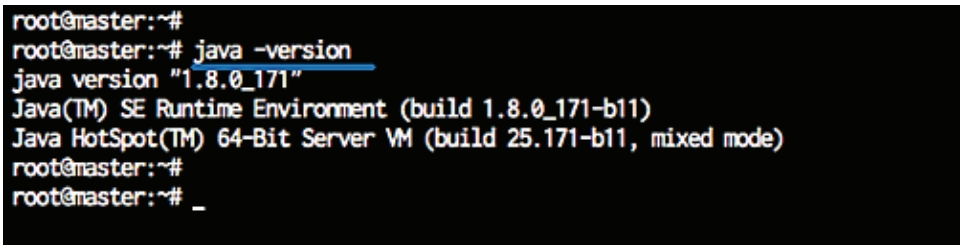
Now install the Java 8 using apt command.

```
sudo apt install openjdk-8-jdk -y
```


When the installation is complete, check the java version installed on the system.

```
java -version
```

And you will get the Java OpenJDK 1.8 is now installed on the Ubuntu 18.04 system.



1.1.5 Install Maven:

Download Maven from *https://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/* and export the path

Install Git:

Run this command to install git : apt-get install git

1.1.6 Installation of Jenkins on Ubuntu

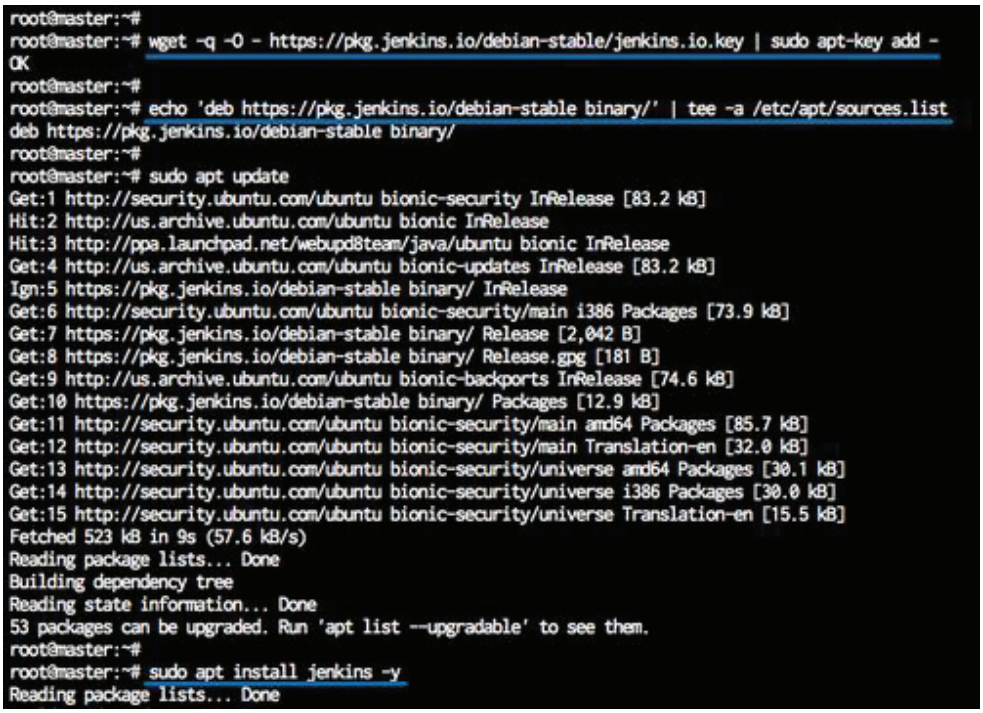
Jenkins provides an Ubuntu repository for the installation packages and we will install Jenkins from this repository.

Add Jenkins key and repository to the system with the command below.

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
echo 'deb https://pkg.jenkins.io/debian-stable binary/' | tee -a /etc/apt/sources.list
```

Now update the repository and install Jenkins.

```
sudo apt update
sudo apt install jenkins -y
```



Once the installation is completed, start the Jenkins service and enable it at boot time.

```
systemctl start jenkins
systemctl enable jenkins
```

Jenkins is up and running on the default port '8080'. Check it using netstat command as below.

```
netstat -plntu
```

And you will get the result as below.


```
root@master:~#
root@master:~# systemctl start jenkins
root@master:~# systemctl enable jenkins
jenkins.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
root@master:~#
root@master:~# netstat -plntu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      5688/systemd-resolv
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      715/sshd
tcp6       0      0 :::8080                 :::*                    LISTEN      5539/java
tcp6       0      0 :::22                   :::*                    LISTEN      715/sshd
udp        0      0 127.0.0.53:53           0.0.0.0:*                           5688/systemd-resolv
udp        0      0 10.0.2.15:68            0.0.0.0:*                           1487/systemd-networ
udp     4352      0 0.0.0.0:68              0.0.0.0:*                           1199/dhclient
udp6       0      0 :::33848                :::*                                5539/java
udp6    6016      0 :::5353                 :::*                                5539/java
root@master:~#
root@master:~# _
```

```
root@master:~#
root@master:~# a2enmod proxy
Enabling module proxy.
To activate the new configuration, you need to run:
  systemctl restart apache2
root@master:~# a2enmod proxy_http
Considering dependency proxy for proxy_http:
Module proxy already enabled
Enabling module proxy_http.
To activate the new configuration, you need to run:
  systemctl restart apache2
root@master:~#
root@master:~# systemctl restart apache2
root@master:~#
root@master:~#
root@master:~# cd /etc/apache2/sites-available/
root@master:/etc/apache2/sites-available# vim jenkins.conf
root@master:/etc/apache2/sites-available#
root@master:/etc/apache2/sites-available#
root@master:/etc/apache2/sites-available# a2ensite jenkins
Enabling site jenkins.
To activate the new configuration, you need to run:
  systemctl reload apache2
root@master:/etc/apache2/sites-available#
root@master:/etc/apache2/sites-available# systemctl restart apache2
```

1.1.8 - Configure UFW Firewall

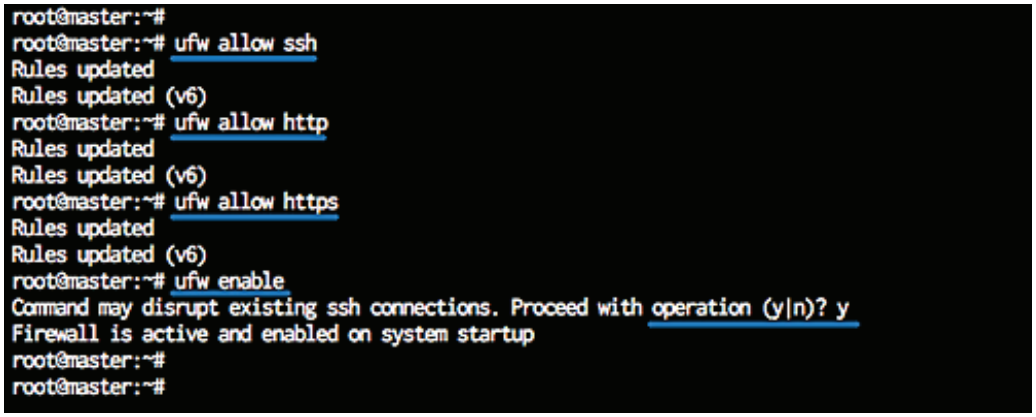
Before you enable the UFW firewall on Ubuntu server, you need to add some services port such as HTTP, SSH and HTTPS.

Add HTTP, SSH, and HTTPS services to the ufw firewall.

```
ufw allow http

ufw allow ssh
ufw allow https
```

Now start and enable the ufw firewall.

```
ufw enable
```



The UFW firewall is now enabled, and the *HTTP* port has been added.

1.1.9 - Configure Jenkins on Ubuntu server

Jenkins will run on the domain name or IP address of the host '*http://<Jenkins-hostname:8080>*'. Open your web browser and type in the URL.

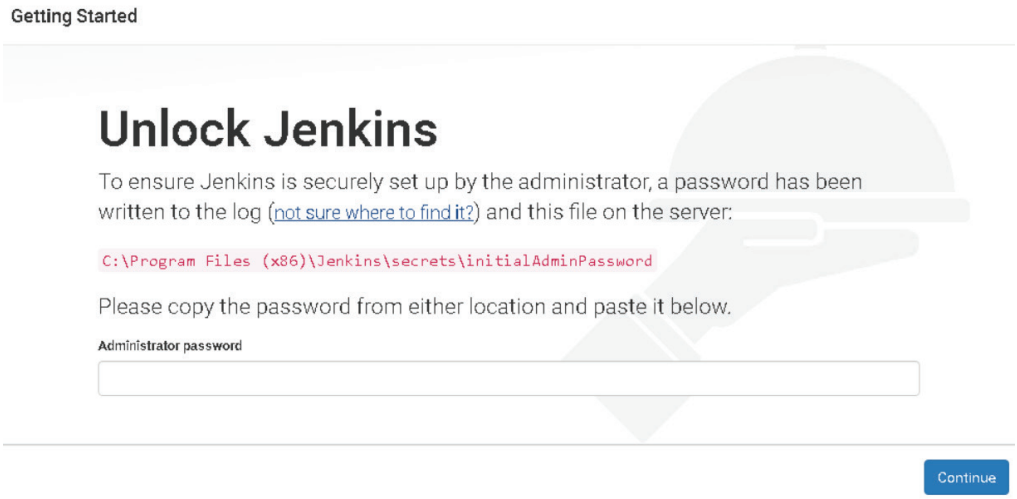
You will get the initial admin password screen. A password has been already generated by Jenkins, so you to copy the password and paste it in the password box.

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

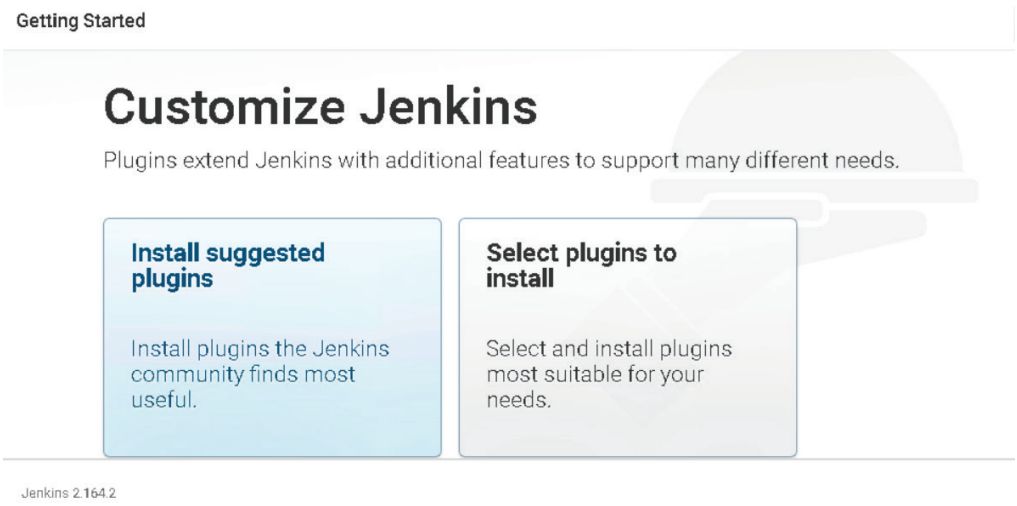


```
root@master:~#
root@master:~#
root@master:~# cat /var/lib/jenkins/secrets/initialAdminPassword
fbff82dcdf514fe693f2c3f161df33ff
root@master:~#
root@master:~#
```

Paste the results to the screen and click 'Continue'.



Choose 'Install Suggested Plugins', click on it.



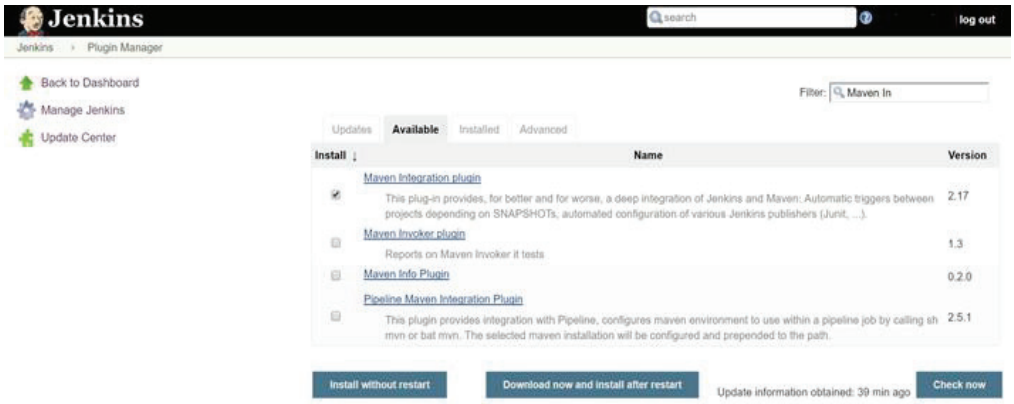
Create username, Password and login to Jenkins portal.

After the plugins installation is complete, we need to create a new admin password. Type in your admin username, password, email etc. and click on 'Save and Continue'.

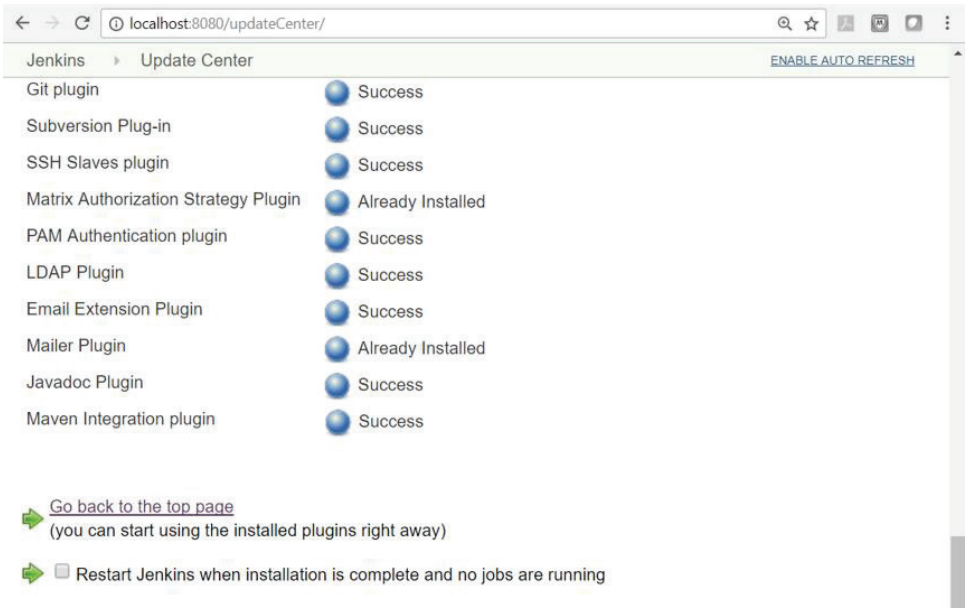
Now click the 'Start using Jenkins' button.

1.2 Jenkins plugins

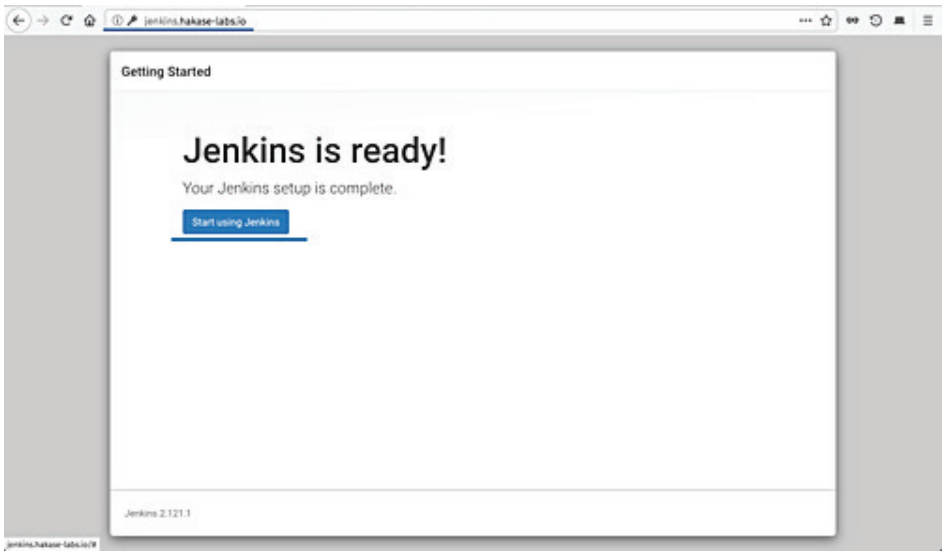
- Go to the Jenkins dashboard. Click on Manage Jenkins on left hand side and then Manage Plugins:
- Go to Available tab and select “Maven integration plugin” to install. Click on Install without restart:



- Verify the successful plugin installation, as demonstrated in this next screenshot:

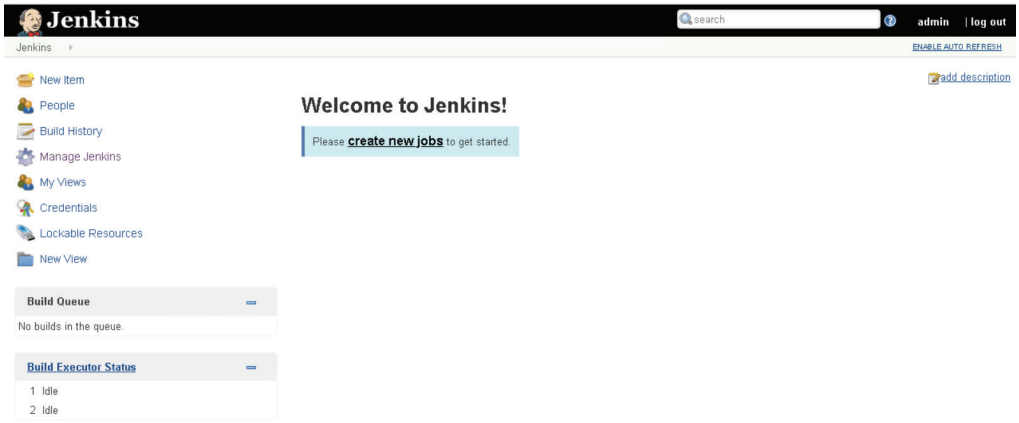


If installation is pending with restart, then restart Jenkins.



And you will be redirected to the Jenkins admin dashboard.

Jenkins installation and Configuration has been completed successfully.



1.2.1 Jenkins Security

Security is one of the important requirements for any product to protect the information and avoid executing malicious code.

A hacker actor could access your environment to launch a DDoS attack or a bot or other mischief.

Handling "Least privileges" concepts helps you manage the **AAA** concepts:

- Authentication — Control who can access the system
- Authorization — Control what each user can do on the system
- Accounting — Monitor your system to ensure that only valid processes are executing

How Jenkins will execute a Pipeline job:

A simple overview of how Jenkins executes a Pipeline helps us understand the security considerations:

- By default, the Pipeline executes with the full privileges of the Jenkins administrator
- Configure the Jenkins to execute Pipelines with fewer privileges

- The Pipeline logic, conditions in the Groovy script, loops, etc execute on the master
- Creates a workspace for each build that runs
- Stores files created for that build
- The Pipeline calls steps that run on agents

How Jenkins can do mischief:

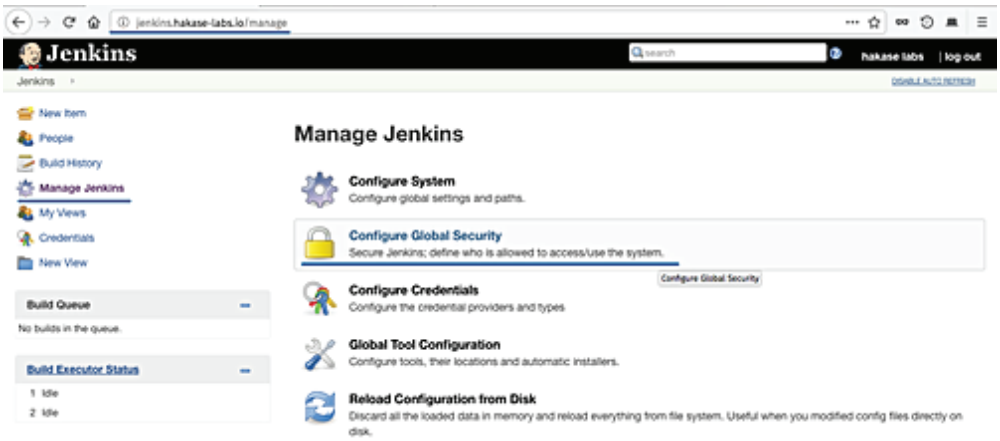
- A malicious Pipeline could reconfigure the Jenkins instance, delete files, launch malware attacks
- A trusted user could visit an infected web site and accidentally introduce malicious code into the Jenkins instance

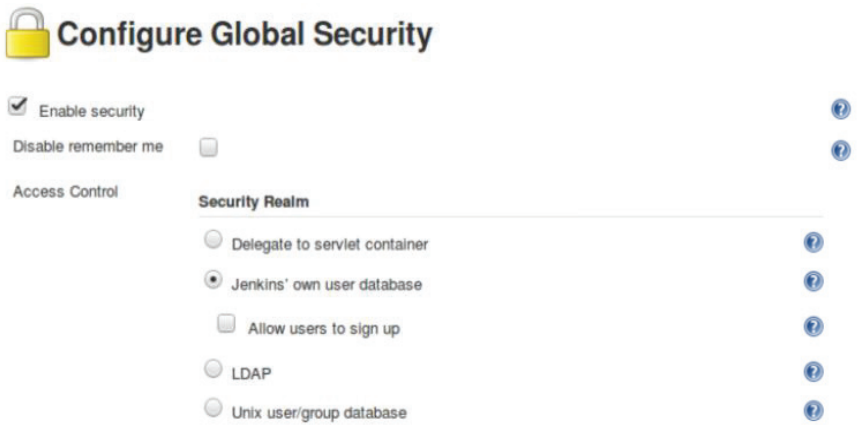
Authentication and authorization is the best way to provide access to the resources as per the user role.

Enable Security in “Manager Jenkins” section as shown below:

Lab:

From the Jenkins admin dashboard, we can configure the standard security settings for Jenkins, click on 'Manage Jenkins' and then 'Configure Global Security'.





Permissions are grouped into following categories:

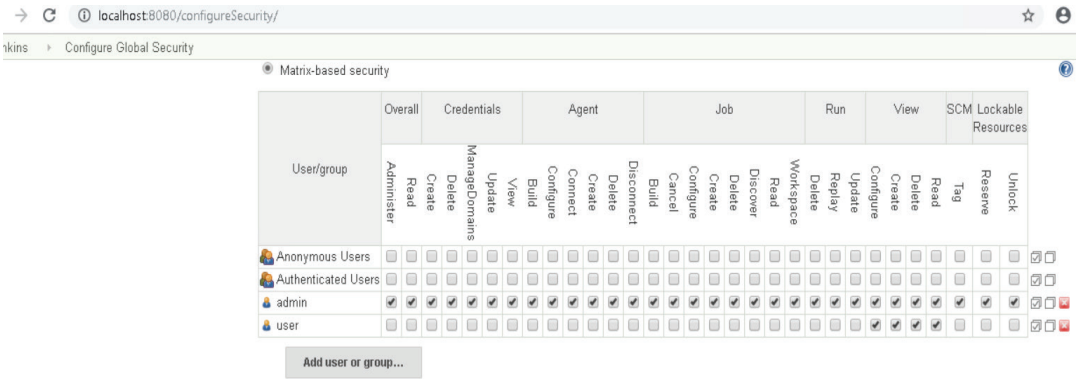
- Overall
- Credentials
- Agent
- Job
- Run
- View
- SCM

Create a test Jenkins user from this link: *http://localhost:8080/securityRealm/addUser*

Jenkins provides several authorization methods in the 'Access Control' section. We will be using the 'Matrix-based Security', so we can control all user privileges.

Add the 'user' user at the box 'User/Group' and click add.

Give the 'user' user all privileges by checking all options, and click 'Save' button.



Please note you need to provide all the privileges to logged in/admin user, otherwise you cannot login to the user once you click on the save button.

You will be redirected to the dashboard, and if there is login option, just type your admin user and password.

Create sample users and provide only Job view privileges and Job created Privileges and login using the user credentials and test the behavior.

1.3 Installation and Configuration of git/Java/maven on Build server (Windows)

Let's prepare the build server for next sections. Install the below components on Jenkins Slave machine and master also. Even though we are not executing build on master, for practice purpose if you want to run builds on master server, you need to install all the below required components:

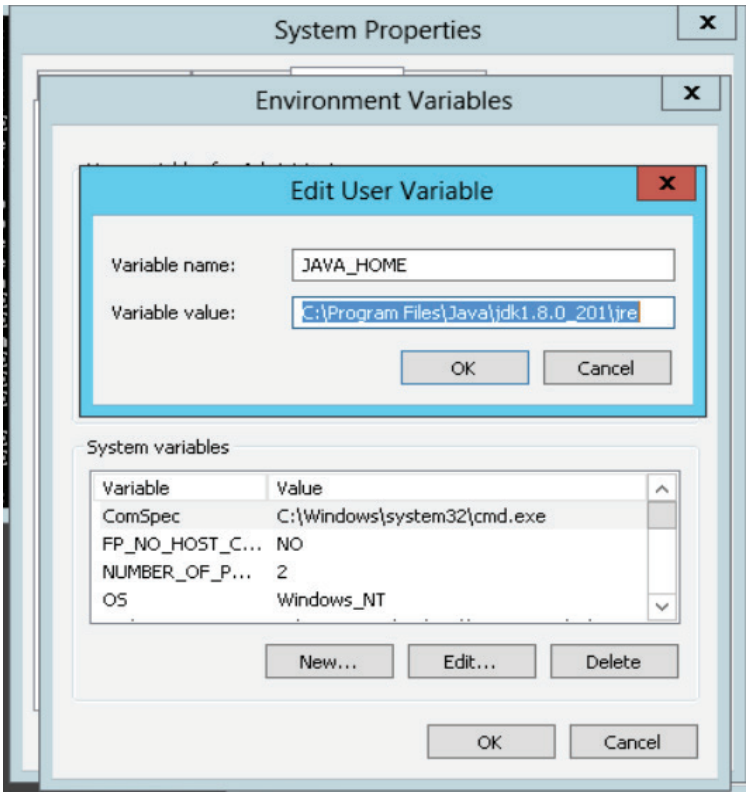
Lab:

Install and configure Java:

- Download Java from below link and install it

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

- Set JAVA_HOME under user variable section

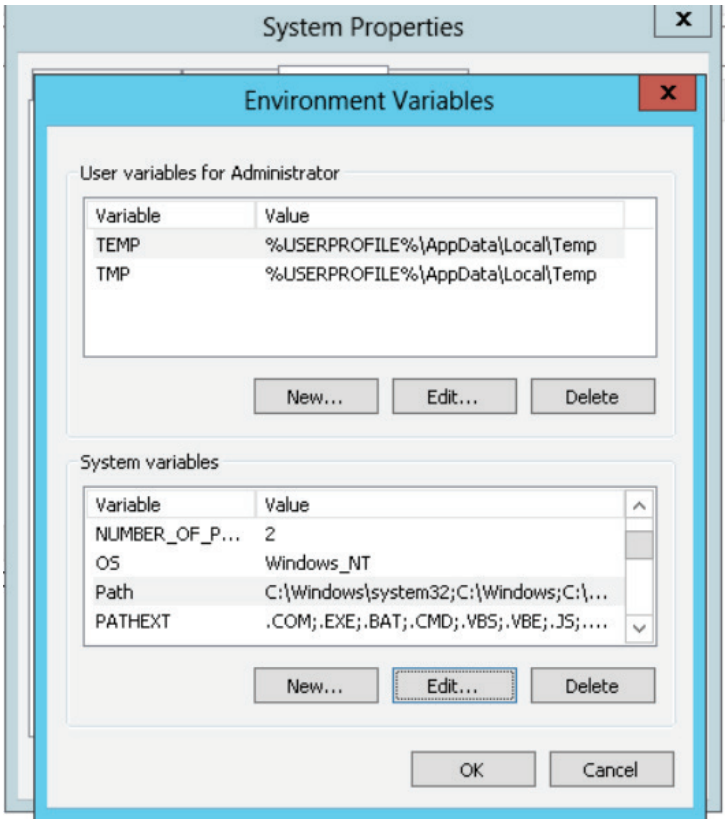


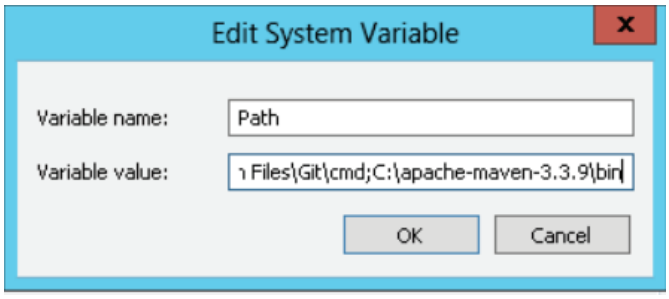
Configure Maven:

1. Download maven from this link -

https://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/

2. Unzip maven to a directory and add the User path variable





3. Download and Install Git from the below link based on your OS version

https://git-scm.com/downloads

Jenkins Configuration: (Windows/Ubuntu master node)

Understanding JENKINS_HOME directory:

The .jenkins is the main directory that contains all the details for Jenkins installation files, configurations, plugins, build job configuration, and so on.

It is important to keep the .jenkins directory at a location where a good amount of free space is available. By default, Jenkins creates .jenkins (or JENKINS_HOME) at a specific location considering the operating systems.

For example, in Windows it is available at C:\Users\<USER_NAME>\.jenkins.

In Ubuntu, go to /etc/default/Jenkins.

Let's discuss about the important directories present in the JENKINS_HOME directory.

config.xml	Jenkins root configuration file
fingerprints	It stores fingerprint records, if any
plugins	It is a root directory for all Jenkins plugins
jobs	It is a root directory for all Jenkins jobs
logs	It stores all log files
secrets	It is a root directory for the secret + key for credential decryption
users	It stores all user-related details in Jenkins
war	It stores all details related to the JENKINS_WAR file
workspace	It stores all the files and artifacts related to different build jobs, and it moves content to jobs directory when archiving elements.

1.4 Jenkins job, parameters, build, post-build actions and pipeline

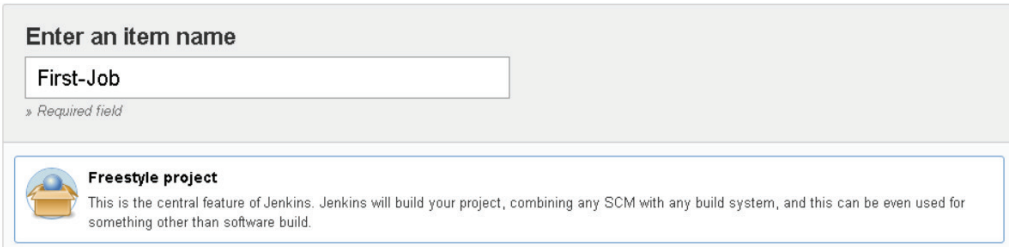
Create first Jenkins freestyle job:

The Jenkins freestyle job is the basic job type in Jenkins CI. _

Create a Jenkins freestyle job

Click on the New Item link in the top left-hand side.

In the first page enter – *First-Job* and then select “create a Freestyle project” and click on the OK button.

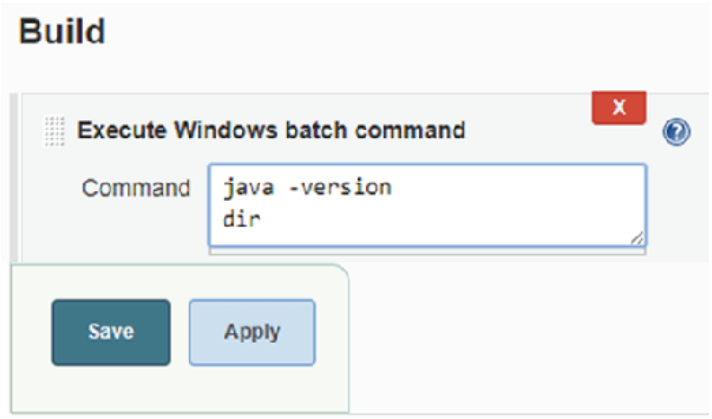


Once the job is created it will redirect to Job configuration page. Click on the button “Add build step" and select *Execute Windows batch command.*

In this example, we are executing a batch command to run java version and list the directory. The expected output will print java version that is installed on the system and the directory from where the job is executed.

The command to add looks as follows:

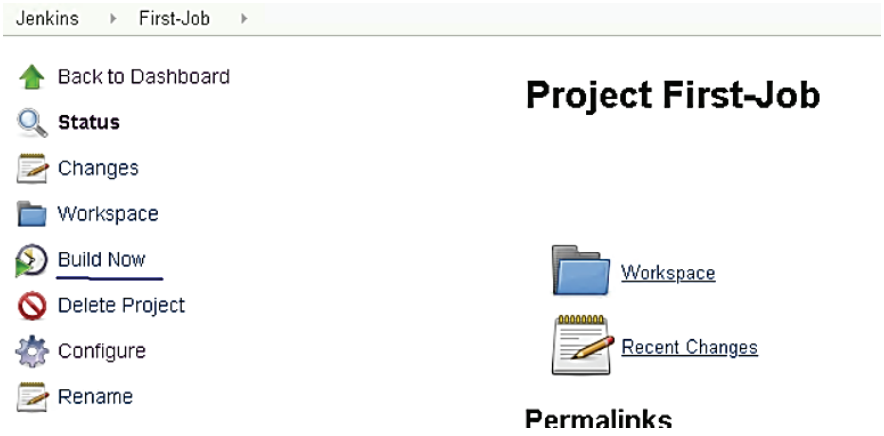
```
java -version
dir
```



Add batch and shell commands.

Run the Jenkins build job

It’s time to see the execution of freestyle Jenkins job that you just created. To do so, click on the Build Now option.



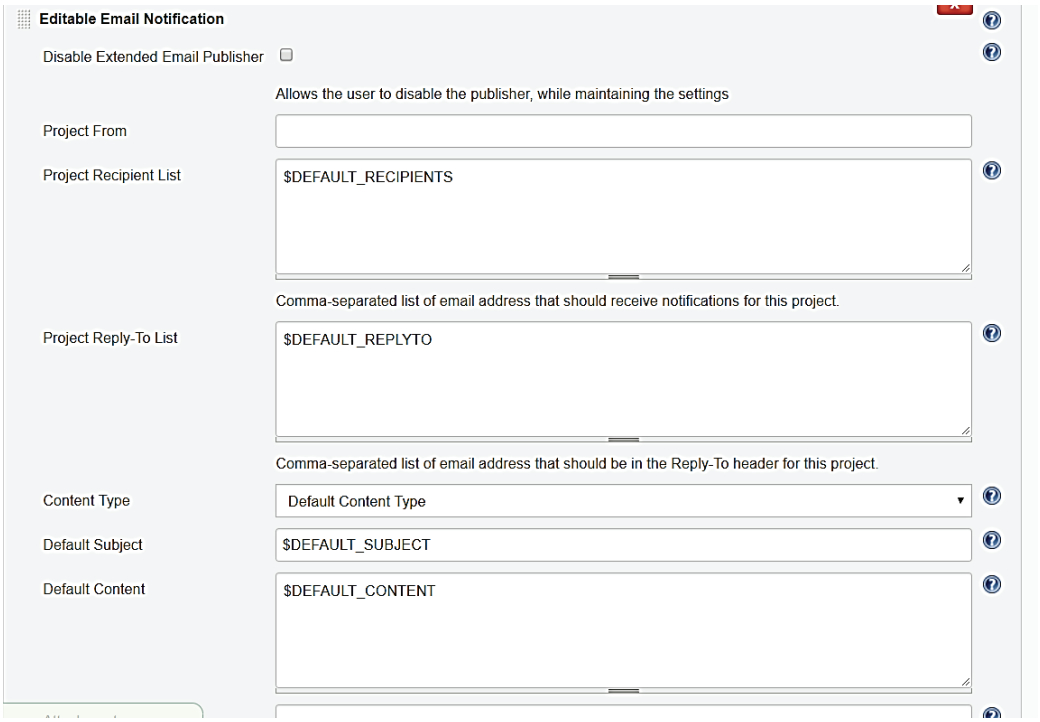
If the build is executed successfully, a new entry with Build number will appear in the Build History box, it starts with number one. Click on the number to view the build job output. In the console output you can see the results of running both the batch commands java -version command, as well as dir command.


```
Started by user admin
Building in workspace C:\Program Files (x86)\Jenkins\workspace\First-Job
[First-Job] $ cmd /c call C:\Windows\TEMP\jenkins4916819243220699 85.bat

C:\Program Files (x86)\Jenkins\workspace\First-Job>java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

C:\Program Files (x86)\Jenkins\workspace\First-Job>dir
 Volume in drive C has no label.
 Volume Serial Number is 4C39-A686

 Directory of C:\Program Files (x86)\Jenkins\workspace\First-Job

04/14/2019  03:55 PM    <DIR>          .
04/14/2019  03:55 PM    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)   2,087,763,968 bytes free

C:\Program Files (x86)\Jenkins\workspace\First-Job>exit 0
Finished: SUCCESS
```

Email notifications (Post Build action):

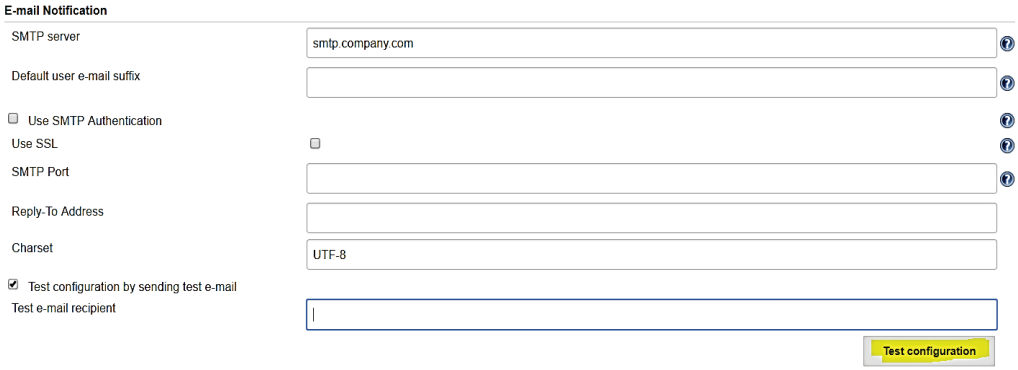
To enable email notification for every build follow the below procedure.

Go to Manage Jenkins -> Manage Plugins -> Available and install “Email extension Template”

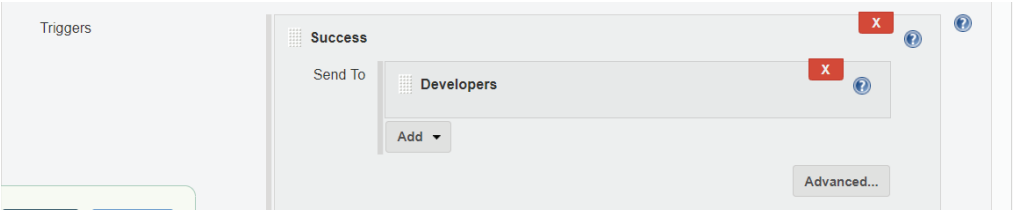


$DEFAULT_RECIPIENTS and $DEFAULT_REPLYTO can be defined in email settings

If Default recipients is not configured in settings page, you can type email address separated by space in the Email settings.



Click on Advanced at the bottom and select the email trigger:



As per the email template the developer will receive an email if this Job is built successfully every time.

1.5 Jenkins Agent/Slave configuration with Windows/Ubuntu master hosts

Requirement:

Jenkins Master – Windows / Ubuntu 18.0+

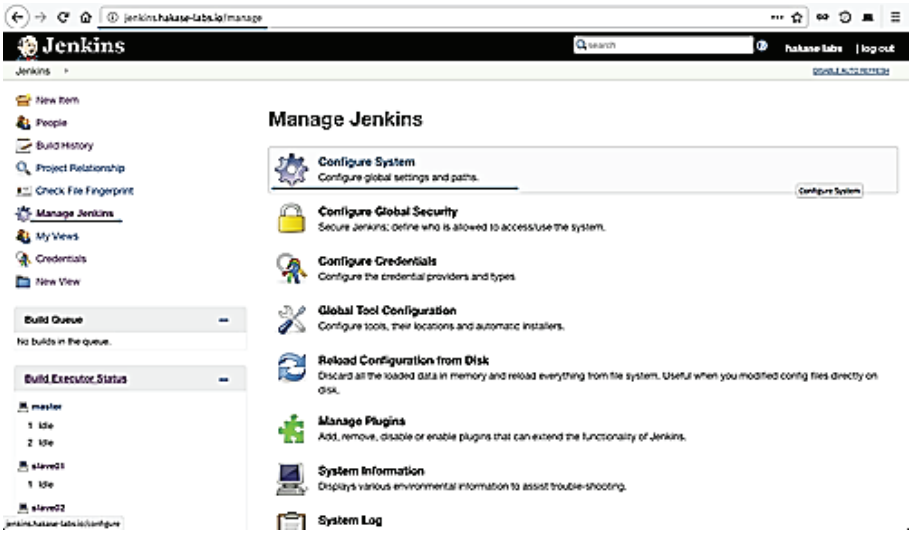
Jenkins Slave (Build server) – CentOS 7/Ubuntu 18.0 + (JDK 1.8+ and maven 3.3.9)

Jenkins Slave (Deployment) - CentOS 7/Ubuntu 18.0 + (JDK 1.8+)

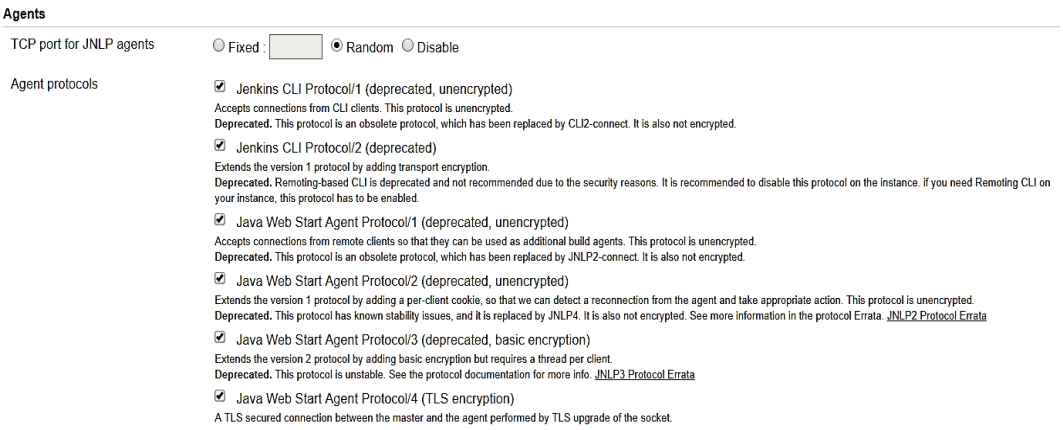
Configure Slave Agent Nodes to Execute Build (Windows Master)

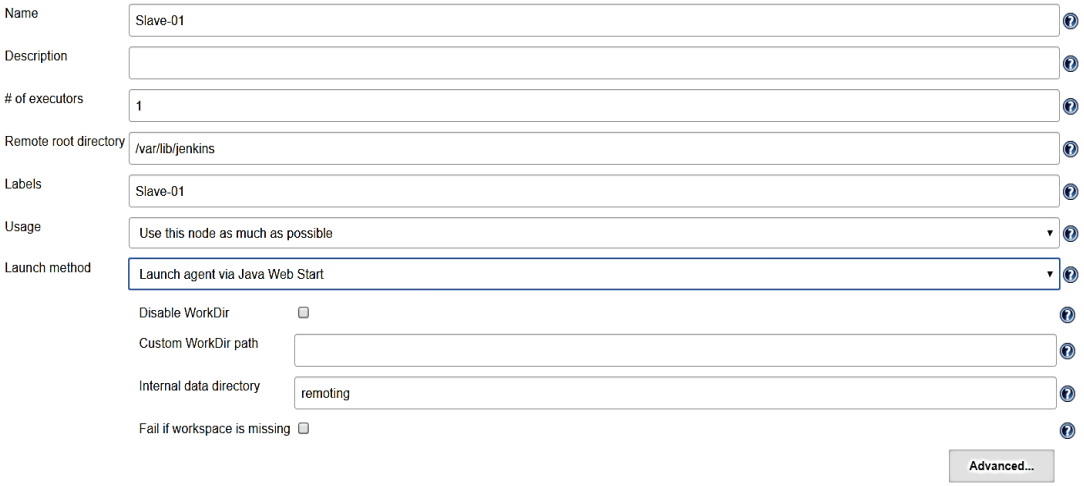
In this step, we will configure the build to execute on the slave agent.

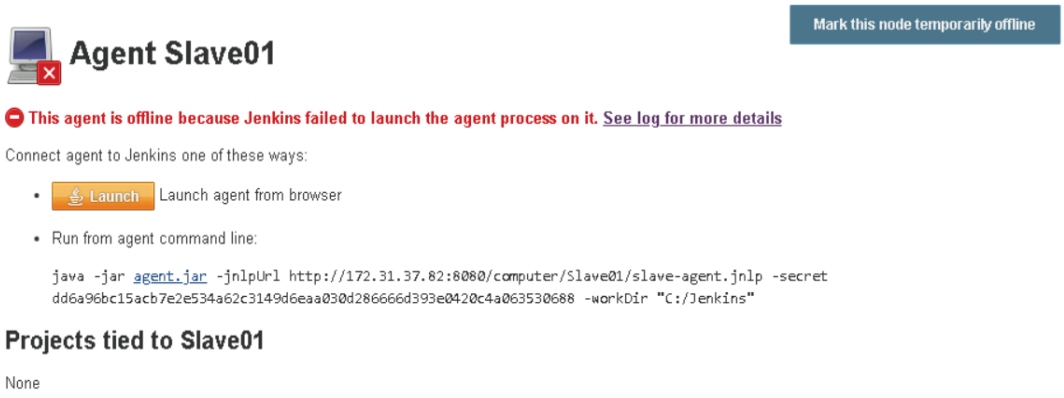
Click on the 'Manage Jenkins' menu -> ‘Configure System'.



From Windows master the best way to connect to a slave is to connect Slave nodes to master node using Java web start method. To enable, navigate to **Manage Jenkins -> Configure Global security** and enable all the options similar to below screen







Download agent.jar and copy it to Slave node and run the above command from command prompt. This will connect the agent to master node.

Jenkins Master (Ubuntu 18.0 +) – Please ignore this step if you would like to use Windows as Jenkins master host

Step 1 - Configure Jenkins Master Credentials (Ubuntu 18.0 + Optional step)

When the master Jenkins server is installed, you need to configure the master server itself. There are different ways to configure and start Jenkins agent nodes. The agents can be launched through SSH, windows administrative account, and via Java Web Start (JNLP), you can pick the best way depending on your infrastructure, environment setup and operating system.

For this exercise, we will launch the agent nodes through ssh, and we need to setup Jenkins credentials on our master server.
Generate SSH Key

We will use the ssh key authentication to setup our jenkisn agent nodes, First step is to generate the ssh key for the Jenkins user and upload the key to each server node manually using 'ssh-copy-id'.

On the Jenkins master server, login to the Jenkins user and generate the ssh key.

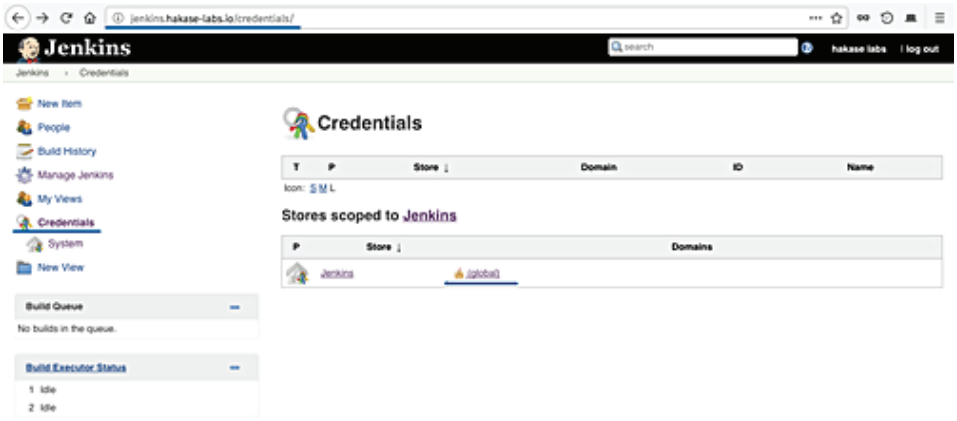
```
su - jenkins
ssh-keygen
```

And you will get the 'id_rsa' private and 'id_rsa.pub' public key in the '.ssh' directory.


```
root@master:~#
root@master:~# su - jenkins
jenkins@master:~$
jenkins@master:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa):
Created directory '/var/lib/jenkins/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/jenkins/.ssh/id_rsa.
Your public key has been saved in /var/lib/jenkins/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:PorTnAJbfKXNqRdkNmPAI1dBS1sziWLk4+j+1GtkBY4 jenkins@master
The key's randomart image is:
+---[RSA 2048]----+
|     o.o=o+.     |
|    ..B.o+.o     |
|     +o*o.       |
|     oE.X .      |
|   .. .XS=       |
|  ..o ooB        |
|   +.+.=+.       |
|  ..oo=.oo       |
|    o+oo.        |
+----[SHA256]-----+
jenkins@master:~$
jenkins@master:~$ ls -lah ~/.ssh/
total 16K
drwx------  2 jenkins jenkins 4.0K Jun 12 09:19 .
drwxr-xr-x 15 jenkins jenkins 4.0K Jun 12 09:19 ..
-rw-------  1 jenkins jenkins 1.7K Jun 12 09:19 id_rsa
-rw-r--r--  1 jenkins jenkins  396 Jun 12 09:19 id_rsa.pub
jenkins@master:~$
jenkins@master:~$ _
```

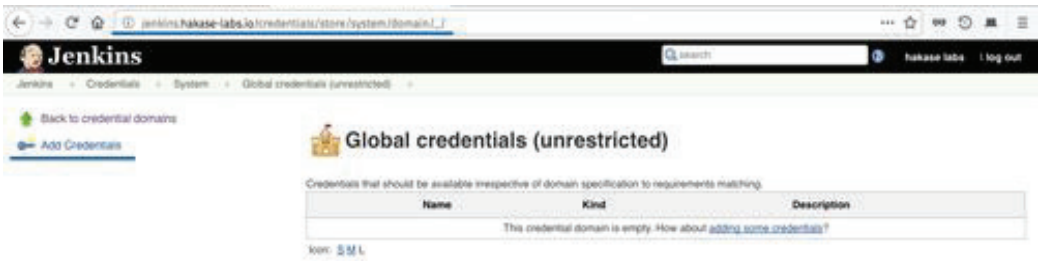
Setup Credentials on Jenkins

Open your Jenkins dashboard and click on the 'Credentials' menu on the left.



And click the 'global' domain link.

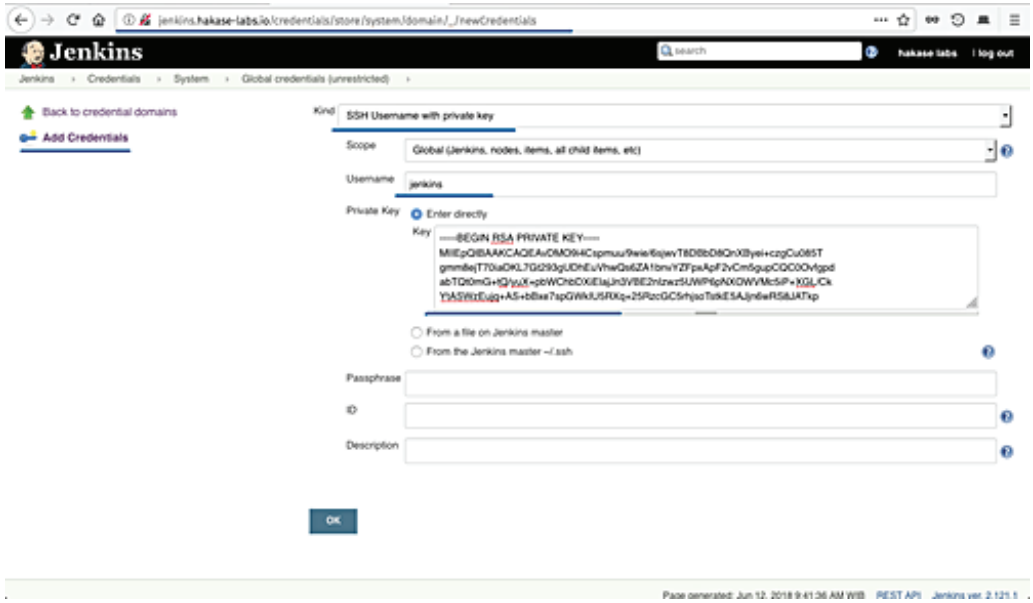
Now click 'Add Credentials'.



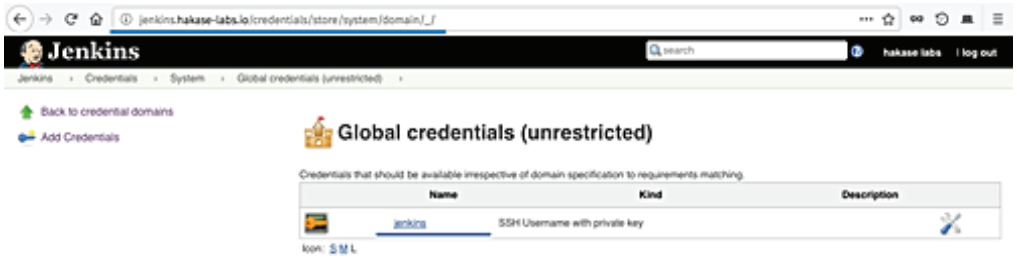
Now choose the authentication method.

- Kind: SSH Username with private key
- Scope: Global
- Username: jenkins
- Private key: Enter directly and paste the 'id_rsa' private key of Jenkins user from the master server.

Click 'OK'.



And the Jenkins credential with ssh auth key method have been created.



Set up Slave Nodes

Now we will setup slave nodes server by installing java on those server, and create a new Jenkins user.
Install Java

Install the 'software-properties-common' packages and add the java PPA repository.

```
sudo apt install software-properties-common apt-transport-https -y
sudo add-apt-repository ppa:openjdk-r/ppa -y
```

Now install java OpenJDK using apt command below.

```
sudo apt install openjdk-8-jdk -y
```

After the installation is complete, check the installed java version.

```
java -version
```

And you will get Java OpenJDK 1.8 installed on the system.

```
root@slave01:~#
root@slave01:~# java -version
openjdk version "1.8.0_171"
OpenJDK Runtime Environment (build 1.8.0_171-8u171-b11-0ubuntu0.18.04.1-b11)
OpenJDK 64-Bit Server VM (build 25.171-b11, mixed mode)
root@slave01:~#
root@slave01:~#
```

Add New Jenkins User

Now add the 'Jenkins' user to all agent nodes.

Run the command below.

```
useradd -m -s /bin/bash Jenkins
passwd Jenkins
```

The 'Jenkins' user for agent nodes has been created.

```
root@slave01:~#
root@slave01:~# useradd -m -s /bin/bash jenkins
root@slave01:~# passwd jenkins
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@slave01:~#
root@slave01:~#
```

Copy the SSH Key from Master to Slave if the Jenkins master is Ubuntu machine

Next, we need to upload the key 'id_rsa.pub' from the master to slave server nodes. We need to upload to the node agent using 'ssh-copy-id' command as below.

```
ssh-copy-id jenkins@10.x.x.x
```

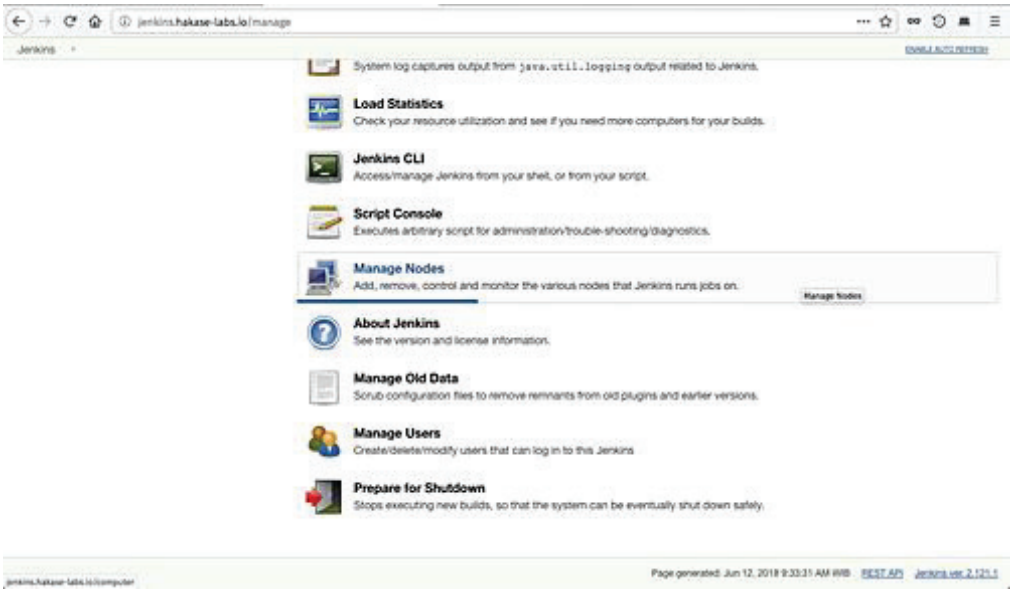
Type the Jenkins user password.

The ssh key 'id_rsa.pub' has been uploaded to all agent nodes.

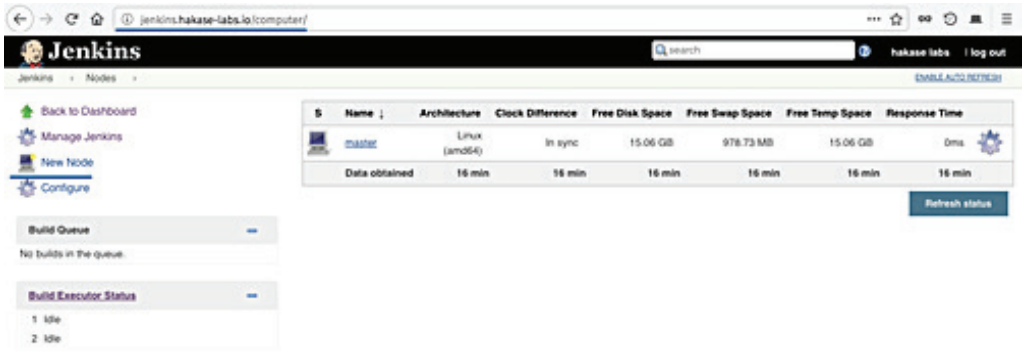

```
root@master:~#
root@master:~# su - jenkins
jenkins@master:~$
jenkins@master:~$ ssh-copy-id jenkins@10.0.15.21
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/var/lib/jenkins/.ssh/id_rsa.pub"
The authenticity of host '10.0.15.21 (10.0.15.21)' can't be established.
ECDSA key fingerprint is SHA256:jhNSnee5eer9c773jYabt9Ahp9SfWI2nqpQ+2uDVOfY.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
jenkins@10.0.15.21's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'jenkins@10.0.15.21'"
and check to make sure that only the key(s) you wanted were added.

jenkins@master:~$
jenkins@master:~$ ssh 'jenkins@10.0.15.21'
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.15.0-20-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Add New Slave Nodes

On the Jenkins dashboard, click the 'Manage Jenkins' menu, and click 'Manage Nodes'.



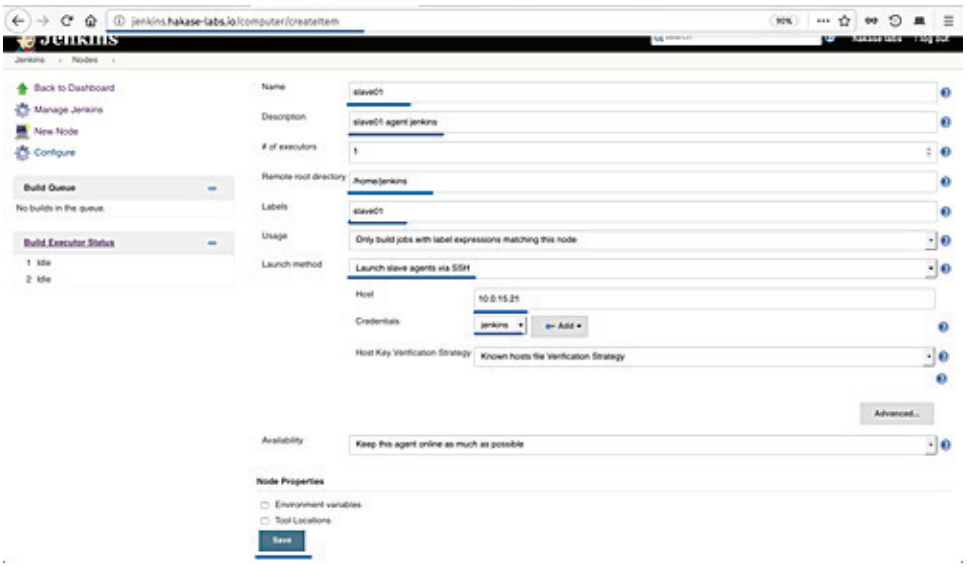
Click the 'New Node'.



Type the node name 'slave01', choose the 'permanent agent', and click 'OK'.

Now type node information details.

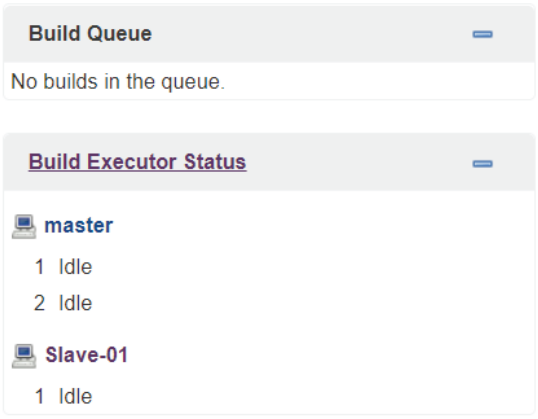
- Description: Slave-01 node agent server
- Remote root directory: /var/lib/jenkins
- Labels: Slave-01
- Launch method: Launch slave agent via SSH, type the host ip address '10.x,x.x', choose the authentication using 'Jenkins' credential.



Now click 'Save' button and wait for the master server to connect to all agent nodes and launch the agent services.

Below are the results when the master server is connected to all agent nodes.

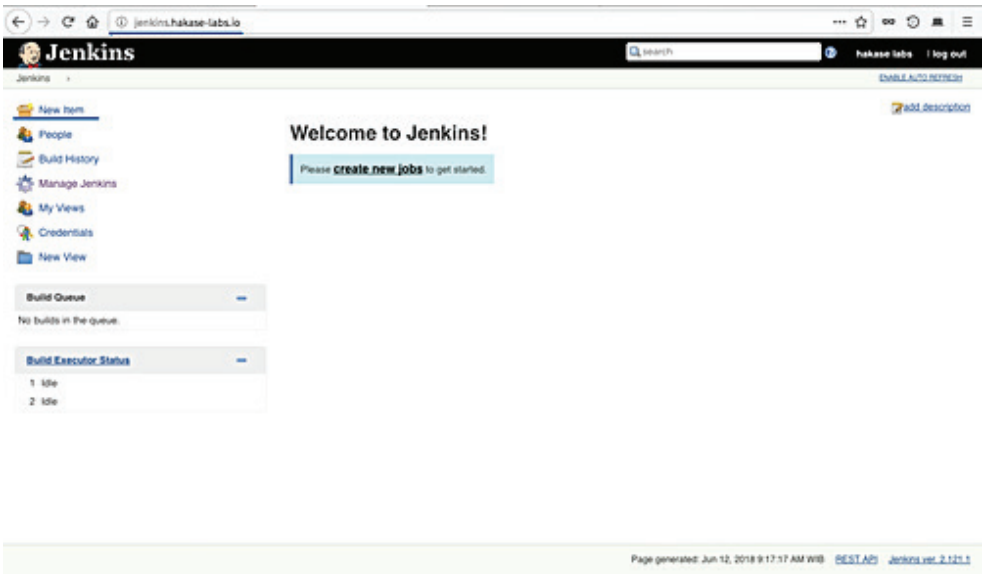
```
SSHLauncher{host='[redacted]', port=22, credentialsId='Build-CentOS', jvmOptions='', javaPath='', prefixStartSlaveCmd='', suffixStartSlaveCmd='',
launchTimeoutSeconds=210, maxNumRetries=10, retryWaitTime=15,
sshHostKeyVerificationStrategy=hudson.plugins.sshslaves.verifiers.ManuallyTrustedKeyVerificationStrategy, tcpNoDelay=true, trackCredentials=true}
[04/21/19 14:04:54] [SSH] Opening SSH connection to [redacted]:22.
[04/21/19 14:04:55] [SSH] SSH host key matches key seen previously for this host. Connection will be allowed.
[04/21/19 14:04:56] [SSH] Authentication successful.
[04/21/19 14:04:57] [SSH] The remote user's environment is:
```



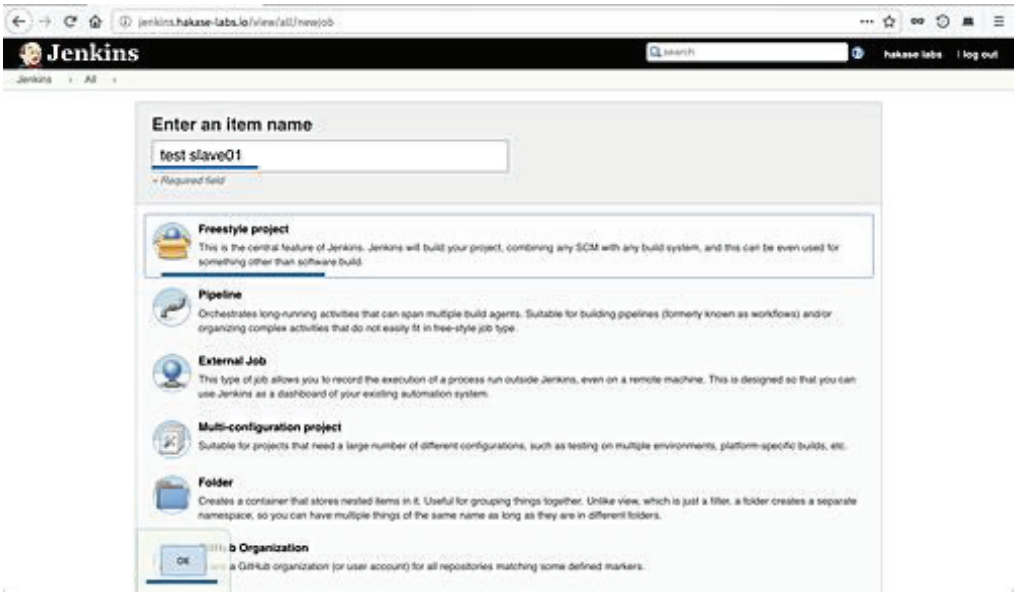
Slave node is added to the Jenkins master server.

Now we want to create a new simple build for Jenkins, and we want to execute the build on the bot 'slave01' agent node.

On the Jenkins dashboard, click the 'New Item' menu.

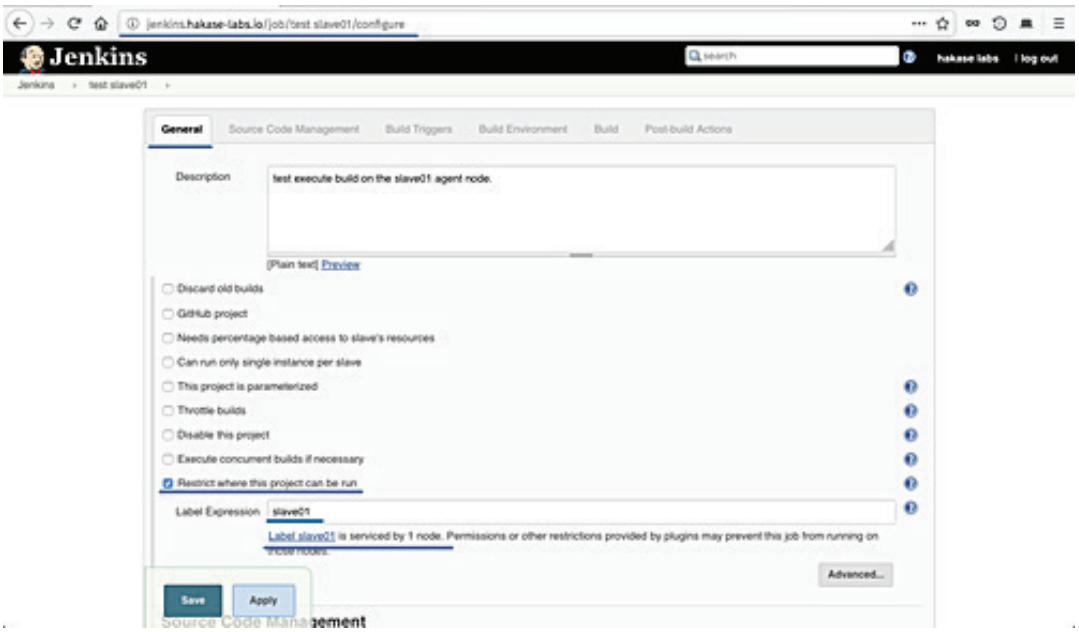


Type the item name, choose the freestyle project, and click 'OK'.



In the general section, enter the job description and select the 'Restrict where this project can be run' option.

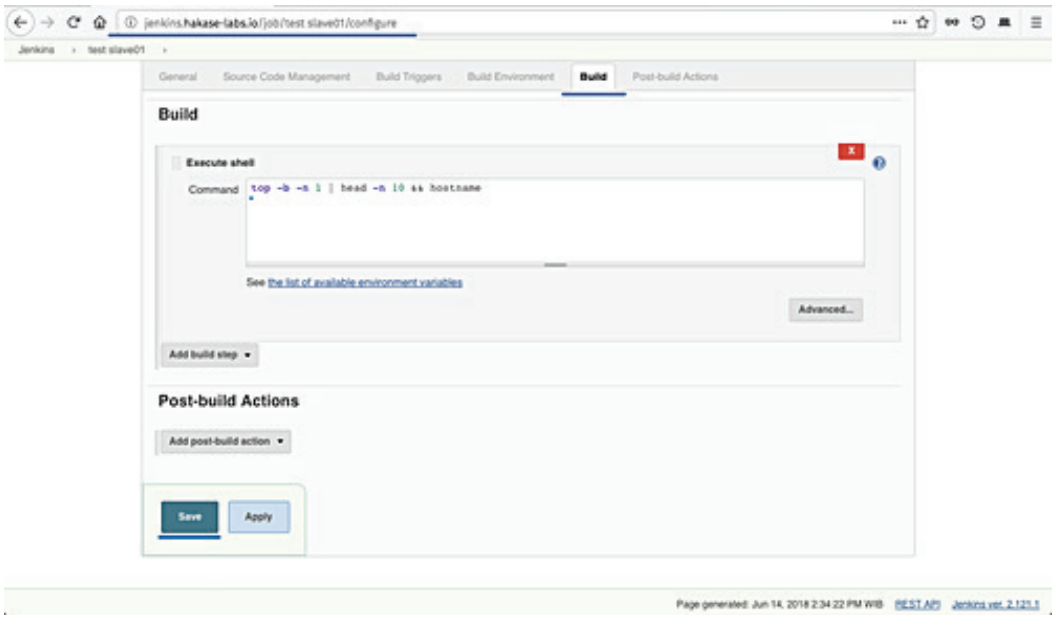
In the 'Label Expression', enter the node as 'Slave-01'.



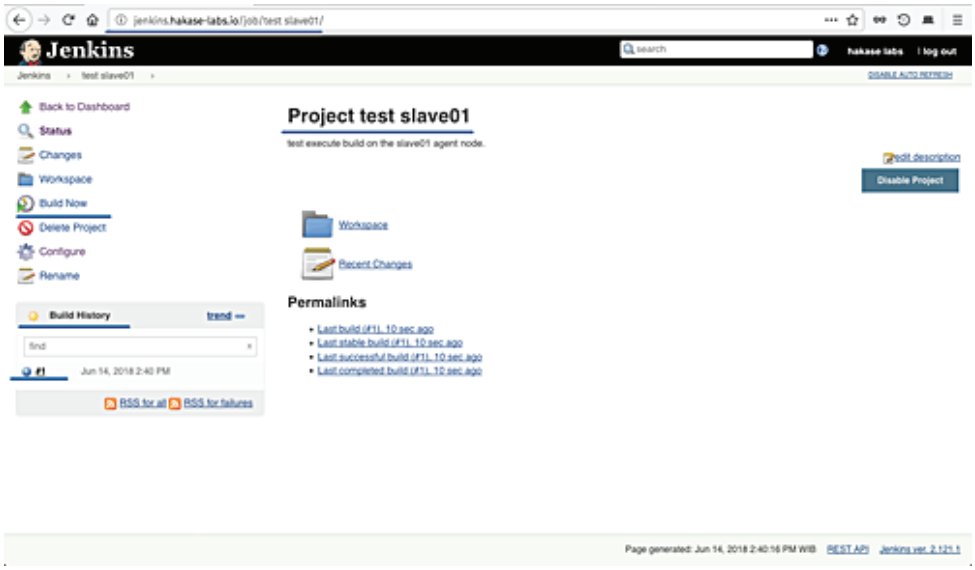
Move to the build section and choose the 'Execute shell' option, type the command as below.

```
top -b -n 1 | head -n 10 && hostname
```

Click 'Save' button, and you will be redirected to the job page.

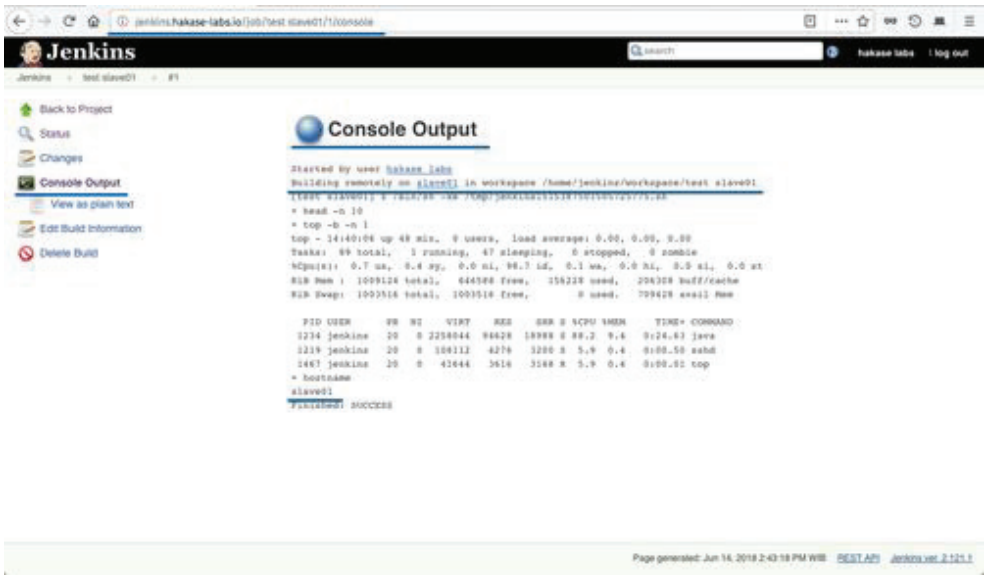


Click the 'Build Now' to build the project, and then click item on the 'Build History' section.



And the following is my result.

Build on the 'slave01' agent node.



Installation and configuration of Jenkins master/slave architecture and the distributed builds Jenkins has been completed successfully.

1.6 Configuring Jenkins with git plugin

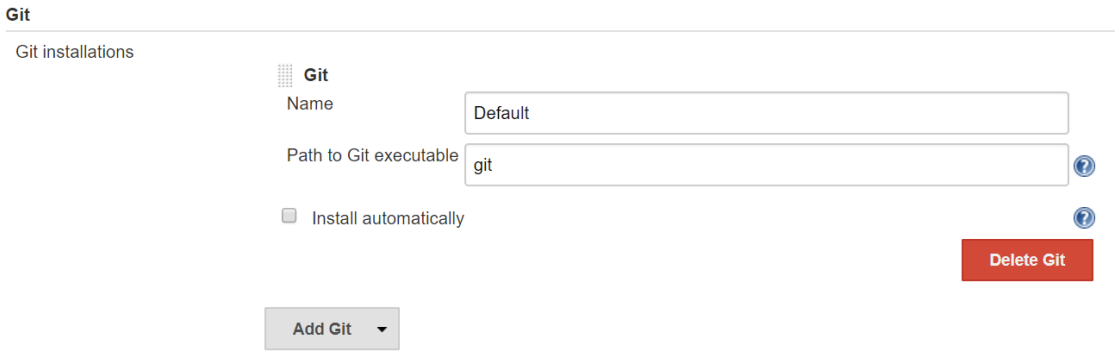
Jenkins can be integrated with many source code repositories, and Git is one of them. Let's consider the case of an application that is Java-based and the source code is stored in Git, Gitlab, and/or GitHub.

We need to tell Jenkins where Git is installed in the local system.

How to do it...

- Open the Jenkins dashboard.
- Go to Manage Jenkins.
- Go to Global Tool Configuration to configure the tools, their locations, and automatic installers.
- Go to the Git section.

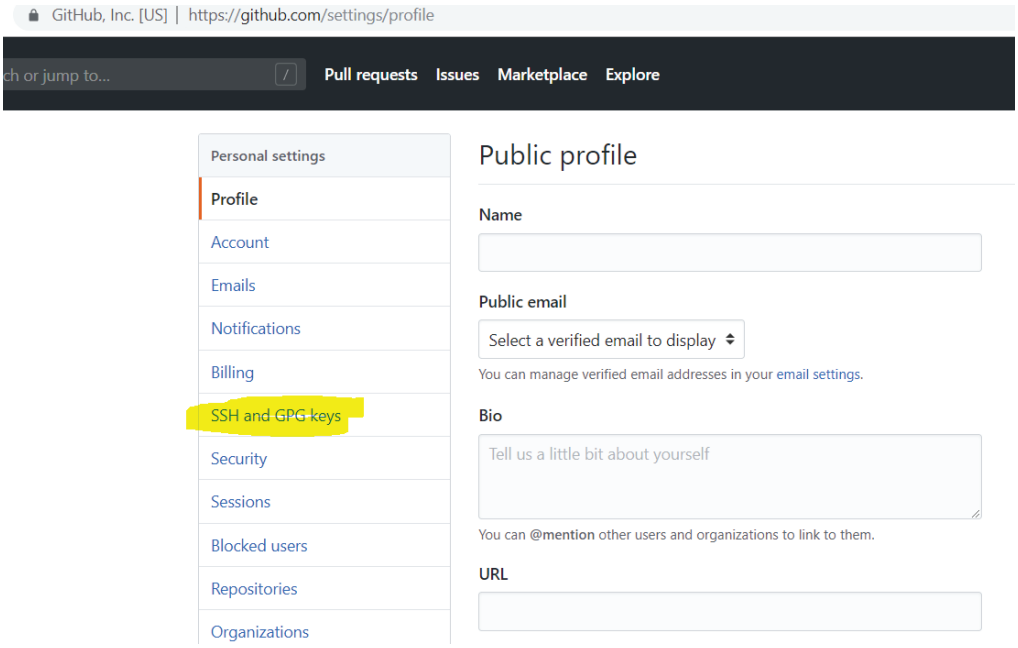
- Give the Name and click on Install Automatically, or provide a path to Git.
- You can add multiple Gits based on the version or for the specific agent. You need to give a meaningful name so it can be identified easily while configuring the build job:



In order to connect to github without entering username/password every time, generate SSH key from your slave machine/other machine

- Open Git bash window and type > sshkeygen –t RSA
- This utility will create a public and private key in C:\Users\user\.ssh\id_rsa.pub
- Copy the contents of id_rsa.put file to SSH keys section of your github account

Navigate to github.com/settings/profile -> SSH and GPG Keys -> New SSH Key and copy id_rsa.pub from your machine to this window.



For more information refer - *https://help.github.com/en/articles/connecting-to-github-with-ssh*

1.7 Creating a Java application (V 1.0)

- Clone a Java web application using this link

Repository URL: *https://github.com/mgna-repo/webapp.git*

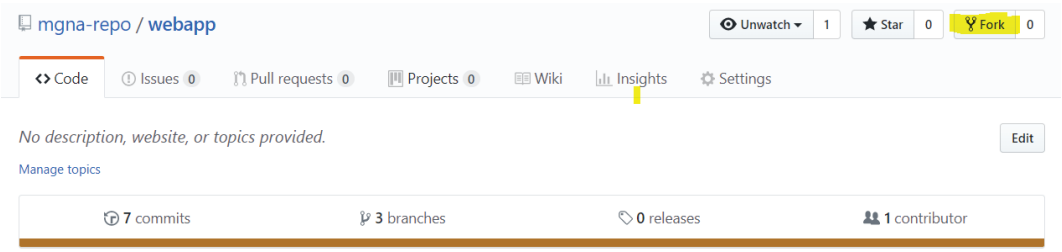
This is a simple java based web application that will start print a message in the browser.

The repository has 3 branches – master, develop and feature.

The current development work can be carried out in develop branch and minor feature related work can be carried out in feature branch. The code from feature branch should be merged to develop branch in regular intervals.

At the end of the release the code from develop branch should be merged into master branch and create a tag.

Open the above URL in a browser and clone the repository. So that you are not making changes to the original repository and you can work on your fork.



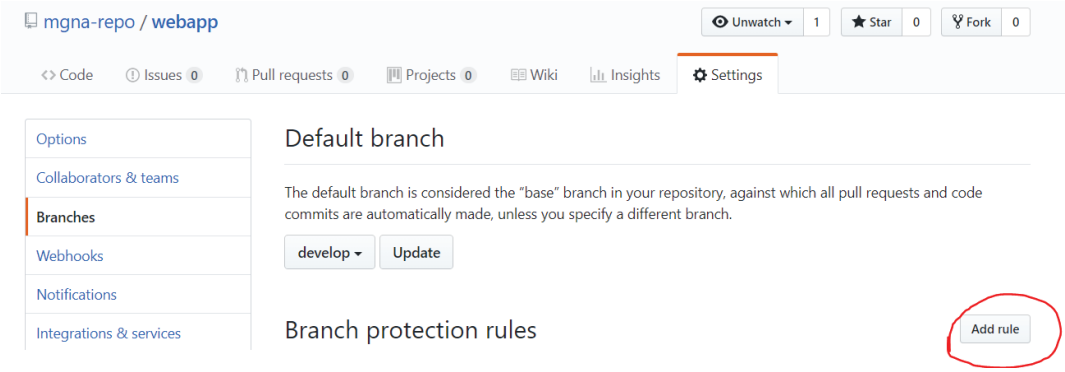
Click on Fork button and select your name. The repository will be forked on your name.

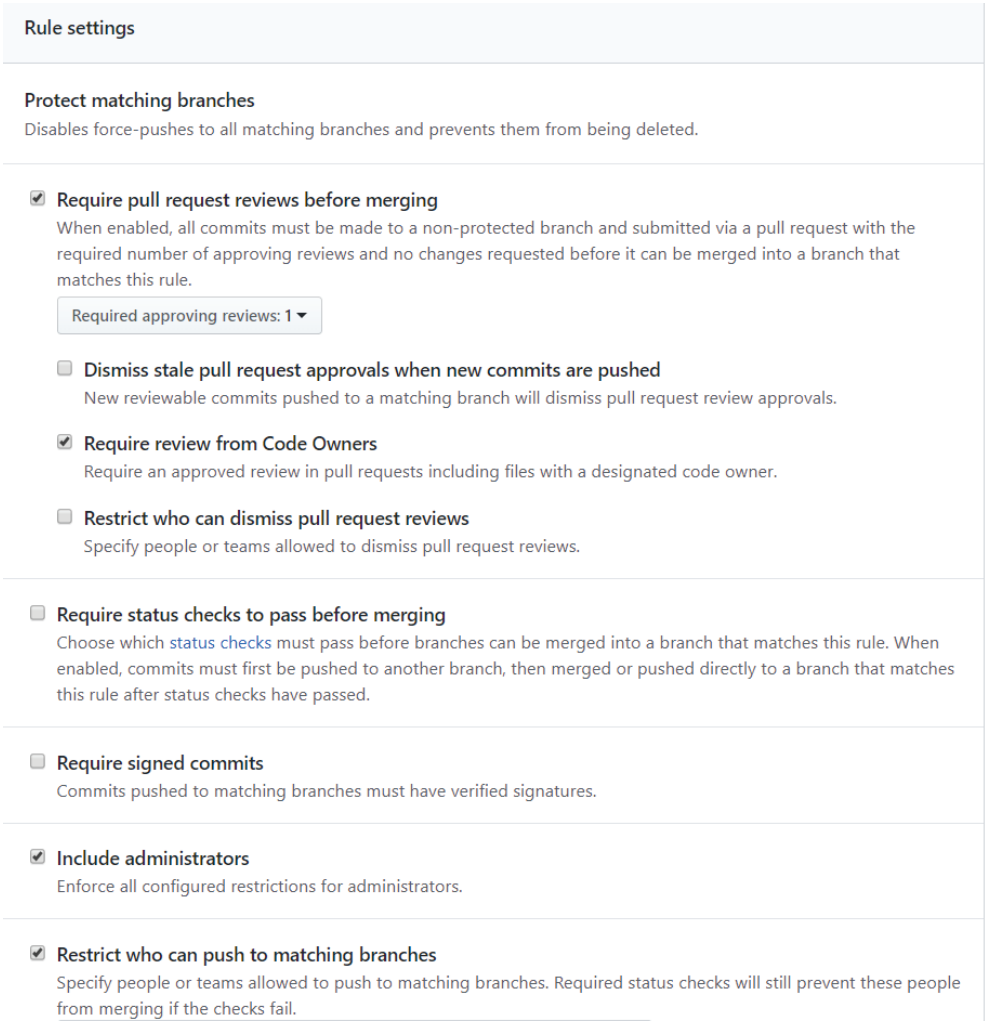
A **fork** is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, **forks** are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

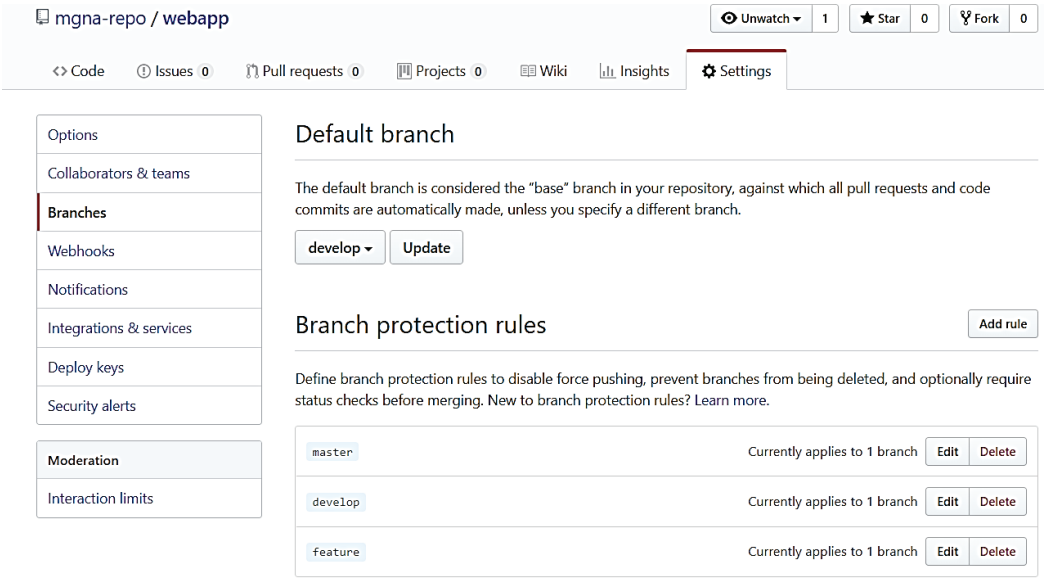
Please note the above git branch has 3 repos and all are protected. You need to work on your fork and raise a pull request to update the main repository.

Jenkins or other build tools will use the main repository link to build your project.

Below is the screenshot of branch protection settings:

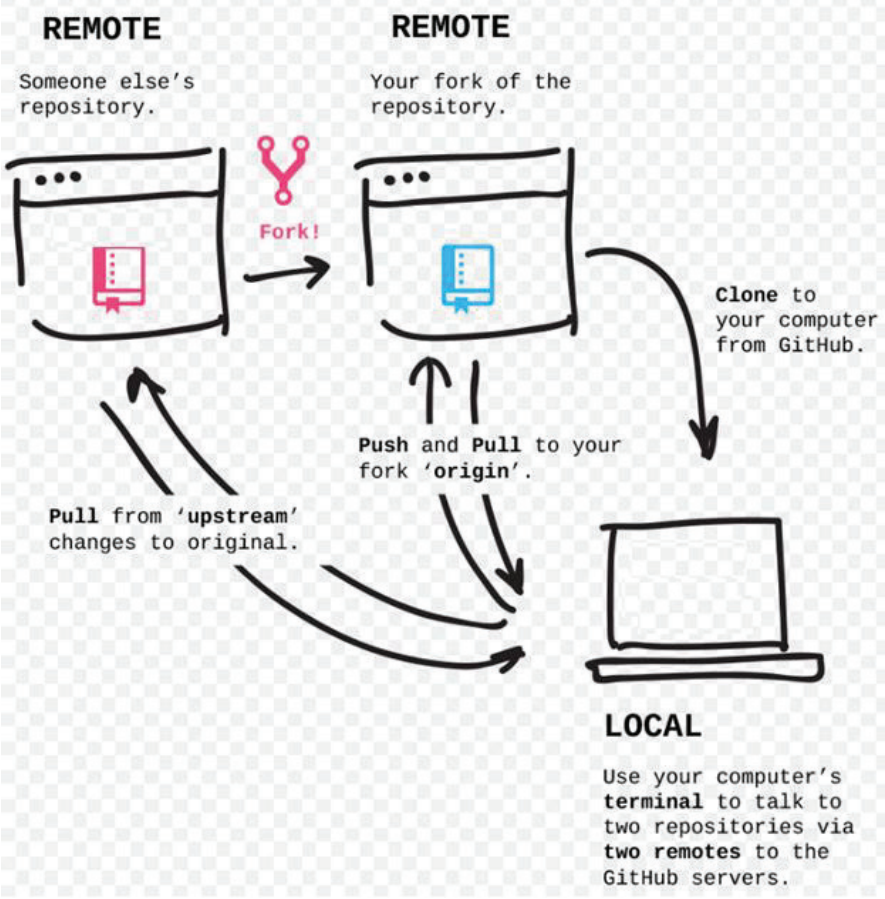






Note: The above branch protection settings will be applied only to the main repository and not the fork.

In any organization there will be a single repository for the application and multiple forks.

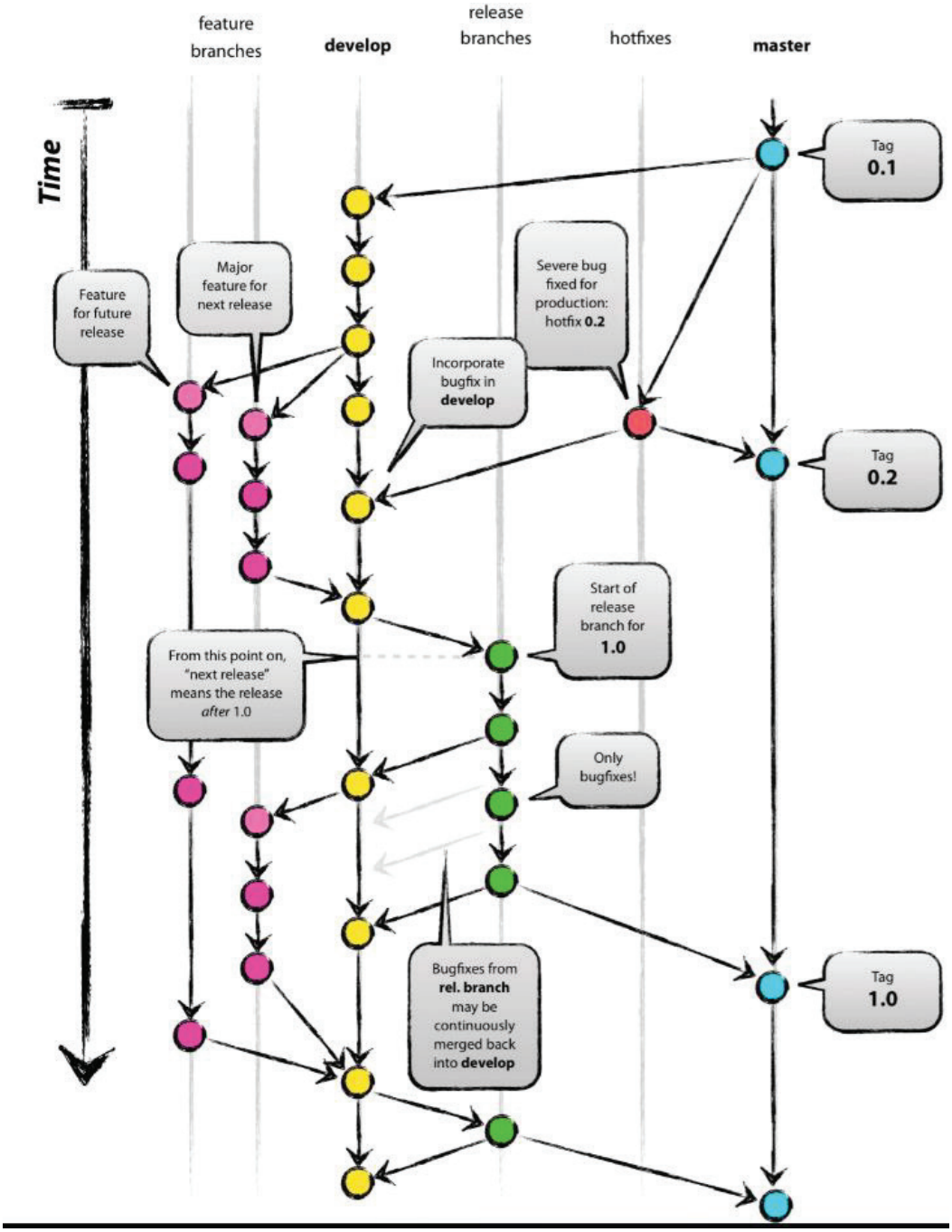


In the above picture fork concept is explained in detail.

Origin: Your forked repository

Upstream: Main repository where the code can be merged through pull requests from your fork

Below is an overview of successful git branching strategy:



May branch off from:

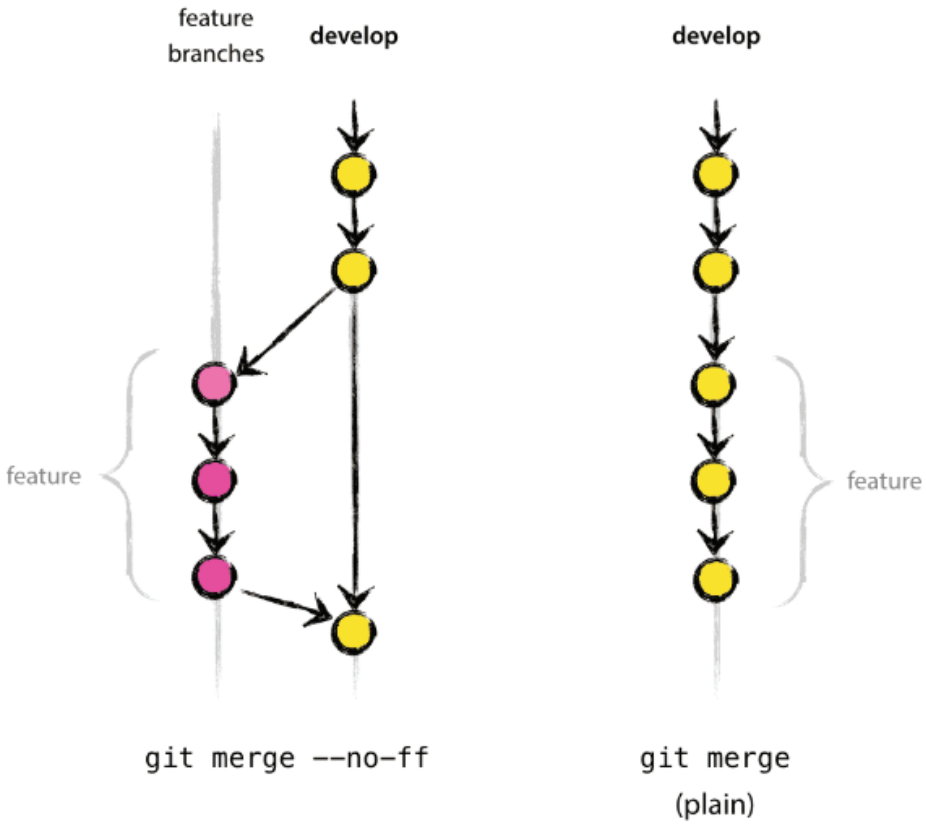
develop

Must merge back into:

develop

Branch naming convention:

anything except master, develop, feature-*, release-*, or hotfix-*



Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a

feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop(to definitely add a new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Feature branches typically exist in developer repos only, not in origin.

- Visit the Plugin tutorial, *https://wiki.jenkins.io/display/JENKINS/Plugin+tutorial* for more details.

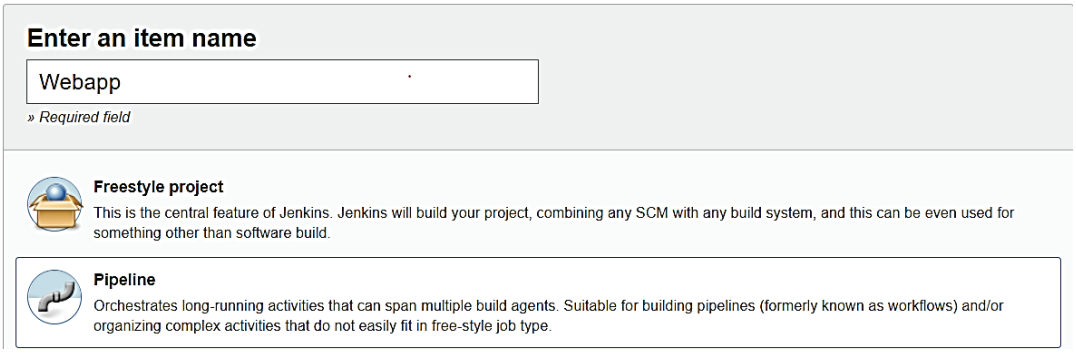
1.8 Create a new Jenkins pipeline

Jenkins Pipeline is a combination of plugins that support the integration and implementation of continuous delivery **pipelines** using **Jenkins**. A **pipeline** has an extensible automation server for creating simple or complex delivery **pipelines** "as code," via **pipeline** DSL (Domain-specific Language).

Sections in Jenkins Pipeline:

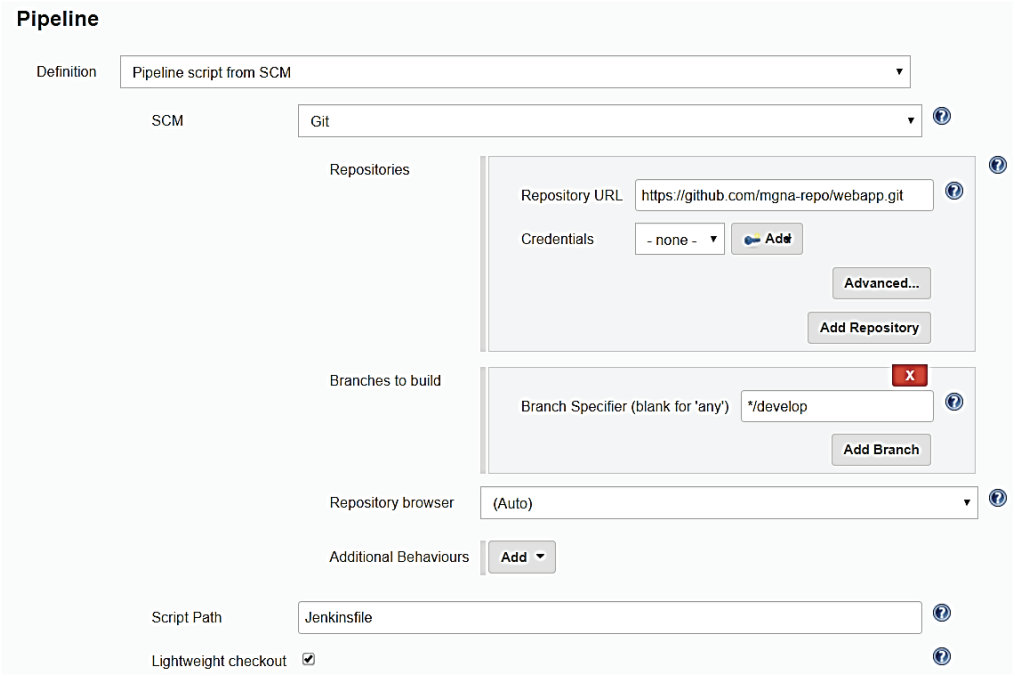
- **agent** — Specifies where the Pipeline or a specific **stage** executes
- **stage** — A conceptually distinct subset of the Pipeline, such as "MyBuild", "MyTest",
- **steps** — A series of distinct tasks inside a stage.

Create a new Jenkins Pipeline job:



Click save after typing in the name selecting the Pipeline

Configure Pipeline job settings

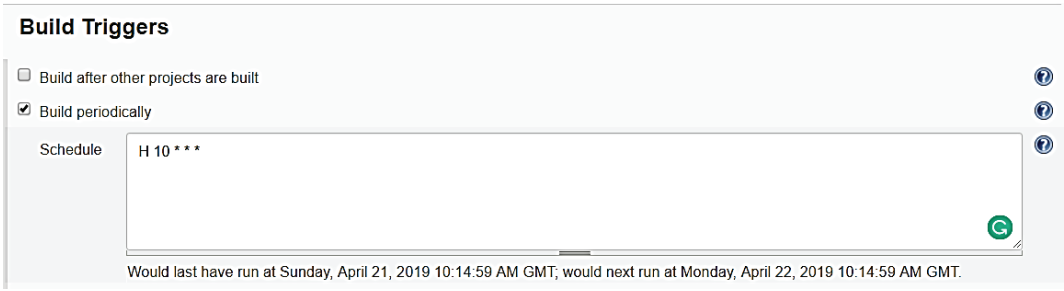


Note: In the above screen provide your forked repository

Example: *https://github.com/abhishek/webapp.git*

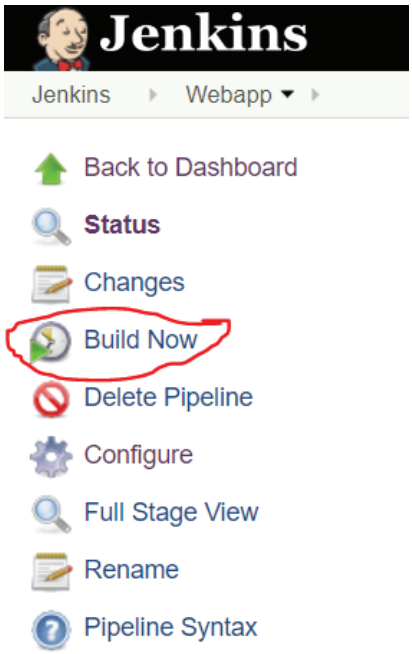
Fork can be deleted at any point of time and recreated again.

Configure Build trigger to scan the repository periodically as below:

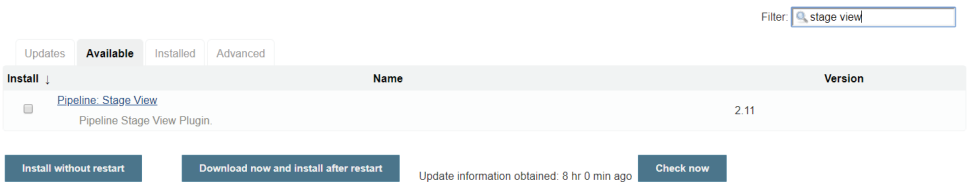


Click Save and click on Build Now and review the output.

Click on Build now to build the web application:



Install Stage View Plugin to review the Build output in stages.

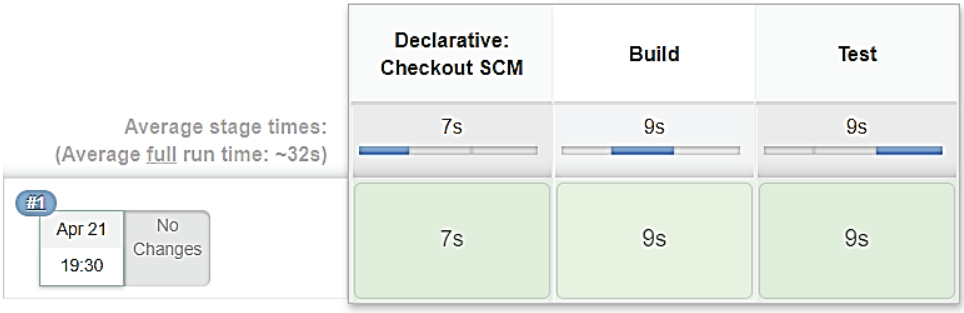


Once the Stage View plugin is installed click on the job and the stages will be displayed like below:

Pipeline Pipeline-Job

Recent Changes

Stage View



In this pipeline Job we are building a sample web application and running a Junit test case to validate if the web server is returning correct output as per the test case.

Generate Sonarqube report for the Project:

```
stage('Sonar-Report') {
   steps {
       sh 'mvn clean install sonar:sonar -Dsonar.host.url=http://[redacted]:9000 -Dsonar.analysis.mode=publish'
   }
}
```

stage('Sonar-Report') {

steps {

sh 'mvn clean install sonar:sonar -Dsonar.host.url=http://10.x.x.x:9000 -Dsonar.analysis.mode=publish'

}

```
}
```

Please note this step need a running Sonarqube server.

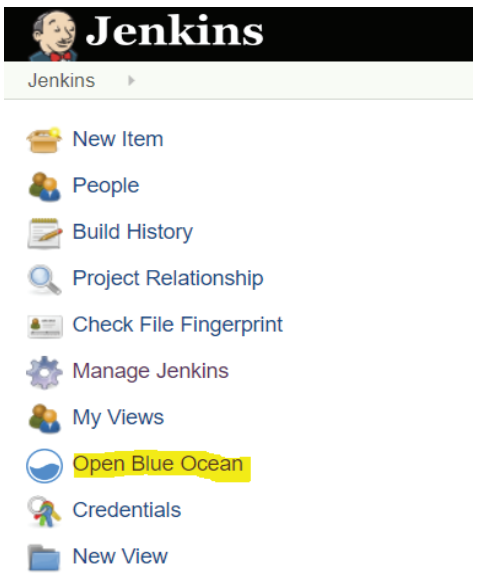
After saving Jenkins file click save and run the job again.

Declarative: Checkout SCM	Build	Test	Sonar-Report	Deploy
5s	9s	9s	1min 24s	3s
5s	9s	9s	1min 24s	3s

Jenkins pipeline to poll the feature branch

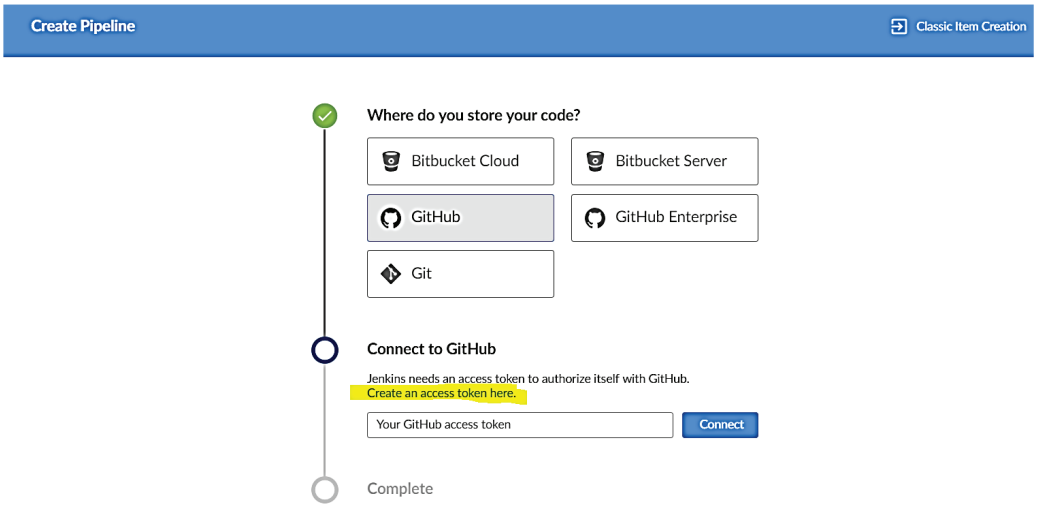
To use multi-branch pipeline feature we need to install Blue Ocean Plugin.

Install Blue Ocean Plugin from Manage Plugins section. After installing the Plugin click on Blue Ocean plugin



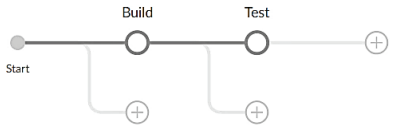
Click on New Pipeline

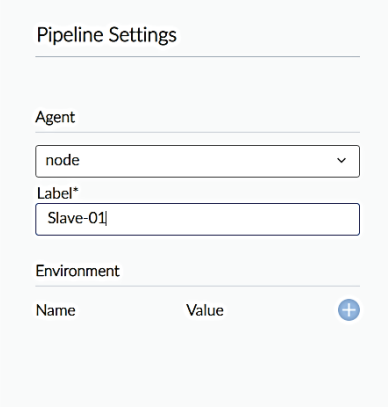
Select Github



Create access token by following the instructions in the above screen.

- Under organization select mgna-repo
- Select webapp as your repository
- Click on create Pipeline
- Select agent Slave-01 and click save





The pipeline will build all the branches as per the instructions in Jenkinsfile for that particular branch.

After the successful execution of the build Open a browser and check the result of the below links:

http://<slave-01>:9999

Hello!!! My version is 1.0 and I am built from Develop branch on port 9999!

http://<slave-01>:9997

Hello!!! My version is 1.0 and I am built from Feature branch on port 9997!

1.9 Merging local changes to the version control system (Git)

- Clone your fork repo git clone forkrepo.git
- Add upstream to your fork

    git remote add upstream *https://github.com/mgna-repo/webapp.git*

In order to push the local changes:

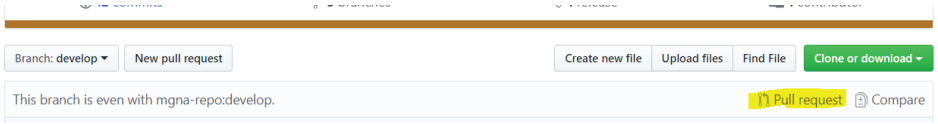
- Make some changes on your laptop
- Run below commands and push the code to your fork

  git add <filename>

  git commit –m “message”

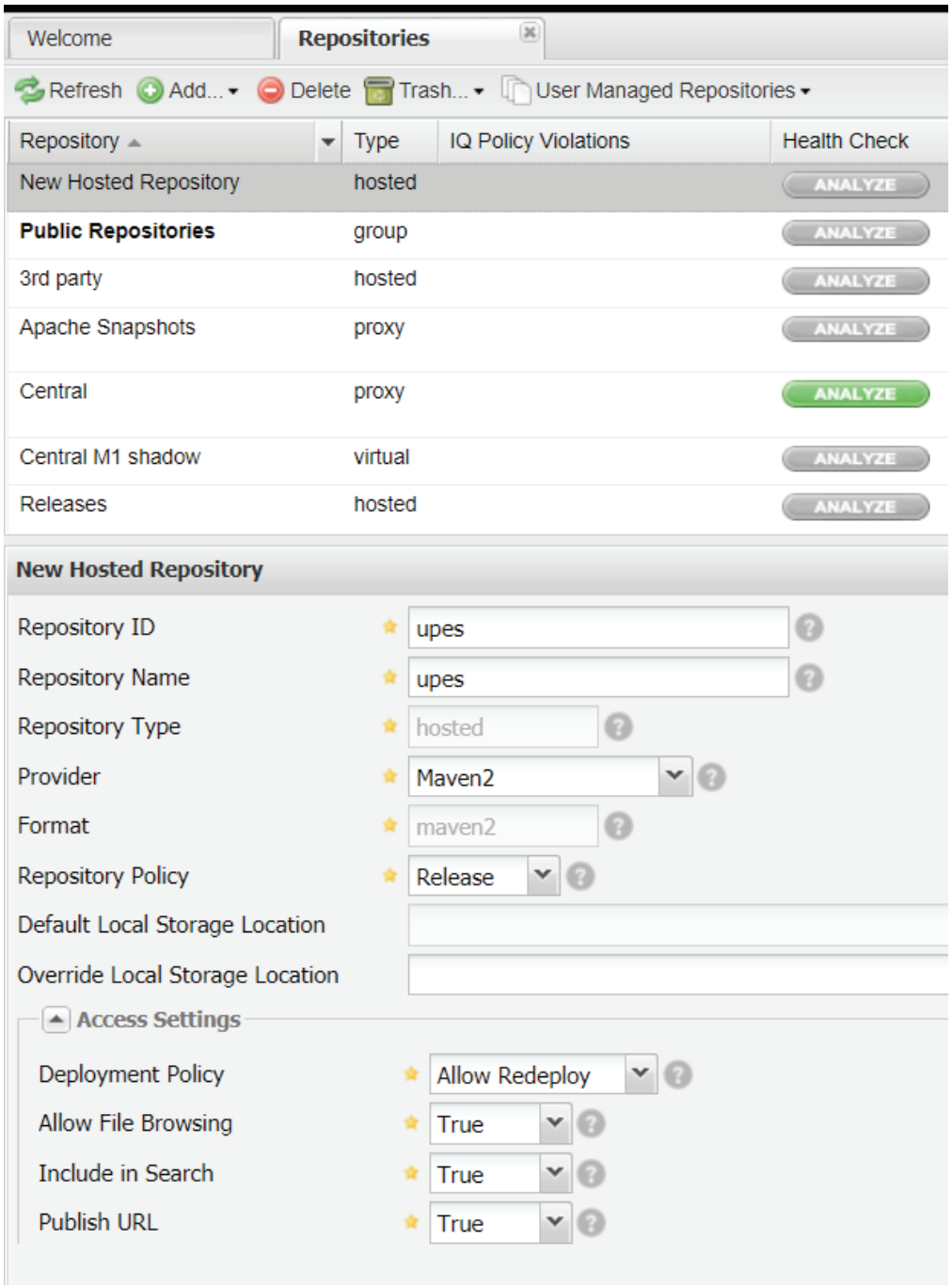
  git push

- The local changes are now pushed to your Fork. Raise a pull request from your fork page to the main repository in order to publish your changes.



1.10 Installing/Configuring Nexus

- Download Nexus from this Link - https://download.sonatype.com/nexus/professional-bundle/nexus-professional-2.14.13-01-bundle.zip
- Unzip the Nexus to nexus-professional-2.14.13-01\bin
- Open a command prompt in administrator mode and run nexus install
- Run nexus start command to start the nexus service.
- Login to nexus using admin/admin and create a hosted repository called upes with below settings



1.11 Publishing artifacts to Nexus

Add repositories section to your pom.xml

```
<repositori
es>

                        <repository>

                              <id>upes</id>

                              <name>Releases</name>


                        <url>http://localhost:8081/nexus/content/repositories/u
                  pes/</url>

                              </repository>

                        </repositories>
```

```
<distributionManagement>

            <repository>

                  <id>upes</id>

                  <name>Releases</name>

                  <url>http://localhost:8081/nexus/content/repositories/upes/</url>

            </repository>

      </distributionManagement>
```

Edit .m2/settings.xml and add nexus server credentials

```
<settings>

<servers>
```

```
<server>
 <id>upes</id>
 <username>admin</username>
 <password>admin123</password>
</server>
</servers>
</settings>
```

Type **mvn clean deploy** after adding the above entries in your pom.xml

Maven will deploy the artifacts to your nexus server. This artifacts can be used by any build server or other team that is dependent in your project.

If the artifacts are successfully uploaded, then browse the nexus URL and check your artifact.

http://localhost:8081/nexus/content/repositories/upes/com/puppet/sample/java-webapp/1.0/

Index of /repositories/upes/com/puppet/sample/java-webapp/1.0

Name	Last Modified	Size	Description
Parent Directory			
java-webapp-1.0.jar	Sat May 18 10:09:31 UTC 2019	6264085	
java-webapp-1.0.jar.md5	Sat May 18 10:09:44 UTC 2019	32	
java-webapp-1.0.jar.sha1	Sat May 18 10:09:43 UTC 2019	40	
java-webapp-1.0.pom	Sat May 18 10:09:45 UTC 2019	2368	
java-webapp-1.0.pom.md5	Sat May 18 10:09:48 UTC 2019	32	
java-webapp-1.0.pom.sha1	Sat May 18 10:09:47 UTC 2019	40	

1.12 Installing/Configuring Sonarqube

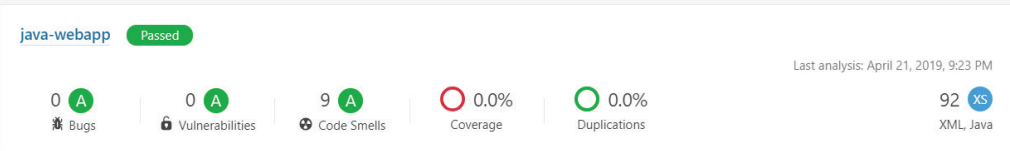
https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.7.zip

- Download the SonarQube Community Edition from the above link
- Unzip it, let's say in *C:\sonarqube* or */opt/sonarqube*
- Start the SonarQube Server:

```
# On Windows, execute:

C:\sonarqube\bin\windows-x86-xx\StartSonar.bat


# On other operating systems, as a non-root user execute:

/opt/sonarqube/bin/[OS]/sonar.sh console
```

- Log in to http://localhost:9000 with System Administrator credentials (admin/admin) and follow the embedded tutorial to analyze your first project.

This is the sample Sonarqube report for this project.



Note: This sample web application has less code, so the report is not displaying any vulnerabilities. You can use any other web application and just run below command to review the sample sonarqube report:

'mvn clean install sonar:sonar -Dsonar.host.url=http://10.x.x.x:9000 -Dsonar.analysis.mode=publish'

1.13 Publishing Static code analysis to Sonarqube

Lab:

Open a command prompt and navigate to your project directory where pom.xml is present and run below command to publish your static code analysis to Sonarqube server

mvn clean install sonar:sonar -Dsonar.host.url=*http://localhost:9000 -Dsonar.analysis.mode=publish org.codehaus.sonar-plugins.pdf-report:maven-pdfreport-plugin:1.3:generate*

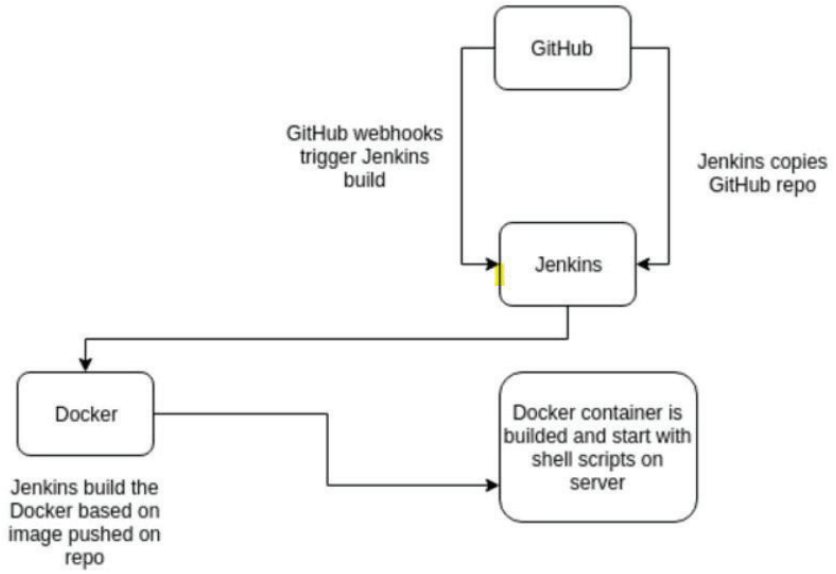
Note: If you install Sonarqube on any other server, replace localhost with the IP address in the above command

1.14 Use Jenkins as a Continuous Integration server

Continuous Integration with Jenkins

A Three Part Process

The CI workflow described in this article is composed of three steps. The developer first pushes a commit to GitHub, which in turn uses a webhook to notify Jenkins of the update. Jenkins can then pull the GitHub repository, build the Docker container which contains our stack and then run the test. If the test passes, Jenkins will push the code to the master branch.



Before we conclude this chapter, here is a list of the key practices of Continuous Integration (as defined by Martin Fowler in 2006) with the examples of the ways in which Jenkins can be used to help you achieve them:

- **Maintain a Single Source Repository**: Jenkins can interact with all modern source code and version control repositories—some abilities are built-in, others can be added as extensions.
- **Automate the Build**: As described earlier in the use cases, this is one of the core aims of Jenkins and often the main driver to start using Jenkins.
- **Make Your Build Self-Testing**: This is usually the second step in setting up a CI environment with Jenkins—once you automate the building of the code, automating the tests as well is a natural progression.
- **Everyone Commits To the Mainline Every Day**: We can't really force developers to do this, unfortunately. However, we can quite easily highlight and report who is doing—or not doing—what, which should eventually help them learn to follow this best practice.

- **Every Commit Should Build the Mainline on an Integration Machine**: Builds can be triggered by developer commits, and Jenkins Slave Nodes can be used to build and provide accurate replica environments to build upon.
- **Fix Broken Builds Immediately**: This is another developer best practice that needs to be adopted—when Jenkins shows red, the top focus should be on fixing the issue until it shows green. No one should commit new changes while the build is broken, and Jenkins can be configured to communicate the current status in the most effective way.
- **Keep the Build Fast**: By offloading and spreading work to distributed Slave Nodes and by breaking down builds to identify and focus on the areas that have changed, Jenkins can be tuned to provide a rapid response to changes—a good target would be to check in a change and obtain a clear indication of the result or impact under 10 minutes.
- **Test in a Clone of the Production Environment**: After compiling the new change, downstream Jenkins jobs can be created that will prepare the environment and take it to the required level—applying database changes, starting up dependent processes, and deploying other prerequisites. Using virtual machines or containers in conjunction with Jenkins to automatically start up environments in a known-good state can be very useful here.
- **Make it Easy for Anyone to Get the Latest Executable**: Jenkins can be set up to act as a web server hosting the latest version at a known location so that everyone (and other processes/consumers) can easily fetch it, or it can also be used to send out details and links to interested parties whenever a new version has been uploaded to Nexus, Artifactory, and so on.
- **Everyone can see what's happening**: There are many ways in which Jenkins communications can be extended—email alerts, desktop notifications, Information Radiators, RSS feeds, Instant Messaging, and many more—from lava lamps and traffic lights to the ubiquitous toy rocket launchers!
- **Automate Deployment**: This is usually a logical progression of the Build -> Test -> Deploy automation sequence, and Jenkins can help in many ways; with Slave Nodes running on the deployment host, or jobs set up to connect to the target and update it with the most recently built artifact.

1.15 Deploying the application to staging/prod environment

Let’s deploy the web application to an application server.

Add a new CentOS/Ubuntu Slave with Java Installed on it. Add the node with tag “Deployment Server”.

Let’s create a deployment script on Slave-01. Login to the machine and create deployment.sh in ./var/deployment

Deploying application to staging environment

deployment.sh

/usr/bin/sshpass -p 'Password' scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no /var/lib/jenkins/workspace/webapp/target/java-webapp-1.0.jar root@10.x.x.x:/var/staging/java-webapp-1.0.jar

/usr/bin/sshpass -p 'Password' ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@10.x.x.x “java –jar /var/staging/java-webapp-1.0.jar 2> /dev/null &”

Please edit the username, password and IP address of your deployment server.

The above script will copy the compiled web application to deployment server, copies to the deployment server and starts the application.

Now add deploy section to your Jenkinsfile as below:


```
pipeline {
    agent {
        label 'Slave-01'
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }

         stage('Deploy') {
            steps {
                sh '/var/deployment/./deployment.sh'
            }
        }
    }
}
```

Note: If you are doing this as a group exercise, one participant can change the code on your local machine, push it to the fork and then raise a pull request to update the main repo.

If you want to test this application without raising a pull request to the main repository, then make the changes on your Fork and use your Fork git hub link in Jenkins server.

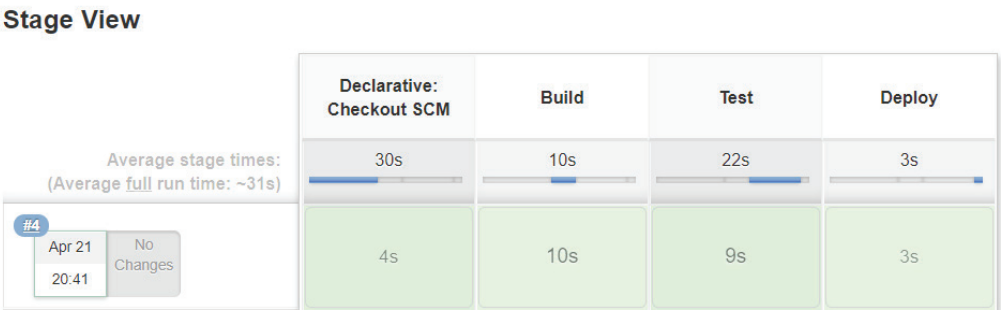
Go to Pipeline-Job in Jenkins and click on Build Now

Once the build is completed successfully, the web application is hosted on <deployment-server>:9999 <ip>:9999

If you would like to run the job again include this command at the top of the script - fuser -k 9999/tcp

This command will kill the application that is deployed on port 9999

The successful pipeline Job result will look like below:



Deploying the application to production:

Consider the application V 1.0 is tested by QA team that is deployed to staging environment. Product manager would take decision to release the product to production if no major bugs are detected in the testing.

Staging and Production environment will be identical, just the users are different.

Create a new Job in Jenkins Production-Pipeline and use a new deployment script to deploy the application to production.

In deployment-prod.sh mention the ip-address, username and password of production server and provide the instructions.

In some organizations they use Blue-Green deployments to deploy their application to the production environments. They simply switch the load balancer entry where the application is hosted on 2 servers.

1.16 Adding new features (V 2.0) to existing application

Since the application from develop branch is deployed to the prod server, merge the code from develop to master. This is a best practice to merge the code to master branch at the end of the release.

git checkout master

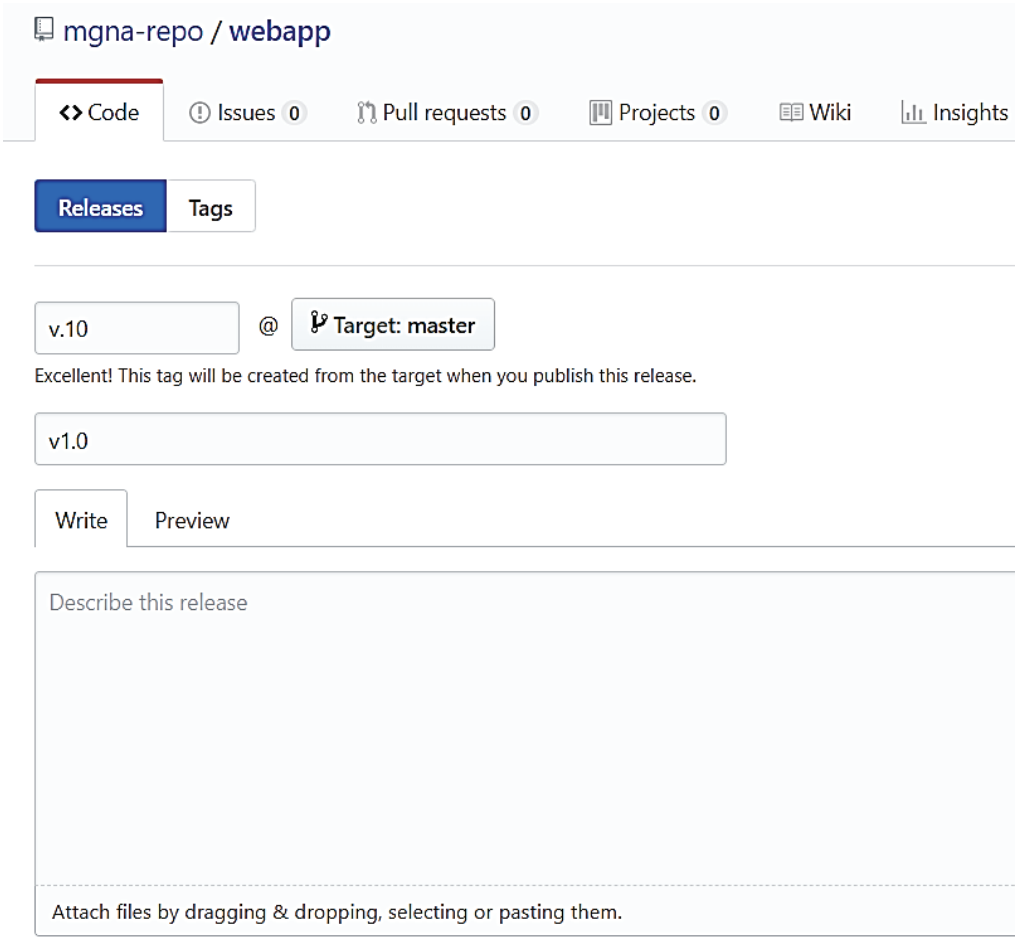
git merge origin/master

git merge -X theirs develop

git push origin master

Create a tag from master branch at the end of release 1.0

- Click on releases in your repository page and click on new releases
- Select from master and type the tag name like below:



Since the v.10 is released merge the code from feature to develop branch and deploy it to staging environment. Test it and then deploy the application to production as v2.0.

Note: A sample web application is used in this example. You can replace this with your own web application and test the Pipeline/Deployment scenarios.

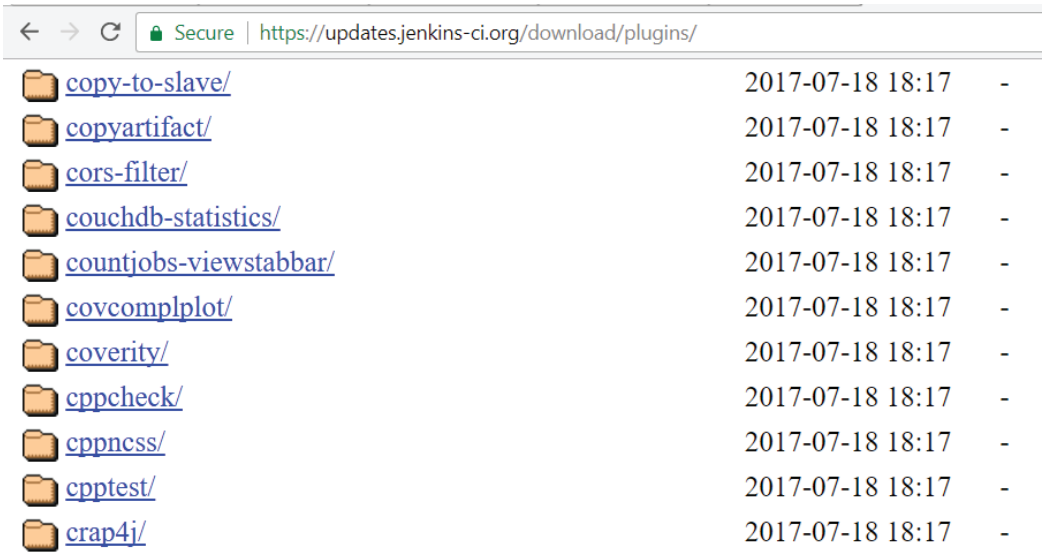
1.17 Uploading plugins manually in Jenkins

There will be instances where you have all the plugins downloaded from https://updates.jenkins-ci.org/download/plugins/, or you may create your own plugin and you want to utilize that in Jenkins. The below procedure will explain how to upload these plugins in Jenkins.

Download the required plugin *https://updates.jenkins-ci.org/download/plugins/* or create a plugin using the Plugin tutorial, *https://wiki.jenkins.io/display/JENKINS/Plugin+tutorial*.

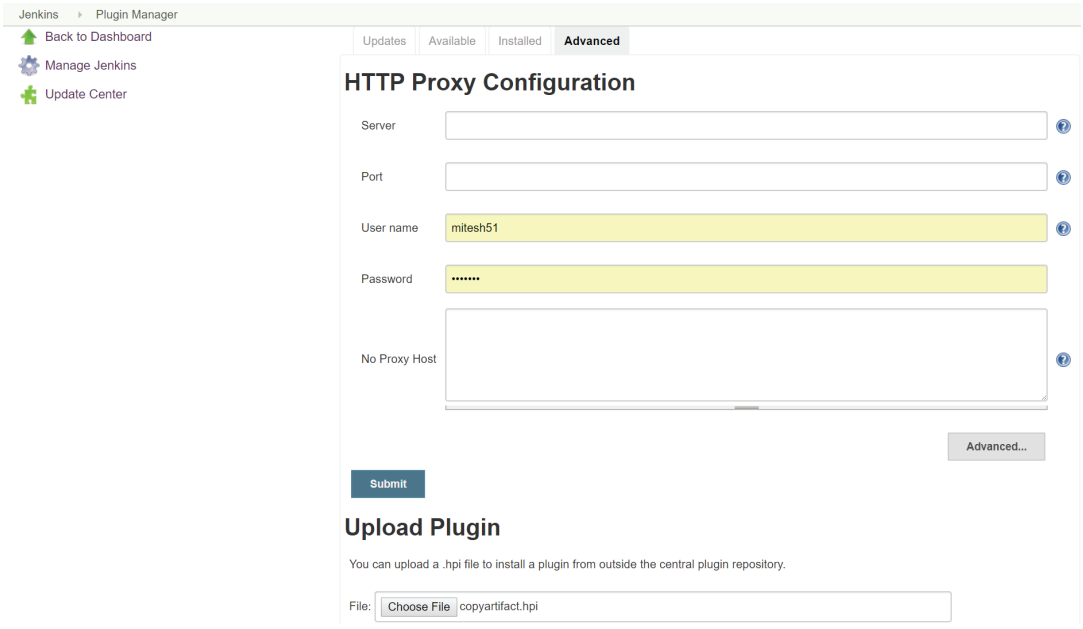
For this section, we will download a plugin and upload it from the Jenkins dashboard:

- Download the copyartifact/ plugin (.hpi) file:

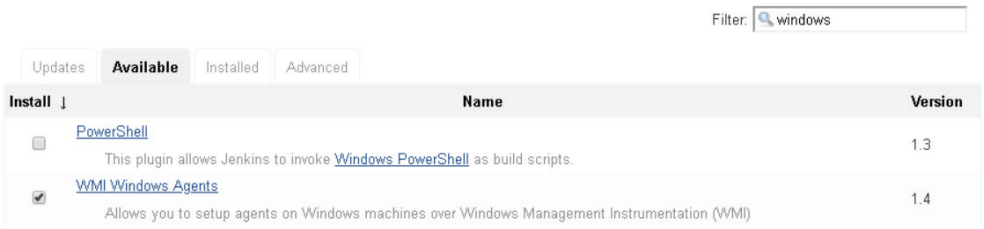


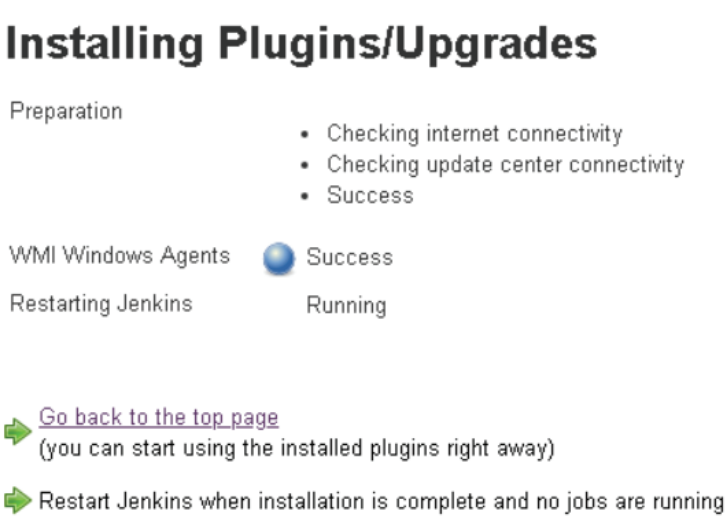
- Download the latest version to your system:

- Go to the Jenkins dashboard.
- Click on Manage Jenkins.
- Click on Manage plugins.
- Go to the Advanced tab. Click on Choose File in the Upload Plugin section:



- Click on Upload.





1.18 Backup Management in Jenkins Server

It is a known fact that Jenkins always requires some disk space in order to execute the build jobs and archive the same. This is configured by using the JENKINS_HOME. This path can be configured whenever required.

Backup and Restore

In Jenkins, all the settings, build logs and archives of the artifacts are stored under the JENKINS_HOME directory. The easy way is to just keep this folder separately as a new back and whenever the same needs to be used, just copy it back.

The build jobs created under this directory contains all the details of each and every individual jobs configured in the Jenkins install. Any job can be moved from one installation directory to another just by copying the jobs.

Also, Jenkins provides the feature of renaming an existing job on the fly. To reflect the applied changes, just click on the reload config from the user interface of the Jenkins.

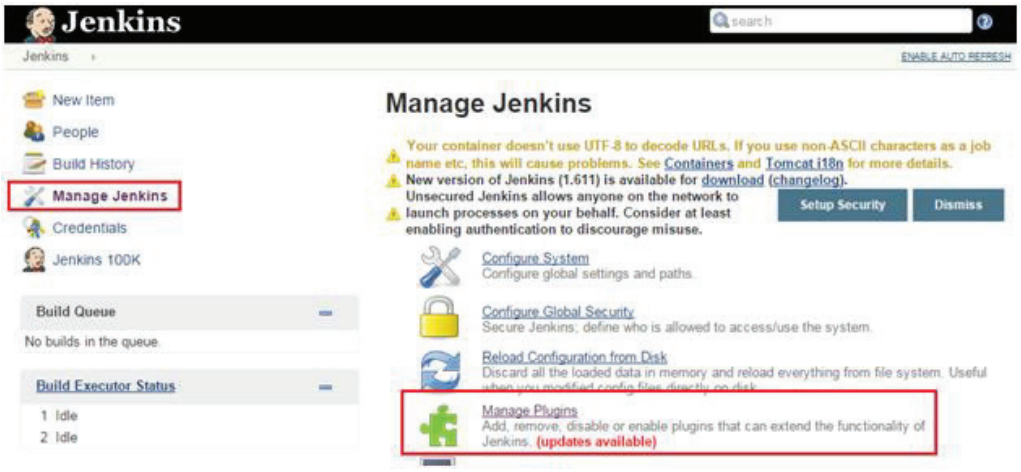
Archiving unused Jobs

Sometimes there may be some build jobs which remains unused for a longer time. In that case, there is an easy way to archive the same. Just navigate to the JENKINS_HOME directory and create an archive folder for the job which needs to be archived.

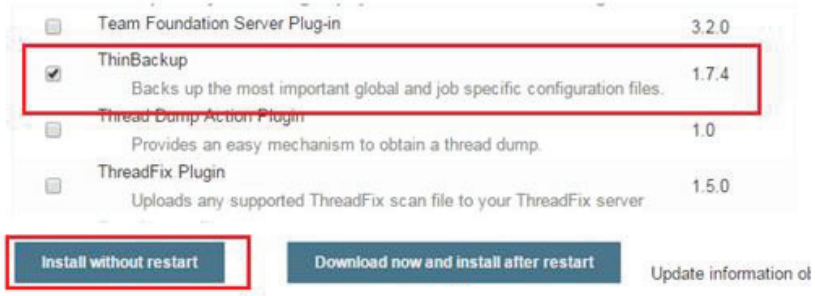
ThinBackup Plugin

Since, Jenkins can be enormously extended by the usage of several plugins, there comes a plugin which can be used for the backup management in Jenkins - ThinBackup plugin. This plugin backs up the job specific configuration files.

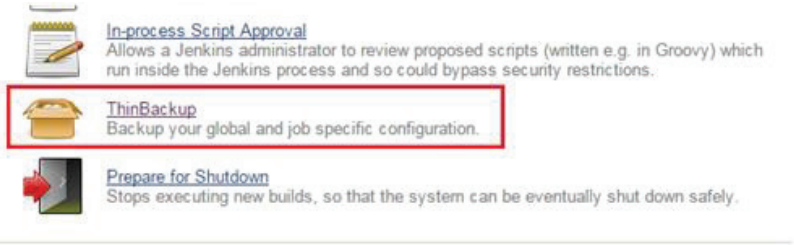
From the Jenkins home page, click on Manage Jenkins and in the next page click on Manage plugins.



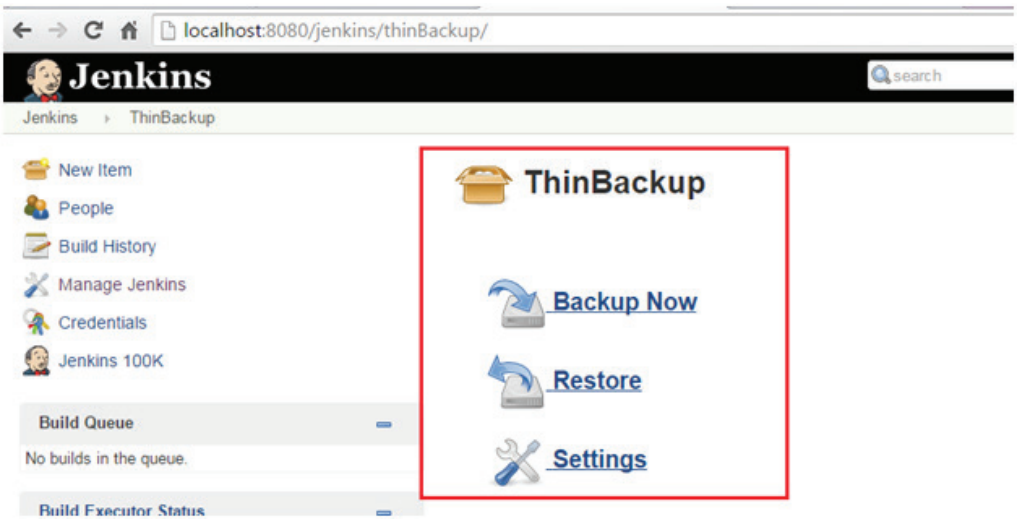
In the next page, under the Available plugins tab, search/look for ThinBackup plugin. Select the same and click on Install without restart button.



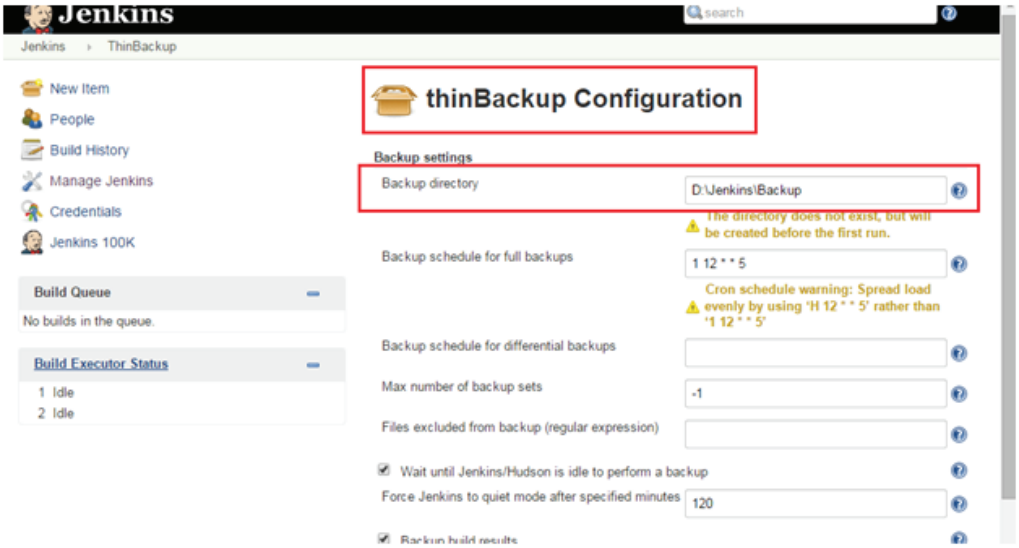
Just restart the Jenkins to see the plugin installed successfully. From the home page, click on Manage Jenkins link and in the next page, from the list if can be found that the ThinBackup plugin installed. Just click on the same and the below page will be displayed.



Click on the Settings link to configure the backup options.



Just provide the configuration details as required and save the settings.



Release Notes

B. TECH CSE with Specialization in DevOps

Semester Four -Year 02

Release Components.
Facilitator Guide, Facilitator Course Presentations, Student Guide, Mock exams and relevant lab guide.

Current Release Version.
1.0.0

Current Release Date.
6 Feb 2019

Course Description.
Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications.

Bugs reported	Not applicable for version 1.0.0
Next planned release	Version 2.0.0 Sept. 2019