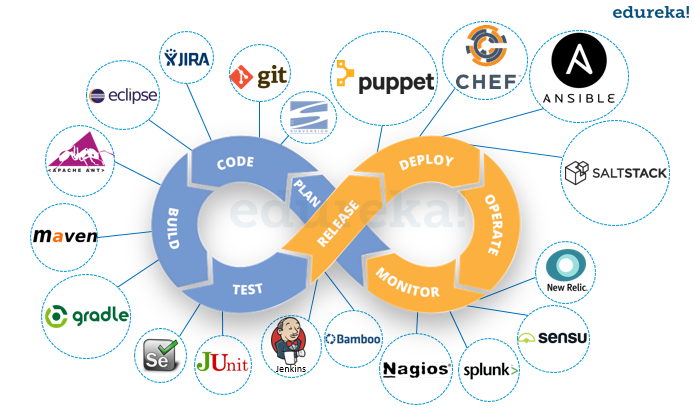
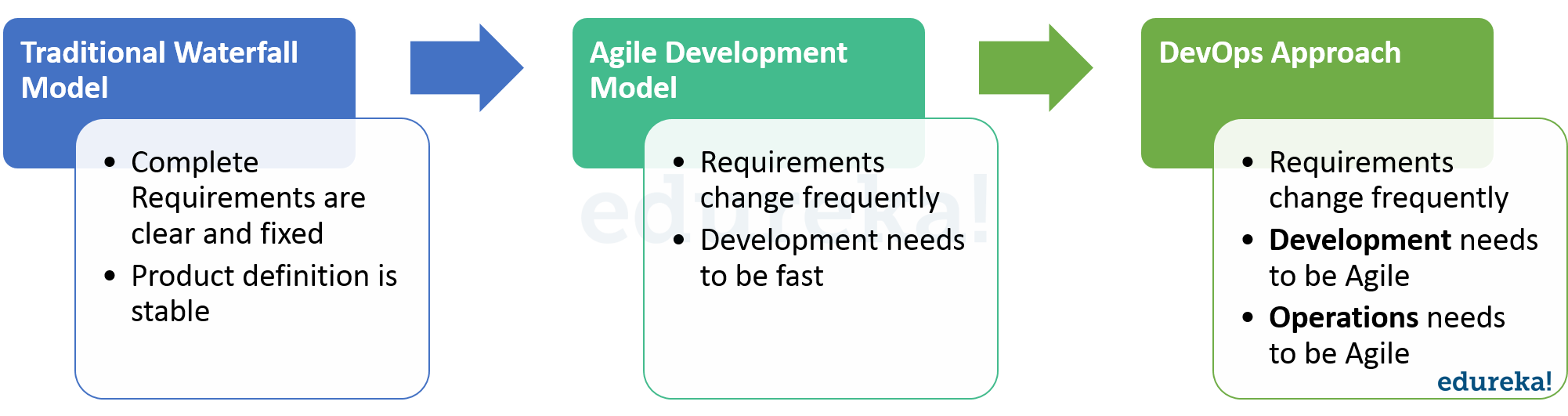
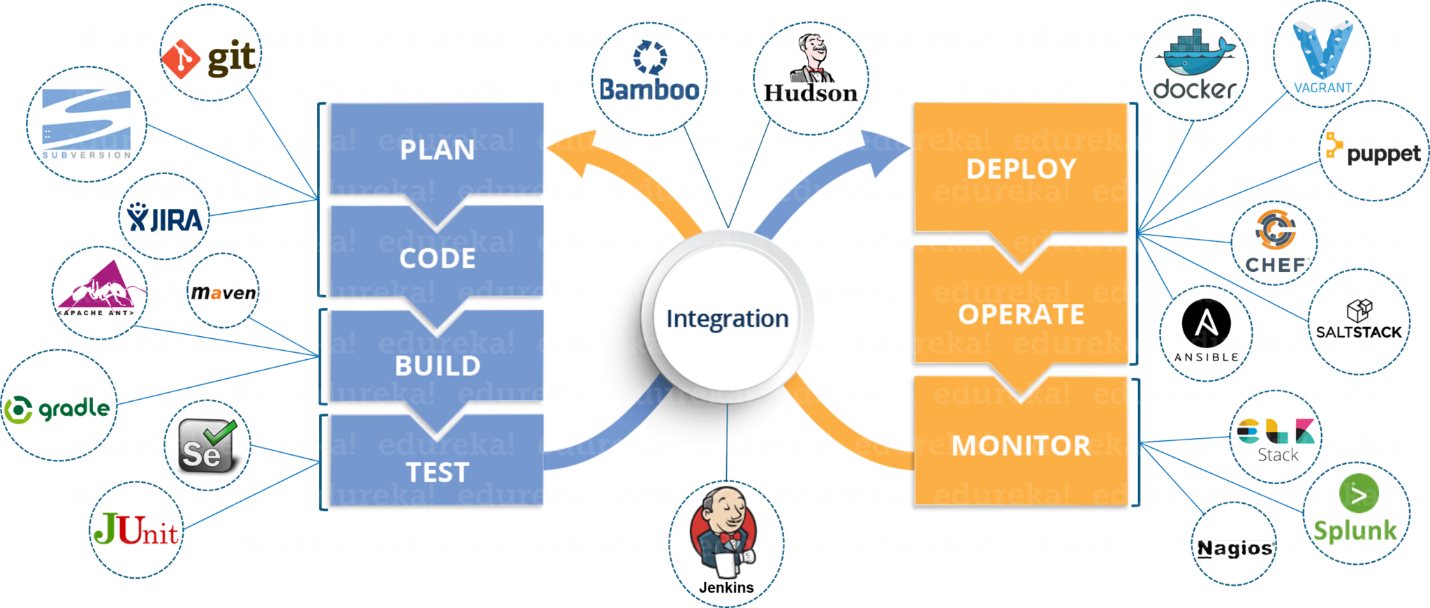
1. **Devops**







Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005 to develop Linux Kernel. Git has the functionality, performance, security and flexibility that most teams and individual developers need.

**What is Git – Why Git** **Came Into Existence?**

Version Control is the management of changes to documents, computer programs, large websites and other collection of information.

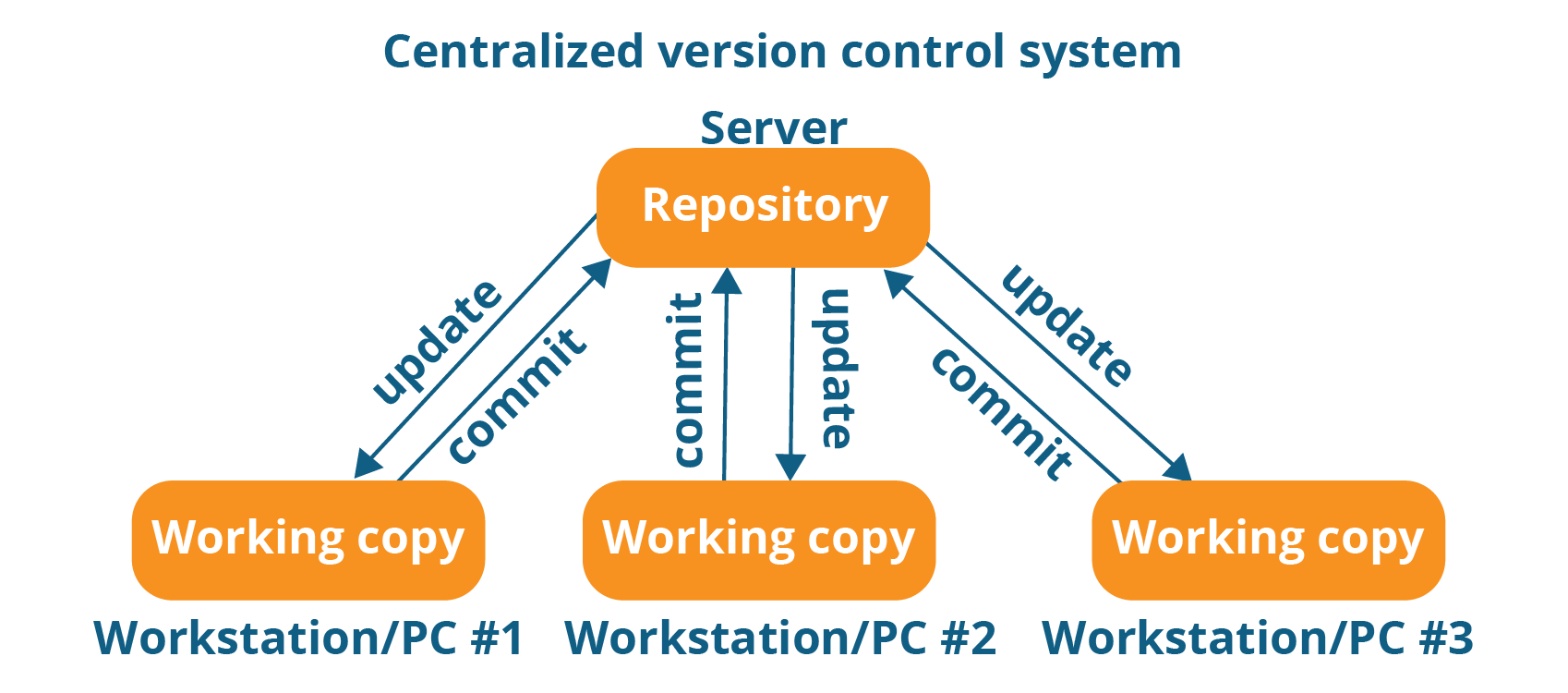
There are two types of VCS:

* Centralized Version Control System (CVCS)
* Distributed Version Control System (DVCS)

## ****Centralized VCS****

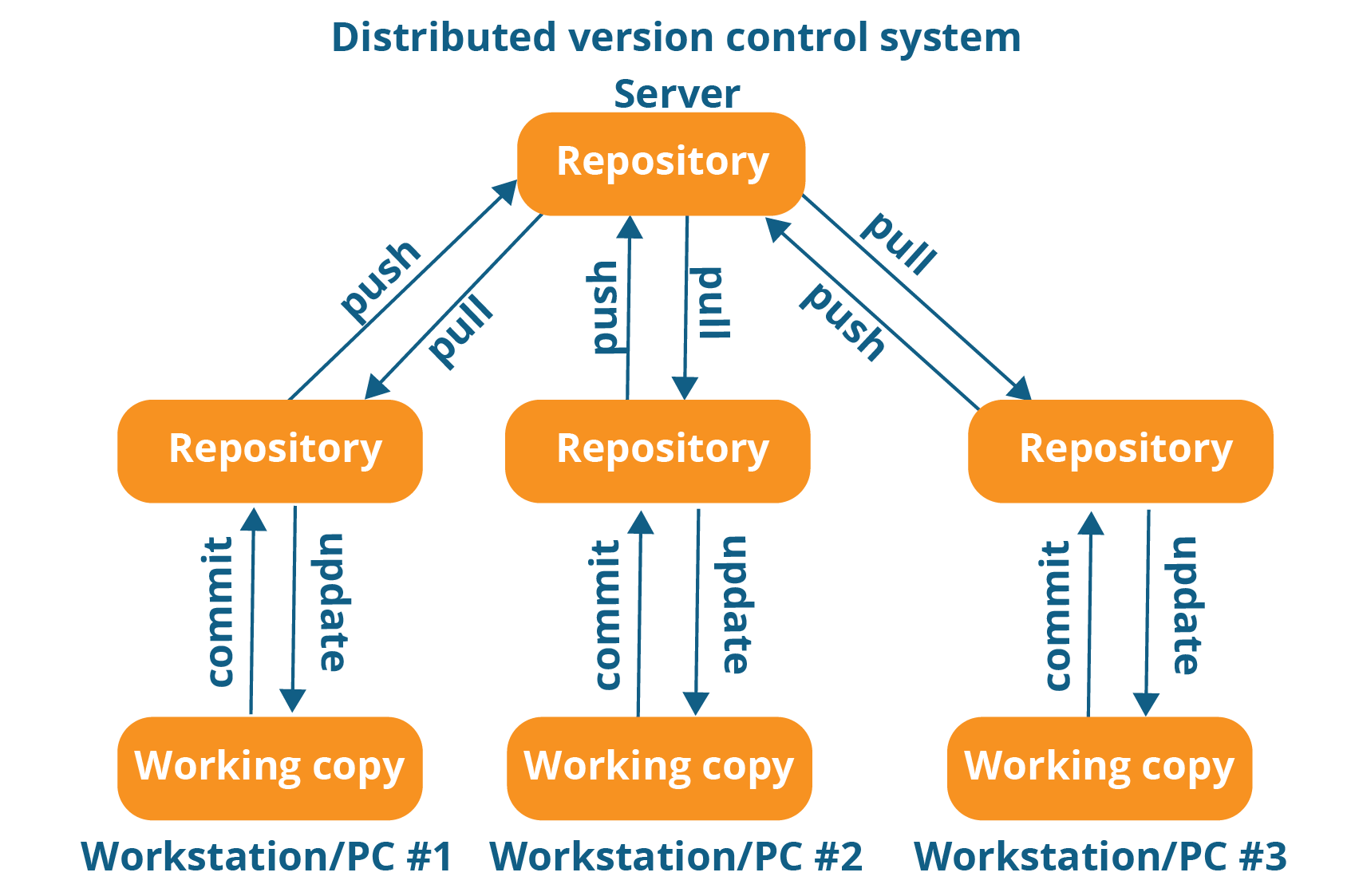
Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

Please refer to the diagram below to get a better idea of CVCS:



## ****Distributed VCS****

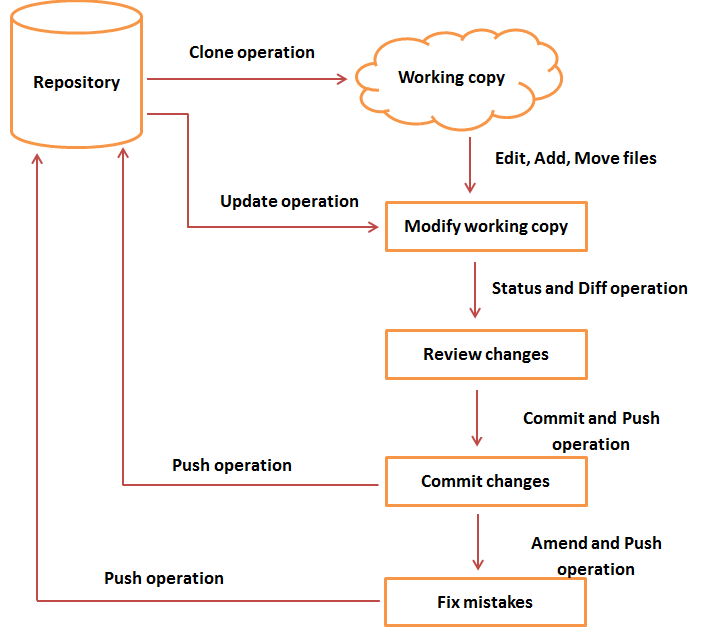
In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.



**GIT**

Workflow is as follows:

* You clone the Git repository as a working copy.
* You modify the working copy by adding/editing files.
* If necessary, you also update the working copy by taking other developer's changes.
* You review the changes before commit.
* You commit changes. If everything is fine, then you push the changes to the repository.
* After committing, if you realize something is wrong, then you correct the last commit and push the



**Git COMMANDS**

git config --global user.name "username"

git config --global user.email "email@example.com"

git config --list

git clone

git --help

git --version

git add filename

git add .

git commit -m "any message"

git commit -a -m "any message"

git push

git pull

git log

git status

git init

git remote add anyname url\_of\_empty\_repository

git push origin master

git branch branchname

git checkout branchname

git merge branchname1 branchname2 (master stl) master

git rebase branchname1 branchname2

git branch -d branchname1 //delete the merged branch

git push origin --delete branchname // delete the mranch from remote

git diff

git revert

git reset

**------------------------------------------------------------------------------------------------------------------------**

**ANT COMMANDS**

ant clean

ant

ant compile

**------------------------------------------------------------------------------------------------------------------------**

**MAVEN COMMANDS**

mvn clean

mvn compile

mvn verify

mvn install

mvn clean install

mvn test

**------------------------------------------------------------------------------------------------------------------------**

**GRADLE COMMANDS**

gradle clean

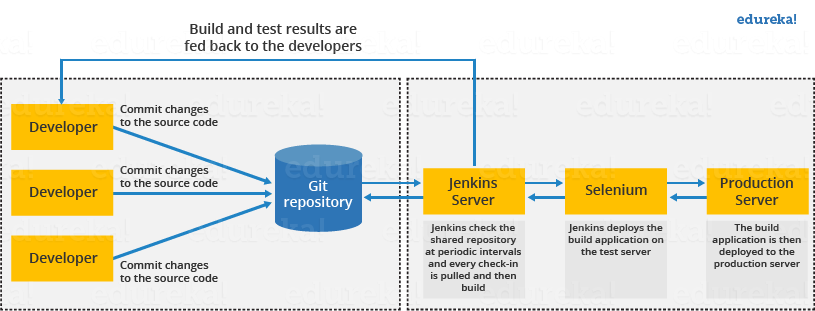
gradle assemble

gradle build

**Jenkins**

Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. You can integrate Jenkins with a number of testing and deployment technologies.

## Jenkins Architecture



his single Jenkins server was not enough to meet certain requirements like:

* Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
* If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.

To address the above stated needs, Jenkins distributed architecture was introduced.

**Jenkins Distributed Architecture**

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

**Jenkins Master**

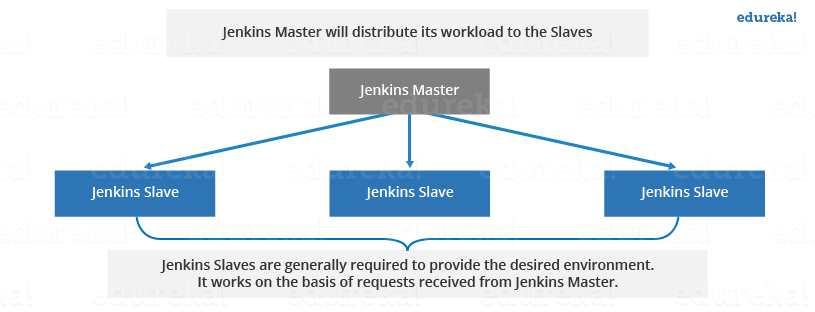
Your main Jenkins server is the Master. The Master’s job is to handle:

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

**Jenkins Slave**

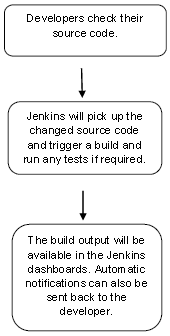
A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

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



**Jenkis**

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place.



**CICD Tutorial**

**Deployment Server**

1.Apache Tomcat

2.Cloud Server – not an freeware

3.Splunk

Ansible Tutorial

Configuration management –

1. maintains configuration of the product performance by keeping a record
2. Updating detailed information which describes an enterprise’s hardware and software.

Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices.,

To install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks.List out the IP addresses of your nodes in the inventory(host file) and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

Ansible is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It’s a human-readable data serialization language & is commonly used for configuration files, but could be used in many applications where data is being stored(Readable format).

Advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed. Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

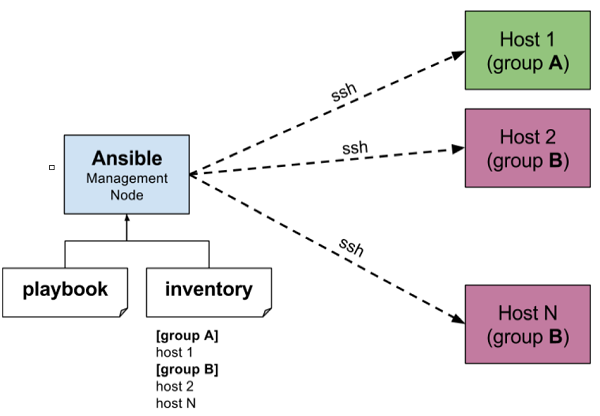
Ansible is designed for multi-tier deployment. Ansible does not manage one system at time, it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely **agentless** which means Ansible works by connecting your nodes through **ssh(by default).**

After connecting to your nodes, Ansible pushes small programs called as “Ansible Modules”. Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file).

## WorkFlow Ansible

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished.

Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



## Installation Process

Mainly, there are two types of machines when we talk about deployment −

* **Control machine** − Machine from where we can manage other machines.
* **Remote machine** − Machines which are handled/controlled by control machine.

There can be multiple remote machines which are handled by one control machine. So, for managing remote machines we have to install Ansible on control machine.

### **Control Machine Requirements**

Ansible can be run from any machine with Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed.

**Note** − Windows does not support control machine.

By default, Ansible uses **ssh** to manage remote machine.

### **Installation through Apt on Ubuntu Machine**

For installing Ansible you have to configure PPA on your machine. For this, you have to run the following line of code −

$ sudo apt-get update

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update

$ sudo apt-get install ansible

. Just run Ansible–version to check the version and just to check whether Ansible was installed properly or not.

**YAML Basics**

Every **YAML** file optionally starts with “---” and ends with “...”.

YAML uses simple key-value pair to represent the data. The dictionary is represented in key: value pair.

**Note** − There should be space between : and value.

### **Example: A student record**

--- #Optional YAML start syntax

james:

name: james john

rollNo: 34

div: B

sex: male

… #Optional YAML end syntax

## Some common words related to Ansible.

**Service/Server** − A process on the machine that provides the service.

**Machine** − A physical server, vm(virtual machine) or a container.

**Target machine** − A machine we are about to configure with Ansible.

**Task** − An action(run this, delete that) etc managed by Ansible.

**Playbook** − The yml file where Ansible commands are written and yml is executed on a machine.

## File Transfer

You can use the Ad-hoc commands for doing **SCP** (Secure Copy Protocol) lots of files in parallel on multiple machines.

### **Transferring file to many servers/machines**

$ Ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"

### **Creating new directory**

$ Ansible abc -m file -a "dest = /path/user1/new mode = 777 owner = user1 group = user1 state = directory"

### **Deleting whole directory and files**

$ Ansible abc -m file -a "dest = /path/user1/new state = absent"

## Create a Playbook

Let us start by writing a sample YAML file. We will walk through each section written in a yaml file.

---

name: install and configure DB

hosts: testServer

become: yes

vars:

oracle\_db\_port\_value : 1521

tasks:

-name: Install the Oracle DB

yum: <code to install the DB>

-name: Ensure the installed service is enabled and running

service:

name: <your service name>

## he Different YAML Tags

Let us now go through the different YAML tags. The different tags are described below −

### **name**

This tag specifies the name of the Ansible playbook.

### **hosts**

This tag specifies the lists of hosts or host group against which we want to run the task.

### **vars**

Vars tag lets you define the variables which you can use in your playbook.

### **tasks**

All playbooks should contain tasks or a list of tasks to be executed.

**Roles**

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

$ ansible-galaxy -h

### **Usage**

ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup] [--help] [options] ...

### **Options**

* **-h, --help** − Show this help message and exit.
* **-v, --verbose** − Verbose mode (-vvv for more, -vvvv to enable connection debugging)
* **--version** − Show program's version number and exit.

### **Creating a Role Directory**

The above command has created the role directories.

$ ansible-galaxy init vivekrole

ERROR! The API server (https://galaxy.ansible.com/api/) is not responding, please try again later.

$ ansible-galaxy init --force --offline vivekrole

- vivekrole was created successfully

$ tree vivekrole/

vivekrole/

├── defaults

│ └── main.yml

├── files ├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── README.md ├── tasks

│ └── main.yml

├── templates ├── tests │ ├── inventory

│ └── test.yml

└── vars

└── main.yml

8 directories, 8 files

## Exception Handling in Playbooks

Exception handling in Ansible is similar to exception handling in any programming language. An example of the exception handling in playbook is shown below.

tasks:

- name: Name of the task to be executed

block:

- debug: msg = 'Just a debug message , relevant for logging'

- command: <the command to execute>

rescue:

- debug: msg = 'There was an exception.. '

- command: <Rescue mechanism for the above exception occurred)

always:

- debug: msg = "this will execute in all scenarios. Always will get logged"

Following is the syntax for exception handling.

* **rescue** and **always** are the keywords specific to exception handling.
* Block is where the code is written (anything to be executed on the Unix machine).
* If the command written inside the block feature fails, then the execution reaches rescue block and it gets executed. In case there is no error in the command under block feature, then rescue will not be executed.
* **Always** gets executed in all cases.
* So if we compare the same with java, then it is similar to try, catch and finally block.
* Here, **Block** is similar to **try block** where you write the code to be executed and **rescue** is similar to **catch block** and **always** is similar to **finally**.

### **Sample Hosts File**

This is the content of hosts file −#File name: hosts

#Description: Inventory file for your application. Defines machine type abc

node to deploy specific artifacts

# Defines machine type def node to upload

metadata.

[abc-node]

#server1 ansible\_host = <target machine for DU deployment> ansible\_user = <Ansible

user> ansible\_connection = ssh

server1 ansible\_host = <your host name> ansible\_user = <your unix user>

ansible\_connection = ssh

[def-node]

#server2 ansible\_host = <target machine for artifact upload>

ansible\_user = <Ansible user> ansible\_connection = ssh

server2 ansible\_host = <host> ansible\_user = <user> ansible\_connection = ssh

|  |
| --- |
| Control machine |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | useradd ansible |
|  | passwd ansible |
|  | visudo |
|  | ansible ALL=(ALL) NOPASSWD: ALL |
|  | su - ansible |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | sudo yum install epel-release |
|  | sudo yum install ansible |
|  | ansible --version |
|  | sudo vi /etc/ansible/ansible.cfg |
|  | ssh-keygen -t rsa |
|  | ls -lah |
|  | cd .ssh |
|  | ls -ltr |
|  | ip a |
|  | sudo vi /etc/hosts |
|  | ssh-copy-id hari03041 |
|  | ssh-copy-id hari03042 |
|  | ssh-copy-id hari03043 |
|  | ssh-copy-id hari03044 |
|  |  |
|  | cat /etc/ansible/ansible.cfg | grep -v '#' |
|  | sudo vi /etc/ansible/hosts // create a group and give only IP |
|  | sudo touch /var/log/ansible.log |
|  | sudo chown ansible:ansible /var/log/ansible.log |
|  | ls -ld /var/log/ansible.log |
|  | ansible -m ping all |
|  |  |
|  | Client System |
|  | \*\*\*\*\* |
|  | useradd ansible |
|  | passwd ansible |
|  | visudo |
|  | su - ansible |
|  |  |
|  | ip a |
|  | uname -n |
|  | sudo vi /etc/hosts |
|  | cd .ssh |
|  | ls -ltr |
|  | cat authorized\_keys |
|  |  |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Ansible Server |
|  | mkdir playbook |
|  |  |
|  | vi first.yml |
|  | - hosts: web |
|  | user: ansible |
|  | sudo: yes |
|  | connection: ssh |
|  | gather\_facts: no |
|  | tasks: |
|  | - name: Copy from the files directory test file |
|  | action: copy src=/home/ansible/files/test4.txt dest=/home/ansible/test4.txt owner=ansible group=ansible mode=0655 backup=yes |
|  |  |
|  |  |
|  |  |
|  | vi tree.yml |
|  | - hosts: web |
|  | user: ansible |
|  | sudo: yes |
|  | connection: ssh |
|  | tasks: |
|  | - name: Install tree software |
|  | action: package name=tree state=latest |
|  |  |
|  |  |
|  | tree |
|  |  |
|  |  |
|  | vi webserver.yml |
|  | - hosts: web |
|  | user: ansible |
|  | sudo: yes |
|  | connection: ssh |
|  | gather\_facts: no |
|  | tasks: |
|  | - name: Install Apache Web Server |
|  | action: yum name=httpd state=installed |
|  | - name: Restart HTTPD |
|  | action: service name=httpd state=restarted |
|  |  |
|  | systemctl status httpd |
|  |  |
|  |  |
|  | ansible -m ping all |
|  | ansible-playbook first.yml |
|  | ansible-playbook tree.yml |
|  | ansible-playbook webserver.yml |

**Maven – Build Tool**

*Maven is a Java tool, so you must have*[*Java*](http://www.oracle.com/technetwork/java/javase/downloads/index.html)*installed in order to proceed.*

1. mvn --version

**Set Environment Variables**

1. Maven home: D:\apache-maven-3.0.5\bin\..
2. Java home: C:\Program Files\Java\jdk1.6.0\_25\jre

### Creating a Project

You will need somewhere for your project to reside, create a directory somewhere and start a shell in that directory. On your command line, execute the following Maven goal:

1. mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

nder this directory you will notice the following [standard project structure](https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html).

1. my-app
2. |-- pom.xml
3. `-- src
4. |-- main
5. | `-- java
6. | `-- com
7. | `-- mycompany
8. | `-- app
9. | `-- App.java
10. `-- test
11. `-- java
12. `-- com
13. `-- mycompany
14. `-- app
15. `-- AppTest.java

The src/main/java directory contains the project source code, the src/test/java directory contains the test source, and the pom.xml file is the project's Project Object Model, or POM.

#### The POM

The pom.xml file is the core of a project's configuration in Maven. It is a single configuration file that contains the majority of information required to build a project in just the way you want.

1. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3. <modelVersion>4.0.0</modelVersion>
5. <groupId>com.mycompany.app</groupId>
6. <artifactId>my-app</artifactId>
7. <version>1.0-SNAPSHOT</version>
8. <packaging>jar</packaging>
10. <name>Maven Quick Start Archetype</name>
11. <url>http://maven.apache.org</url>
13. <dependencies>
14. <dependency>
15. <groupId>junit</groupId>
16. <artifactId>junit</artifactId>
17. <version>4.8.2</version>
18. <scope>test</scope>
19. </dependency>
20. </dependencies>
21. </project>

#### Build the Project

1. mvn package

Rather than a goal, this is a *phase*. A phase is a step in the [build lifecycle](https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html), which is an ordered sequence of phases. For example, if we execute the *compile* phase, the phases that actually get executed are:

1. validate
2. generate-sources
3. process-sources
4. generate-resources
5. process-resources
6. compile

You may test the newly compiled and packaged JAR with the following command:

1. java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App

Which will print the quintessential:

1. Hello World!

### Running Maven Tools

#### Maven Phases

Although hardly a comprehensive list, these are the most common *default* lifecycle phases executed.

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

There are two other Maven lifecycles of note beyond the *default* list above. They are

* **clean**: cleans up artifacts created by prior builds
* **site**: generates site documentation for this project

For example, *package*executes *jar:jar* if the project type is a JAR, and *war:war* if the project type is - you guessed it - a WAR.

An interesting thing to note is that phases and goals may be executed in sequence.

1. mvn clean dependency:copy-dependencies package

This command will clean the project, copy dependencies, and package the project (executing all phases up to *package*

#### Generating the Site

1. mvn site

This phase generates a site based upon information on the project's pom. You can look at the documentation generated under target/site.

**Apache Ant** is a Java based build tool from Apache Software Foundation. Apache Ant's build files are written in XML. Apache ANT -automate the build and deployment process.

ANT stands for Another Neat Tool. It is a Java-based build tool from Apache

Need for a Build Tool

On an average, a developer spends a substantial amount of time doing mundane tasks like build and deployment that include:

* Compiling the code
* Packaging the binaries
* Deploying the binaries to the test server
* Testing the changes
* Copying the code from one location to another

To automate and simplify the above tasks, Apache Ant is useful. It is an Operating System build and deployment tool that can be executed from the command line.

Ant was created by **James Duncan Davidson** (the original author of Tomcat).

## Installing Apache Ant

* Ensure that the JAVA\_HOME environment variable is set to the folder where your JDK is installed.
* Download the binaries from [https://ant.apache.org](https://ant.apache.org/)
* Create a new environment variable called **ANT\_HOME** that points to the Ant installation folder, in this case **c:\apache-ant-1.8.2-bin**folder.
* Append the path to the Apache Ant batch file to the PATH environment variable. In our case this would be the **c:\apache-ant-1.8.2-bin\bin**folder.

## Verifying Apache Ant Installation

C:\>ant -version

For this exercise, create a file called build.xml anywhere in your computer with the following contents in it −

<?xml version = "1.0"?>

<project name = "Hello World Project" default = "info">

<target name = "info">

<echo>Hello World - Welcome to Apache Ant!</echo>

</target>

</project>

A target is a collection of tasks that you want to run as one unit. Targets can have dependencies on other targets. For example, a **deploy** target may have a dependency on the **package** target, the **package** target may have a dependency on the **compile** target and so forth. Dependencies are denoted using the **depends** attribute. For example −

<target name = "deploy" depends = "package">

....

</target>

<target name = "package" depends = "clean,compile">

....

</target>

<target name = "clean" >

....

</target>

<target name = "compile" >

....

</target>

To run the ant build file, open up command prompt and navigate to the folder where the build.xml resides, and type **ant info**. You could also type **ant**instead. Both will work, because **info** is the default target in the build file. You should see the following output −

C:\>ant

Buildfile: C:\build.xml

info: [echo] Hello World - Welcome to Apache Ant!

BUILD SUCCESSFUL

Total time: 0 seconds

**Puppet Tutorial**

Puppet is a configuration management technology to manage the infrastructure on physical or virtual machines. It is an open-source software configuration management tool developed using Ruby which helps in managing complex infrastructure.

Puppet should have an understanding of the system administration, infrastructure, and network protocol communication.

Puppet follows client-server model, where one machine in any cluster acts as client known as puppet master and the other acts as server known as slave on nodes. Puppet has the capability to manage any system from scratch, starting from initial configuration till endof-life of any particular machine.

## Features of Puppet System

Following are the most important features of Puppet.

### **Idempotency**

Puppet, one can safely run the same set of configuration multiple times on the same machine. In this flow, Puppet checks for the current status of the target machine and will only make changes when there is any specific change in the configuration.

Idempotency helps in managing any particular machine throughout its lifecycle starting from the creation of machine, configurational changes in the machine, till the end-of-life.

## Why Chef?

Chef is a configuration management technology used to automate the infrastructure provisioning. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company’s server. It has the capability to get integrated with any of the cloud technology.

In DevOps, we use Chef to deploy and manage servers and applications in-house and on the cloud.

# **Prerequisites**

an understanding of system administration, infrastructure and network protocol communication. To automate the infrastructure provisioning, one should have a command over basic Ruby script writing and the underlying system where one wants to use Chef.

## Features of Chef

Following are the most prominent features of Chef −

* Chef uses popular Ruby language to create a domain-specific language.
* Chef does not make assumptions on the current status of a node. It uses its mechanisms to get the current status of machine.
* Chef is ideal for deploying and managing the cloud server, storage, and software.

## Advantages of Chef

Chef offers the following advantages −

* **Lower barrier for entry** − As Chef uses native Ruby language for configuration, a standard configuration language it can be easily picked up by anyone having some development experience.
* **Excellent integration with cloud** − Using the knife utility, it can be easily integrated with any of the cloud technologies. It is the best tool for an organization that wishes to distribute its infrastructure on multi-cloud environment.

## Chef − Related Technologies

Following is the list of Chef related technologies.

### **Puppet**

Puppet provides a standard way of delivering and operating software, no matter where it runs. It is an automated administrative engine for Linux, Unix, and Windows system that performs administrative tasks based on centralized specification.

The primary **features of Puppet** are as follows −

* Implementing new systems with a uniform configuration.
* Updating the systems and upgrading the security and software packages.
* Incorporating new features and adding dexterous capabilities.
* Customizing configurations for ensuring the availability of data sources.
* Optimizing the available resources and minimizing the cost.
* Simplifying the roles and enabling the team to focus on core and productive issues.
* Getting a bird’s eye view of the available infrastructure.

### **Ansible**

Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy. Avoid writing scripts or custom code to deploy and update your applications — automate in a language that approaches plain English, using SSH, with no agents to install on remote systems.

The primary **features of Ansible** are as follows −

* Simple and easy to learn
* Written in Python
* Agentless
* YAML-based Playbooks
* Ansible galaxy

### **SaltStack**

SaltStack is used for data-driven configuration. It is a new approach of infrastructure management built on dynamic communication bus. It is used for data-driven orchestration, remote execution for any infrastructure, and configuration management for any app stack.

### **Fabric**

**Fabric** is a Python-based programming language, which is developed as an API of Python which needs to be imported in Python code in order to configure and manage an infrastructure.

**Docker**

**Docker -** [**https://katacoda.com/**](https://katacoda.com/)

**Docker commands**

**1.To create a new container**

**Usage: docker create [OPTIONS] IMAGE [COMMAND] [ARG...] [flags**

**docker create –v /config –name cont1 imagename**

**2.** To copy files into a container you use the command docker cp. The following command will copy the config.conf file into our dataContainer and the directory config.

**docker cp config.conf cont1:/config/**

**3.** Using the --volumes-from <container> option we can use the mount volumes from other containers inside the container being launched. In this case, we'll launch an Ubuntu container which has reference to our Data Container. When we list the config directory, it will show the files from the attached container.

docker run --volumes-from dataContainer ubuntu ls /config

4.If we wanted to move the Data Container to another machine then we can export it to a .tar file.

docker export dataContainer > dataContainer.tar

5.command lists all running containers, the image used to start the container and uptime.

Docker ps

6.The command docker inspect <friendly-name|container-id>provides more details about a running container, such as IP address.

7.The command docker logs <friendly-name|container-id> will display messages the container has written to standard error or standard out.

8.

docker run -d --name redisHostPort -p 6379:6379 redis:latest

9. option -p 6379 enables her to expose Redis but on a randomly available port. She decides to test her theory using docker run -d --name redisDynamic -p 6379 redis:latest

10. he command docker run ubuntu ps launches an Ubuntu container and executes the command *ps* to view all the processes running in a container.

Using docker run -it ubuntu bash allows

#### Docker Images

Docker images are built based on a Dockerfile. A Dockerfile defines all the steps required to create a Docker image with your application configured and ready to be run as a container. The image itself contains everything, from operating system to dependencies and configuration required to run your application

Create your *Dockerfile* for building your image by copying the contents below into the editor.

Copy to EditorFROM nginx:alpine

COPY . /usr/share/nginx/html

The first line defines our base image. The second line copies the content of the current directory into a particular location inside the container.

Build our static HTML image using the build command below.

docker build -t webserver-image:v1 .

You can view a list of all the images on the host using

docker images.

Launch our newly built image providing the friendly name and tag. As it's a web server, bind port 80 to our host using the *-p* parameter.

docker run -d -p 80:80 webserver-image:v1

Once started, you'll be able to access the results of port 80 via curl docker

To render the requests in the browser use the following links

**Docker commands**

|  |
| --- |
| docker ps |
|  | docker ps -a |
|  | dokcer images |
|  | dokcer images -a |
|  | docker info |
|  | docker pull <imagename>:<Tag> |
|  | docker run -d <image\_ID> /bin/bash |
|  | docker run -it <image\_ID> /bin/bash |
|  | docker restart <Container\_Name> |
|  | docker exec -it <Container\_Name> /bin/bash |
|  | docker attach <Container\_Name> |
|  | docker logs <Container\_Name> |
|  | docker stop <Container\_Name> |
|  | docker inspect <Container\_Name> |
|  | docker commit -m "What did you do to the image" -a "Author Name" container-id repository/new\_image\_name |
|  | docker login -u <Docker\_Username> |
|  | docker images -a |
|  | docker push Docker\_Username/Repository:Tag |
|  | docker export <Container\_Name> > <Exporting filename with tar(mybuild4.tar)> |
|  | #docker export practical\_elion > pushpa.tar |
|  | docker import mybuild4.tar mybuild:importv5 |
|  | docker rm $(docker ps -a -q) |
|  | docker rmi $(docker images -q) |
|  | docker tag <imageID> username/repository:tag # Tag <image> for upload to registry |
|  | docker push username/repository:tag # Upload tagged image to registry |
|  | docker run username/repository:tag # Run image from a registry |
|  |  |
|  | dodocker inspect <container-id> |

|  |
| --- |
| sudo yum -y install docker |
|  | sudo systemctl start docker |
|  | sudo systemctl enable docker |
|  | sudo groupadd docker |
|  | sudo usermod -aG docker jenkins |
|  | sudo systemctl restart jenkins |
|  | sudo systemctl restart docker |
|  |  |
|  |  |
|  |  |
|  | https://docs.ansible.com/ansible/modules\_by\_category.html |
|  |  |
|  |  |
|  |  |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*JENKINS SLAVE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  |  |
|  | sudo yum -y install git java-1.8.0-openjdk |
|  | sudo useradd -d /var/lib/jenkins jenkins |
|  | ssh-keygen |
|  | cat /home/user/.ssh/id\_rsa.pub |
|  | sudo mkdir /var/lib/jenkins/.ssh |
|  | sudo vi /var/lib/jenkins/.ssh/authorized\_keys |
|  | sudo chown -R jenkins:jenkins /var/lib/jenkins/.ssh |
|  | cat /home/user/.ssh/id\_rsa //copy this key |
|  |  |
|  | go to Jenis // Manage Jenkins // Manage Nodes // new node// node name / select permanent Agent/ ok |
|  | Remote root directory - /var/lib/jenkins |
|  | Launch method - Launch slave agents via SSH |
|  | Host - slave system IP |
|  | credentials - add - kind - SSH Usename with Private Key. |
|  | username jenkins |
|  | privatekey- past the id\_rsa key // save |
|  | Host key verication - Manually trusted key verication strategy |