

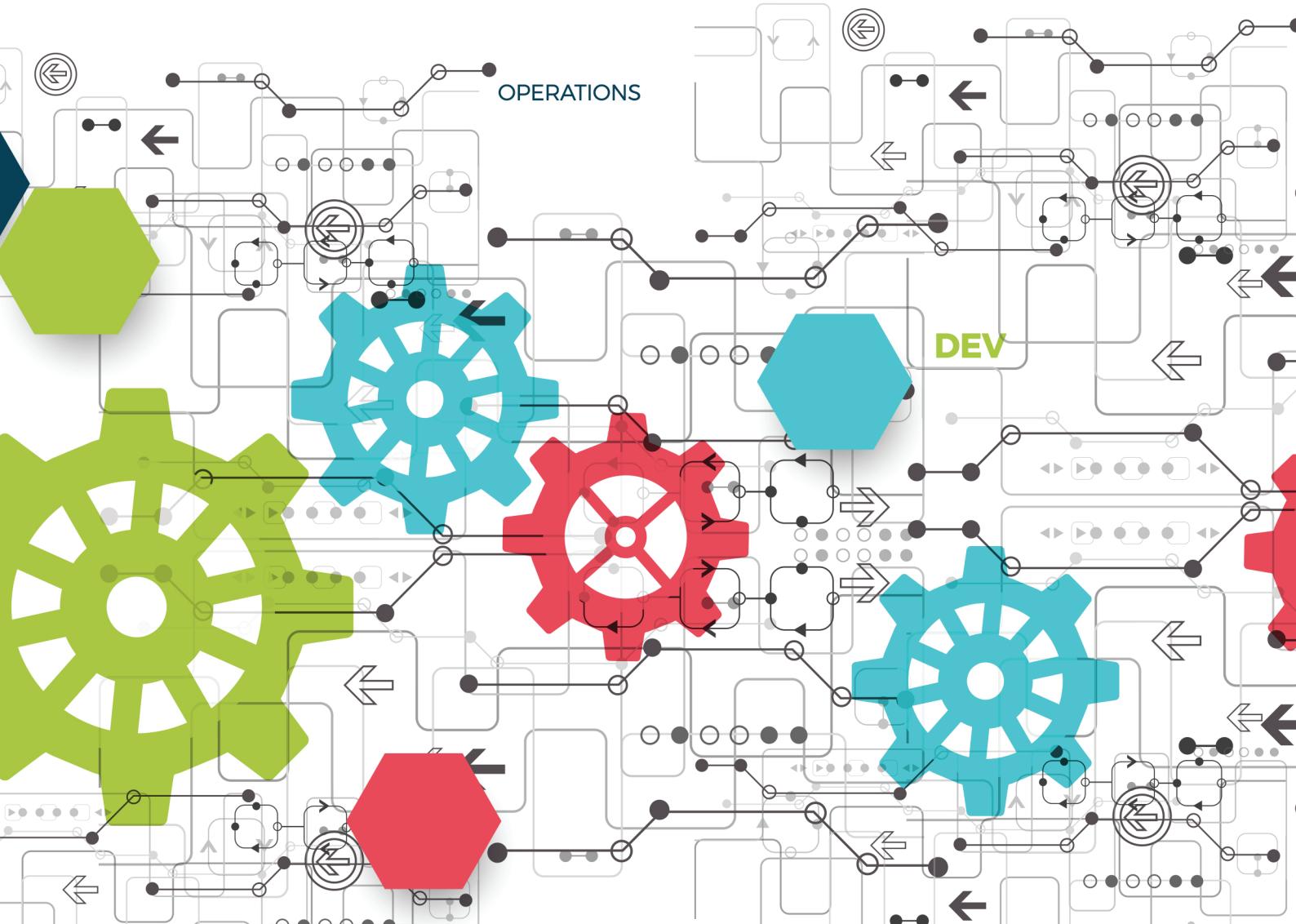


B.Tech Computer Science
and Engineering in DevOps

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

Copyright and Trademark Information for Partners/Stakeholders.

The course B.TECH computer science and engineering with Specialization in DevOps is designed and developed by Xebia Academy and is licenced to University of Petroleum and Energy Studies (UPES), Dehradun.

Content and Publishing Partners
ODW Inc | www.odw.rocks

www.xebia.com

Copyright © 2018 Xebia. All rights reserved.

Please note that the information contained in this classroom material is subject to change without notice. Furthermore, this material contains proprietary information that is protected by copyright. No part of this material may be photocopied, reproduced, or translated to another language without the prior consent of Xebia or ODW Inc. Any such complaints can be raised at sales@odw.rocks

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.

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

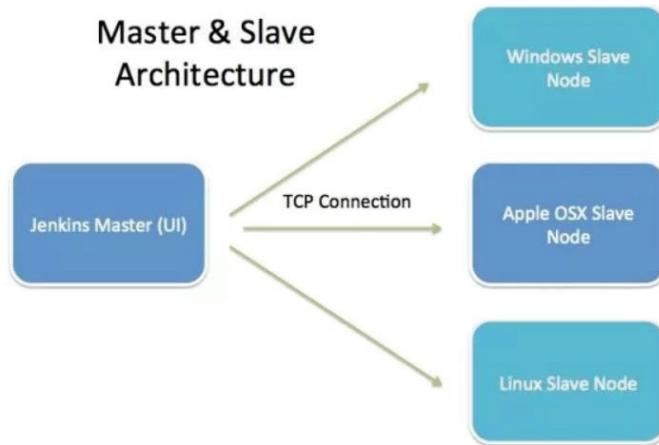
1.1 Introduction to Jenkins and setup/configuration

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:

Getting Started

The screenshot shows the 'Unlock Jenkins' step in the Jenkins setup wizard. It features a large circular icon with a gear and a keyhole. The text 'Unlock Jenkins' is prominently displayed at the top. Below it, a message states: 'To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:'. A red box highlights the path 'C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword'. Below this, instructions say 'Please copy the password from either location and paste it below.' A text input field labeled 'Administrator password' is shown, with a red box around its border. At the bottom right is a blue 'Continue' button.

Choose 'Install Suggested Plugins', click on it.

[Getting Started](#)

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.164.2

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
```

```
root@master:~# sudo apt install software-properties-common apt-transport-https -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
apt-transport-https is already the newest version (1.6.1).
software-properties-common is already the newest version (0.96.24.32.3).
0 upgraded, 0 newly installed, 0 to remove and 42 not upgraded.
root@master:~# sudo add-apt-repository ppa:webupd8team/java -y
Hit:1 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease [15.4 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:5 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease
Get:6 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic/main i386 Packages [1,556 B]
Get:7 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic/main amd64 Packages [1,556 B]
Get:8 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic/main Translation-en [928 B]
Fetched 19.4 kB in 7s (2,960 B/s)
Reading package lists... Done
```

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.

```
root@master:~#  
root@master:~# java -version  
java version "1.8.0_171"  
Java(TM) SE Runtime Environment (build 1.8.0_171-b11)  
Java HotSpot(TM) 64-Bit Server VM (build 25.171-b11, mixed mode)  
root@master:~#  
root@master:~# _
```

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
```

```
root@master:~#
root@master:~# wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
OK
root@master:~#
root@master:~# echo 'deb https://pkg.jenkins.io/debian-stable binary/' | tee -a /etc/apt/sources.list
deb https://pkg.jenkins.io/debian-stable binary/
root@master:~#
root@master:~# sudo apt update
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Hit:3 http://ppa.launchpad.net/webupd8team/java/ubuntu bionic InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [83.2 kB]
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:6 http://security.ubuntu.com/ubuntu bionic-security/main i386 Packages [73.9 kB]
Get:7 https://pkg.jenkins.io/debian-stable binary/ Release [2,042 B]
Get:8 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [181 B]
Get:9 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:10 https://pkg.jenkins.io/debian-stable binary/ Packages [12.9 kB]
Get:11 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [85.7 kB]
Get:12 http://security.ubuntu.com/ubuntu bionic-security/main Translation-en [32.0 kB]
Get:13 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [30.1 kB]
Get:14 http://security.ubuntu.com/ubuntu bionic-security/universe i386 Packages [30.0 kB]
Get:15 http://security.ubuntu.com/ubuntu bionic-security/universe Translation-en [15.5 kB]
Fetched 523 kB in 9s (57.6 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
53 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@master:~#
root@master:~# sudo apt install jenkins -y
Reading package lists... Done
```

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 Local Address          Foreign Address        State      PID/Program name
tcp     0    0.0.0.0:53            0.0.0.0:*           LISTEN    5688/systemd-resolv
tcp     0    0.0.0.0:22            0.0.0.0:*           LISTEN    715/sshd
tcp6    0    :::8080              ::*:*                LISTEN    5539/java
tcp6    0    0.0.0.0:22            ::*:*                LISTEN    715/sshd
udp     0    0.0.0.0:53            0.0.0.0:*           LISTEN    5688/systemd-resolv
udp     0    0.0.0.0:68            0.0.0.0:*           LISTEN    1487/systemd-networ
udp     4352   0.0.0.0:68          0.0.0.0:*           LISTEN    1199/dhclient
udp6    0    0.0.0.0:33848         ::*:*                LISTEN    5539/java
udp6    6016   0.0.0.0:5353        ::*:*                LISTEN    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# 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
```

```
root@master:~# ufw allow ssh
Rules updated
Rules updated (v6)
root@master:~# ufw allow http
Rules updated
Rules updated (v6)
root@master:~# ufw allow https
Rules updated
Rules updated (v6)
root@master:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
root@master:~#
root@master:~#
```

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'.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

C:\Program Files (x86)\Jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Continue

Choose 'Install Suggested Plugins', click on it.

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.164.2

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:

The screenshot shows the Jenkins Plugin Manager interface. The top navigation bar includes 'Jenkins', 'Plugin Manager', 'search', and 'log out'. Below the navigation is a sidebar with links: 'Back to Dashboard', 'Manage Jenkins', and 'Update Center'. The main content area has tabs: 'Updates' (selected), 'Available', 'Installed', and 'Advanced'. A search bar at the top right is set to 'Filter: Maven'. The 'Available' tab displays a list of plugins:

Install	Name	Version
<input checked="" type="checkbox"/>	Maven Integration plugin	2.17
<input type="checkbox"/>	Maven Invoker plugin	1.3
<input type="checkbox"/>	Maven Info Plugin	0.2.0
<input type="checkbox"/>	Pipeline Maven Integration Plugin	2.5.1

Below the table are three buttons: 'Install without restart', 'Download now and install after restart', and 'Check now'. A status message says 'Update information obtained: 39 min ago'.

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

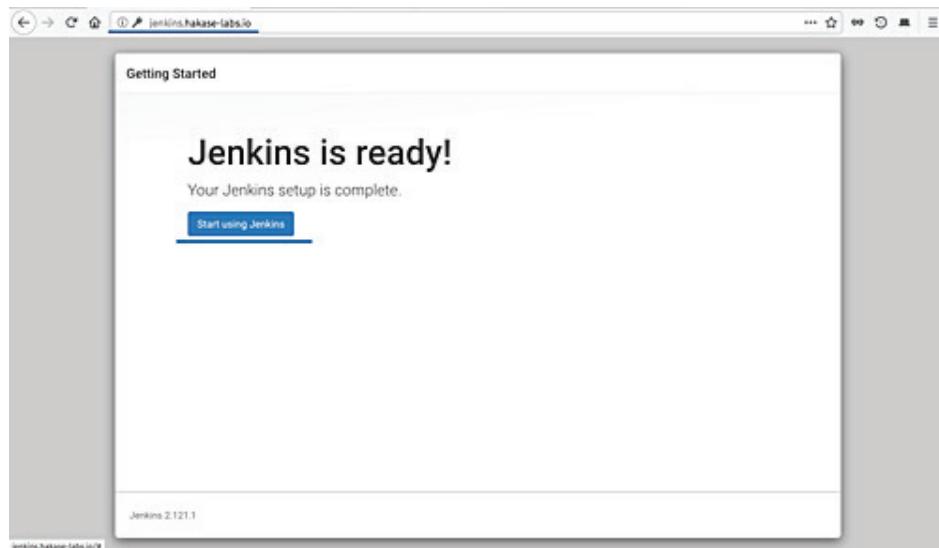
The screenshot shows the Jenkins Update Center page at `localhost:8080/updateCenter/`. The page lists various Jenkins plugins and their installation status:

Plugin	Status
Git plugin	Success
Subversion Plug-in	Success
SSH Slaves plugin	Success
Matrix Authorization Strategy Plugin	Already Installed
PAM Authentication plugin	Success
LDAP Plugin	Success
Email Extension Plugin	Success
Mailer Plugin	Already Installed
Javadoc Plugin	Success
Maven Integration plugin	Success

Below the table, there are two informational messages:

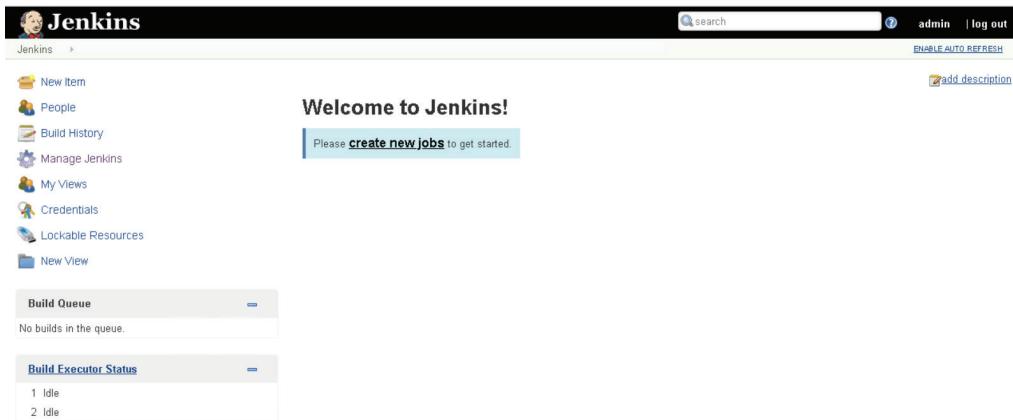
- Go back to the top page**
(you can start using the installed plugins right away)
- Restart Jenkins when installation is complete and no jobs are running**

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'.

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is currently selected), 'My Views', 'Credentials', and 'New View'. Below these are 'Build Queue' (empty) and 'Build Executor Status' (2 idle). The main content area is titled 'Manage Jenkins' and contains several configuration options: 'Configure System' (Configure global settings and paths), 'Configure Global Security' (Secure Jenkins; define who is allowed to access/use the system, currently selected), 'Configure Credentials' (Configure the credential providers and types), 'Global Tool Configuration' (Configure tools, their locations and automatic installers), and '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). At the top right, there are links for 'Search', 'nakase labs', and 'log out'.



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:

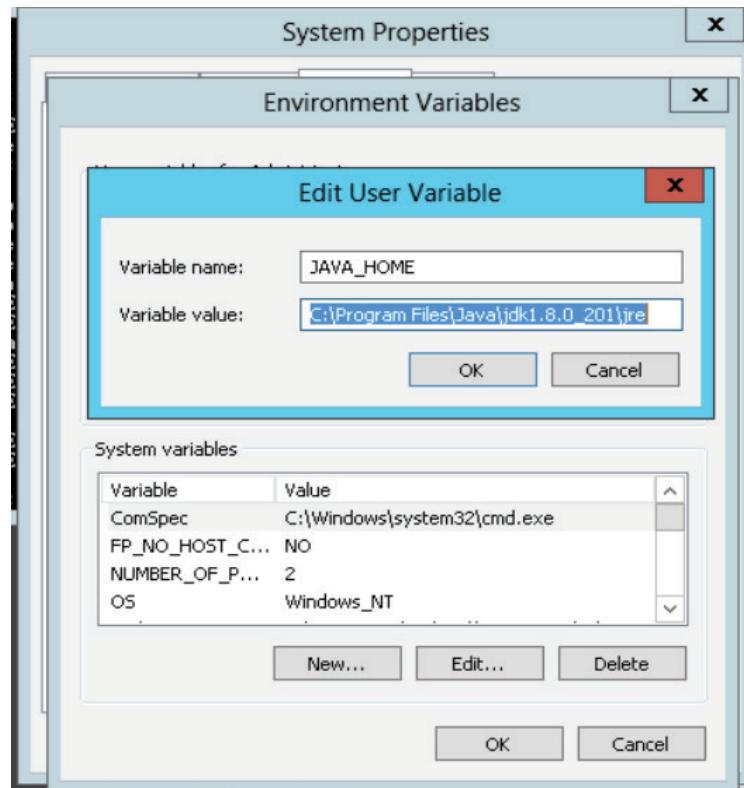
Lab:

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

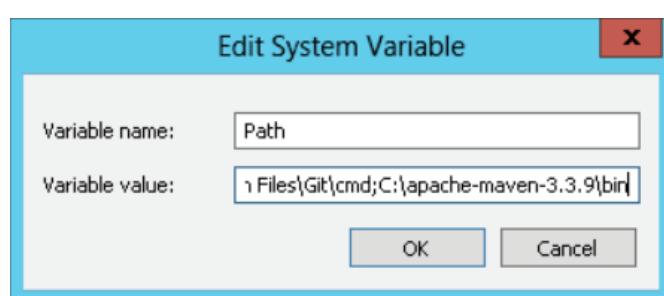
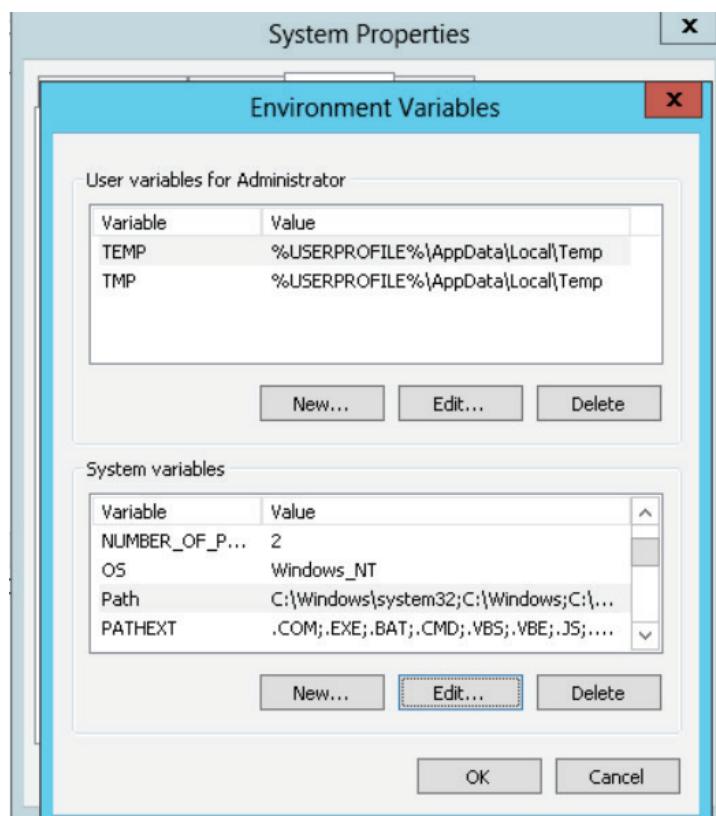


Configure Maven:

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.

Enter an item name
First-Job
» Required field

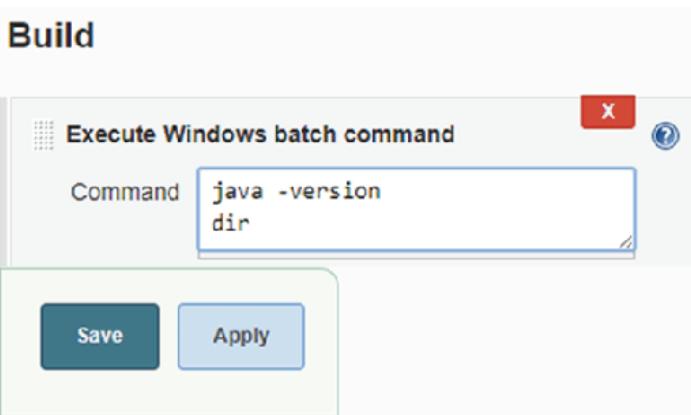
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

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.

The screenshot shows the Jenkins project page for 'First-Job'. The 'Build Now' button is underlined, indicating it has been clicked. To the right, there are links for 'Workspace' (with a folder icon) and 'Recent Changes' (with a document icon). Below these are sections for 'Permalinks'.

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.



Console Output

```
Started by user admin
Building in workspace C:\Program Files (x86)\Jenkins\workspace\First-Job
[First-Job] $ cmd /c call C:\Windows\TEMP\jenkins491681924322069985.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

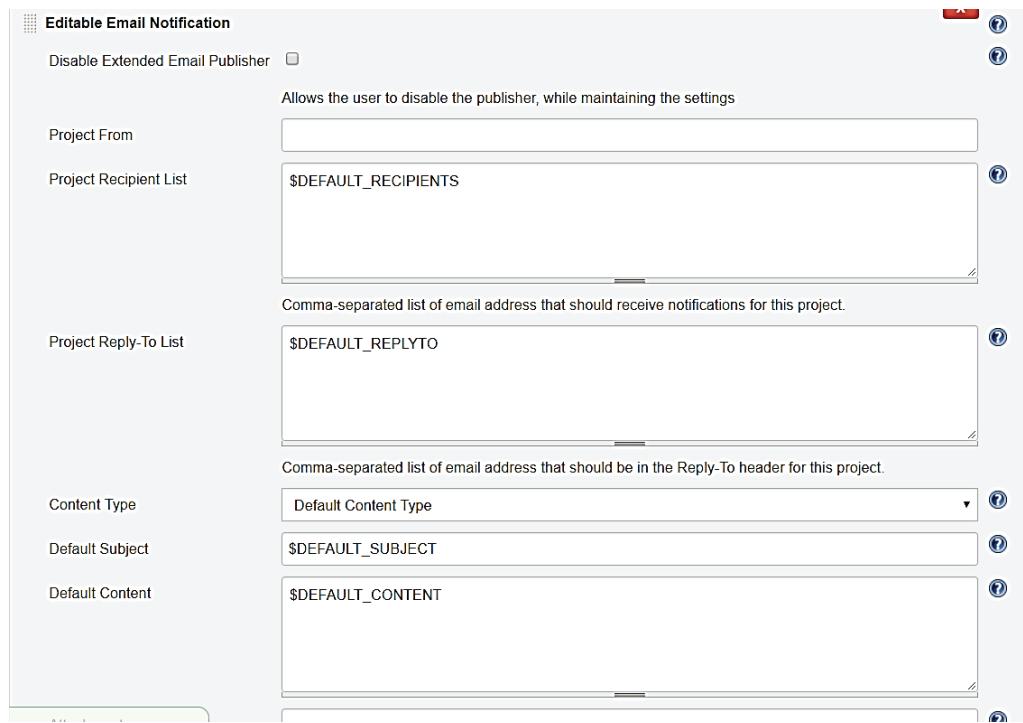
04/14/2019  03:55 PM    <DIR>      .
04/14/2019  03:55 PM    <DIR>      ..
              0 File(s)       0 bytes
              2 Dir(s)   2,087,763,968 bytes free

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”



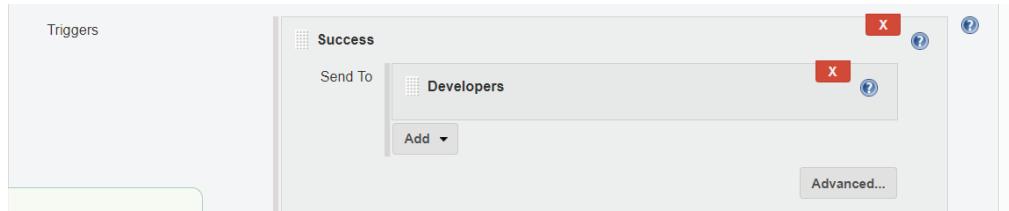
\$DEFAULT_RECIPIENTS and \$DEFAULT_REPLYTO can be defined in email settings

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

The screenshot shows the 'E-mail Notification' configuration page. It includes the following fields:

- SMTP server**: A text input field containing "smtp.company.com".
- Default user e-mail suffix**: A text input field.
- Use SMTP Authentication**: An unchecked checkbox.
- Use SSL**: An unchecked checkbox.
- SMTP Port**: A text input field.
- Reply-To Address**: A text input field.
- Charset**: A text input field containing "UTF-8".
- Test configuration by sending test e-mail**: A checked checkbox.
- Test e-mail recipient**: A text input field containing a recipient address.
- Test configuration**: A yellow button at the bottom right.

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'.

The screenshot shows the Jenkins Manage Jenkins page. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins' (which is selected), 'My Views', 'Credentials', and 'New View'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (showing two master nodes: 'master' with 1 idle and 2 busy, and two slave nodes: 'slave01' and 'slave02', both with 1 idle). The main content area is titled 'Manage Jenkins' and contains several configuration links: 'Configure System' (selected), 'Configure Global Security', 'Configure Credentials', 'Global Tool Configuration', 'Reload Configuration from Disk', 'Manage Plugins', 'System Information', and 'System Log'. A 'Configure Systems' button is located at the top right of the main content area.

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

The screenshot shows the 'Agents' tab of the 'Configure Global Security' screen. It has two sections: 'TCP port for JNLP agents' (radio buttons for Fixed, Random, or Disable, with 'Random' selected) and 'Agent protocols' (checkboxes for various JNLP protocols). All checkboxes are checked except for 'Jenkins CLI Protocol/1 (deprecated, unencrypted)', which is described as unencrypted and deprecated. Other protocols listed include 'Jenkins CLI Protocol/2 (deprecated)', 'Java Web Start Agent Protocol/1 (deprecated, unencrypted)', 'Java Web Start Agent Protocol/2 (deprecated, unencrypted)', 'Java Web Start Agent Protocol/3 (deprecated, basic encryption)', and 'Java Web Start Agent Protocol/4 (TLS encryption)', which is described as a TLS secured connection.

Name	Slave-01	?
Description		?
# of executors	1	?
Remote root directory	/var/lib/jenkins	?
Labels	Slave-01	?
Usage	Use this node as much as possible	?
Launch method	Launch agent via Java Web Start	?
Disable WorkDir	<input type="checkbox"/>	?
Custom WorkDir path		?
Internal data directory	remoting	?
Fail if workspace is missing	<input type="checkbox"/>	?
Advanced...		



 This agent is offline because Jenkins failed to launch the agent process on it. [See log for more details](#)

Connect agent to Jenkins one of these ways:

-  Launch Launch agent from browser
- Run from agent command line:

```
java -jar agent.jar -jnlpUrl http://172.31.37.82:8080/computer/Slave01/slave-agent.jnlp -secret dd6a96bc15acb7e2e534a62c3149d6ea030d286666d393e0420c4a063530688 -workDir "C:/Jenkins"
```

Projects tied to Slave01

None

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 jenkin 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:PorTnAJbfXQNqRdkNmPAI1dBS1szWLk4+j+1GtkBY4 jenkins@master
The key's randomart image is:
+---[RSA 2048]---+
|   o.o=+.
|   ..B.+.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.

T	P	Store	Domain	ID	Name
Icon: SM	L	Jenkins			
		global			

And click the 'global' domain link.

Now click 'Add Credentials'.

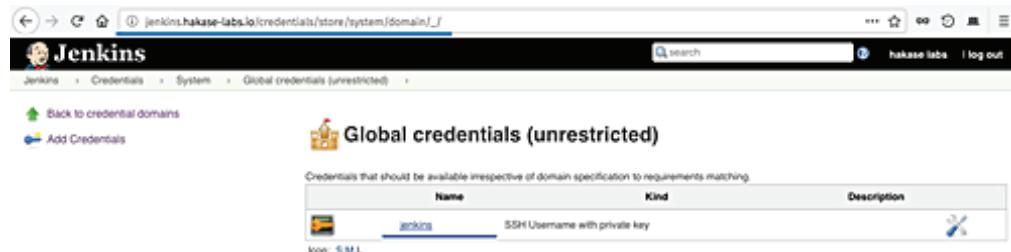
Name	Kind	Description
This credential domain is empty. How about adding some credentials?		

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.



The screenshot shows the Jenkins Global credentials (unrestricted) page. The URL is `jenkins.hakase-labs.io/credentials/store/system/domain/_/`. The page title is "Global credentials (unrestricted)". It displays a table with one row, showing a credential named "jenkins" of type "SSH Username with private key".

Name	Kind	Description
jenkins	SSH Username with private key	

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.
ED25519 key fingerprint is SHA256:jHNSne5e9r9c773jYabt9Ahp9SFWI2npQ+2uDMFY.
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.

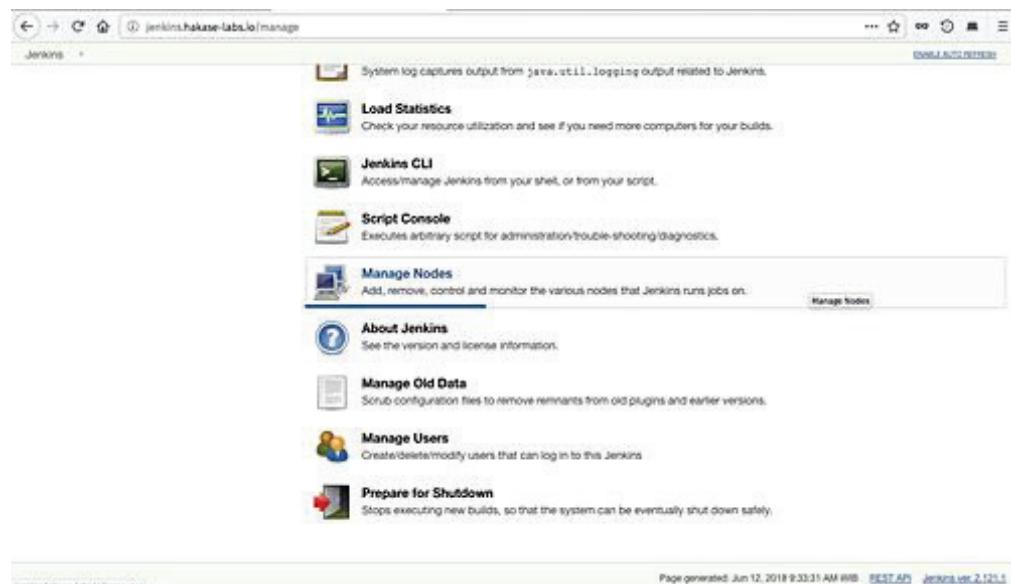
jenkins@master:~$ 
jenkins@master:~$ ssh 'jenkins@10.0.15.21'
Welcome to Ubuntu 18.04 LTS (GNU/Linux 4.15.0-20-generic x86_64)

 * 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'.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Linux (amd64)	In sync	15.06 GB	978.73 MB	15.06 GB	0ms
	Data obtained		16 min	16 min	16 min	16 min	16 min

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='15.242.140.21', 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 15.242.140.21:22.  
[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:  
osname: Cent OS  
osversion: 7.4.1708  
archname: x86_64  
arch: amd64  
cpu: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz  
osfamily: Linux  
osfullname: Cent OS 7.4.1708 (Core)  
osmajorversion: 7  
osminorversion: 4  
osrelease: 17.08  
osv: 3.10.0-862.1.2.el7.x86_64  
osvfull: 3.10.0-862.1.2.el7  
osvrel: 17.08  
osvrelfull: Cent OS 7.4.1708 (Core)
```

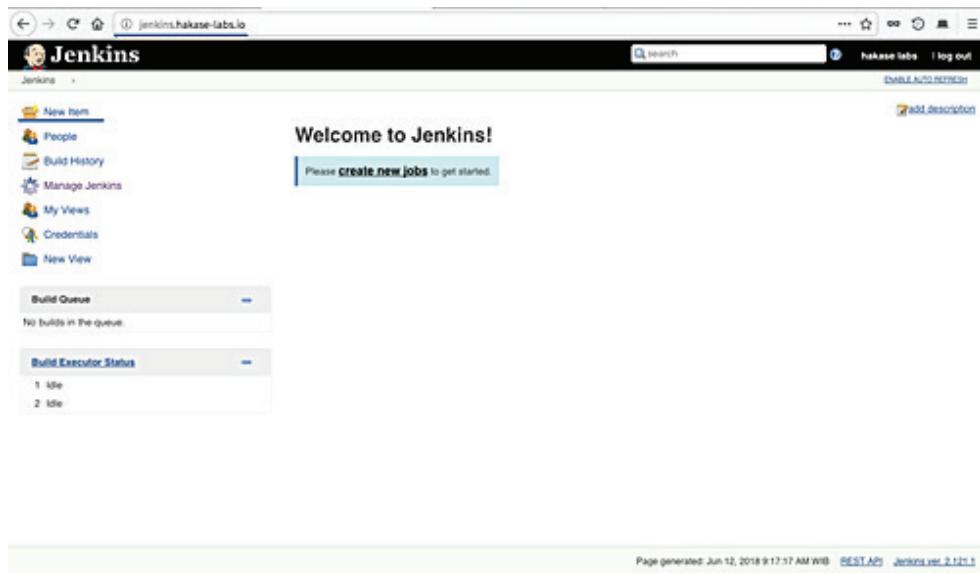
The screenshot shows two panels of the Jenkins dashboard. The top panel is titled "Build Queue" and contains the message "No builds in the queue." The bottom panel is titled "Build Executor Status" and lists two nodes: "master" and "Slave-01". The "master" node has 1 idle executor, and the "Slave-01" node has 1 idle executor.

Node	Idle Executors
master	1
Slave-01	1

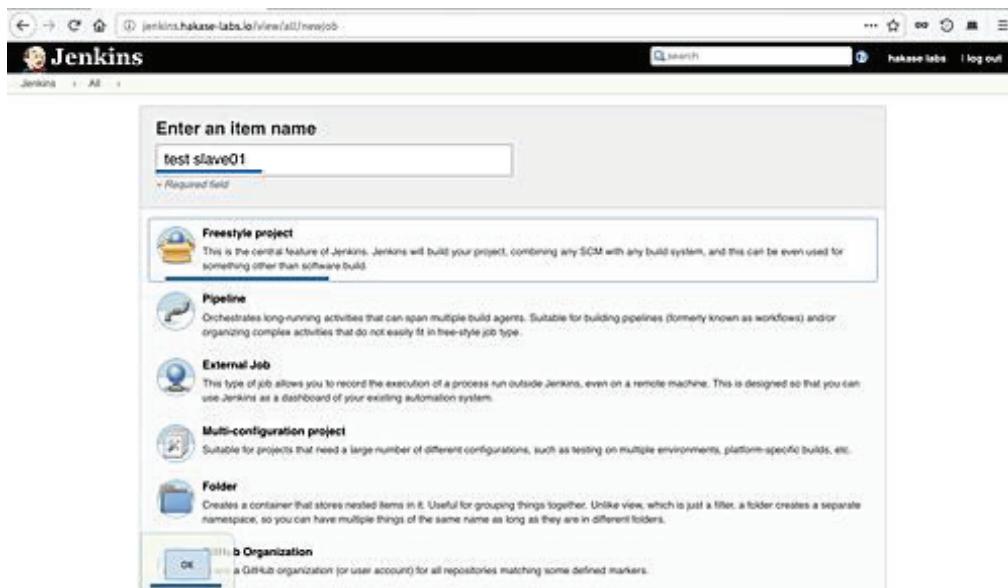
Slave node is added to the Jenkins master server.

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.

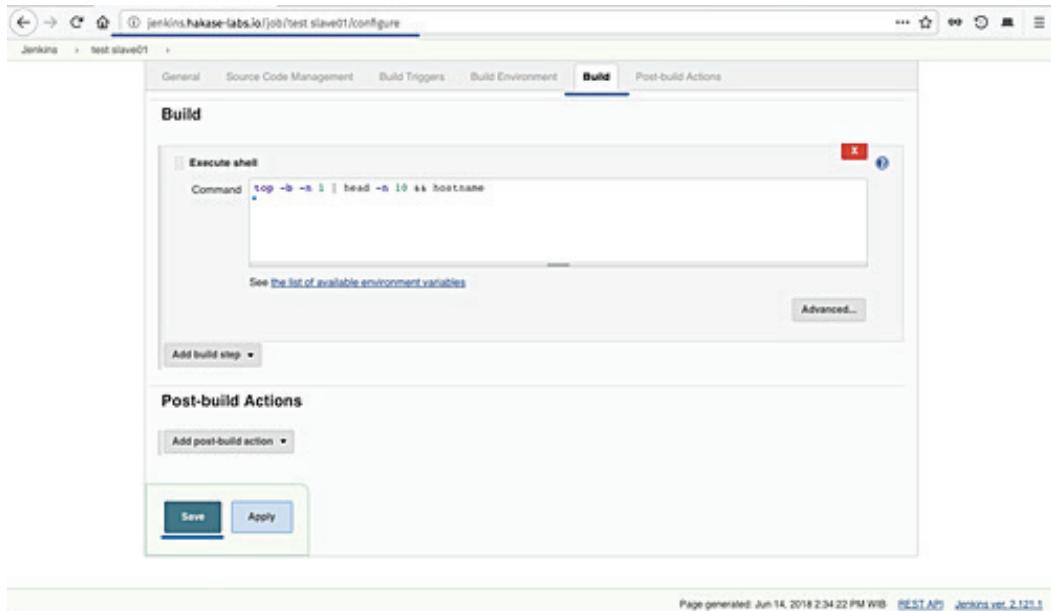
In the 'Label Expression', enter the node as 'Slave-01'.

The screenshot shows the Jenkins configuration interface for a job named 'test slave01'. The 'General' tab is active. In the 'Label Expression' field, the value 'slave01' is specified. A note below the field indicates that the job is serviced by one node and that permissions or other restrictions from plugins might prevent it from running on master nodes. At the bottom of the screen, there are 'Save' and 'Apply' buttons.

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.

The screenshot shows the Jenkins project page for 'test slave01'. The 'Build Now' link is highlighted in the left sidebar. The main content area shows the project name 'Project test slave01' and a list of recent builds. The 'Build History' section shows one build from Jun 14, 2018 2:40 PM. The 'Permalinks' section lists four recent builds.

And the following is my result.

Build on the 'slave01' agent node.

The screenshot shows the Jenkins interface with the URL <http://jenkins.hakase-labs.local/test-slave01/>. The left sidebar shows navigation links: Back to Project, Status, Changes, Console Output (selected), View as plain text, Edit Build Information, and Delete Build. The main content area is titled "Console Output". It displays the following text:

```
Started by user hakase_labs
Building remotely on slave01 in workspace /home/jenkins/workspace/test-slave01
Last 100 lines: [14:40:04] + readlink -f /usr/lib/jvm/jdk-11.0.1+13-amd64/jre/bin/java
* head -n 10
* top -b -n 1
top: - 14:40:04 up 48 min, 9 users, load average: 0.00, 0.00, 0.00
Tasks: 99 total, 1 running, 47 sleeping, 0 stopped, 0 zombie
CPU(s): 0.7 us, 0.4 sy, 0.0 ni, 98.7 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
SLB Mem: 1099124 total, 644388 free, 358238 used, 209308 buff/available
SLB Swap: 1099318 total, 1003518 free, 0 used, 799428 avail Mem

PID USER    PR  NI    VIRT    RES    SHR % CPU(s)    TIME+ COMMAND
1234 jenkins  20   0  2258044  84428 18988  8 0.7  9.4  0:26.63 java
1219 jenkins  20   0  104112  4278  3200  2 5.9  0.4  0:00..50 sshd
1467 jenkins  20   0  43444  3618  3188  2 5.9  0.4  0:00..52 top
* bootstrap
slave01
Finished: SUCCESS
```

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

1.6 Configuring Jenkins with git plugin

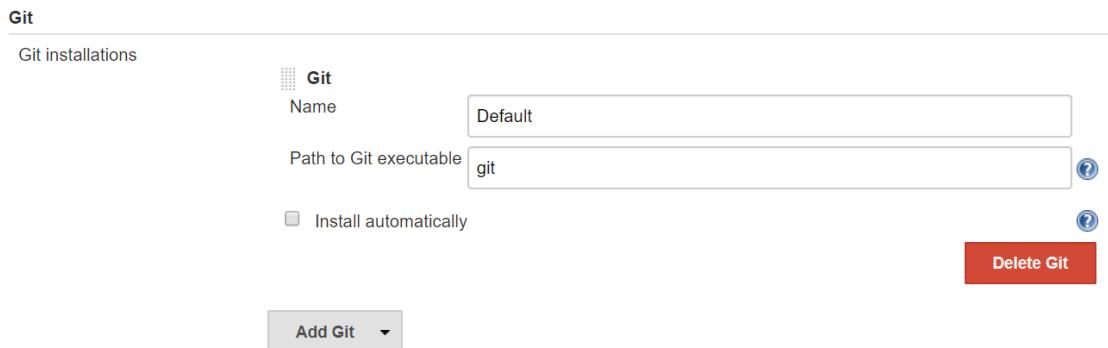
Jenkins can be integrated with many source code repositories, and Git is one of them. Let's consider the case of an application that is Java-based and the source code is stored in Git, Gitlab, and/or GitHub.

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.pub 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.

The screenshot shows the GitHub profile settings page at <https://github.com/settings/profile>. The left sidebar lists various settings sections: Personal settings, Profile (which is selected and highlighted in orange), Account, Emails, Notifications, Billing, SSH and GPG keys (which is highlighted with a yellow box), Security, Sessions, Blocked users, Repositories, and Organizations. The main right-hand area is titled "Public profile" and contains fields for Name, Public email (with a dropdown menu "Select a verified email to display"), Bio (with placeholder text "Tell us a little bit about yourself" and a note "You can @mention other users and organizations to link to them."), and URL.

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>

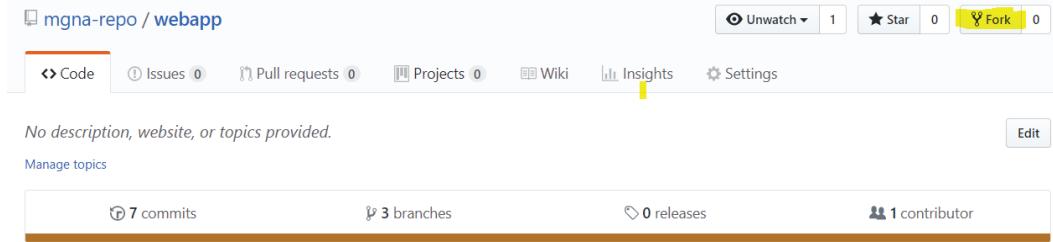
This is a simple java based web application that will start print a message in the browser.

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.



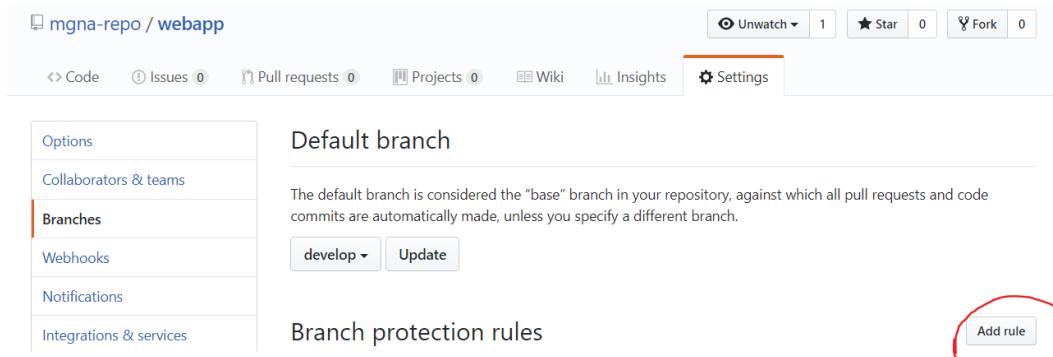
Click on Fork button and select your name. The repository will be forked on your name.

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:



Rule settings

Protect matching branches
Disables force-pushes to all matching branches and prevents them from being deleted.

Require pull request reviews before merging
When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of approving reviews and no changes requested before it can be merged into a branch that matches this rule.

Dismiss stale pull request approvals when new commits are pushed
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

Require review from Code Owners
Require an approved review in pull requests including files with a designated code owner.

Restrict who can dismiss pull request reviews
Specify people or teams allowed to dismiss pull request reviews.

Require status checks to pass before merging
Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

Require signed commits
Commits pushed to matching branches must have verified signatures.

Include administrators
Enforce all configured restrictions for administrators.

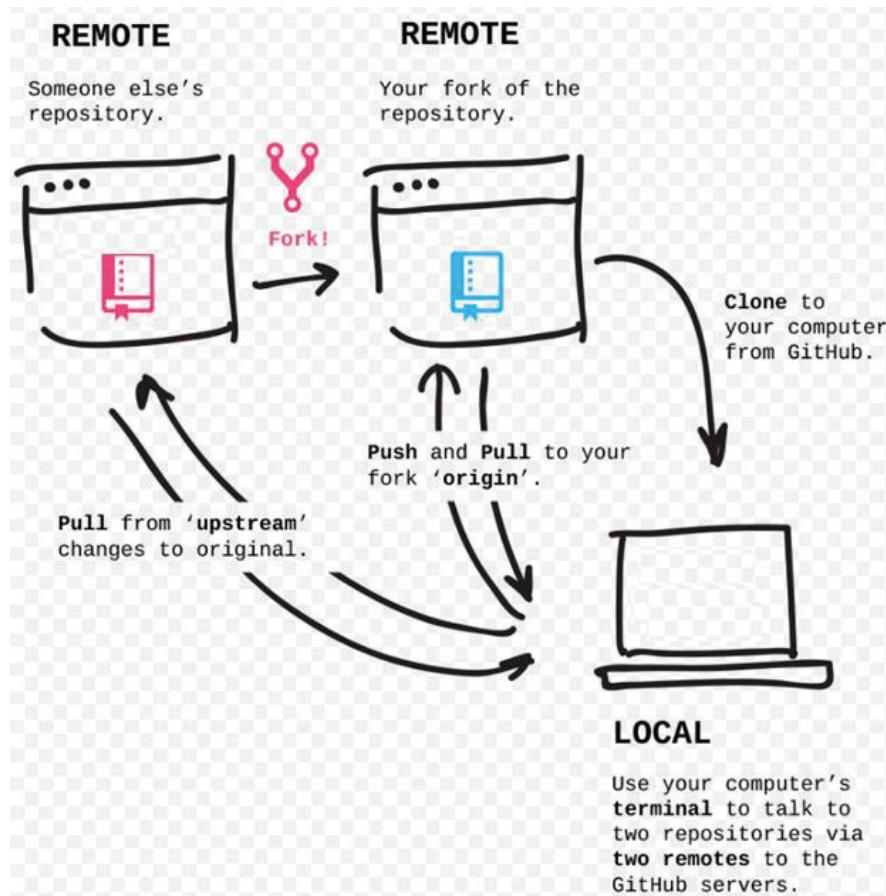
Restrict who can push to matching branches
Specify people or teams allowed to push to matching branches. Required status checks will still prevent these people from merging if the checks fail.

The screenshot shows the GitHub repository settings for 'myna-repo / webapp'. The 'Branches' tab is selected. On the left, a sidebar lists options like Options, Collaborators & teams, Branches (which is selected), Webhooks, Notifications, Integrations & services, Deploy keys, and Security alerts. Below the sidebar, the 'Default branch' section shows 'develop' as the current default branch. The 'Branch protection rules' section contains three rules: 'master', 'develop', and 'feature', each applying to one branch. An 'Add rule' button is available.

Branch	Currently applies to 1 branch	Edit	Delete
master	Currently applies to 1 branch	Edit	Delete
develop	Currently applies to 1 branch	Edit	Delete
feature	Currently applies to 1 branch	Edit	Delete

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.

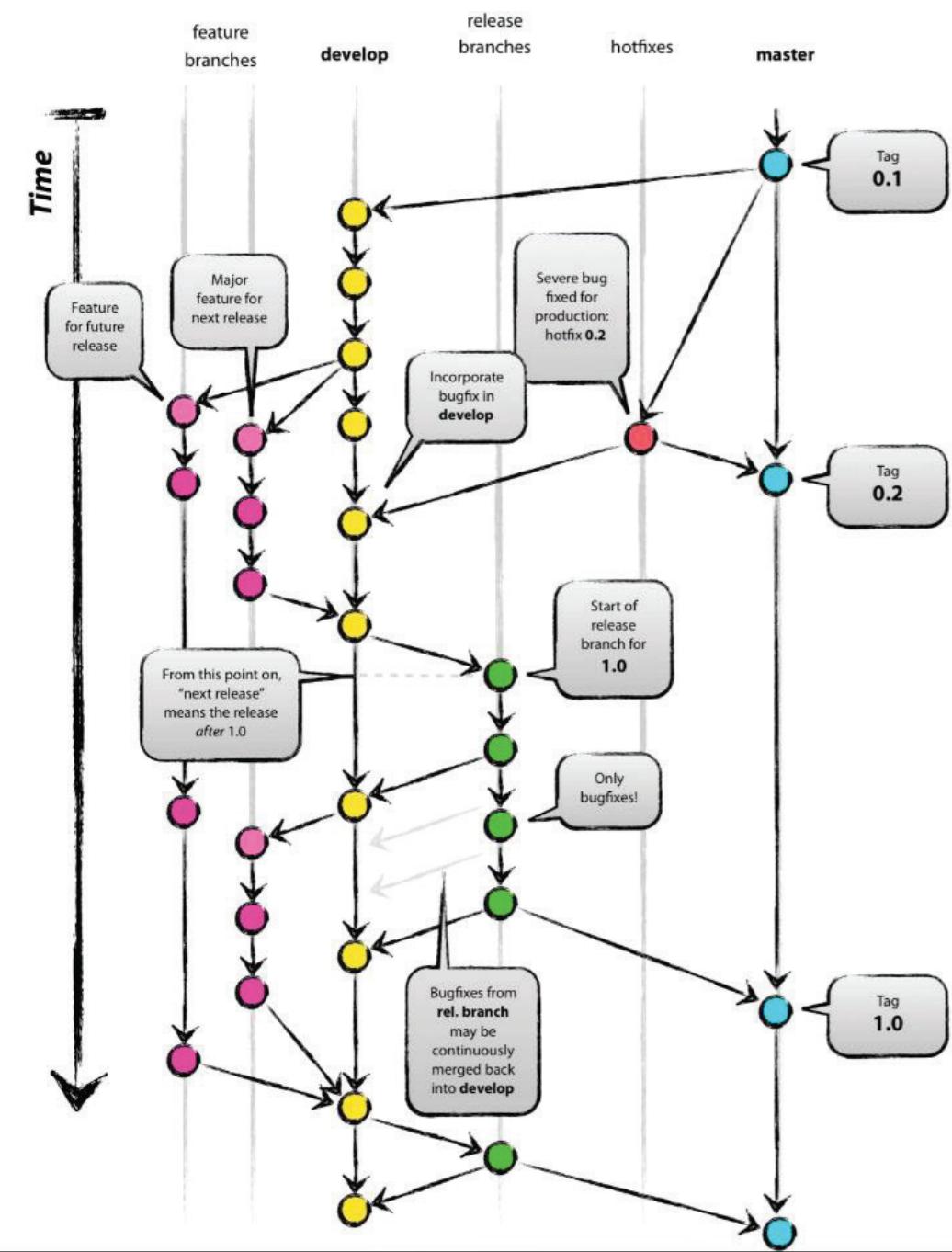


In the above picture fork concept is explained in detail.

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:

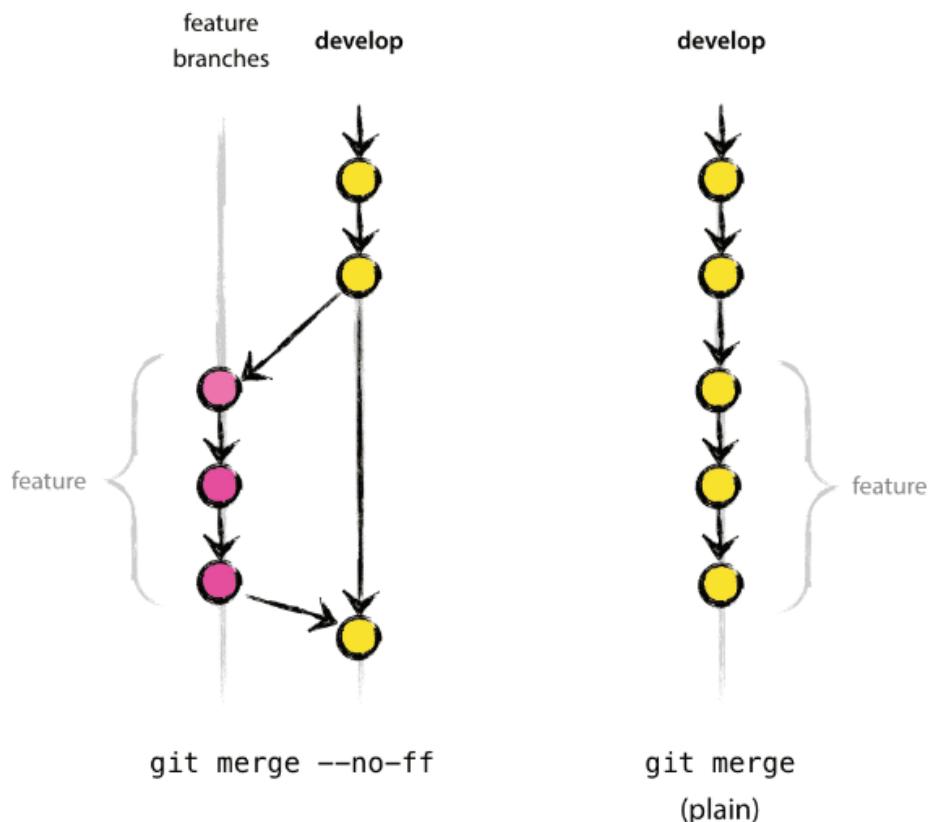
develop

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:

Enter an item name

» Required field



Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Click save after typing in the name selecting the Pipeline

Configure Pipeline job settings

Pipeline

Definition	Pipeline script from SCM
SCM	Git
Repositories	
Repository URL: https://github.com/mgna-repo/webapp.git ? Credentials: - none - Add Advanced... Add Repository	
Branches to build	
Branch Specifier (blank for 'any'): */develop X ? Add Branch	
Repository browser	
(Auto) ?	
Additional Behaviours	
Script Path: Jenkinsfile ? Lightweight checkout: <input checked="" type="checkbox"/> ?	

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.

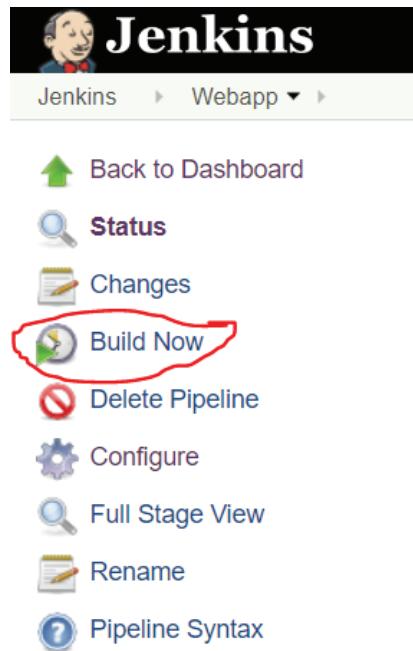
Configure Build trigger to scan the repository periodically as below:

Build Triggers

<input type="checkbox"/> Build after other projects are built	
<input checked="" type="checkbox"/> Build periodically	
Schedule: H 10 * * *	
Would last have run at Sunday, April 21, 2019 10:14:59 AM GMT; would next run at Monday, April 22, 2019 10:14:59 AM GMT. G	

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

Click on Build now to build the web application:

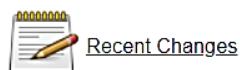


Install Stage View Plugin to review the Build output in stages.

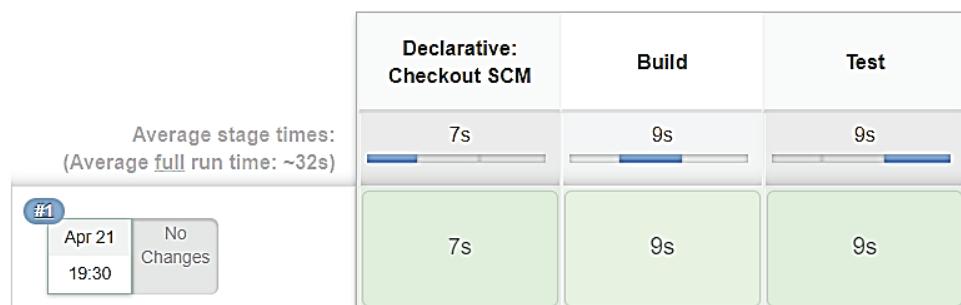
A screenshot of the Jenkins plugin manager. At the top, there's a search bar with the text "stage view". Below it, there are tabs for "Updates", "Available" (which is selected), "Installed", and "Advanced". A table lists available plugins: "Pipeline Stage View" by "Pipeline Stage View Plugin" (version 2.11). At the bottom, there are buttons for "Install without restart" (highlighted in blue), "Download now and install after restart", and "Check now". A note says "Update information obtained: 8 hr 0 min ago".

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

Pipeline Pipeline-Job



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:

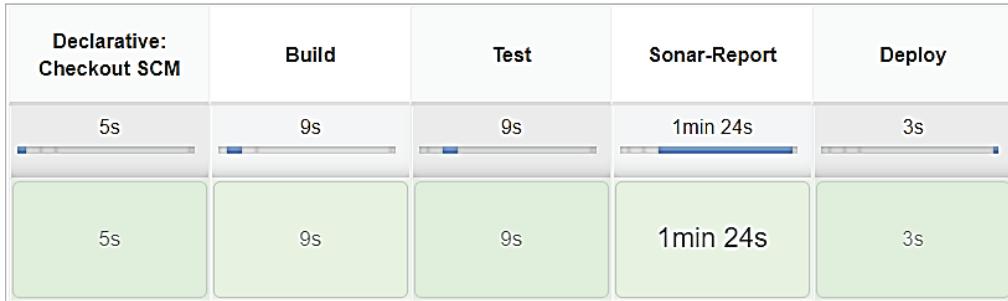
```
stage('Sonar-Report') {
    steps {
        sh 'mvn clean install sonar:sonar -Dsonar.host.url=http://10.x.x.x:9000 -Dsonar.analysis.mode=publish'
    }
}

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.

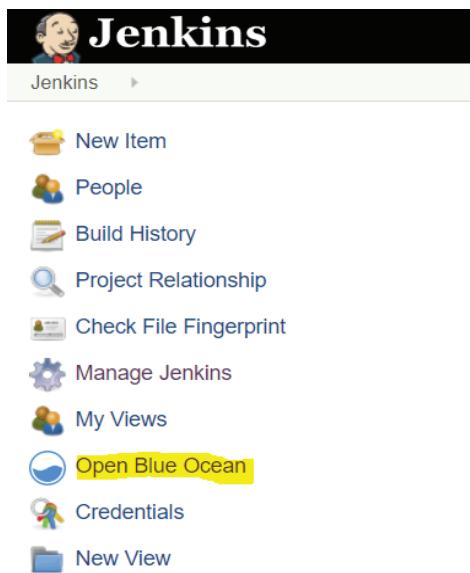
After saving Jenkins file click save and run the job again.



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

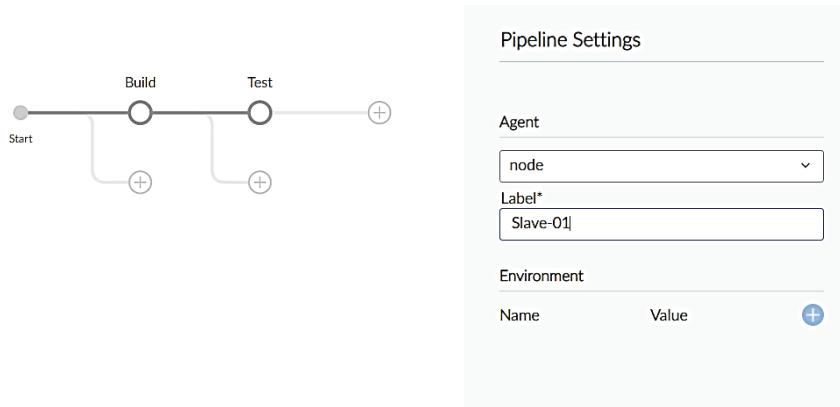
The screenshot shows the Jenkins dashboard with a blue header bar. In the top right corner, there are links for 'Pipelines', 'Administration', and 'Logout'. Below the header is a search bar with the placeholder 'Search pipelines...'. A prominent green button labeled 'New Pipeline' is located in the top right corner of the main content area. The main content area displays a table with one row, 'First-Job', which has a yellow gear icon under 'HEALTH'.

Select Github

The screenshot shows the 'Create Pipeline' wizard. The first step, 'Where do you store your code?', is displayed. It features a vertical flow diagram with three stages: a green checkmark at the top, a blue circle in the middle, and a grey circle at the bottom. To the right of the stages are two columns of options. The first column contains 'Bitbucket Cloud' and 'GitHub'. The second column contains 'Bitbucket Server' and 'GitHub Enterprise'. Below these is a single option 'Git'. Under the GitHub option, there is a note: 'Jenkins needs an access token to authorize itself with GitHub.' followed by a link 'Create an access token here.' A text input field 'Your GitHub access token' and a 'Connect' button are also present. At the bottom of the screen, there is a 'Complete' button.

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

Screenshot of the Sonatype Nexus Repository Manager interface showing the 'Repositories' screen and a configuration dialog for a new hosted repository.

Repositories Screen:

Repository	Type	IQ Policy Violations	Health Check
New Hosted Repository	hosted		ANALYZE
Public Repositories	group		ANALYZE
3rd party	hosted		ANALYZE
Apache Snapshots	proxy		ANALYZE
Central	proxy		ANALYZE
Central M1 shadow	virtual		ANALYZE
Releases	hosted		ANALYZE

New Hosted Repository Configuration:

General Settings:

- Repository ID: upes
- Repository Name: upes
- Repository Type: hosted
- Provider: Maven2
- Format: maven2
- Repository Policy: Release

Access Settings:

- Deployment Policy: Allow Redeploy
- Allow File Browsing: True
- Include in Search: True
- Publish URL: True

1.11 Publishing artifacts to Nexus

Add repositories section to your pom.xml

```
<repositories>
    <repository>
        <id>upes</id>
        <name>Releases</name>
        <url>http://localhost:8081/nexus/content/repositories/upes/</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
<u>Parent Directory</u>			
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

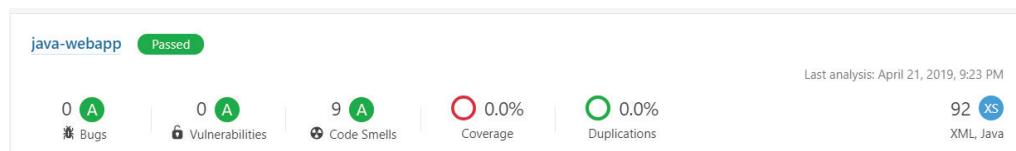
<https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.7.zip>

- 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
```

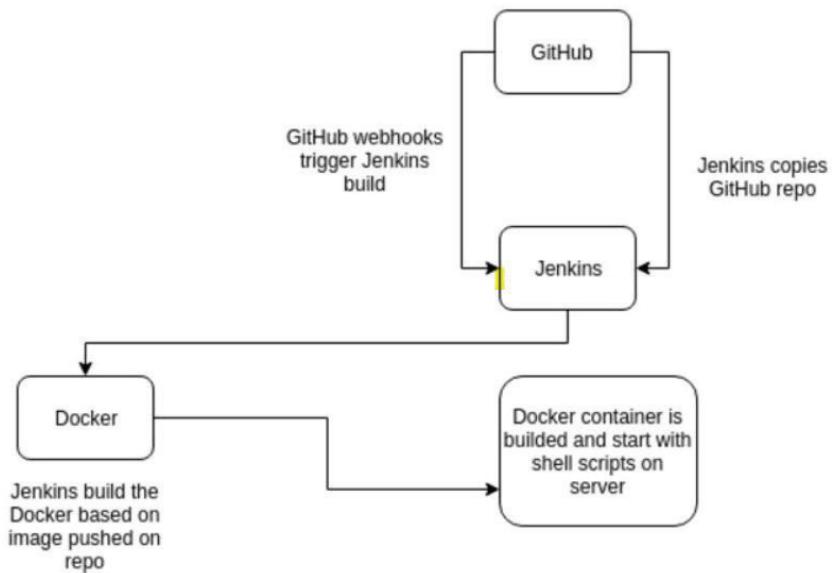
Note: If you install Sonarqube on any other server, replace localhost with the IP address in the above command

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 > /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:

Stage View



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:

magna-repo / webapp

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Releases Tags

v.10 @ Target: master

Excellent! This tag will be created from the target when you publish this release.

v1.0

Write Preview

Describe this release

Attach files by dragging & dropping, selecting or pasting them.

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:



A screenshot of a web browser displaying a list of Jenkins plugins for download. The address bar shows the URL: https://updates.jenkins-ci.org/download/plugins/. The page lists 14 plugins, each with a folder icon, the plugin name, a timestamp, and a download link. The plugins listed are: copy-to-slave/, copyartifact/, cors-filter/, couchdb-statistics/, countjobs-viewstabbar/, covcomplplot/, coverity/, cppcheck/, cppncss/, cptest/, and crap4j/.

copy-to-slave/	2017-07-18 18:17	-
copyartifact/	2017-07-18 18:17	-
cors-filter/	2017-07-18 18:17	-
couchdb-statistics/	2017-07-18 18:17	-
countjobs-viewstabbar/	2017-07-18 18:17	-
covcomplplot/	2017-07-18 18:17	-
coverity/	2017-07-18 18:17	-
cppcheck/	2017-07-18 18:17	-
cppncss/	2017-07-18 18:17	-
cptest/	2017-07-18 18:17	-
crap4j/	2017-07-18 18:17	-

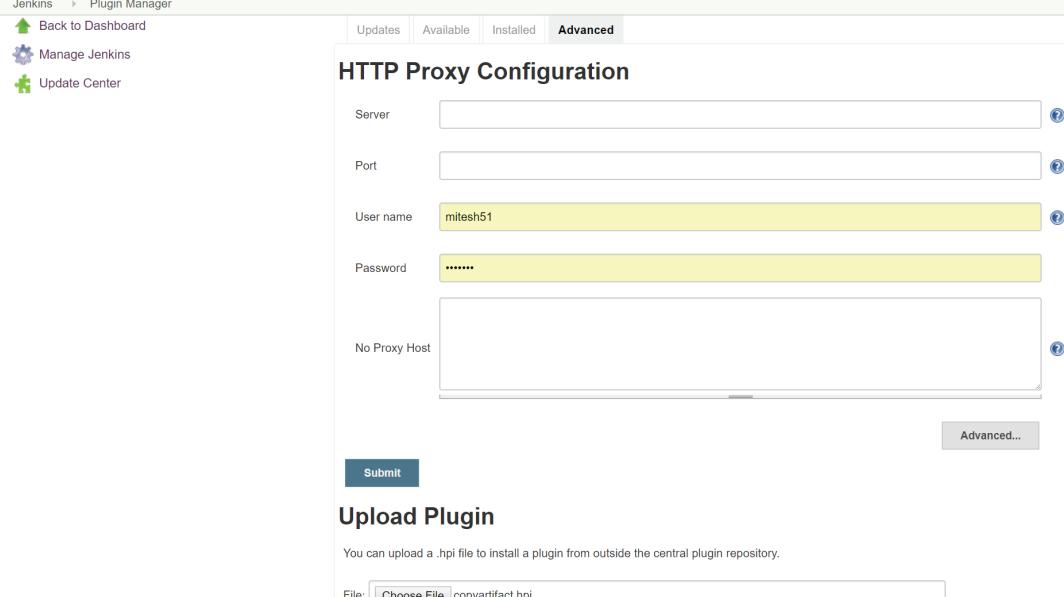
- Download the latest version to your system:

← → ⌂ ⓘ <https://updates.jenkins-ci.org/download/plugins/copyartifact/>

copyartifact

-  [permalink to the latest](#)
-  [1.38.1](#)
-  [1.38](#)
-  [1.37](#)
-  [1.36.1](#)
-  [1.36](#)

- 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:



Jenkins > Plugin Manager

Back to Dashboard | Manage Jenkins | Update Center

Updates Available Installed Advanced

HTTP Proxy Configuration

Server: []

Port: []

User name: mitesh51

Password: [REDACTED]

No Proxy Host: []

Advanced...

Upload Plugin

You can upload a .hpi file to install a plugin from outside the central plugin repository.

File:

- Click on Upload.

Available			
Install	Name	Version	
<input type="checkbox"/>	PowerShell This plugin allows Jenkins to invoke Windows PowerShell as build scripts.	1.3	
<input checked="" type="checkbox"/>	WMI Windows Agents Allows you to setup agents on Windows machines over Windows Management Instrumentation (WMI)	1.4	

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

WMI Windows Agents Success

Restarting Jenkins Running

➔ [Go back to the top page](#)
(you can start using the installed plugins right away)

➔ Restart Jenkins when installation is complete and no jobs are running

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.

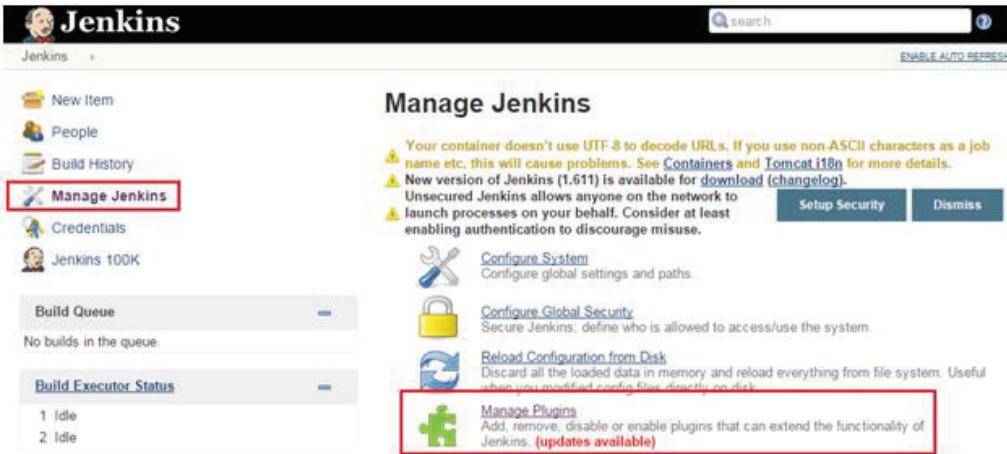
Archiving unused Jobs

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.



The screenshot shows the Jenkins Manage Jenkins page. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is highlighted with a red box), 'Credentials', and 'Jenkins 100K'. The main content area is titled 'Manage Jenkins' and contains several configuration options with yellow warning icons:

- 'Your container doesn't use UTF-8 to decode URLs. If you use non-ASCII characters as a job name etc, this will cause problems. See [Containers](#) and [Tomcat i18n](#) for more details.'
- 'New version of Jenkins (1.611) is available for [download](#) (changelog). Unsecured Jenkins allows anyone on the network to launch processes on your behalf. Consider at least enabling authentication to discourage misuse.'

Below these are three buttons: 'Setup Security' (disabled), 'Dismiss', and 'Configure System' (with a wrench icon). Further down are 'Configure Global Security' (with a lock icon), 'Reload Configuration from Disk' (with a circular arrow icon), and 'Manage Plugins' (with a puzzle piece icon). The 'Manage Plugins' button is also highlighted with a red box.

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

The screenshot shows the Jenkins 'Available Plugins' page. A red box highlights the 'ThinBackup' plugin entry. The entry includes the plugin name, version (1.7.4), a brief description ('Backs up the most important global and job specific configuration files.'), and three buttons at the bottom: 'Install without restart' (which is highlighted with a red box), 'Download now and install after restart', and 'Update information'.

Plugin	Version
Team Foundation Server Plug-in	3.2.0
ThinBackup	1.7.4
Thread Dump Action Plugin	1.0
ThreadFix Plugin	1.5.0

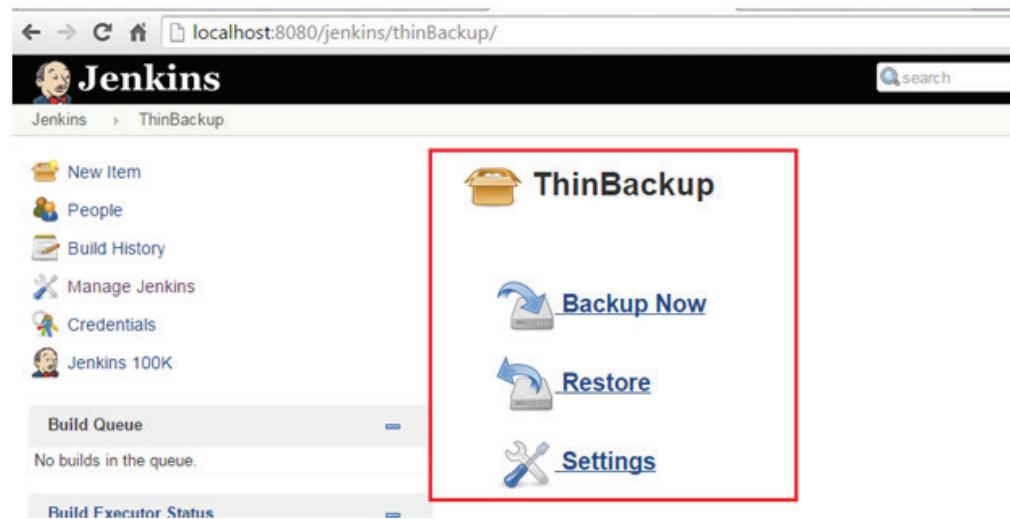
Install without restart **Download now and install after restart** Update information

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.

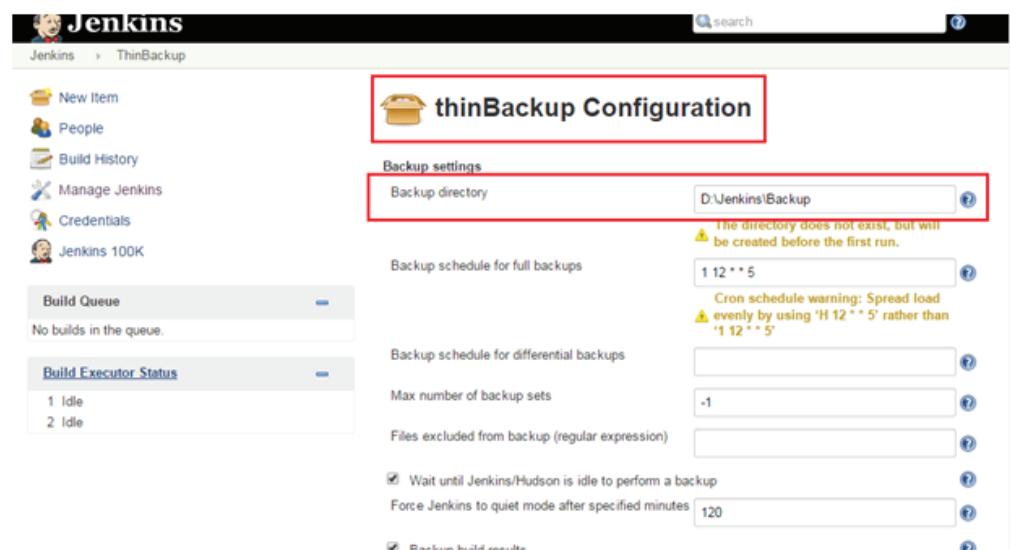
The screenshot shows the Jenkins 'Manage Jenkins' page under the 'Available Plugins' section. A red box highlights the 'ThinBackup' plugin entry. The entry includes the plugin name, icon, and description ('Backup your global and job specific configuration.').

Plugin	Description
In-process Script Approval	Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.
ThinBackup	Backup your global and job specific configuration.
Prepare for Shutdown	Stops executing new builds, so that the system can be eventually shut down safely.

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.

Copyright © 2018 Xebia. All rights reserved.

Please note that the information contained in this classroom material is subject to change without notice. Furthermore, this material contains proprietary information that is protected by copyright. No part of this material may be photocopied, reproduced, or translated to another language without the prior consent of Xebia or ODW Inc. Any such complaints can be raised at sales@odw.rocks

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