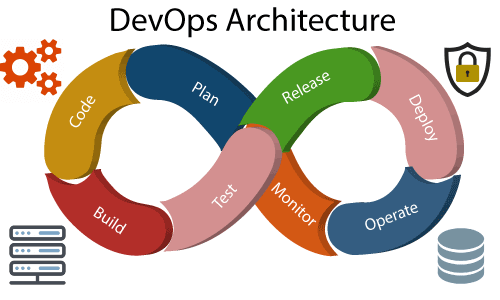
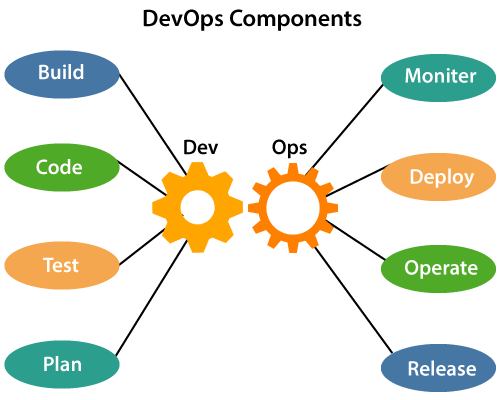
**WHAT IS AND WHY DEVOPS**

DevOps is an IT culture that encourages communication, collaboration, integration and automation among software developers and IT operations in order to improve the speed and quality of delivering software.The DevOps ideals extend agile development practices by further streamlining the movement of software change thru the build, validate, and deploy and delivery stages, while empowering cross-functional teams with full ownership of software applications – from design thru production support

DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

Software Development Life Cycle ( SDLC )

**Plan 🡪 Code 🡪 Build 🡪 Test 🡪 Release 🡪 Deploy 🡪 Operate**

**Microservice Architecture :**

* Microservices design differs from monolithic in that they involve building applications as a set of small, independent services.
* Each microservice focuses on a specific element of business functionality.
* Independent deployment, Independent Coding, Fault tolerance, Increase agility

**Devops Tools**

* **Source Code Management :-**

GitHub , SVN, Bazaar, Perforce, VCS, GitLab, Bitbucket, Team Foundation Server (TFS),Kallithea, Mercurial, Gerrit

* **Build and Release :-**

Maven, Gradle, Scala Oriented Buildtool (SBT ), Terraform, Bower, Apache Ant

* **Deploy Tools :-**

Puupet, Chef, Ansible, CFEngine, Salt

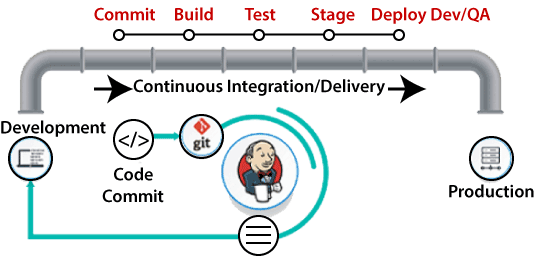
* **Continous Integration :-**

Jenkins, Bamboo, Teamcity, Travis CI, Go CD, Circle CI, Buildbot, Apache Gump

* **Provisioning Tools :-**

Docker, Vagrant, Packer, Vault, Kubernetes

* **Continuous Monitoring :-**

Nagios, Sensu, Prometheus, Ganglia, Monit, ELK

**GIT**

Git is an open-source distributed version control system. It is designed to handle minor to major projects with high speed and efficiency.

**Features of Git:-**

**Open Source**  
Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

**Scalable**  
Git is scalable, which means when the number of users increases, the Git can easily handle such situations.

**Distributed**  
One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository.

**Security**  
Git is secure. It uses the **SHA1 (Secure Hash Function)** to name and identify objects within its repository

**Speed**  
Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**.

**Supports non-linear development**  
Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development.

**Data Assurance**  
The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID.

**Offline Working**  
One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work.

**Undo Mistakes**  
One additional benefit of Git is we can **Undo** mistakes.

**Track the Changes**  
Git facilitates with some exciting features such as **Diff, Log,** and **Status**, which allows us to track changes so we can **check the status, compare** our files or branches.

$ apt-get update

$ apt-get install git-core

$ git --version

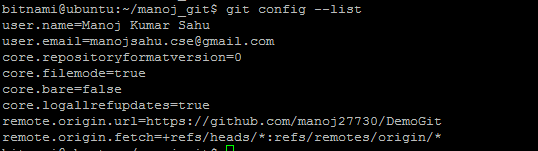
git version 2.24.0

**Configure the Git for the First use**

$ git config --global user.name "javaTpoint"

$ git config --global user.email "javatpoint@xyz"

$ git config --list

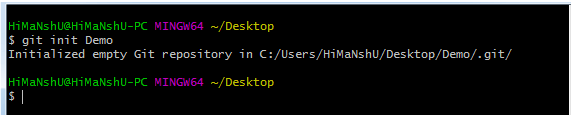


**Git Configuration Levels :**

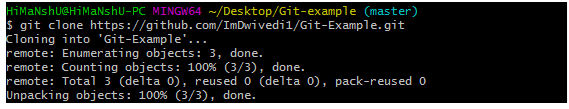
**Local** - It is the default level in Git. Git config will write to a local level if no configuration option is given. Local configuration values are stored in **.git/config** directory as a file.

**Global** - The global level configuration is user-specific configuration. User-specific means, it is applied to an individual operating system user. Global configuration values are stored in a user's home directory. **~ /.gitconfig** on UNIX systems and **C:\Users\\.gitconfig** on windows as a file format

**System** - The system-level configuration is applied across an entire system. The entire system means all users on an operating system and all repositories. The system-level configuration file stores in a **gitconfig** file off the system directory. **$(prefix)/etc/gitconfig** on UNIX systems nd **C:\ProgramData\Git\config** on Windows.

create a local repository - git init Demo

$ git clone URL



add one or more files to staging (Index) area

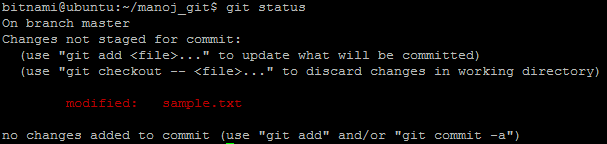
$ git add Filename

$ git add\*

$ git commit -m " Commit Message"

git commit -a

$ git status  -- to display the state of the working directory and the staging area



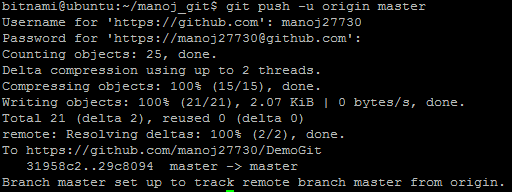
git branch –a -- list all branches in local and remote repositories

git branch -- List local branches

git branch –r -- listing the remote branches from Git Bash

git show-branch -- Command for seeing the branches and their commits

git show-branch –r -- listing the remote tracking branches



$ git push origin master -- Sends the changes made on the master branch, to your remote repository

$ git push --all  -- pushes all the branches to the server repository

$ git pull URL  -- receive data from GitHub

git config –list -- verify your Git settings of the local repository

The heads of the branches are stored in **.git/refs/heads/** directory.

[CentOS]$ ls -1 .git/refs/heads/

master

[CentOS]$ cat .git/refs/heads/master

570837e7d58fa4bccd86cb575d884502188b0c49

**DOCKER**

Docker is an **open-source centralized platform designed** to create, deploy, and run applications. Docker uses **container** on the host's operating system to run applications.

It allows applications to use the same **Linux kernel** as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.

**Docker Components :-**

**Docker client, Docker server, Docker machine, Docker hub, Docker composes**

**Why Docker :-**

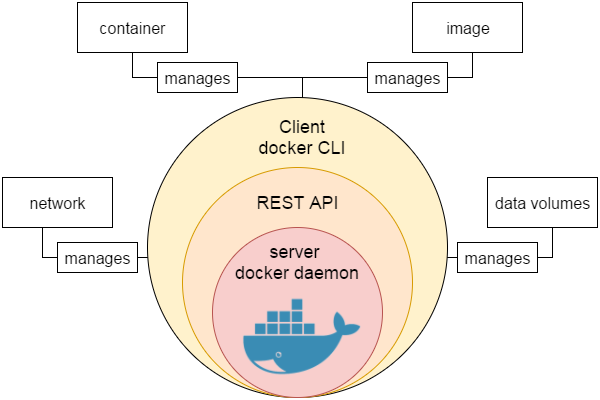
Docker allows us to easily install and run software without worrying about setup or dependencies.Operators use Docker to run and manage apps in isolated containers for better compute density.Docker allows you to use a remote repository to share your container with others.

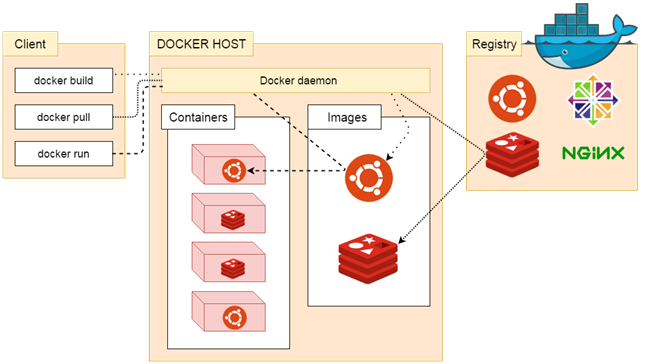
**Docker Engine :-**It is a client server application that contains the following major components.

* A server which is a type of long-running program called a daemon process.
* The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface client.

Docker Architecture

* Docker follows client-server architecture. Its architecture consists mainly three parts.
* 1) **Client:** Docker provides Command Line Interface (CLI) tools to client to interact with Docker daemon. Client can build, run and stop application. Client can also interact to Docker\_Host remotely.
* 2) **Docker\_Host:** It contains Containers, Images, and Docker daemon. It provides complete environment to execute and run your application.
* 3) **Registry:** It is global repository of images. You can access and use these images to run your application in Docker environment.





VMs vs Containers

VMs Containers

Heavyweight Lightweight

Limited performance Native performance

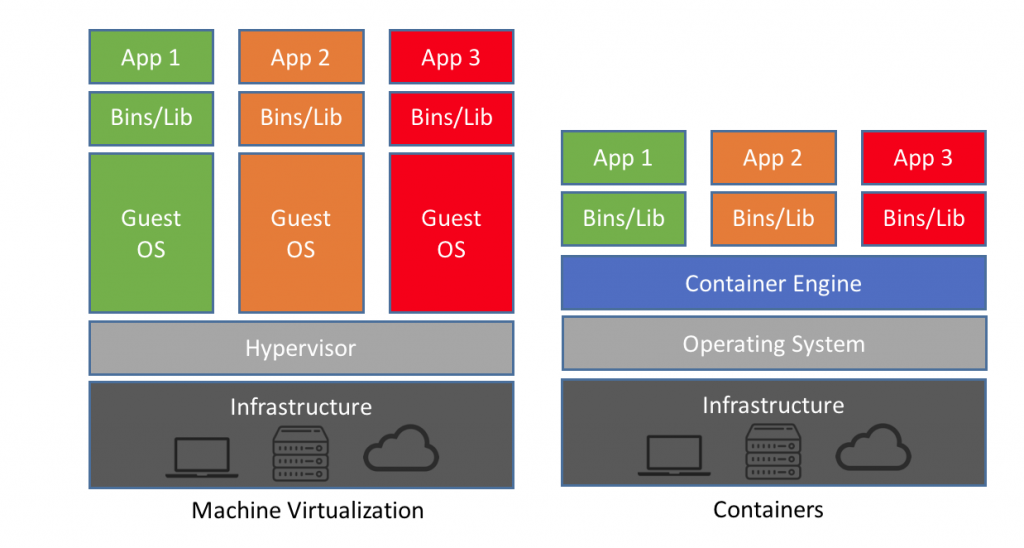
Each VM runs in its own OS All containers share the host OS

Hardware-level virtualization OS virtualization

Startup time in minutes Startup time in milliseconds

Allocates required memory requires less memory space

Fully isolated and hence more secure Process-level isolation, possibly less secure



# Docker Installation:-

* It only works on a 64-bit Linux installation.
* It requires Linux kernel version 3.10 or higher

# $ uname -r

# Login as a root user by using sudo command

# $ apt-get update

# $ apt-get install apt-transport-https ca-certificates

# Add the new GPG key

# $ sudo apt-key adv \

# --keyserver hkp://ha.pool.sks-keyservers.net:80 \

# --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

# Verify that APT is pulling from the right repository

# $ apt-cache policy docker-engine

# Install the recommended packages.

# $ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual

# Install docker-engine

# $ sudo apt-get install docker-engine

# Start the docker daemon.

# $ sudo service docker start

# 

# Verify that docker is installed correctly by running the hello-world image

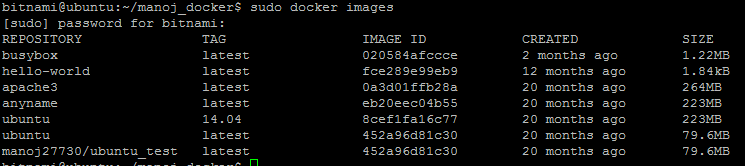
# $ sudo docker run hello-world

# Docker Image :-

# In Docker, everything is based on Images. An image is a combination of a file system and parameters.Images can be downloaded from Docker Hub using the Docker run command.

docker run image --- Pull an image from hub

docker images --- Display all the docker images



Repository – Image name

TAG − This is used to logically tag images.

Image ID − This is used to uniquely identify the image.

Created − The number of days since the image was created.

Virtual Size − The size of the image

docker rmi ImageID -- remove Docker images

docker images –q -- Return only the Image ID’s of the images

sudo docker inspect <Repository> -- show detailed information on the Image

# Docker Container :-

# Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.Deploy the containers anywhere in physical, virtual even in cloud. There can be multiple containers in a single host machine and each container is isolated from one another.

# In simple terms, an image is a template, and a container is a copy of that template. You can have multiple containers (copies) of the same image

# Run an image with below command in an interactive mode

$ sudo docker run -i -t alpine /bin/bash

$ sudo docker run –it centos /bin/bash

Then hit Crtl+p and you will return to your OS shell

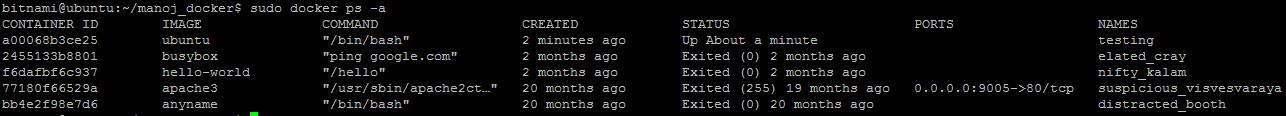
docker ps – list the containers currently running

docker ps –a – list all the containers

bitnami@ubuntu:~/manoj\_docker$ sudo docker run -it --name testing docker.io/ubuntu /bin/bash

It will create a container but once you exit the container it will stop. So start again,

sudo docker start testing --- Start the container



Again move into the container using below

sudo docker exec -it testing /bin/bash

sudo docker history 452a96d81c30 -- see all the commands that were run with an image via a container

sudo docker run -d -it --name testing1 ubuntu /bin/bash – start a container in daemon mode

sudo docker attach <container ID> -- Get into a container

docker top ContainerID --  top processes within a container

docker stop ContainerID -- stop a running container

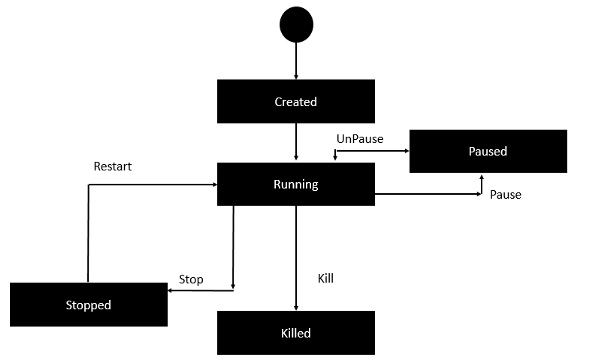
docker rm ContainerID -- delete a container

docker stats ContainerID -- provide the statistics of a running container

docker pause/unpause ContainerID -- pause the processes in a running container

docker kill ContainerID

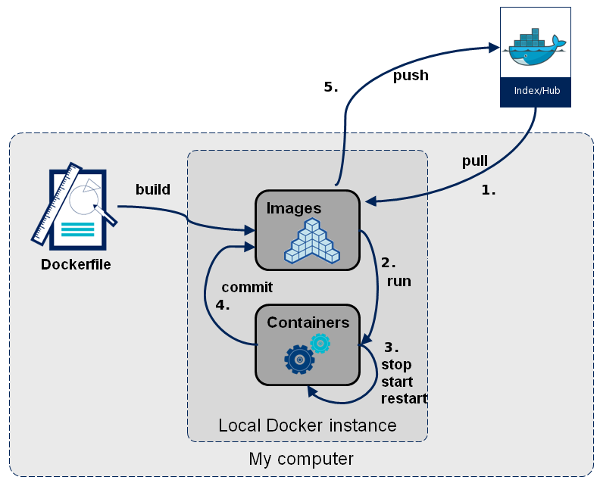
sudo docker search ansible – search a container



sudo docker save ubuntu > ubuntu\_save.tar  
sudo docker export ubuntu > ubuntu\_export.tar

docker run -i --expose=22 b5593e60c33b bash -- expose port while starting the container

sudo docker commit CONTAINER\_ID nginx-template – edit and save the container.



# .

# DOCKER FILE :-

# Docker also gives you the capability to create your own Docker images, and it can be done with the help of Docker Files. A Docker File is a simple text file with instructions on how to build your images.

# A Dockerfile is a text document that contains commands that are used to assemble an image. We can use any command that call on the command line. Docker builds images automatically by reading the instructions from the Dockerfile.

# Below are some dockerfile commands you must know:-

| **Command** | **Description** |
| --- | --- |
| ADD | Copies a file from the host system onto the container |
| CMD | The command that runs when the container starts |
| ENTRYPOINT |  |
| ENV | Sets an environment variable in the new container |
| EXPOSE | Opens a port for linked containers |
| FROM | The base image to use in the build. This is mandatory and must be the first command in the file. |
| MAINTAINER | An optional value for the maintainer of the script |
| ONBUILD | A command that is triggered when the image in the Dcokerfile is used as a base for another image |
| RUN | Executes a command and save the result as a new layer |
| USER | Sets the default user within the container |
| VOLUME | Creates a shared volume that can be shared among containers or by the host machine |
| WORKDIR | Set the default working directory for the container |

docker build -- build the Docker File

docker build -t ImageName:TagName dir

docker run -- run the new image created

sudo docker login – Login to docker repository

docker pull demousr/demorep

docker tag imageID Repositoryname

docker push Repositoryname

sudo docker run -p 8080:8080 -p 50000:50000 jenkins

sudo docker network ls -- list all the networks associated with Docker on the host

**Sample Docker Files**

FROM ubuntu

RUN apt-get update

RUN apt-get install –y apache2

RUN apt-get install –y apache2-utils

RUN apt-get clean

EXPOSE 80 CMD [“apache2ctl”, “-D”, “FOREGROUND”]

sudo docker build –t=”mywebserver”

sudo docker run –d –p 80:80 mywebserver

FROM ubuntu:latest

MAINTAINER Andrew Odewahn "odewahn@oreilly.com"

RUN apt-get update

RUN apt-get install -y python python-pip wget

RUN pip install Flask

ADD hello.py /home/hello.py

WORKDIR /home

#Download base image ubuntu 16.04  
FROM ubuntu:16.04  
   
# Update Software repository  
RUN apt-get update  
   
# Install nginx, php-fpm and supervisord from ubuntu repository  
RUN apt-get install -y nginx php7.0-fpm supervisor && \  
    rm -rf /var/lib/apt/lists/\*  
   
#Define the ENV variable  
ENV nginx\_vhost /etc/nginx/sites-available/default  
ENV php\_conf /etc/php/7.0/fpm/php.ini  
ENV nginx\_conf /etc/nginx/nginx.conf  
ENV supervisor\_conf /etc/supervisor/supervisord.conf  
   
# Enable php-fpm on nginx virtualhost configuration  
COPY default ${nginx\_vhost}  
RUN sed -i -e 's/;cgi.fix\_pathinfo=1/cgi.fix\_pathinfo=0/g' ${php\_conf} && \  
    echo "\ndaemon off;" >> ${nginx\_conf}  
   
#Copy supervisor configuration  
COPY supervisord.conf ${supervisor\_conf}  
   
RUN mkdir -p /run/php && \  
    chown -R www-data:www-data /var/www/html && \  
    chown -R www-data:www-data /run/php  
# Volume configuration  
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d", "/var/log/nginx", "/var/www/html"]  
   
# Configure Services and Port  
COPY start.sh /start.sh  
CMD ["./start.sh"]  
   
EXPOSE 80 443

docker build -t nginx\_image

mkdir -p /webroot

docker run -d -v /webroot:/var/www/html -p 80:80 --name hakase nginx\_image

**Docker Compose :-**

**Docker Compose** is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

In this chapter, we will see how to get started with Docker Compose. Then, we will look at how to get a simple service with MySQL and NGNIX up and running using Docker Compose.

## Docker Compose ─ Installation

The following steps need to be followed to get Docker Compose up and running.

**Step 1** − Download the necessary files from **github** using the following command −

curl -L "https://github.com/docker/compose/releases/download/1.10.0-rc2/dockercompose

-$(uname -s) -$(uname -m)" -o /home/demo/docker-compose

The above command will download the latest version of Docker Compose which at the time of writing this article is **1.10.0-rc2**. It will then store it in the directory **/home/demo/**.



**Step 2** − Next, we need to provide **execute privileges** to the downloaded Docker Compose file, using the following command −

chmod +x /home/demo/docker-compose



We can then use the following command to see the **compose** version.

### Syntax

docker-compose version

### Parameters

* **version** − This is used to specify that we want the details of the version of **Docker Compose**.

### Output

The version details of Docker Compose will be displayed.

### Example

The following example shows how to get the **docker-compose** version.

sudo ./docker-compose -version

### Output

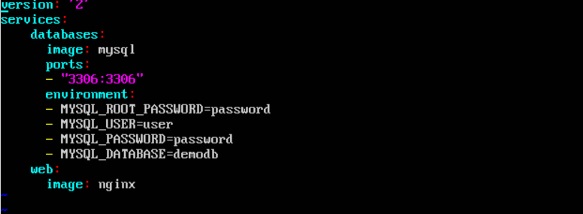
You will then get the following output −



## Creating Your First Docker-Compose File

Now let’s go ahead and create our first Docker Compose file. All Docker Compose files are YAML files. You can create one using the vim editor. So execute the following command to create the **compose** file −

sudo vim docker-compose.yml



Let’s take a close look at the various details of this file −

* The **database** and **web** keyword are used to define two separate services. One will be running our **mysql** database and the other will be our **nginx** web server.
* The **image** keyword is used to specify the image from **dockerhub** for our **mysql** and **nginx** containers
* For the database, we are using the ports keyword to mention the ports that need to be exposed for **mysql**.
* And then, we also specify the environment variables for **mysql** which are required to run **mysql**.

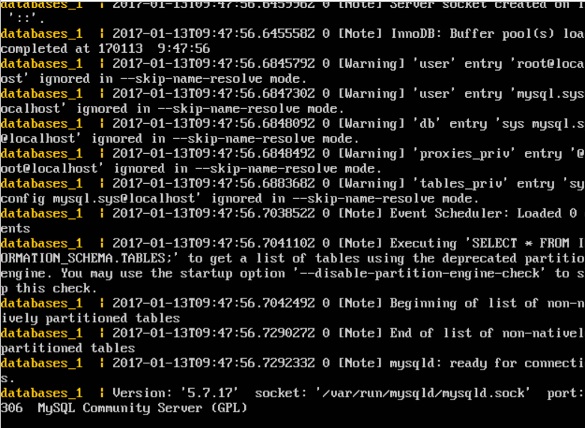
Now let’s run our Docker Compose file using the following command −

sudo ./docker-compose up

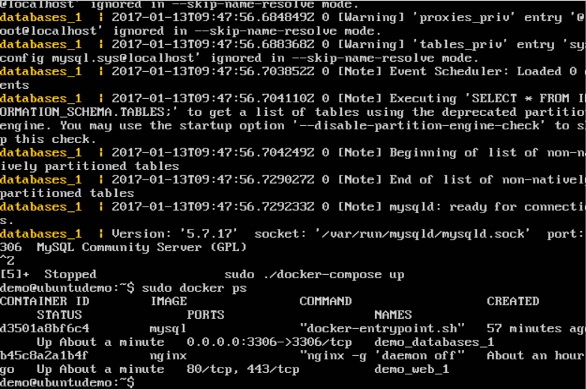
This command will take the **docker-compose.yml** file in your local directory and start building the containers.



Once executed, all the images will start downloading and the containers will start automatically.



And when you do a **docker ps**, you can see that the containers are indeed up and running.



<https://gabrieltanner.org/blog/docker-compose>

**ANSIBLE**

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

# Ansible is easy to deploy because it does not use any **agents** or **custom security** infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server. Ansible is entirely 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.

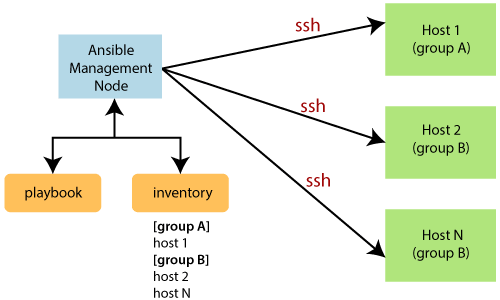
# Why Use Ansible

# Ansible is free to use by everyone.Ansible is very consistent and lightweight, and no constraints regarding the operating system or underlying hardware are present.

# It is very secure due to its agentless capabilities and open SSH security features.Ansible does not need any special system administrator skills to install and use it.Ansible has a smooth learning curve determined by the comprehensive documentation and easy to learn structure and configuration.

# Its modularity regarding plugins, inventories, modules, and playbooks make Ansible perfect companion orchestrate large environments.[next →](https://www.javatpoint.com/ansible-architecture)[← prev](https://www.javatpoint.com/ansible)

# Ansible Workflow:-



**Installation :-**

step 1 - sudo ifconfig | grep inet ---> Check system settings

step 2 - sudo apt-add-repository ppa:ansible/ansible -y -----> add official Ansible PPA repository on the system

sudo apt-get update

sudo apt-get install ansible -y ---> Ubuntu/Debian/Linux Mint

sudo yum install ansible -y ---> RHEL/CentOS/Fedora

step 3 - ansible --version ---> verify the version

step 4 - ssh-keygen -t rsa -b 4096 -C "admin@tecmintlocal.com" ---> create a SSH key in the master

step 5 - ssh-copy-id tecmint@192.168.0.112 ---> copy the created key to remote server’s

step 6 - ssh tecmint@192.168.0.112 ---> After copying all SSH Keys to remote host, now perform a ssh key authentication on all remote hosts to check whether authentication working or not.

step 7 - sudo vim /etc/ansible/hosts ---> Creating Inventory File for Remote Hosts

[web-servers]

192.168.0.112

192.168.0.113

192.168.0.114

step 8 - ansible -m ping web-servers --- check our all 3 server by just doing a ping from my localhost

ansible-playbook <name.yml>-🡪 Run ansible playbook

**Ansible Ad-hoc commands:-**

**1**. **Parallelism and shell commands**

You can reboot your company server in 12 parallel forks at the same time. For this, you need to set up the SSHagent for connection.

$ ssh-agent bash

$ ssh-add ~/.ssh/id\_rsa

To run reboot for all your company servers in the group, 'abc', in 12 parallel forks:

$ ansible abc -a "/sbin/reboot" -f 12

By default, Ansible will run the above ad-hoc commands from the current user account. If you want to change then pass the username in ad-hoc command as follows:

$ ansible abc -a "/sbin/reboot" -f 12 -u username

**2. File Transfer**

You can use ad-hoc commands for doing SCP (secure copy protocol) which means lots of files in parallel on multiple machines or servers.

Transferring file on many machines or servers

$ 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 = 888 owner = user1 group = user1 state = directory"

Deleting all directory and files

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

**3. Managing Packages**

Ad-hoc commands are available for apt and yum module. Here are the following ad-hoc commands using yum.

Below command checks, if the yum package is installed or not, but not update it.

$ ansible abc -m yum -a "name = demo-tomcat-1 state = present"

Below command checks the package is not installed.

$ ansible abc -m yum -a "name = demo-tomcat-1 state = absent"

And below command checks the latest version of package is installed.

$ ansible abc -m yum -a "name = demo-tomcat-1 state = latest"

**4. Managing Users and Groups**

You can manage, create, and remove a user account on your managed nodes with ad-hoc commands.

$ ansible all -m user -a "name=foo password=<crypted password here>"

$ ansible all -m user -a "name=foo state=absent"

**5. Managing Services**

Ensure a service is started on all the webservers.

$ ansible webservers -m service -a "name=httpd state=started"

Alternatively, restart a service on all webservers:

$ ansible webservers -m service -a "name=httpd state=restarted"

Ensure a service is stopped:

$ ansible webservers -m service -a "name=httpd state=stopped"

**6. Gathering Facts**

Fact represents the discovered variables about a system. You can use the facts to implement conditional execution of tasks, and also used to get ad-hoc information about your systems. To see all the facts:

$ ansible all -m setup

**Ansible Playbook:-**

Playbooks are the files where the Ansible code is written. Playbooks are written in YAML format. **YAML** means "Yet Another Markup Language," so there is not much syntax needed. **Playbooks** are one of the core features of Ansible and tell Ansible what to execute, and it is used in complex scenarios.

Playbooks contain the steps which the user wants to execute on a particular machine. And playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.Each playbook is an aggregation of one or more plays in it. Playbooks are structured using Plays. There can be more than one play inside a playbook.

## YAML Tags:-

**Name**

This tag specifies the name of the Ansible playbook. As in what this playbook will be doing. Any logical name can be given to the playbook.

**Hosts**

This tag specifies the lists of hosts or host group against which we want to run the task. The hosts field/tag is mandatory. It tells Ansible on which hosts to run the listed tasks. The tasks can be run on the same machine or on a remote machine. One can run the tasks on multiple machines and hence hosts tag can have a group of hosts’ entry as well.

**Vars**

Vars tag lets you define the variables which you can use in your playbook. Usage is similar to variables in any programming language.

**Tasks**

All playbooks should contain tasks or a list of tasks to be executed. Tasks are a list of actions one needs to perform. A tasks field contains the name of the task. This works as the help text for the user. It is not mandatory but proves useful in debugging the playbook. Each task internally links to a piece of code called a module. A module that should be executed, and arguments that are required for the module you want to execute

**Sample Ansible playbooks**

**Install Apache:-**

---

- hosts: apache

sudo: yes

tasks:

- name: install apache2

apt: name=apache2 update\_cache=yes state=latest

- name: enabled mod\_rewrite

apache2\_module: name=rewrite state=present

notify:

- restart apache2

handlers:

- name: restart apache2

service: name=apache2 state=restarted

------

---

- hosts: webservers

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

**Install vsftpd:-**

- name: Install FTP package on RHEL

yum:

name: vsftpd

state: present

when: ansible\_distribution == "RedHat"

- name: Install FTP package on Ubuntu

apt:

name: vsftpd

state: present

when: ansible\_distribution == "Ubuntu"

- name: Install Firewalld package on RHEL

yum:

name: firewalld

state: present

when: ansible\_distribution == "RedHat"

**Install Jenkins :-**

---

- name: Install Jenkins

hosts: linux

gather\_facts: false

become: true

tasks:

- name: Install yum

yum:

name:

- wget

- java-1.8.0-openjdk

- name: Download jenkins.repo

get\_url:

url: http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo

dest: /etc/yum.repos.d/jenkins.repo

- name: Import Jenkins Key

rpm\_key:

state: present

key: https://jenkins-ci.org/redhat/jenkins-ci.org.key

- name: Install Jenkins

yum:

name: jenkins

state: present

- name: Start & Enable Jenkins

systemd:

name: jenkins

state: started

enabled: true

- name: Sleep for 30 seconds and continue with play

wait\_for: timeout=30

- name: Get init password Jenkins

shell: cat /var/lib/jenkins/secrets/initialAdminPassword

changed\_when: false

register: result

- name: Print init password Jenkins

debug:

var: result.stdout

**PUPPET**

Puppet is an open source DevOps systems management tool for centralizing and automating the configuration management process.

It is used to configure, manage, deploy, and orchestrate various applications and services across the whole infrastructure of an organization.

Puppet is specially designed to manage the configuration of Linux and Windows systems. It is written in Ruby and uses its unique Domain Specific Language (DSL) to describe system configuration.

**Deployment models of configuration management tools**

There are two deployment models for configuration management tools :

Push-based deployment model: initiated by a master node.

Pull-based deployment model: initiated by agents.

**Push-based deployment model:**

In this deployment model master server pushes the configurations and software to the individual agents. After verifying a secure connection, the master runs commands remotely on the agents. For example, Ansible and Salt Stack.

**Pull-based deployment model.**

In this deployment model, individual servers contact a master server, verify and establish a secure connection, download their configurations and software and then configure themselves accordingly — for example, Puppet and Chef.

## How Puppet works?

Puppet is based on a Pull deployment model, where the agent nodes check in regularly after every **1800** seconds with the master node to see if anything needs to be updated in the agent. If anything needs to be updated the agent pulls the necessary puppet codes from the master and performs required actions.

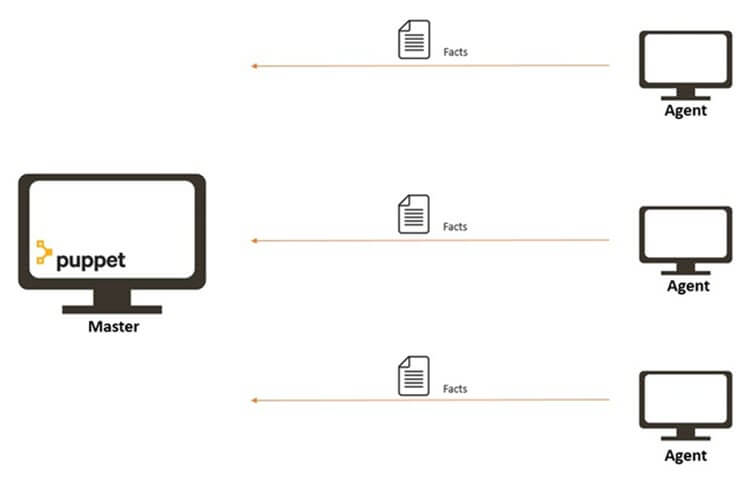
**The Master:**A Linux based machine with Puppet master software installed on it. It is responsible for maintaining configurations in the form of puppet codes. The master node can only be Linux.

**The Agents**:The target machines managed by a puppet with the puppet agent software installed on them.The agent can be configured on any supported operating system such as Linux or Windows or Solaris or Mac OS.

**The communication between master and agent is established through secure certificates.**

Communication between the Master and the Agent:

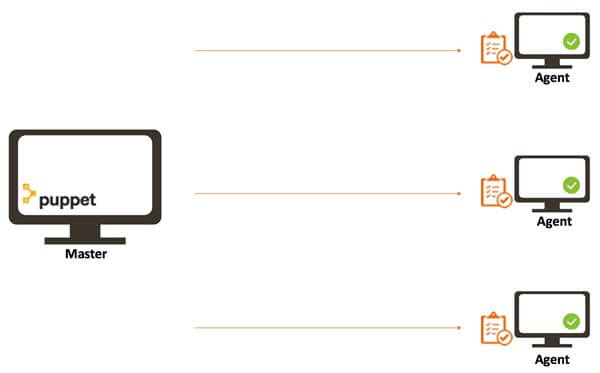
**Step 1)**Once the connectivity is established between the agent and the master, the Puppet agent sends the data about its state to the Puppet master server. These are called Facts: This information includes the hostname, kernel details, IP address, file name details, etc.…



**Step 2)**Puppet Master uses this data and compiles a list with the configuration to be applied to the agent. This list of configuration to be performed on an agent is known as a **catalog.**This could be changed such as package installation, upgrades or removals, File System creation, user creation or deletion, server reboot, IP configuration changes, etc.



**Step 3)**The agent uses this list of configuration to apply any required configuration changes on the node.In case there are no drifts in the configuration, Agent does not perform any configuration changes and leaves the node to run with the same configuration.



**Step 4)**Once it is done the node reports back to puppet master indicating that the configuration has been applied and completed.

## Puppet Blocks

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

### Puppet Resources:

Puppet Resources are the building blocks of Puppet.Resources are the **inbuilt functions** that run at the back end to perform the required operations in puppet.

### Puppet Classes:

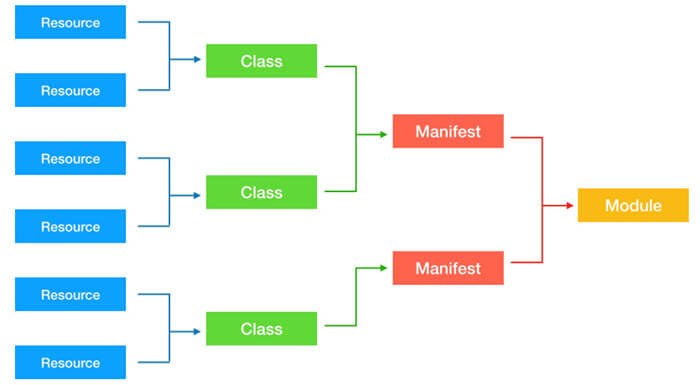
A combination of different resources can be grouped together into a single unit called class.

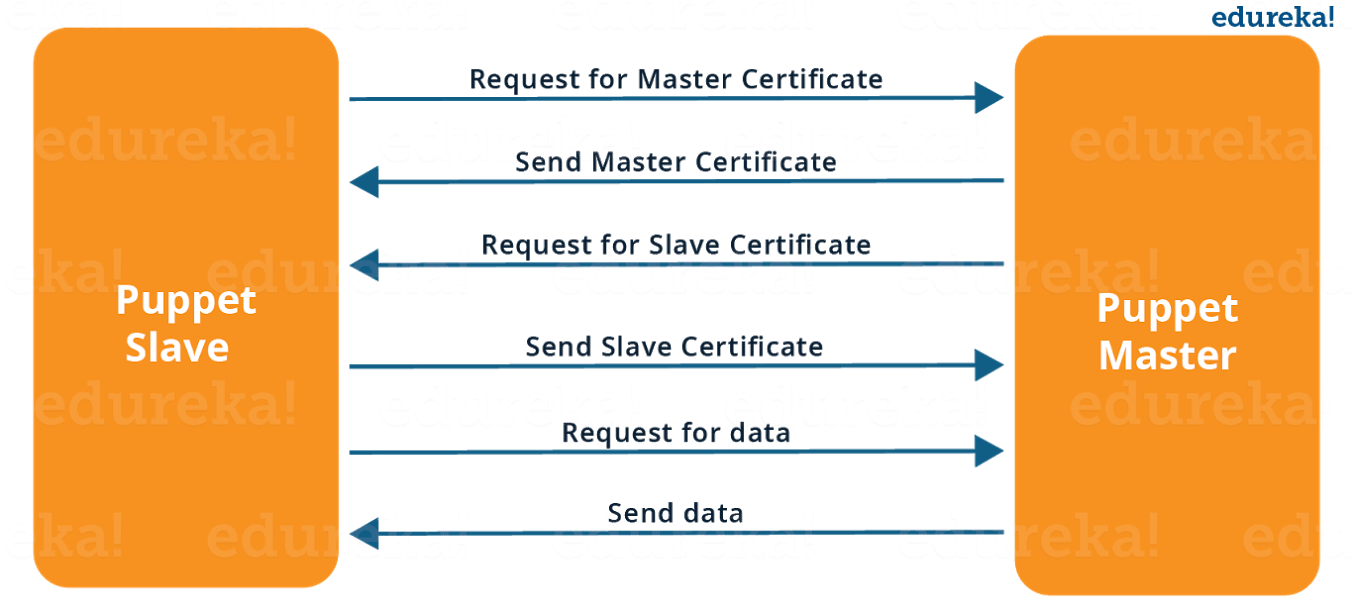
### Puppet Manifest:

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

### Puppet Modules:

Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

**Templates:**Templates use Ruby expressions to define the customized content and variable input. They are used to develop custom content. Templates are defined in manifests and are copied to a location on the system. For example, if one wants to define httpd with a customizable port, then it can be done using the following expression.

**Puppet Master and Slave Communication :-** Puppet Master and Slave communicates through a secure encrypted channel with the help of SSL.

**Puppet Installation**

**Puppet Agent:**

bitnami@ubuntu:~$ cd manoj\_puppet

bitnami@ubuntu:~/manoj\_puppet$ sudo wget <http://apt.puppetlabs.com/puppetlabs-release-trusty.deb>

bitnami@ubuntu:~/manoj\_puppet$ sudo apt-get install puppet

**Puppet Master:**

bitnami@ubuntu:~/manoj\_puppet$ sudo wget <http://apt.puppetlabs.com/puppetlabs-release-trusty.deb>

bitnami@ubuntu:~/manoj\_puppet$ sudo apt-get install puppetmaster

sudo apt-get install puppetserver

## Puppet Pre-Installation

Setup Hostnames -- > hostnamectl set-hostname hostname

vim /etc/hosts -- > Edit the '/etc/hosts' file to configure the FQDN server

Change the IP address and the domain name with your own and paste into it

10.5.5.21   master.hakase-labs.io   master  
10.5.5.22   agent01.hakase-labs.io  agent01

Save and close.

Now restart the hostnamed service to apply a new hostname and FQDN.

*systemctl restart systemd-hostnamed*

And after that, check the hostname and the FQDN using the following command.

*hostname  
hostname -f*

And you will get a new hostname and FQDN has been configured and applied to the system.

**/etc/puppet/puppet.conf 🡪 The puppet configuration file**

**Puppet –V / puppet --version 🡪 check puppet version**

**Generate and Sign Certificates:**

step 1 -- check master services up ( sudo service puppetmaster start )

step 2 -- if step 1 showing fail do sudo service puppetmaster stop and sudo service puppetmaster start

step 3 -- install certificate in master (sudo puppet master --verbose --no-daemonize)

step 4 -- check installed certificate master ( sudo puppet cert list -all ) only master certificate will show

step 5 -- install certificate in agent ( sudo puppet agent -t )

step 6 -- check installed certificate master ( both master and agent certificate should show )

step 7 -- sign the agent certificate in master ( sudo puppet cert --allow-dns-alt-names sign < agent host name > )

step 8 -- check installed certificate master ( both master and agent certificate should show with "+" at starting)

step 9 -- write and apply the site.pp manifest file in master in /etc/puppet/manifest

step 9 -- in agent again run --> sudo puppet agent -t

step 10 -- if any error in step 5 or 9 comment the below 2 lines in puppet.conf file( /etc/puppet/puppet.conf ) in both agent and master

#prerun\_command=/etc/puppet/etckeeper-commit-pre

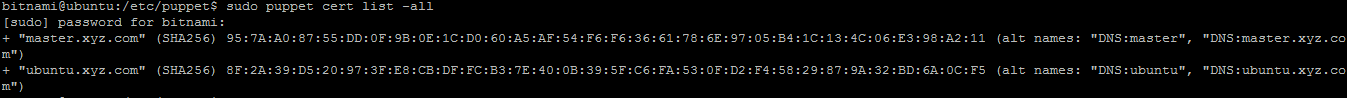
#postrun\_command=/etc/puppet/etckeeper-commit-post

step 11 -- run in agent --> sudo puppet agent --enable

step 12 -- run again in agent --> sudo puppet agent -t

step 13 -- if all run fine test in both the servers at / tmp --> cat example-1.ip

**sudo puppet cert list –all 🡪 show all certificates**

****

**Sudo** puppet agent -t

Info: Downloaded certificate for hostname.example.com from puppet

Info: Using configured environment 'production'

Info: Retrieving pluginfacts

Info: Retrieving plugin

Info: Retrieving locales

Info: Caching catalog for hostname.example.com

Info: Applying configuration version '1547066428'

Info: Creating state file /opt/puppetlabs/puppet/cache/state/state.yaml

Notice: Applied catalog in 0.02 seconds

Puppet agent --fingerprint 🡪 verify the correctness of the sign in the certificates****

By default, the Puppet master listens for client connections on port 8140. If the puppetserver service fails to start, check that the port is not already in use:

****

Add the puppetserver / puppetmaster port '8140' to the firewalld using the following command.

*firewall-cmd --add-port=8140/tcp --permanent  
firewall-cmd –reload*

sudo ufw allow 8140

add the puppetserver service to the startup boot time and start the service

*systemctl enable puppetserver  
systemctl start puppetserver*

**sudo service puppetmaster status --- check puppet master running or not**

****

1. Disable – sudo puppet agent --disable "<MESSAGE>" .
2. Enable – sudo puppet agent –enable

Use the puppet resource command to start and enable the Puppet agent service:

/opt/puppetlabs/bin/puppet resource service puppet ensure=running enable=true

**Note**

On systemd systems, the above command is equivalent to using these two systemctl commands:

systemctl start puppet

systemctl enable puppet

**Puppet Manifest File**

### Manifests

Puppet programs are called manifests. Manifests are composed of puppet code and their filenames use the .pp extension. The default main manifest in Puppet installed via apt is **/etc/puppet/manifests/site.pp**

Manifests are made up of below major components wrapped in a configuration language that includes variables, arrays, conditions, modules

Resources : Individual CIs

Files : Physical files

Templates

Nodes : Specific configuration of each node

Classes : Collection of resources

**Resources :**

In general, a system consists of files, users, services, processes, packages, etc. In Puppet, these are called resources. Resources are the fundamental building blocks in Puppet. All the operations on puppet agents are performed with the help of puppet resources.

Resources are the smallest building block of the Puppet configuration language. They represent a singular element that you wish to evaluate, create, or remove. Puppet comes with many built-in resource types. The stock resource types manipulate system components that you are already familiar with, including:

* Users
* Groups
* Files
* Host file entries
* Packages
* Services

List puppet resources 🡪 bitnami@ubuntu:~$ puppet describe –list

These are the types known to puppet:

augeas - Apply a change or an array of changes to the ...

computer - Computer object management using DirectorySer ...

cron - Installs and manages cron jobs

exec - Executes external commands

file - Manages files, including their content, owner ...

filebucket - A repository for storing and retrieving file ...

group - Manage groups

host - Installs and manages host entries

interface - This represents a router or switch interface

k5login - Manage the `.k5login` file for a user

macauthorization - Manage the Mac OS X authorization database

mailalias - .. no documentation ..

maillist - Manage email lists

mcx - MCX object management using DirectoryService ...

mount - Manages mounted filesystems, including puttin ...

nagios\_command - The Nagios type command

nagios\_contact - The Nagios type contact

nagios\_contactgroup - The Nagios type contactgroup

nagios\_host - The Nagios type host

nagios\_hostdependency - The Nagios type hostdependency

nagios\_hostescalation - The Nagios type hostescalation

nagios\_hostextinfo - The Nagios type hostextinfo

nagios\_hostgroup - The Nagios type hostgroup

nagios\_service - The Nagios type service

nagios\_servicedependency - The Nagios type servicedependency

nagios\_serviceescalation - The Nagios type serviceescalation

nagios\_serviceextinfo - The Nagios type serviceextinfo

nagios\_servicegroup - The Nagios type servicegroup

nagios\_timeperiod - The Nagios type timeperiod

notify - .. no documentation ..

package - Manage packages

resources - This is a metatype that can manage other reso ...

router - .. no documentation ..

schedule - Define schedules for Puppet

scheduled\_task - Installs and manages Windows Scheduled Tasks

selboolean - Manages SELinux booleans on systems with SELi ...

selmodule - Manages loading and unloading of SELinux poli ...

service - Manage running services

ssh\_authorized\_key - Manages SSH authorized keys

sshkey - Installs and manages ssh host keys

stage - A resource type for creating new run stages

tidy - Remove unwanted files based on specific crite ...

user - Manage users

vlan - .. no documentation ..

whit - Whits are internal artifacts of Puppet's curr ...

yumrepo - The client-side description of a yum reposito ...

zfs - Manage zfs

zone - Manages Solaris zones

zpool - Manage zpools

**Puppet Classes:**

Puppet classesare the collection of puppet resources bundled together as a single unit.

Puppet introduced classes to make the structure re-usable and organized.

First, we need to define a class using class definition syntax; classes must be unique and can be declared only once with the same name.

In Puppet, classes are code blocks that can be called in a code elsewhere. Using classes allows you reuse Puppet code, and can make reading manifests easier.

#### Class Definition

A class definition is where the code that composes a class lives. Defining a class makes the class available to be used in manifests, but does not actually evaluate anything.Here is how a class **definition** is formatted:

class <class-name> {

<Resource declarations>

}

Example:

class ntpconfig {

file {

"/etc/ntp.conf":

ensure=> "present", content=> "server 0.centos.pool.ntp.org iburst\n",

}

}

## Class Declaration

To use a defined class in code, use the **include** keyword.

class ntpconfig {

file {

"/etc/ntp.conf":

ensure=> "present",

content=> "server 0.centos.pool.ntp.org iburst\n",

}

}

include ntpconfig

A **normal class declaration** occurs when the include keyword is used in Puppet code, like so:

**include** example\_class

This will cause Puppet to evaluate the code in example\_class.

A **resource-like class declaration** occurs when a class is declared like a resource, like so:

**class** { '**example\_class**': }

**Puppet Module :-**

In Puppet, a module can be defined as a collection of resources, classes, files, definition, and templates. Puppet supports easy re-distribution of modules, which is very helpful in modularity of code as one can write a specified generic module and can use it multiple times with very few simple code changes.

Modules are useful for organizing your Puppet code, because they allow you to split your code into multiple manifests. It is considered best practice to use modules to organize almost all of your Puppet manifests.

To add a module to Puppet, place it in the /etc/puppet/modules directory.

## Puppet Facts

Just before an agent requests for a catalog from the master, the agent first compiles a complete list of information available in itself in the form of a key value pair. The information on the agent is gathered by a tool called facter and each key-value pair is referred as a fact. Following is a common output of facts on an agent.

Puppet supports holding multiple values as an environment variable. This feature is supported in Puppet by using **facter**. In Puppet, facter is a standalone tool that holds the environment level variable. In can be considered similar to env variable of Bash or Linux.

**[root@puppetagent1 ~]# facter**

architecture => x86\_64

augeasversion => 1.0.0

bios\_release\_date => 13/09/2012

bios\_vendor => innotek GmbH

bios\_version => VirtualBox

blockdevice\_sda\_model => VBOX HARDDISK

blockdevice\_sda\_size => 22020587520

blockdevice\_sda\_vendor => ATA

blockdevice\_sr0\_model => CD-ROM

blockdevice\_sr0\_size => 1073741312

blockdevice\_sr0\_vendor => VBOX

blockdevices => sda,sr0

boardmanufacturer => Oracle Corporation

boardproductname => VirtualBox

boardserialnumber => 0

domain => codingbee.dyndns.org

facterversion => 2.1.0

filesystems => ext4,iso9660

fqdn => puppetagent1.codingbee.dyndns.org

hardwareisa => x86\_64

hardwaremodel => x86\_64

hostname => puppetagent1

id => root

interfaces => eth0,lo

ipaddress => 172.228.24.01

ipaddress\_eth0 => 172.228.24.01

ipaddress\_lo => 127.0.0.1

is\_virtual => true

kernel => Linux

kernelmajversion => 2.6

kernelrelease => 2.6.32-431.23.3.el6.x86\_64

kernelversion => 2.6.32

lsbdistcodename => Final

lsbdistdescription => CentOS release 6.5 (Final)

lsbdistid => CentOS

**Sample Manifest files**

## Install LAMP with a Single Manifest :

# execute 'apt-get update'

exec { 'apt-update': # exec resource named 'apt-update'

command => '/usr/bin/apt-get update' # command this resource will run

}

# install apache2 package

package { 'apache2':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure apache2 service is running

service { 'apache2':

**ensure** => running,

}

# install mysql-server package

package { 'mysql-server':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure mysql service is running

service { 'mysql':

**ensure** => running,

}

# install php5 package

package { 'php5':

**require** => Exec['apt-update'], # require 'apt-update' before installing

**ensure** => installed,

}

# ensure info.php file exists

file { '/var/www/html/info.php':

**ensure** => file,

content => '<?php phpinfo(); ?>', # phpinfo code

**require** => Package['apache2'], # require 'apache2' package before creating

}

class testdirs {

# create a directory

file { '/etc/nagios':

ensure => 'directory',

}

# a fuller example, including permissions and ownership

file { '/var/log/nagios':

ensure => 'directory',

owner => 'root',

group => 'root',

mode => '0777',

}}

You can execute any arbitrary command by declaring an exec resource, like the following:

**exec { 'apt-get update':**

**command => '/usr/bin/apt-get update'**

**}**

let’s say you want to execute a command, but you need to make sure a dependency is installed first:

**package { 'python-software-properties':**

**ensure => 'installed'**

**}**

**exec { 'add-repository':**

**command => '/usr/bin/add-apt-repository ppa:ondrej/php5 -y'**

**require => Package['python-software-properties']**

}

The require option receives as parameter a reference to another resource. In this case, we are referring to the Package resource identified as python-software-properties.

**file** { '/tmp/testfile.txt':

**ensure** => present,

**mode** => '0644',

**replace** => true,

**content** => 'holy cow!',

**Install NTP package :-**

**Traditional way :-**

1. rpm -qa | grep -i ntp -- Check ntp packages
2. yum remove ntp -- Remove old package
3. yum install <package>
4. ls -lrt /etc/ntp.conf -- Ensure config file created
5. systemctl status ntp -- Check ntp service status

**Puppet Way** :-

class ntpconfig {

# Installing NTP Package

package {"ntp":

ensure=> "present",

}

# Configuring NTP configuration file

file {"/etc/ntp.conf":

ensure=> "present",

content=> "server 0.centos.pool.ntp.org iburst\n",

}

# Starting NTP services

service {"ntpd":

ensure=> "running",

}

}

puppet parser validate demontp.pp -- Validate any code error

puppet apply demontp.pp

service { 'apache2':

ensure => running,

enable => true,

}

service { 'apache2':

ensure => running / Present / latest / absent

enable => true,

}

**MAVEN**

**Build Process :**

The term build refers to the process by which source code is converted into a stand-alone form that can be run on a computer or to the form itself. One of the most important part of software build is the compilation process where a source code is converted into a executable code.

The Build is a process that covers all the steps required to create a deliverable of your software. In Java the processes include:

* Generating Sources (Sometimes)
* Compiling the Sources
* Compiling test Sources
* Executing Tests (Unit test, Integration test)
* Packaging (into jar, war, ear)
* Running Health checks
* Generating Reports

Java Build Process:

Create sample Java program – App.java

Compile App.java to create App.class -- $Javac App.java

Execute the Code -- $Java App

Using Jar:

Create sample Java program – App.java

Create manifest.mf file

Compile App.java to create App.class -- $Javac App.java

Create Jar file -- $jar cmf App.jar manifest.mf App.class

Execute Jar file -- $java –jar App.jar

## What is Maven?

Maven is an automation and management tool developed by Apache Software Foundation. It was initially released on 13 July 2004. In Yiddish language the meaning of Maven is "accumulator of knowledge".

It is written in Java Language and used to build and manage projects written in C#, Ruby, Scala, and other languages. It allows the developer to create projects using Project Object Model and plugins.

It helps to build projects, dependency, and documentation. Its development process is very similar to ANT.However, it is much advanced than ANT.

Maven is also able to build any number of projects into desired output such as jar, war, metadata.

Maven helps the developer to create a java-based project more easily. Accessibility of new feature created or added in Maven can be easily added to a project in Maven configuration. It increases the performance of project and building process.

**The main feature of Maven is that it can download the project dependency libraries automatically.**

## Installing Maven

What you need to do is:

1. Set the JAVA\_HOME environment variable to point to a valid Java SDK (e.g. Java 8).
2. Download and unzip Maven.
3. Set the M2\_HOME environment variable to point to the directory you unzipped Maven to.
4. Set the M2 environment variable to point to M2\_HOME/bin (%M2\_HOME%\bin on Windows, $M2\_HOME/bin on unix).
5. Add M2 to the PATH environment variable (%M2% on Windows, $M2 on unix).
6. Open a command prompt and type 'mvn -version' (without quotes) and press enter.

After typing in the mvn -version command you should be able to see Maven execute, and the version number of Maven written out to the command prompt.

Note: Maven uses Java when executing, so you need Java installed too (and the JAVA\_HOME environment variable set as explained above). Maven 3.0.5 needs a Java version 1.5 or later. I use Maven 3.3.3 with Java 8 (u45).

## Understanding the problem without Maven

There are many problems that we face during the project development. They are discussed below:

**1) Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

**2) Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

**3) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

Maven simplifies the above mentioned problems. It does mainly following tasks.

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate for new features of Maven

Apache Maven helps to manage

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* mailing list



## Features of Maven

* Simple project setup that follows best practices.
* Consistent usage across all projects.
* Dependency management including automatic updating.
* A large and growing repository of libraries.
* Extensible, with the ability to easily write plugins in Java or scripting languages.
* Instant access to new features with little or no extra configuration.
* **Model-based builds** − Maven is able to build any number of projects into predefined output types such as jar, war, metadata.
* **Coherent site of project information** − Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.
* **Release management and distribution publication** − Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
* **Backward Compatibility** − You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
* **Automatic parent versioning** − No need to specify the parent in the sub module for maintenance.
* **Parallel builds** − It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.
* **Better Error and Integrity Reporting** − Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

Below are the examples of some popular IDEs supporting development with Maven:

* Eclipse
* IntelliJ IDEA
* JBuilder
* NetBeans
* MyEclipse

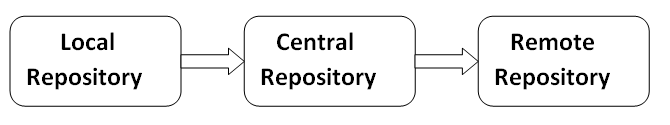
**Maven Repository**

A **maven repository** is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

**Local repository** then **Central repository** then **Remote repository**.



Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

Maven **central repository** is located on the web. It has been created by the apache maven community itself.

The path of central repository is: <https://repo1.maven.org/maven2/>.

Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file.

You can search any repository from Maven official website **mvnrepository.com**.

## How to use Maven

* To configure the Maven, you need to use Project Object Model, which is stored in a pom.xml-file.
* POM includes all the configuration setting related to Maven. Plugins can be configured and edit in the <plugins> tag of a pom.xml file. And developer can use any plugin without much detail of each plugin.
* When user start working on Maven, it provides default setting of configuration, so the user does not need to add every configuration in pom.xml

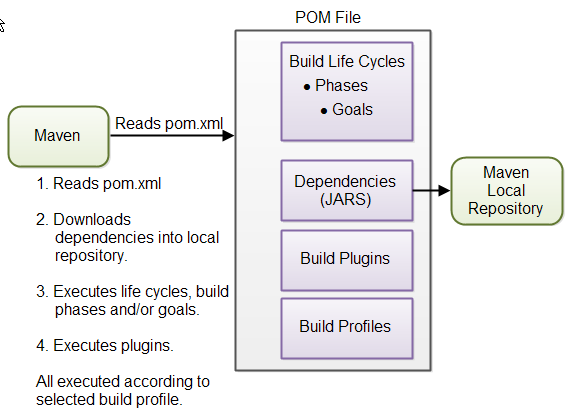
## Steps/process involved in building the project:

* Add / Write the code for application creation and process that into source code repository
* Edit configuration / pom.XML / plugin details
* Build the application
* Save the build process output as WAR or EAR file to a local location or server
* Get the file from local location or server and deploy the file to the production site or
* client site Updated the application document with date and updated version number of the application
* create and generate a report as per the application or requirement.

**POM**

Maven is centered around the concept of POM files (Project Object Model). A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc. The POM contains references to all of these resources. The POM file should be located in the root directory of the project it belongs to.

Here is a diagram illustrating how Maven uses the POM file, and what the POM file primarily contain.



**When you execute a Maven command you give Maven a POM file to execute the commands on. Maven will then execute the command on the resources described in the POM.**

**Build Life Cycles, Phases and Goals**  
The build process in Maven is split up into build life cycles, phases and goals. A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. When you run Maven you pass a command to Maven. This command is the name of a build life cycle, phase or goal. If a life cycle is requested executed, all build phases in that life cycle are executed. If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too.

**Dependencies and Repositories**  
One of the first goals Maven executes is to check the dependencies needed by your project. Dependencies are external JAR files (Java libraries) that your project uses. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository. The local repository is just a directory on your computer's hard disk. You can specify where the local repository should be located if you want to (I do). You can also specify which remote repository to use for downloading dependencies. All this will be explained in more detail later in this tutorial.

**Build Plugins**  
Build plugins are used to insert extra goals into a build phase. If you need to perform a set of actions for your project which are not covered by the standard Maven build phases and goals, you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java if you need to.

**Build Profiles**  
Build profiles are used if you need to build your project in different ways. For instance, you may need to build your project for your local computer, for development and test. And you may need to build it for deployment on your production environment. These two builds may be different. To enable different builds you can add different build profiles to your POM files. When executing Maven you can tell which build profile to use.

**Elements of maven pom.xml file**

For creating the simple pom.xml file, you need to have following elements:

**project** -- It is the root element of pom.xml file.

**modelVersion** -- It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0.

**groupId** -- It is the sub element of project. It specifies the id for the project group.

**artifactId** -- It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs.

**version** -- It is the sub element of project. It specifies the version of the artifact under given group.

**Packaging** -- defines packaging type such as jar, war etc.

**Name** -- defines name of the maven project.

**url** -- defines url of the project.

**dependencies** -- defines dependencies for this project.

**Dependency** -- defines a dependency. It is used inside dependencies.

**Scope** -- defines scope for this maven project. It can be compile, provided, runtime, test and system.

## Super POM

The Super POM is Maven’s default POM. All POMs inherit from a parent or default (despite explicitly defined or not). This base POM is known as the **Super POM**, and contains values inherited by default.

Maven use the effective POM (configuration from super pom plus project configuration) to execute relevant goal. It helps developers to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: **mvn help:effective-pom**

**POM Examples:-**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
7. **<groupId>**com.javatpoint.application1**</groupId>**
8. **<artifactId>**my-app**</artifactId>**
9. **<version>**1**</version>**
10. **</project>**

<project xmlns = "http://maven.apache.org/POM/4.0.0"

xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.project-group</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

</project>

<dependency>

<groupId>com.jenkov</groupId>

<artifactId>java-web-crawler</artifactId>

**<version>1.0-SNAPSHOT</version>**

</dependency>

#### Default POM Configuration

It depends on the type of archtype, you have selected. e.g. for a quickstart project, this is default generated pom.xml file.

|  |
| --- |
| <project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"    xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd;>    <modelVersion>4.0.0</modelVersion>      <groupId>com.howtodoinjava.demo</groupId>    <artifactId>MavenExamples</artifactId>    <version>0.0.1-SNAPSHOT</version>    <packaging>jar</packaging>      <name>MavenExamples</name>    <url>[http://maven.apache.org](http://maven.apache.org/)</url>      <properties>      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>    </properties>      <dependencies>      <dependency>        <groupId>junit</groupId>        <artifactId>junit</artifactId>        <version>3.8.1</version>        <scope>test</scope>      </dependency>    </dependencies>  </project> |



**Build Life Cycles**  
Maven has 3 built-in build life cycles. These are:

1. default
2. clean
3. site

A typical **Maven Build Lifecycle** consists of the following sequence of phases.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Handles** | **Description** |
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| validate | Validating the information | Validates if the project is correct and if all necessary information is available. |
| compile | compilation | Source code compilation is done in this phase. |
| Test | Testing | Tests the compiled source code suitable for testing framework. |
| package | packaging | This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml. |
| install | installation | This phase installs the package in local/remote maven repository. |
| Deploy | Deploying | Copies the final package to the remote repository. |

There are always **pre** and **post** phases to register **goals**, which must run prior to, or after a particular phase.

## Clean Lifecycle

When we execute *mvn post-clean* command, Maven invokes the clean lifecycle consisting of the following phases.

* pre-clean
* clean
* post-clean

Maven Site plugin is generally used to create fresh documentation to create reports, deploy site, etc. It has the following phases −

* pre-site
* site
* post-site
* site-deploy
* $ mvn clean
* **validate project**: validate the project is correct and all necessary information is available:
* $ mvn validate
* **compile project**: compile source code, classes stored in target/classes:
* $ mvn compile
* **test project**: run tests using a suitable unit testing framework:
* $ mvn test
* **package project**: take the compiled code and package it in its distributable format, such as a JAR / WAR:
* $ mvn package
* **verify project**: run any checks to verify the package is valid and meets quality criteria:
* $ mvn verify
* **install project**: install the package into the local repository, for use as a dependency in other projects locally:
* $ mvn install
* **deploy project**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects:
* $ mvn deploy
* **deploy-file**: can be used for deploying a external jar file to repository:

$ mvn deploy:deploy-file -Dfile=/path/to/jar/file -DrepositoryId=repos-server -Durl=http://repos.company.org/test -DgroupId=javax -DartifactId=mail -Dpackaging=jar

**JENKINS**

## What is Continuous Integration?

In Continuous Integration after a code commit, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each commit code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment.

A Continuous Integration Pipeline is a powerful instrument that consists of a set of tools designed to **host**, **monitor**, **compile** and **test** code, or code changes.

## What is Jenkins?

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

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

* Jenkins is used to integrate all DevOps stages with the help of plugins.
* Commonly used Jenkins plugins are Git, Amazon EC2, Maven 2 project, HTML publisher etc.
* Jenkins has well over 1000 plugins and 147,000 active installations along with over 1 million users around the world.
* With Continuous Integration every change made in the source code is built. It performs other functions as well, that depends on the tool used for Continuous Integration.
* Nokia shifted from Nightly build to Continuous Integration.
* Process before Continuous Integration had many flaws. As a result, not only the software delivery was slow but the quality of software was also not up to the mark. Developers also had a tough time in locating and fixing bugs.
* Continuous Integration with Jenkins overcame these shortcomings by continuously triggering a build and test for every change made in the source code.

**For example:** If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

|  |  |
| --- | --- |
| **Before Jenkins** | **After Jenkins** |
| The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time consuming, which in turn slows the software delivery process. | Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases. |
| Developers have to wait for test results | Developers know the test result of every commit made in the source code on the run. |
| The whole process is manual | You only need to commit changes to the source code and Jenkins will automate the rest of the process for you. |

**Advantages of Jenkins**

* It is an open source tool.
* It is free of cost.
* It does not require additional installations or components. Means it is easy to install.
* Easily configurable.
* It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
* It is built in java and hence it is portable.
* It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
* Easy support, since it open source and widely used.
* Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

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:

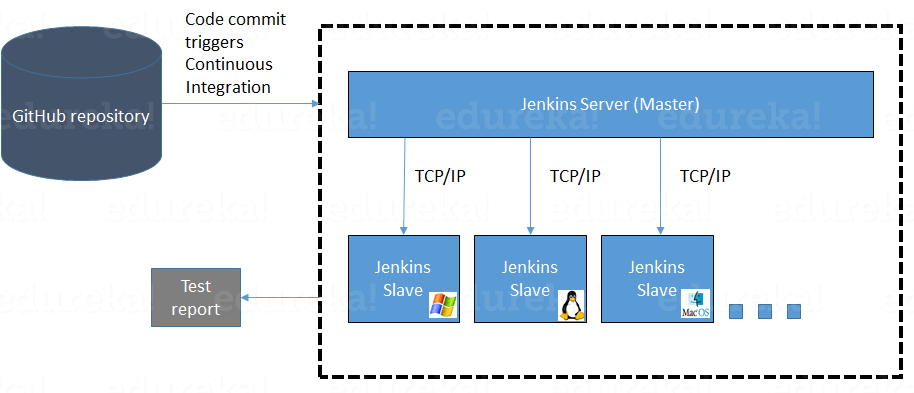
* Scheduling build jobs.
* Dispatching builds to the slaves for the actual execution.
* Monitor the slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master instance of Jenkins can also execute build jobs directly.

## 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.

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



The following functions are performed in the above image:

* Jenkins checks the Git repository at periodic intervals for any changes made in the source code.
* Each builds requires a different testing environment which is not possible for a single Jenkins server. In order to perform testing in different environments Jenkins uses various Slaves as shown in the diagram.
* Jenkins Master requests these Slaves to perform testing and to generate test reports.

Jenkins – Installation:

### Linux

#### Debian/Ubuntu

On Debian-based distributions, such as Ubuntu, you can install Jenkins through apt.

Recent versions are available in [an apt repository](https://pkg.jenkins.io/debian/). Older but stable LTS versions are in [this apt repository](https://pkg.jenkins.io/debian-stable/).

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

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

sudo apt-get update

sudo apt-get install jenkins

Next, start the Jenkins service.

sudo service jenkins start

sudo chkconfig jenkins on

You can check the status of the Jenkins service using this systemctl command:

systemctl status Jenkins / ps –ef | grep jenkins

bitnami@ubuntu:~/tools/jenkins$ sudo ufw allow 8080

Rule added

Rule added (v6)

bitnami@ubuntu:~/tools/jenkins$ ps -ef | grep jenkins

bitnami 2802 2692 0 12:44 pts/1 00:00:00 grep --color=auto jenkins

bitnami@ubuntu:~/tools/jenkins$ sudo ufw status

Status: active

To Action From

-- ------ ----

2181/tcp ALLOW Anywhere

22/tcp ALLOW Anywhere

9092/tcp ALLOW Anywhere

22 ALLOW Anywhere

9000 ALLOW Anywhere

8080 ALLOW Anywhere

2181/tcp (v6) ALLOW Anywhere (v6)

22/tcp (v6) ALLOW Anywhere (v6)

9092/tcp (v6) ALLOW Anywhere (v6)

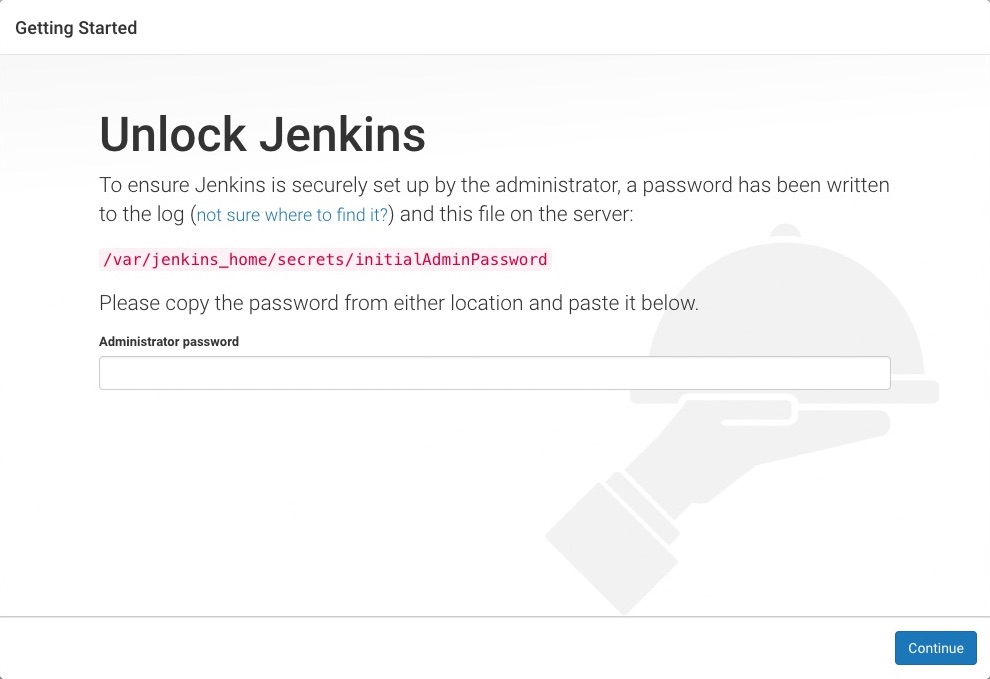
22 (v6) ALLOW Anywhere (v6)

9000 (v6) ALLOW Anywhere (v6)

8080 (v6) ALLOW Anywhere (v6)

Once Jenkins is up and running, one can access Jenkins from the link − [**http://localhost:8080**](http://localhost:8080)

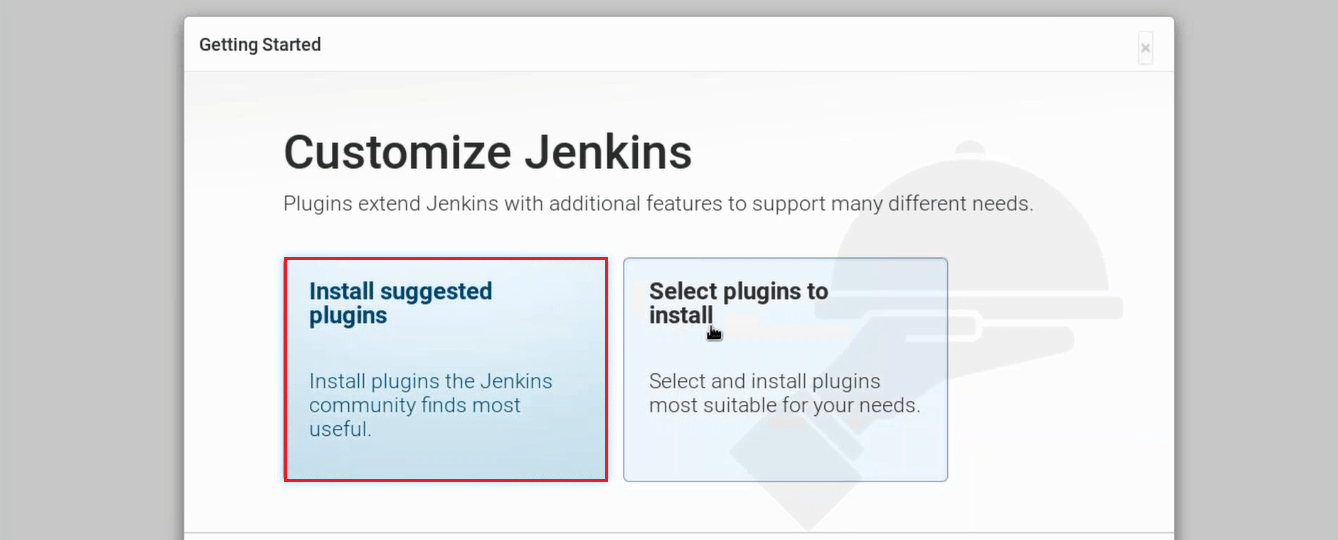
**and** wait until the **Unlock Jenkins** page appears.



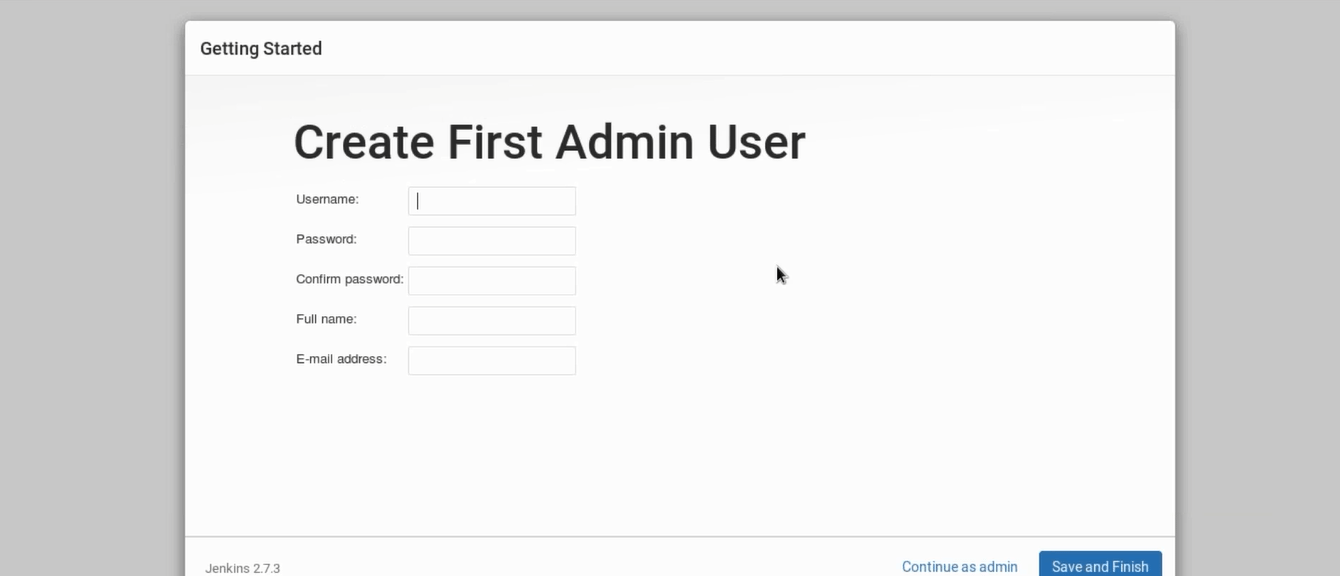
In the terminal, type the following cat command to see the password:

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

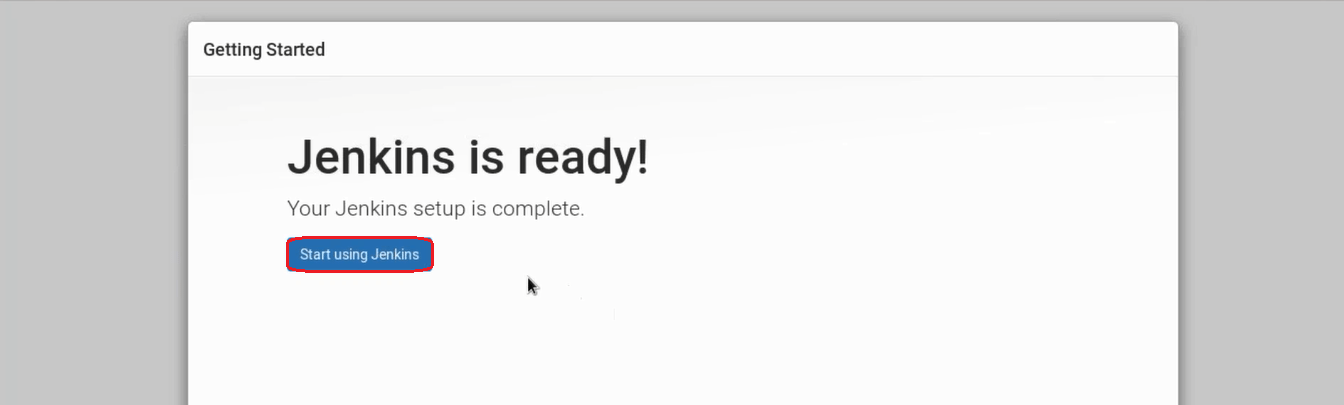
Copy the password from the terminal and paste it into the "Administrator password" field and then click continue. Now, the screen presents the option of installing suggested plugins or selecting specific plugins:



Once the plugins are installed, you will be directed to a page where you have to Create First Admin User. Please fill your relevant details.



After filling this form, click on save and finish.



**Freestyle Project:**

Freestyle build jobs are general-purpose build jobs, which provides maximum flexibility. The freestyle build job is the most flexible and configurable option, and can be used for any type of project. It is relatively straightforward to set up, and many of the options we configure here also appear in other build jobs.

**Multiconfiguration Job:**

The “multiconfiguration project” (also referred to as a “matrix project”) allows you run the same build job on different environments. It is used for testing an application in different environments, with different databases, or even on different build machines.

**Monitor an External Job:**

The “Monitor an external job” build job lets you keep an eye on non-interactive processes, such as cron jobs.

**Maven Project:**

The “maven2/3 project” is a build job specially adapted to Maven projects. Jenkins understands Maven pom files and project structures, and can use the information gleaned from the pom file to reduce the work you need to do to set up your project.

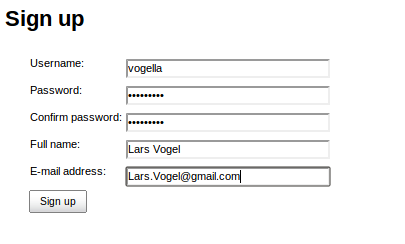
**Multibranch Pipeline Project:**

The [Multibranch Pipeline](https://dzone.com/refcardz/declarative-pipeline-with-jenkins) allows you to automatically create a pipeline for each branch on your Source Code Management (SCM) repository with the help of Jenkinsfile.

### [Assign roles to users](https://www.vogella.com/tutorials/Jenkins/article.html#assign-roles-to-users)

If you want to create a role based authorization Strategy you first need to install the Role-based Authorization Strategy Plugin. Go to **Manage Jenkins**  **Manage Plugins**  **Available** enter Role-based Authorization Strategy in the filter box and select and install the Plugin. To see a list of commonly used Plugins go to [Plugin management](https://www.vogella.com/tutorials/Jenkins/article.html#jenkins_pluginmanagement).

Now go to **Manage Jenkins**  **Manage and Assign Roles**  **Assign Roles** to grant users additional access rights.



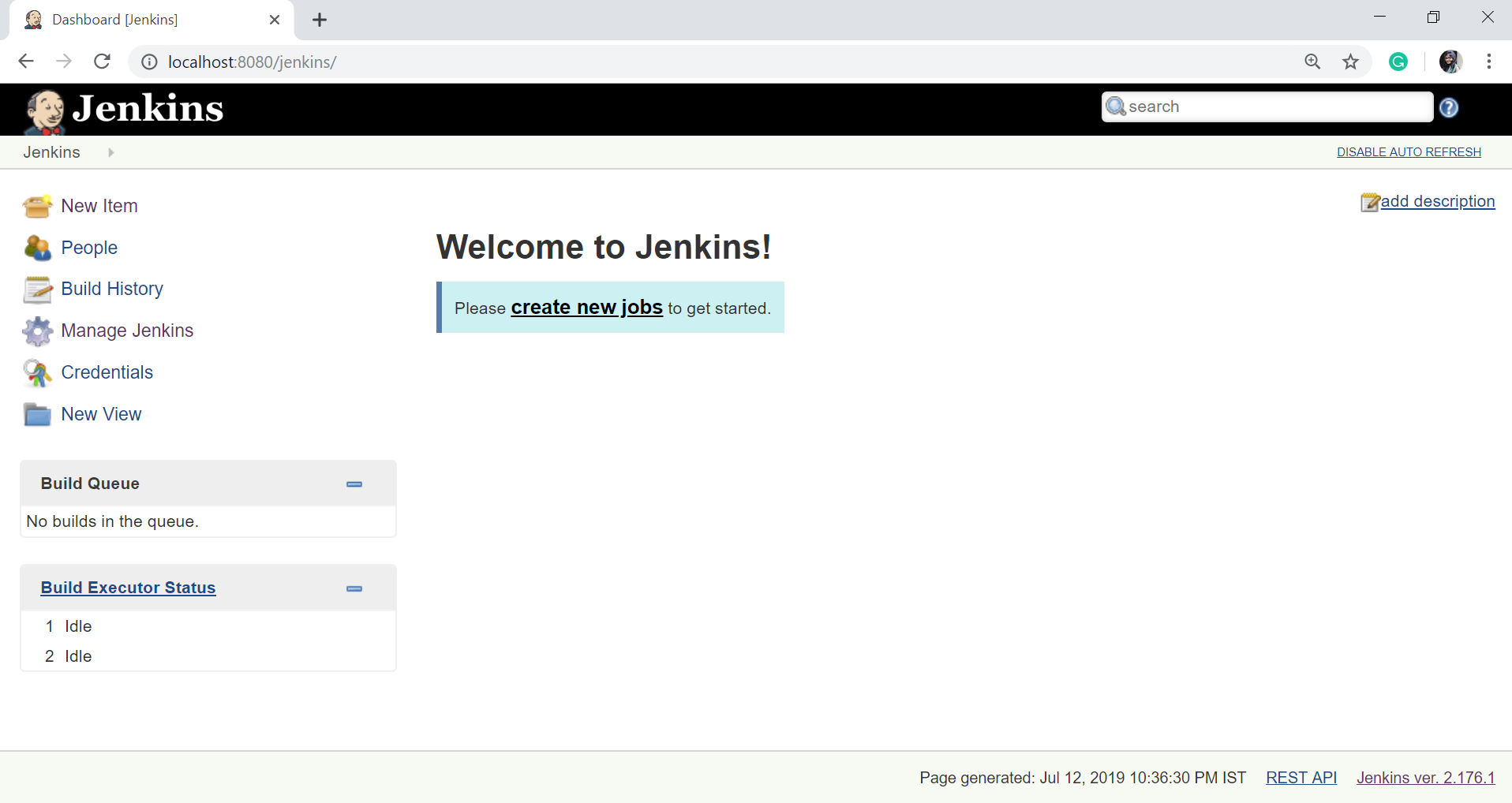
Navigate to Manage Roles to define access restrictions in detail. Pattern is a regex value of the job name. The following grants unregistered users read-only access to your build jobs that start with the L-, C-, I- or M- and only those.



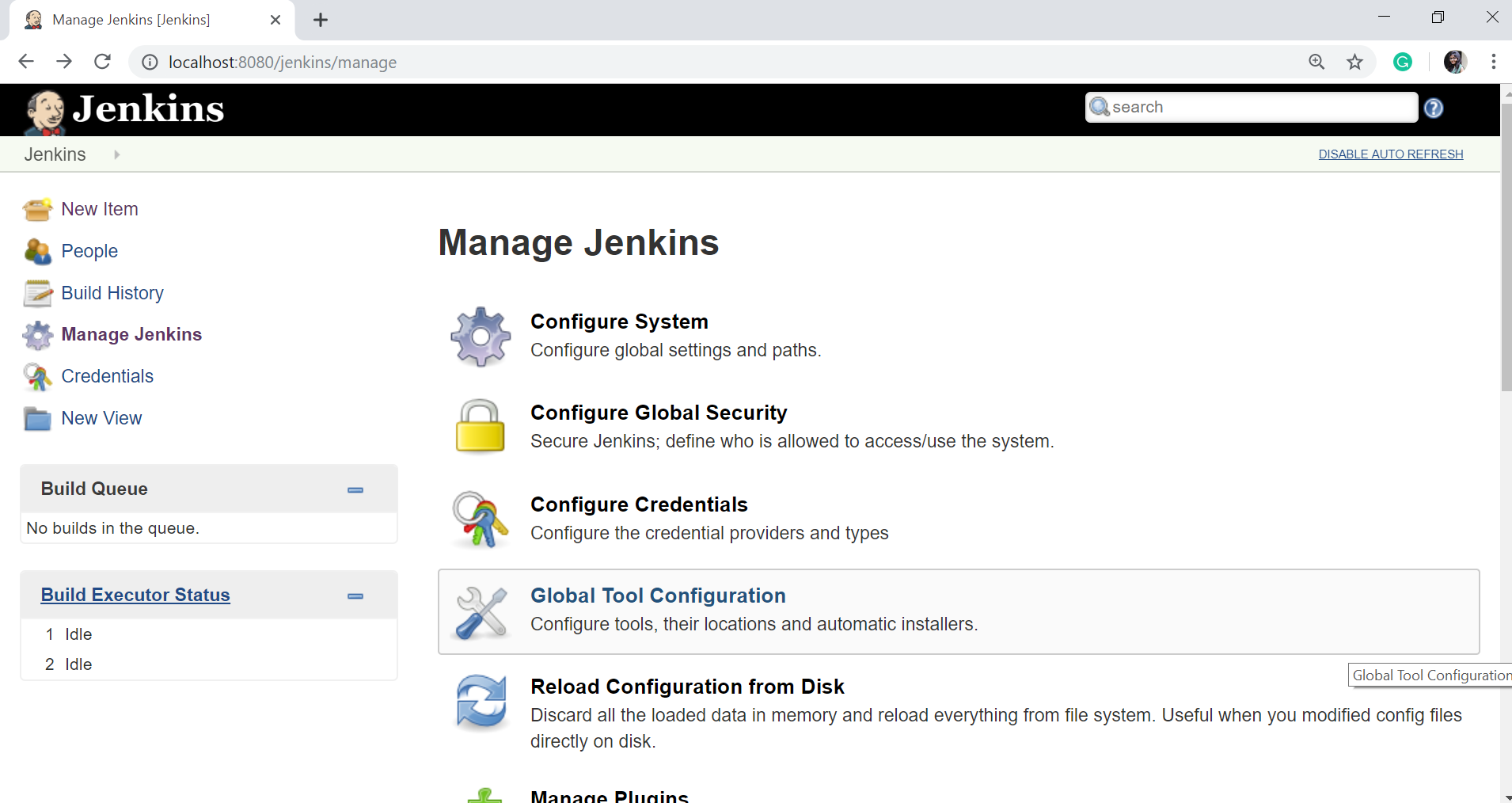
The default port number can be changed in the config file at

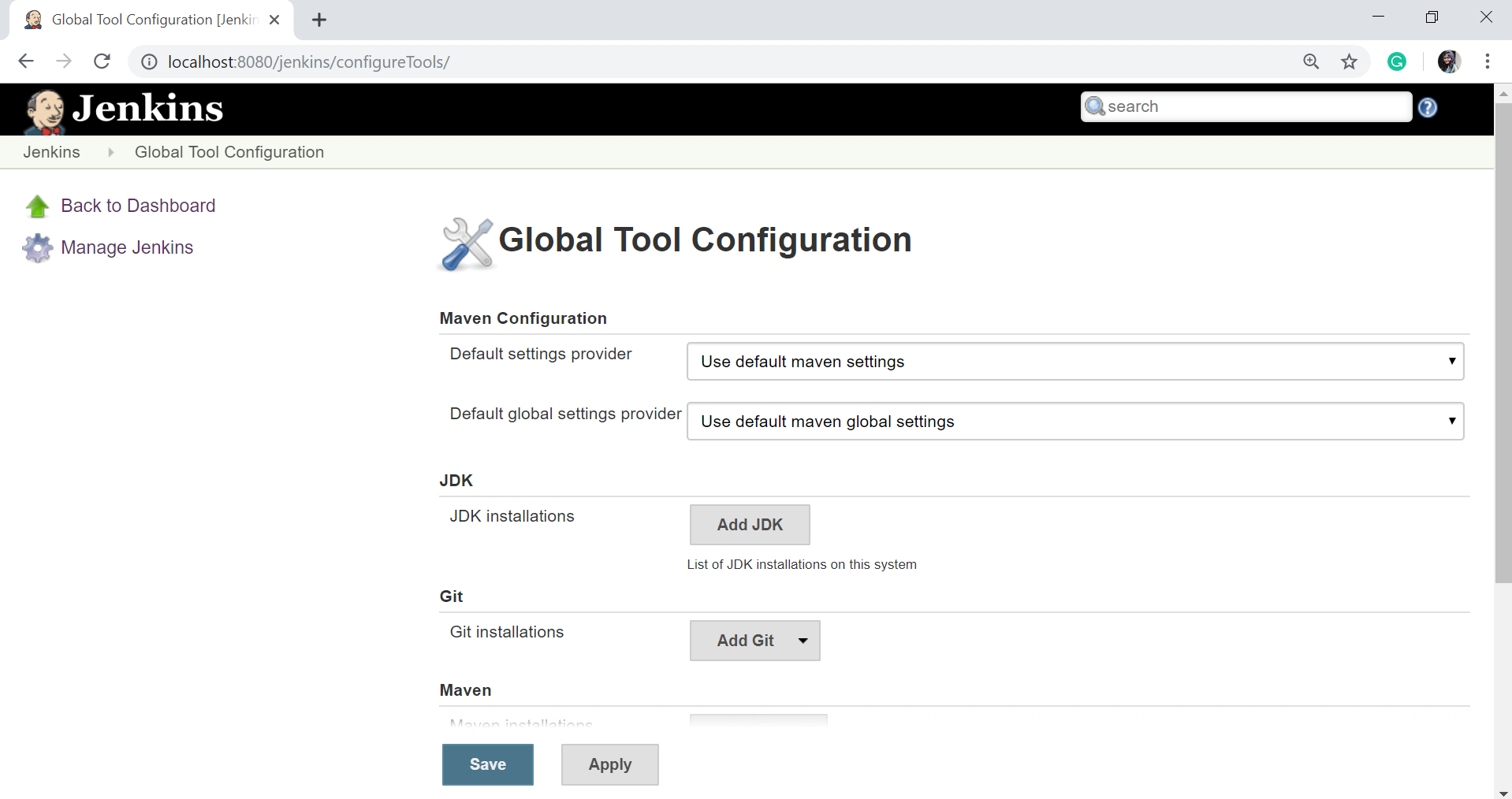
sudo vim /etc/default/jenkins

## Setting Up Java and Maven in Jenkins:

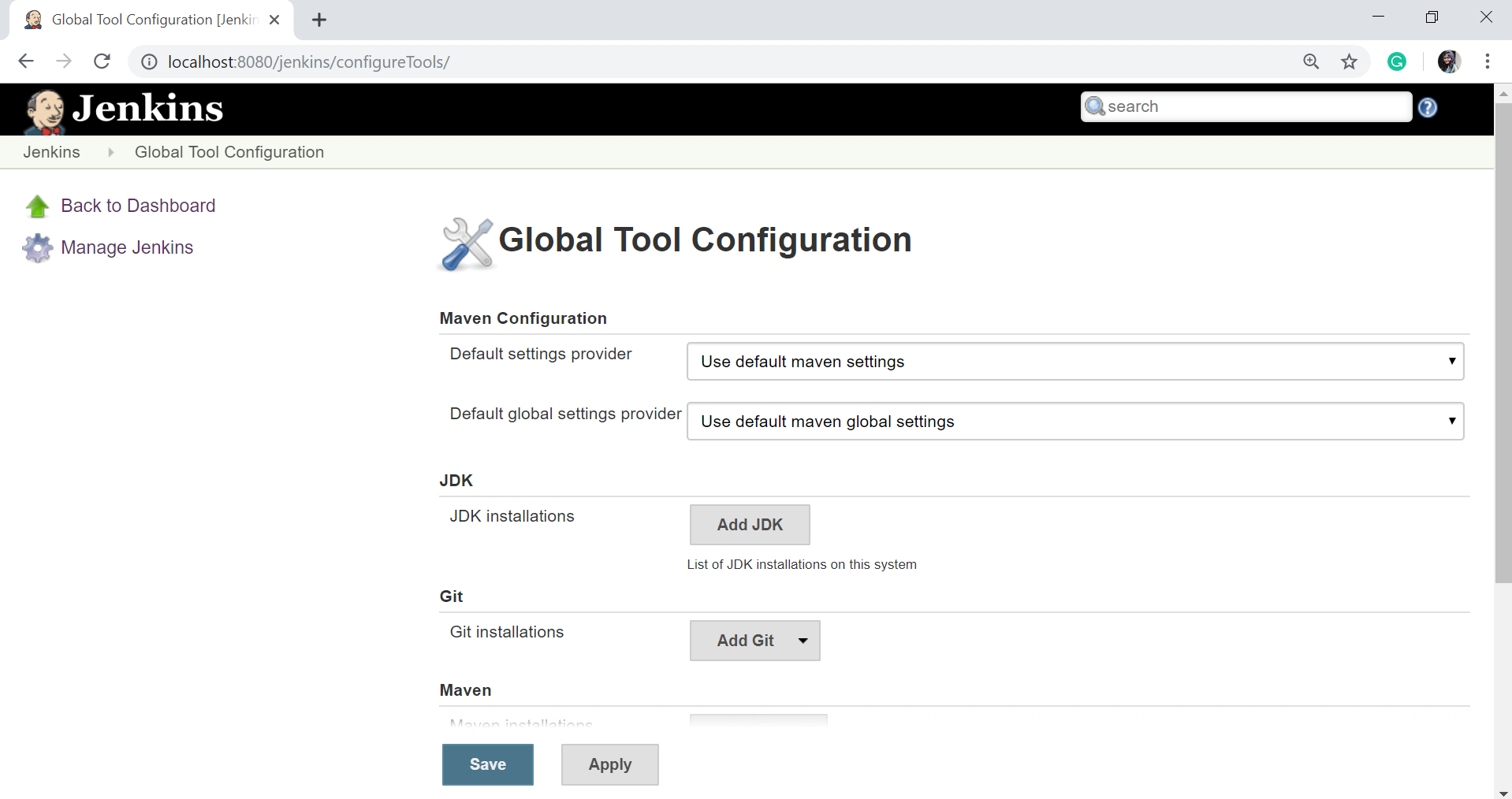


Click on "Global Tool Configuration" option.

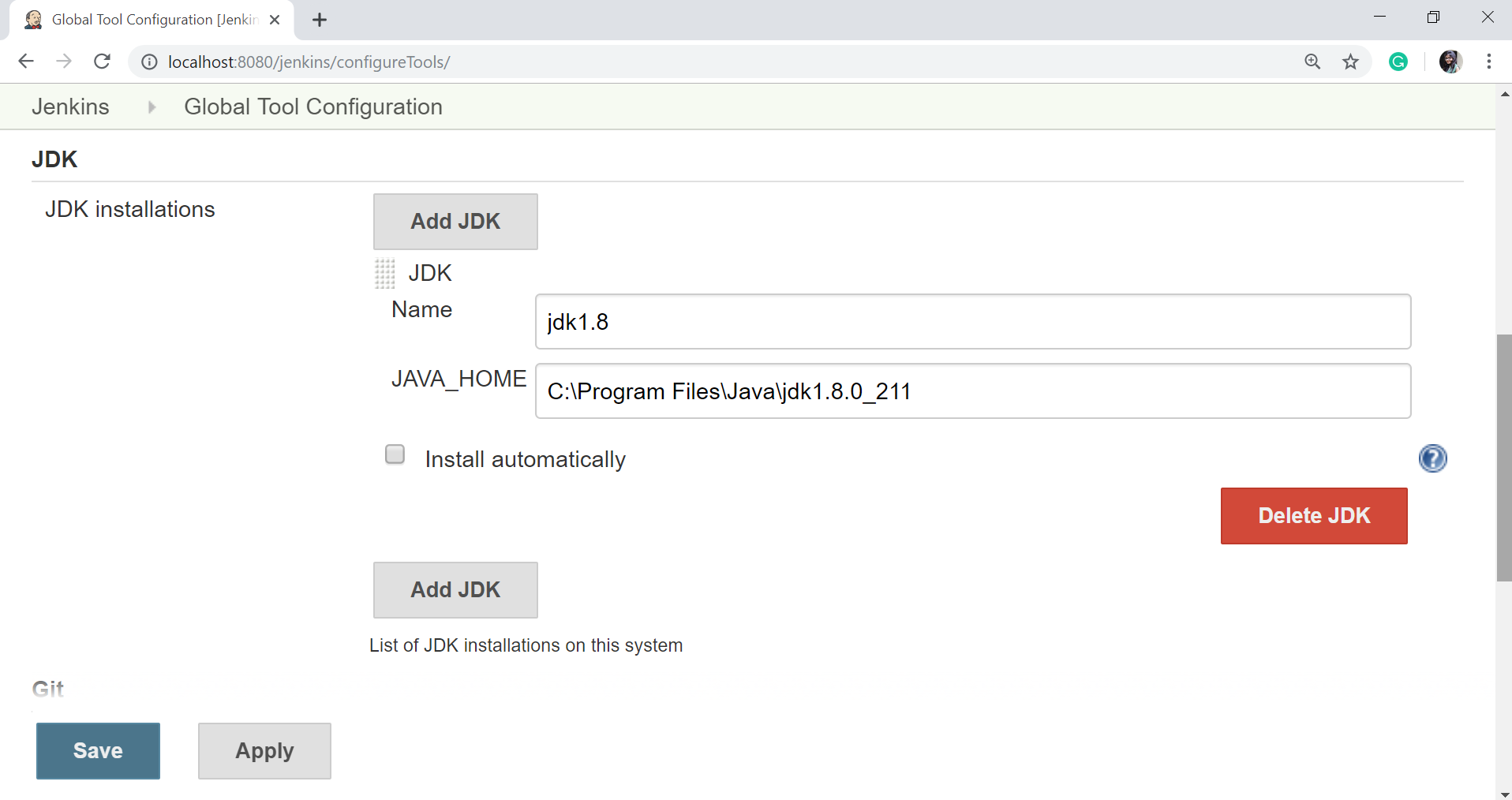




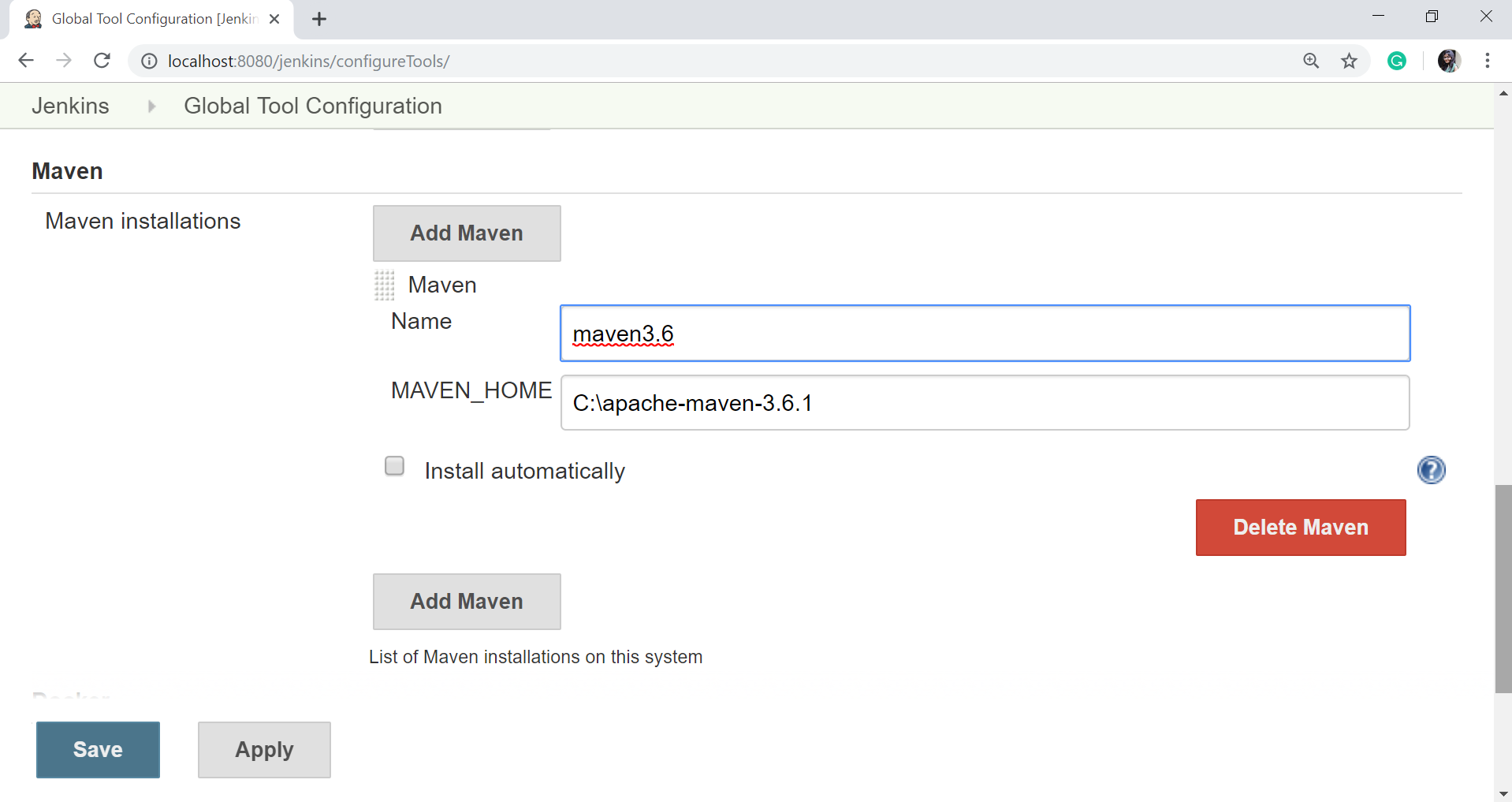
To configure Java, click on "**Add JDK**" button in the JDK section.



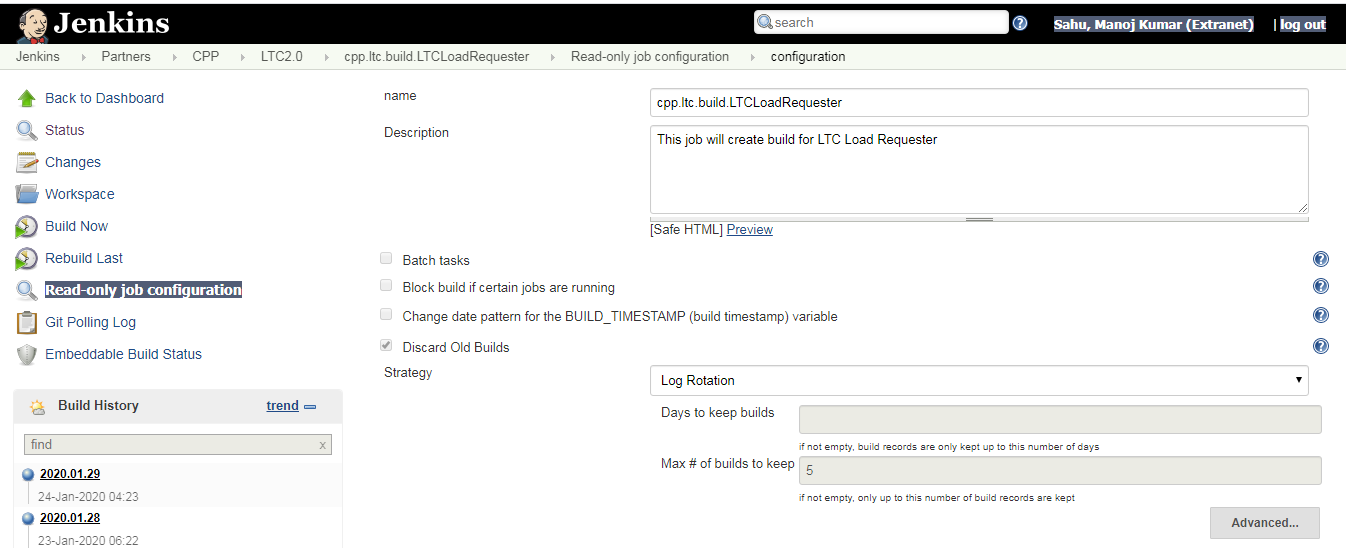
Give a **Name** and **JAVA\_HOME** path, or check on **install automatically** checkbox.



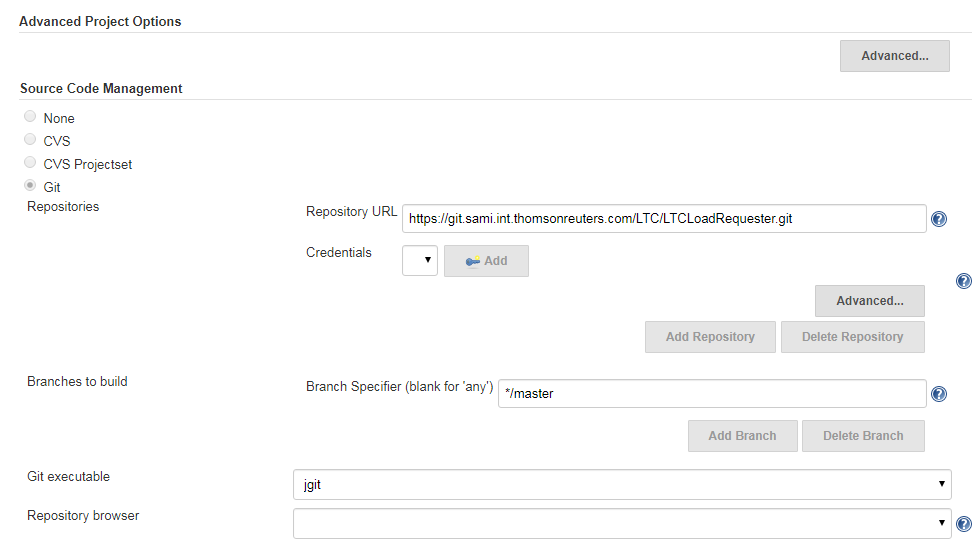
To configure Maven, click on "Add Maven" button in the Maven section, give any **Name** and **MAVEN\_HOME** path or check to install automatically checkbox.

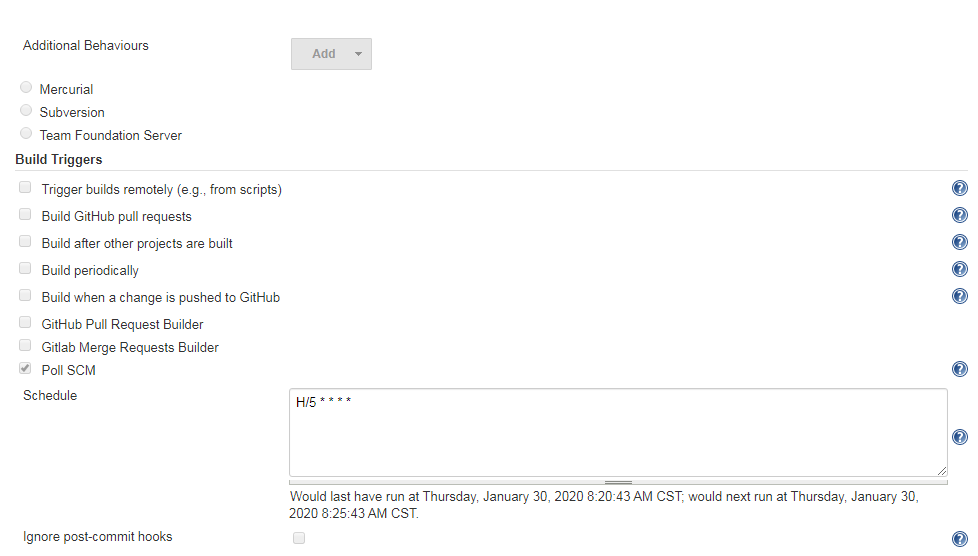


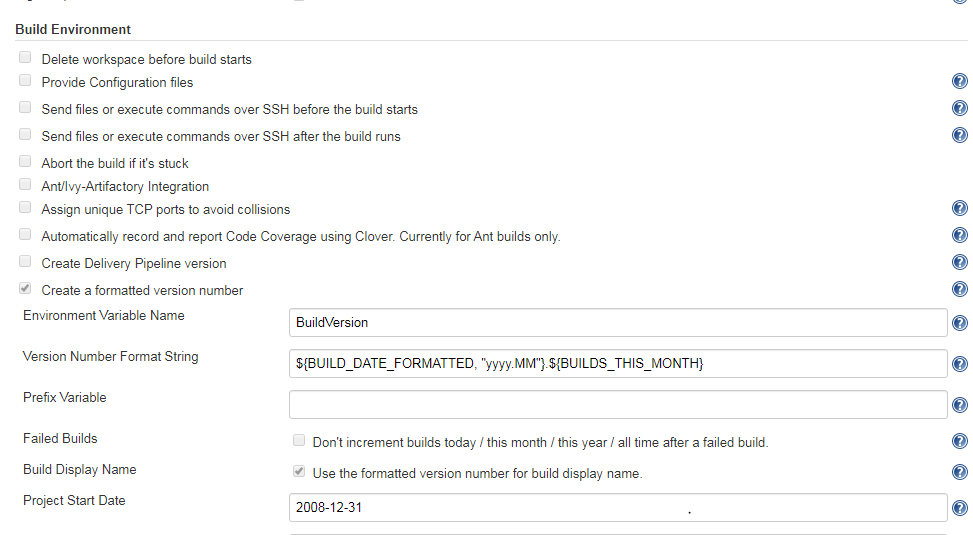
**CPP LTC Build Job :**

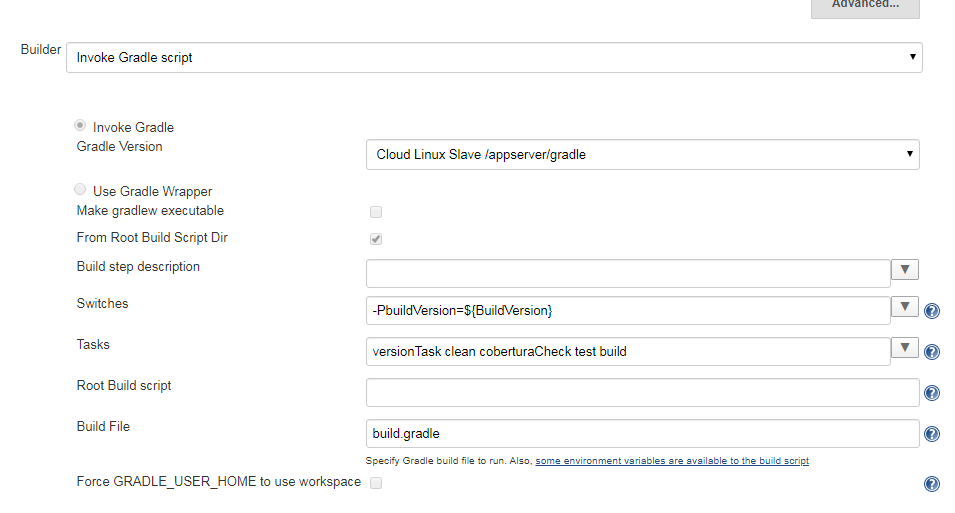
****

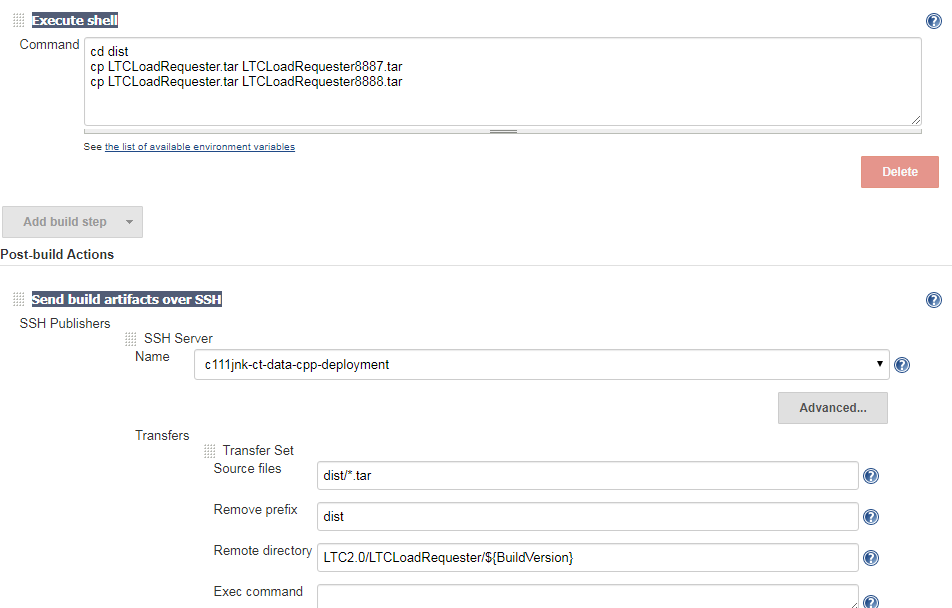
****

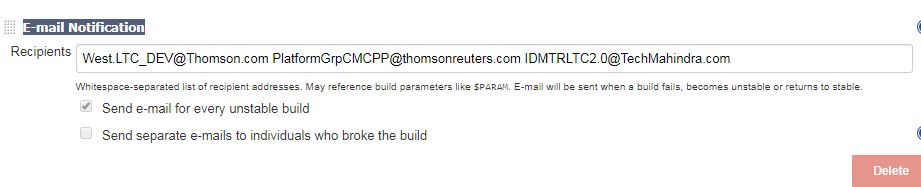
****











Pipeline Job :

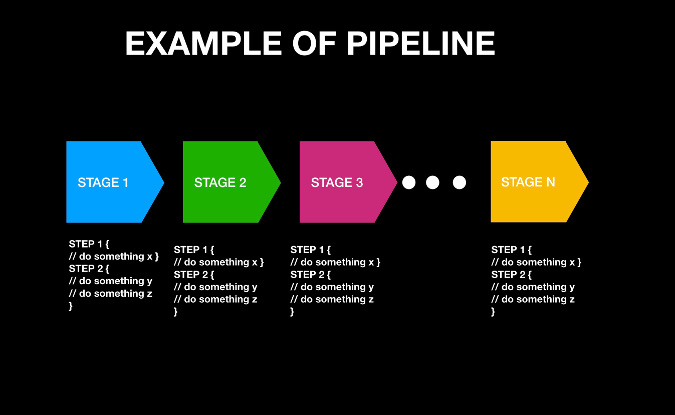
In contrast to freestyle jobs, pipelines enable you to define the whole application lifecycle. Pipeline functionality helps Jenkins to support continuous delivery (CD). The Pipeline plugin was built with requirements for a flexible, extensible, and script-based CD workflow capability in mind.

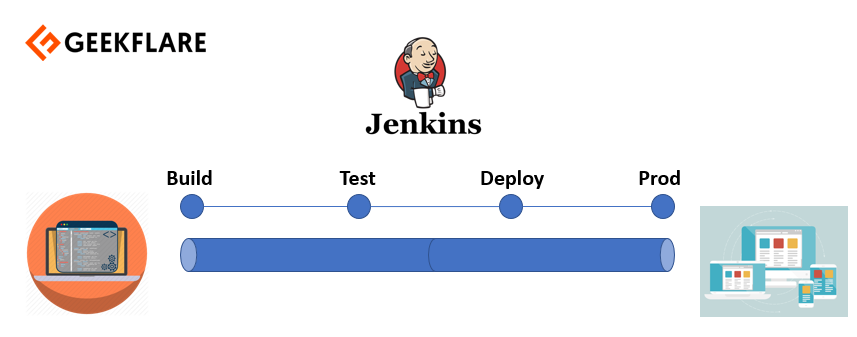
In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

It is a combination of plugins that support the integration and implementation of **continuous delivery pipelines** using Jenkins.

In other words, a Jenkins Pipeline is a collection of jobs or events that brings the software from version control into the hands of the end users by using automation tools. It is used to incorporate continuous delivery in our software development workflow.

A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).





## ****What is a Jenkinsfile?****

Jenkins pipelines can be defined using a text file called **JenkinsFile**

A Jenkinsfile is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system. How is this advantageous? This enables the developers to **access, edit and check the code at all times**.

The benefits of using J**enkinsFile are**:

* You can create pipelines automatically for all branches and execute pull requests with just one **JenkinsFile.**
* You can review your code on the pipeline
* You can audit your Jenkins pipeline
* This is the singular source for your pipeline and can be modified by multiple users.

## Pipeline syntax

Two types of syntax are used for defining your JenkinsFile.

* Declarative
* Scripted

**Declarative:**

Declarative pipeline syntax offers a simple way to create pipelines. It consists of a predefined hierarchy to create Jenkins pipelines. It provides you the ability to control all aspects of a pipeline execution in a simple, straightforward manner.

**Scripted:**

Scripted Jenkins pipeline syntax runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to convert the pipeline into atomic commands.

Both scripted and declarative syntax are different from each other and are defined totally differently.

## ****Pipeline concepts****

### ****Pipeline****

This is a user defined block which contains all the processes such as build, test, deploy, etc. It is a collection of all the stages in a Jenkinsfile. All the stages and steps are defined within this block. It is the key block for a declarative pipeline syntax.



### ****Node****

A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.



There are various mandatory sections which are common to both the declarative and scripted pipelines, such as stages, agent and steps that must be defined within the pipeline. These are explained below:

### ****Agent****

An agent is a directive that can run multiple builds with only one instance of Jenkins. This feature helps to distribute the workload to different agents and execute several projects within a single Jenkins instance. It instructs Jenkins to **allocate an executor** for the builds.

A single agent can be specified for an entire pipeline or specific agents can be allotted to execute each stage within a pipeline. Few of the parameters used with agents are:

##### **Any**

Runs the pipeline/ stage on any available agent.

##### **None**

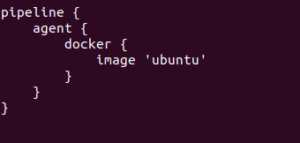
This parameter is applied at the root of the pipeline and it indicates that there is no global agent for the entire pipeline and each stage must specify its own agent.

##### **Label**

Executes the pipeline/stage on the labelled agent.

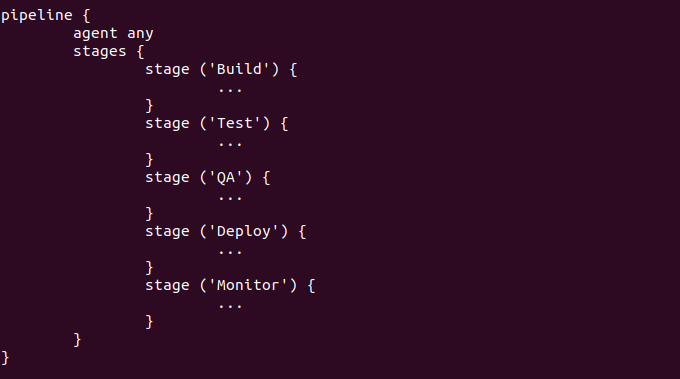
##### **Docker**

This parameter uses docker container as an execution environment for the pipeline or a specific stage. In the below example I’m using docker to pull an ubuntu image. This image can now be used as an execution environment to run multiple commands.



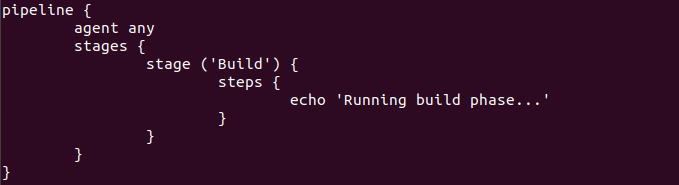
### ****Stages****

This block contains all the work that needs to be carried out. The work is specified in the form of stages. There can be more than one stage within this directive. Each stage performs a specific task. In the following example, I’ve created multiple stages, each performing a specific task.



### ****Steps****

A series of steps can be defined within a stage block. These steps are carried out in sequence to execute a stage. There must be at least one step within a steps directive. In the following example I’ve implemented an echo command within the build stage. This command is executed as a part of the ‘Build’ stage.



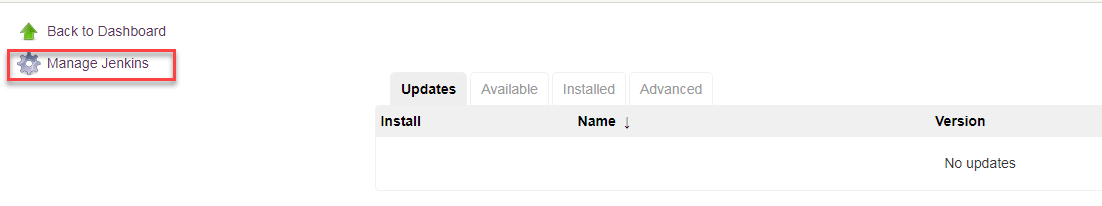
Now that you are familiar with the basic pipeline concepts let’s start of with the Jenkins pipeline.

## Install Build Pipeline Plugin in Jenkins

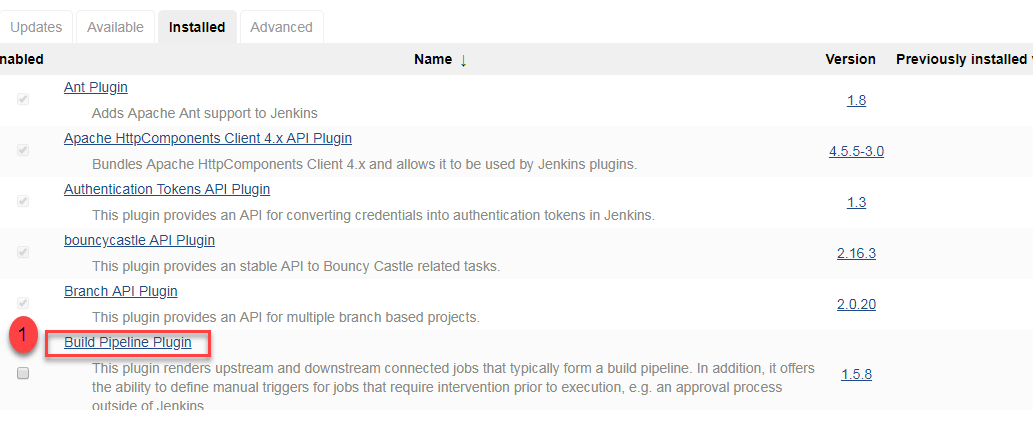
With the **build pipeline** plugin, you can create a pipeline view of incoming and outgoing jobs, and create triggers which require manual intervention.

Here is how you can install the **build pipeline**plugin in your Jenkins:

**Step 1**) The settings for the plugin can be found under **Manage Jenkins > Manage Plugins.**

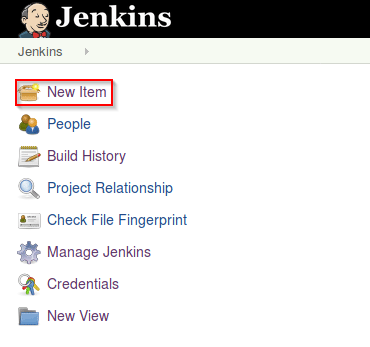


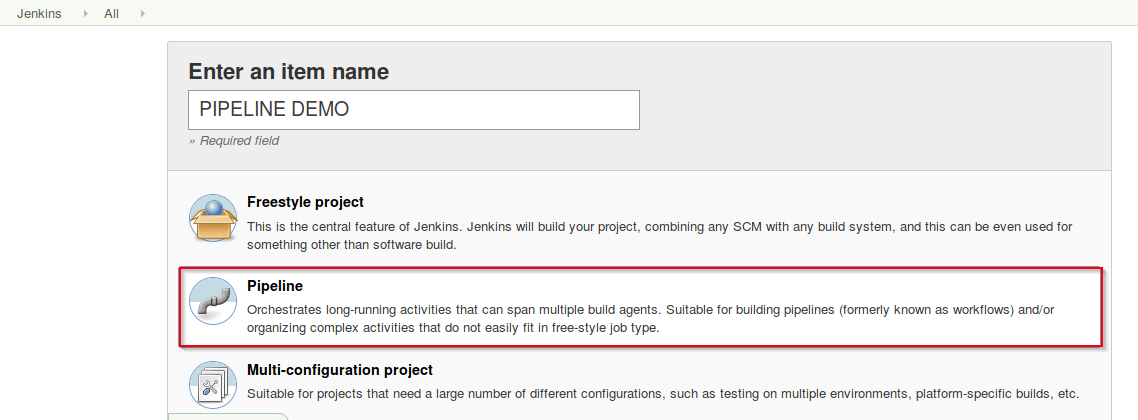
If you have already installed the plugin, it is shown under the installed tab.

**Step 2**) If you do not have the plugin previously installed, it shows up under the **Available**tab.

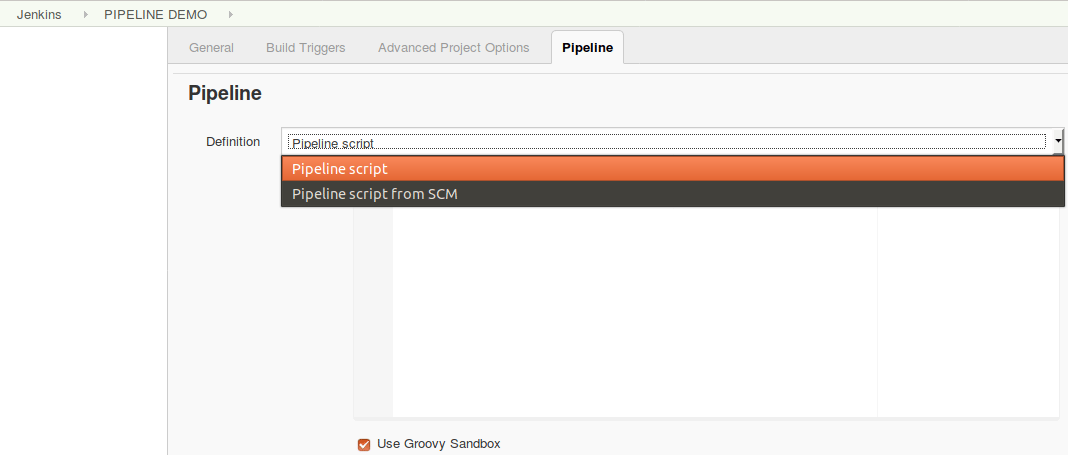
### ****Creating your first Jenkins pipeline.****

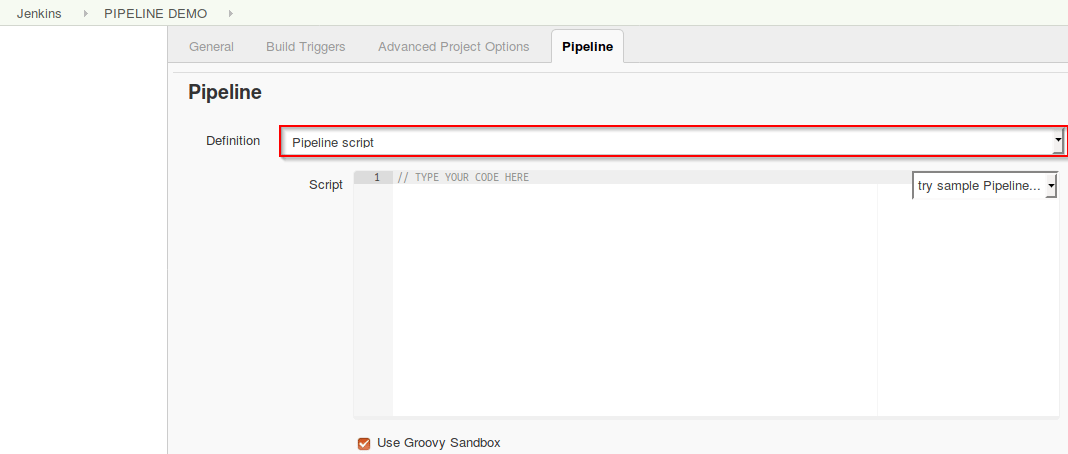
**Step 1**: Log into Jenkins and select ‘New item’ from the dashboard.



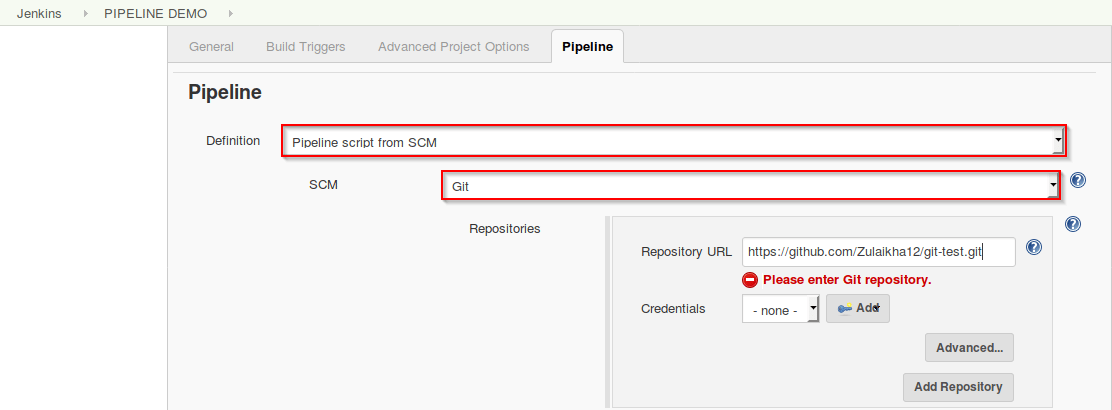
**Step 2**: Next, enter a name for your pipeline and select ‘pipeline’ project. Click on ‘ok’ to proceed.

**Step 3**: Scroll down to the pipeline and choose if you want a declarative pipeline or a scripted one.

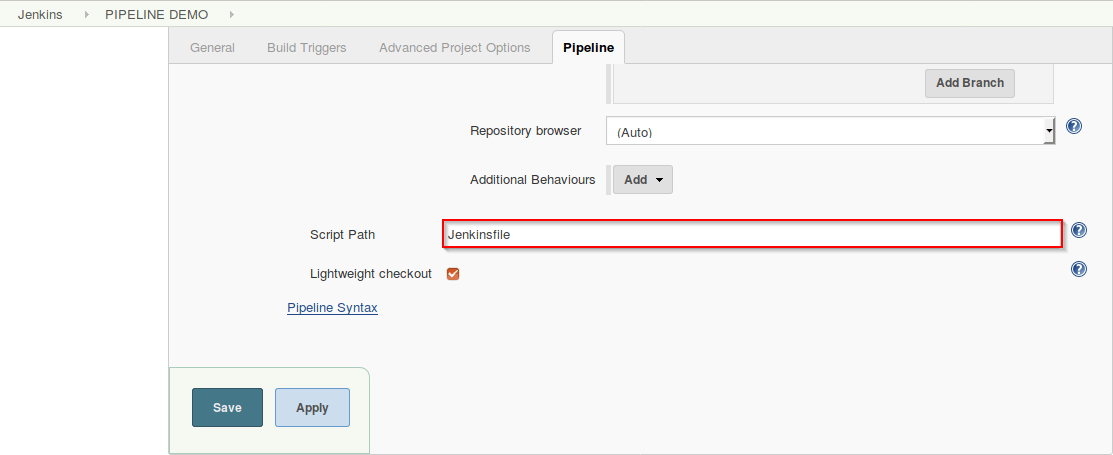


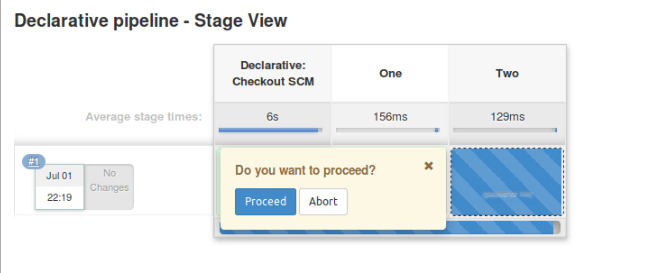
**Step 4a**: If you want a scripted pipeline then choose ‘pipeline script’ and start typing your code

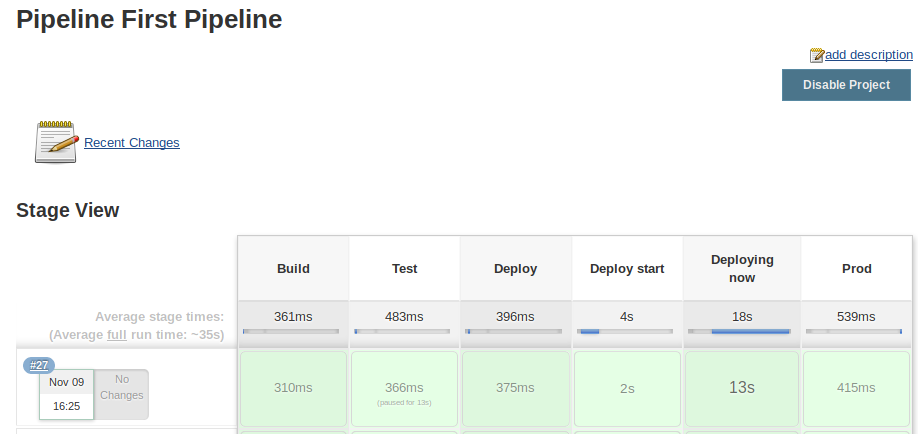
**Step 4b**: If you want a declarative pipeline then select ‘pipeline script from SCM’ and choose your SCM. In my case I’m going to use Git throughout this demo. Enter your repository URL.



**Step 5**: Within the script path is the name of the Jenkinsfile that is going to be accessed from your SCM to run. Finally click on ‘apply’ and ‘save’. You have successfully created your first Jenkins pipeline.







**Sample Jenkins File**:

|  |
| --- |
| pipeline { |
|  | agent any |
|  |  |
|  | stages { |
|  | stage ('Compile Stage') { |
|  |  |
|  | steps { |
|  | withMaven(maven : 'maven\_3\_5\_0') { |
|  | sh 'mvn clean compile' |
|  | } |
|  | } |
|  | } |
|  |  |
|  | stage ('Testing Stage') { |
|  |  |
|  | steps { |
|  | withMaven(maven : 'maven\_3\_5\_0') { |
|  | sh 'mvn test' |
|  | } |
|  | } |
|  | } |
|  |  |
|  |  |
|  | stage ('Deployment Stage') { |
|  | steps { |
|  | withMaven(maven : 'maven\_3\_5\_0') { |
|  | sh 'mvn deploy' |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |

### Java

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'maven:3.3.3' } }

stages {

stage('build') {

steps {

sh 'mvn --version'

}

}

}

}

### Node.js / JavaScript

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'node:6.3' } }

stages {

stage('build') {

steps {

sh 'npm --version'

}

}

}

}

### Ruby

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'ruby' } }

stages {

stage('build') {

steps {

sh 'ruby --version'

}

}

}

}

### Python

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'python:3.5.1' } }

stages {

stage('build') {

steps {

sh 'python --version'

}

}

}

}

### PHP

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'php' } }

stages {

stage('build') {

steps {

sh 'php --version'

}

}

}

}

### Go

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent { docker { image 'golang' } }

stages {

stage('build') {

steps {

sh 'go version'

}

}

}

}

**pipeline {  
  agent { label 'worker\_node1' }  
  stages {  
    stage('Source') { // Get code  
      steps {  
        // get code from our Git repository  
        git 'https://github.com/brentlaster/roarv2'  
      }  
    }  
    stage('Compile') { // Compile and do unit testing  
      tools {  
        gradle 'gradle4'  
      }  
      steps {  
        // run Gradle to execute compile and unit testing  
        sh 'gradle clean compileJava test'  
      }  
    }  
  }  
}**

<https://foxutech.com/how-to-write-a-jenkinsfile/>

<https://dzone.com/articles/implement-ci-for-multibranch-pipeline-in-jenkins>

<https://github.com/TechPrimers/jenkins-example>

**UNIX and Shell Scripting**

**SED Command in Unix:-**

Sed command or **Stream Editor** is very powerful utility offered by Linux/Unix systems. It is mainly used for **text substitution** , find & replace but it can also perform other text manipulations like **insertion**, **deletion**, **search** etc. With SED, we can edit complete files without actually having to open it. Sed also supports the use of regular expressions, which makes sed an even more powerful **test manipulation tool**.

* SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
* SED command in unix supports regular expression which allows it perform complex pattern matching.

sed -n 22,29p testfile.txt --> Print lines from 22 to 29

sed 22,29d testfile.txt --> Display all except some lines

sed -n '2~3p' file.txt --> Do display content of every 3rd line starting with line number 2 or any other line

sed Nd testfile.txt --> Deleting Nth line from a file

sed $d testfile.txt --> Deleting the last line of a file

sed '29,34d' testfile.txt --> Deleting a range of lines of a file ( delete lines 29 to 34)

sed '29,34!d' testfile.txt --> All the lines other 29-34 will be deleted from the file

sed G testfile.txt --> To add a blank line after every non-blank line, we will use option ‘G’

sed '/^$/d' /tmp/data.txt --> delete all empty lines from a file called /tmp/data.txt

sed 's/abc/def/g' myfile.txt --> Replace every instance of abc with def frm whole file

sed 's/abc/def/gi' myfile.txt --> Same with 'i' to ignore case

sed 's/danger/safety/' testfile.txt --> search for word ‘danger’ & replace it with ‘safety’ on every line for the first occurrence only

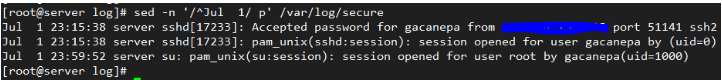
sed ‘s/danger/safety/2’ testfile.txt --> replace ‘danger’ with ‘safety’ only on second occurrence

sed 's/danger/safety/2g' testfile.txt --> replace ‘danger’ on 2nd occurrence of every line from whole file

sed '4 s/danger/safety/' testfile.txt --> substitute the string from 4th line of the file

sed '4-9 s/danger/safety/' testfile.txt --> substitute the string from 4th to 9th line of the file

sed -n '/^Jul 1/ p' /var/log/secure --> search for is Jul 2 at the beginning of each line in /var/log/secure



sed '/danger/a "This is new line with text after match"' testfile.txt --> To add a new line with some content after every pattern match, use option ‘a’

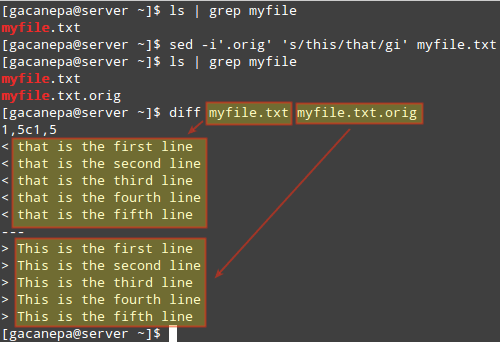
sed '/danger/i "This is new line with text before match" ' testfile.txt --> To add a new line with some content a before every pattern match, use option ‘i’

sed '/danger/c "This will be the new line" ' testfile.txt --> To change a whole line to a new line when a search pattern matches we need to use option ‘c’

**sed '/^#\|^$\| \*#/d' httpd.conf** --> To remove empty lines or those beginning with # from the Apache configuration file

The caret sign followed by the number sign (^#) indicates the beginning of a line, whereas ^$ represents blank lines. The vertical bars indicate boolean operations, whereas the backward slash is used to escape the vertical bars.In this particular case, the Apache configuration file has lines with #’s not at the beginning of some lines, so \*# is used to remove those as well.

**sed -i'.orig' 's/this/that/gi' myfile.txt** --> replace all instances of this or This (ignoring case) with that in myfile.txt, and we will save the original file as myfile.txt.orig



sed 's/^\(.\*\),\(.\*\)$/\, /g' names.txt --> Switching pairs of words



sed -e 's/.\*/testing sed &/' testfile.txt --> To add some content before every line

sed -e 's/#.\*//;/^$/d' testfile.txt --> To remove all commented lines i.e. lines with # & all the empty lines

sed 's/\([^:]\*\).\*/\1/' /etc/passwd --> To get the list of all usernames from /etc/passwd file

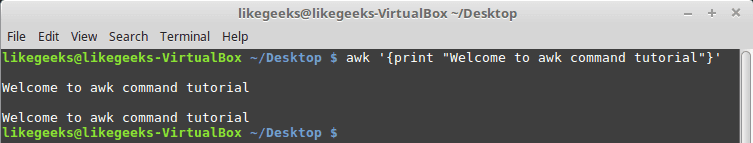
sed -e 's/danger/safety/g' -e 's/hate/love/' testfile.txt --> to perform multiple sed expressions, we can use option ‘e’ to chain the sed commands

**AWK Command in Unix:**

AWK  Stands for ‘**Aho, Weinberger**, and **Kernighan**‘

Awk is a **scripting language** which is used  for  processing or **analyzing text files**. Or we can say that awk is mainly used for grouping of data based on either a **column or field** , or on a **set of columns**. Mainly it’s used for reporting data in a usefull manner. It also employs Begin and End Blocks to process the data.

awk '{print "Welcome to awk command tutorial "}' --> To define an awk script, use braces surrounded by single quotation marks.



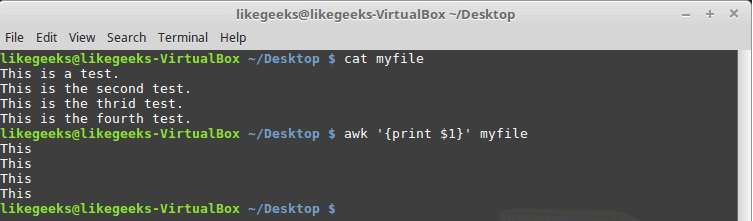
## Using Variables

With awk, you can process text files. Awk assigns some variables for each data field found:

* $0 for the whole line.
* $1 for the first field.
* $2 for the second field.
* $n for the nth field.

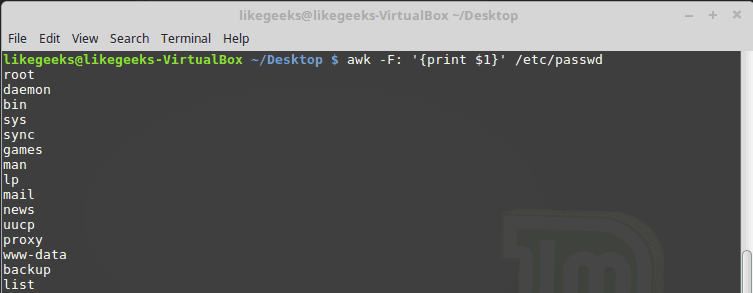
The whitespace character like space or tab is the default separator between fields in awk.

$ awk '{print $1}' myfile



Sometimes the separator in some files is not space nor tab but something else. You can specify it using –F option:

**$** **awk -F: '{print $1}' /etc/passwd**

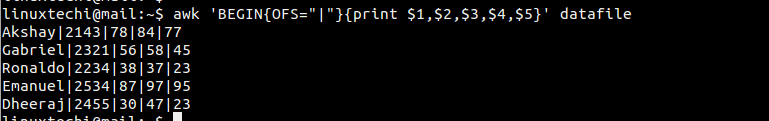


awk '{print}' employee.txt --> By default Awk prints every line of data from the specified file

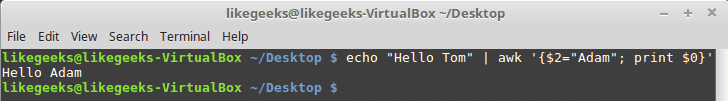
awk -F “,” ‘{print $2, $3;}’ file.txt --> Print only Specific field like 2nd & 3rd

awk ‘/Hari|Ram/’file.txt --> print the lines which contains the word “Hari & Ram”

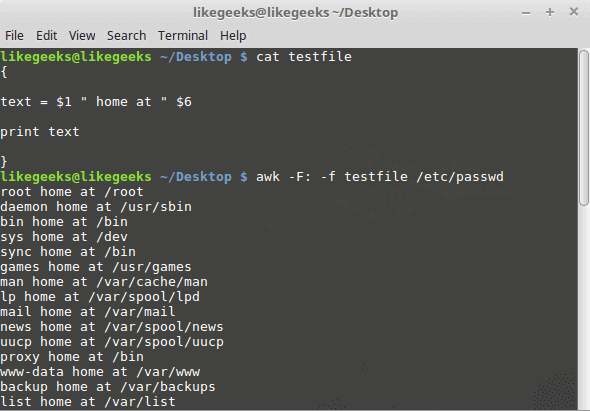
awk ‘BEGIN{OFS=”|”}{print $1,$2,$3,$4,$5}’ datafile.txt



$ echo "Hello Tom" | awk '{$2="Adam"; print $0}'



awk -F: -f testfile /etc/passwd



awk '{print $2 "\t" $3}' file.txt --> print the 2nd and 3rd columns

awk '/a/{++cnt} END {print "Count = ", cnt}' file.txt --> counts the number of instances a matching pattern appears

awk '/a/ {print $3 "\t" $4}' file.txt > Output.txt --> Saving output of AWK to a different file

awk '{print NR "- " $1 }' sample.txt --> print the Row Number (NR), then a dash and space ("- ") and then the first item ($1)

awk '{print NR,$0}' employee.txt --> Display Line Number for every line

awk '{print $1,$NF}' employee.txt --> Display first and last field

awk 'END { print NR }' sample.txt --> count the lines in a file

awk 'NR==3, NR==6 {print NR,$0}' employee.txt --> Display Line From 3 to 6

awk 'NF > 0' geeksforgeeks.txt --> To print any non empty line if present

awk 'length($0) > 10' file.txt --> Printing lines with more than 10 characters

**To print the squares of first numbers from 1 to n say 6:**

$ awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i\*i; }'

square of 1 is 1

square of 2 is 4

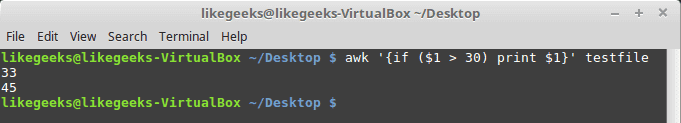
square of 3 is 9

square of 4 is 16

square of 5 is 25

square of 6 is 36

$ awk '{if ($1 > 30) print $1}' testfile



1. **[0-9]** means a single number
2. **[a-z]** means match a single lower case letter
3. **[A-Z]** means match a single upper case letter
4. **[a-zA-Z]** means match a single letter
5. **[a-zA-Z 0-9]** means match a single letter or number

<https://likegeeks.com/awk-command/>

### Cut Command in Unix :

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text.

cut -b 1,2,3 state.txt --> List without ranges

cut -b 1-3,5-7 state.txt --> List with ranges

cut -b 1- state.txt --> 1- indicate from 1st byte to end byte of a line

cut -b -3 state.txt --> -3 indicate from 1st byte to 3rd byte of a line

cut -c 2,5,7 state.txt --> prints second, fifth and seventh character from each line

cut -c 1-7 state.txt --> prints first seven characters of each line

**Sort Command in Unix :**

SORT command is used to sort a file, arranging the records in a particular order. It is important to notice that sort command don’t actually sort the files but only print the sorted output, until your redirect the output.

When we have a mix file with both uppercase and lowercase letters then first the lower case letters would be sorted

sort filename.txt --> Sorting a file

sort -o filename.txt inputfile.txt --> write the output to a new file

sort -r inputfile.txt --> perform a reverse-order sort

sort -n filename.txt --> sort the file with numeric data present inside

sort -nr filename.txt --> sort the file with numeric data in reverse order present inside

sort -nk2 lsl.txt --> Sort the contents of file ‘lsl.txt‘ on the basis of 2nd column with numeric value

sort -k9 lsl.txt --> Sort the contents of file ‘lsl.txt‘ on the basis of 9th column

ls -l /home/$USER | sort -nk5 --> Using pipe

sort -c filename.txt --> check if the file given is already sorted or not

sort -u filename.txt --> sort and remove duplicates pass the -u option to sort

sort -M filename.txt --> sort by month

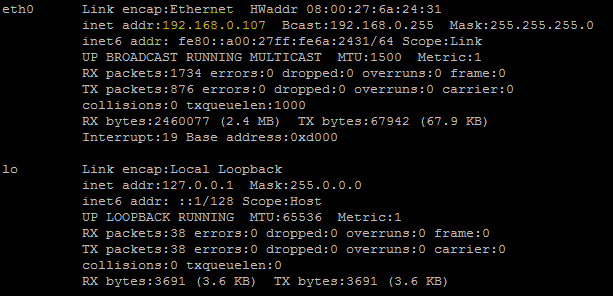
sort lsl.txt lsla.txt --> Sort the contents of two files on standard output in one go

sort -u lsl.txt lsla.txt --> sort, merge and remove duplicates from the two files

ls -l /home/$USER | sort -t "," -nk2,5 -k9 --> Sort the output of ‘ls -l‘ command on the basis of field 2,5 (Numeric) and 9 (Non-Numeric)

**Networking Commands:-**

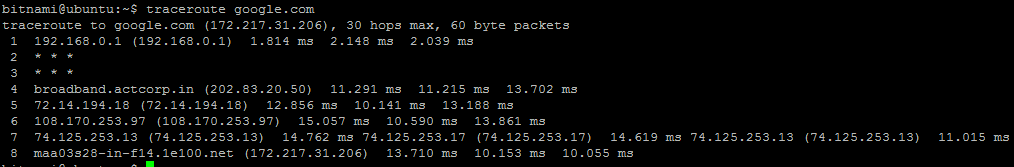
**Ifconfig** - ifconfig (interface configurator) command is use to initialize an interface, assign IP Address to interface and enable or disable interface on demand. With this command you can view IP Address and Hardware / MAC address assign to interface and also MTU (Maximum transmission unit) size.



## Featured snippet from the web

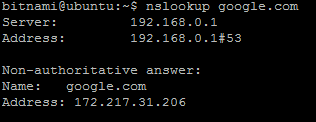
**MTU** (**Maximum Transmission Unit**) is related to TCP/IP networking in **Linux**/BSD/UNIX oses. It refers to the size (in bytes) of the **largest** datagram that a given layer of a communications protocol can pass at a time.

**traceroute:- traceroute** is a network troubleshooting utility which shows number of hops taken to reach destination also determine packets traveling path. Below we are tracing route to global **DNS server IP Address** and able to reach destination also shows path of that packet is traveling.

****

sudo apt-get install traceroute – Install traceroute if not installed

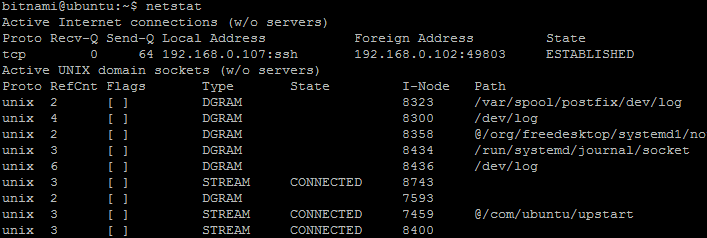
**nslookup :-**  **Nslookup**(stands for “Name Server Lookup”) is a useful command for getting information from DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS related problems.



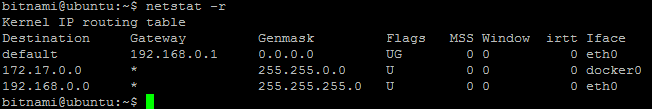
You can also do the reverse DNS look-up by providing the IP Address as argument to nslookup.

**nslookup 192.168.0.10**

**netstat :-** [Netstat](https://geekflare.com/netstat/)command allows you a simple way to review each of your network connections and open sockets. It is used to display routing table, connection information, the status of ports, etc. This command works with Linux Network Subsystem. This command basically displays the content of /proc/net file defined in the Linux file system.

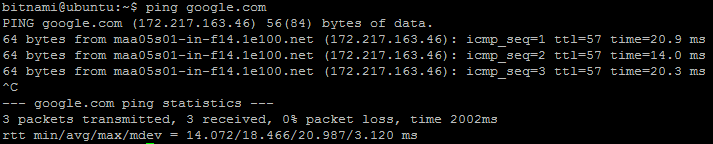


To displays routing table information use option as**–r / use command route**



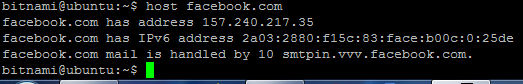
**Ping:-**

**PING** (**Packet INternet Groper**) command is the best way to test connectivity between **two nodes**. Whether it is **Local Area Network** (**LAN**) or **Wide Area Network** (**WAN**). Ping use **ICMP** (**Internet Control Message Protocol**) to communicate to other devices. Once the packets are received by the destined computer, it starts sending the packets back. This command keeps executing until it is interrupted.



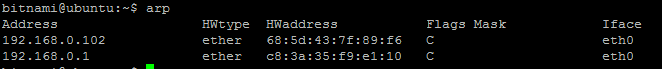
**HOST:-**

host command is used to find domain name associated with the IP address or find IP address associated with domain name. The returned IP address is either IPv4 or IPv6.

****

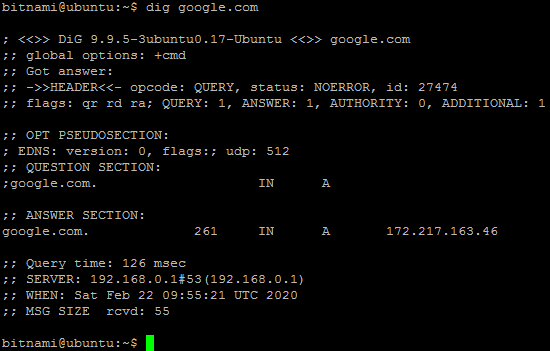
**ARP :-**

**arp command** manipulates the System’s ARP cache. It also allows a complete dump of the ARP cache. ARP stands for Address Resolution Protocol. The primary function of this protocol is to resolve the IP address of a system to its mac address, and hence it works between level 2(Data link layer) and level 3(Network layer).

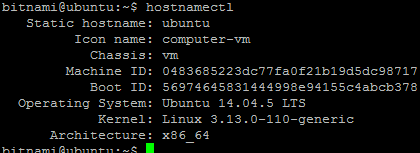


**DIG:-**

The dig command stands for**D**omain **I**nformation **G**roper. This command is used for task related to DNS lookup to query DNS name servers. It mainly deals with troubleshoot DNS related problems.



**Hostnamectl:-**



**NMAP:-**

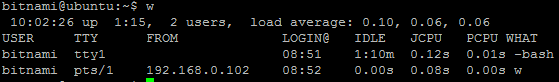
The **Nmap** aka **Network Mapper** is an open source and a very versatile tool for Linux system/network administrators. **Nmap** is used for **exploring networks**, **perform security scans**, **network audit** and **finding open ports** on remote machine. It scans for Live hosts, Operating systems, packet filters and open ports running on remote hosts.

****

**W:-**

w prints a summary of the current activity on the system, including what each user is doing, and their processes.

Also list the logged in users and system load average for the past 1, 5, and 15 minutes.



**More commands**

**Nohup** - Nohup, short for no hang up is a command in Linux systems that keep processes running even after exiting the shell or terminal.

Nohup prevents the processes or jobs from receiving the SIGHUP (Signal Hang UP) signal. This is a signal that is sent to a process upon closing or exiting the terminal.

If you want to keep your processes/jobs running, precede the command with nohup



From the output above, the output of the command has been saved to nohup.out to verify this run,

**cat nohup.out**

**How to Run Linux Commands in Background:**Typically when you run a command in the terminal, you have to wait until the command finishes before you can enter another one. This is called running the command in the foreground or foreground process. When a process runs in the foreground, it occupies your shell, and you can interact with it using the input devices

A background process is a process/command that is started from a terminal and runs in the background, without interaction from the user.

To run a command in the background, add the ampersand symbol (&) at the end of the command:

**command &**

The shell job ID (surrounded with brackets) and process ID will be printed on the terminal:

**[1] 25177**

You can have multiple processes running in the background at the same time.

The background process will continue to write messages to the terminal from which you invoked the command. To suppress the stdout and stderr messages use the following syntax:

**command > /dev/null 2>&1 &**

>/dev/null 2>&1 means redirect stdout to /dev/null and stderr to stdout.

Use the jobs utility to display the status of all stopped and background jobs in the current shell session:

**jobs -l**

The output includes the job number, process ID, job state, and the command that started the job:

[1]+ 25177 Running ping google.com &

If you have multiple background jobs, include % and the job ID after the command:

**fg %1**

## Move a Foreground Process to Background

To move a running foreground process in the background:

1. Stop the process by typing Ctrl+Z.
2. Move the stopped process to the background by typing bg.

One way is to remove the job from the shell’s job control using the disown shell builtin:

**disown**

**Compress and un-compress:-**

$zip myfile.zip filename.txt

$zip –d filename.zip file.txt --> Removes the file from the zip archive

$zip –u filename.zip file.txt --> Updates the file in the zip archive

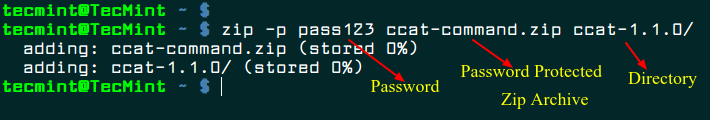
$zip –m filename.zip file.txt --> Deletes the original files after zipping

$zip –r filename.zip directory\_name --> To zip a directory recursively, use the -r option

$zip –x filename.zip file\_to\_be\_excluded --> Exclude a file

$zip -p pass123 ccat-command.zip ccat-1.1.0/ --> Protect with password

$ zip -e ccat-command.zip ccat-1.1.0/ --> Protect a zip file with password



$unzip latest.zip --> Unzip a zip file

$unzip -q filename.zip --> suppress the printing of these messages

$unzip filename.zip -d /path/to/directory --> unzip a ZIP file to a different directory than the current one, use the -d

$unzip -P PasswOrd filename.zip --> Unzip a password protected file

**TAR command :-**

The Linux ‘tar’ stands for **tape archive**, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

The tar is most widely used command to create compressed archive files and that can be moved easily from one disk to another disk or machine to machine.

**Syntax:**

**tar [options] [archive-file] [file or directory to be archived]**

**Options:**  
**-c :** Creates Archive  
**-x :** Extract the archive  
**-f :** creates archive with given filename  
**-t :** displays or lists files in archived file  
**-u :** archives and adds to an existing archive file  
**-v :** Displays Verbose Information  
**-A :** Concatenates the archive files  
**-z :** zip, tells tar command that create tar file using gzip  
**-j :** filter archive tar file using tbzip  
**-W :** Verify a archive file  
**-r :** update or add file or directory in already existed .tar file

#tar -cvf tecmint-14-09-12.tar /home/tecmint/ --> create a tar archive file tecmint-14-09-12.tar for a directory /home/tecmint in current working directory

# tar cvzf MyImages-14-09-12.tar.gz /home/MyImages --> create a compressed MyImages-14-09-12.tar.gz file for the directory /home/MyImages. (Note : tar.gz and tgz both are similar)

OR

# tar cvzf MyImages-14-09-12.tgz /home/MyImages

# tar cvfj Phpfiles-org.tar.bz2 /home/php --> bz2 feature compress and create archive file less than the size of the gzip

# tar -xvf public\_html-14-09-12.tar --> untar the file public\_html-14-09-12.tar in present working directory

#tar -xvf thumbnails-14-09-12.tar.gz --> Uncompress tar.gz archive file

# tar -tvf uploadprogress.tar --> To list the contents of tar archive file

# tar -xvf cleanfiles.sh.tar cleanfiles.sh --> To extract a single file called cleanfiles.sh from cleanfiles.sh.tar

# tar -xvf Phpfiles-org.tar --wildcards '\*.php' --> to extract a group of all files whose pattern begins with .php from a tar

# tar -rvf tecmint-14-09-12.tar xyz.txt --> To add files or directories to existing tar archived file we use the option r (append)

**SCP Command :-**

**scp** (secure copy) command in Linux system is used to copy file(s) between servers in a secure way. It uses the same authentication and security as it is used in the Secure Shell (SSH) protocol.

**Options:**

* **scp –P port:**Specifies the port to connect on the remote host.
* **scp –p:**Preserves modification times, access times, and modes from the original file.
* **scp –q:**Disables the progress meter.
* **scp –r:**Recursively copy entire directories.
* **scp –S program:**Name of program to use for the encrypted connection. The program must understand ssh(1) options.
* **scp –v:**Verbose mode. Causes *scp*and *ssh*to print debugging messages about their progress. This is helpful in debugging connection, authentication, and configuration problems.

scp file.txt [remote\_username@10.10.0.2:/remote/directory](mailto:remote_username@10.10.0.2:/remote/directory) - To copy a file from a local to a remote system

scp -P 2322 file.txt [remote\_username@10.10.0.2:/remote/directory](mailto:remote_username@10.10.0.2:/remote/directory) - specify the port using the -P argument

scp -r /local/directory [remote\_username@10.10.0.2:/remote/directory](mailto:remote_username@10.10.0.2:/remote/directory) - To copy a directory from a local to remote system, use the -r option

scp remote\_username@10.10.0.2:/remote/file.txt /local/directory - copy a file named file.txt from a remote server with IP 10.10.0.2

scp user1@host1.com:/files/file.txt [user2@host2.com:/files](mailto:user2@host2.com:/files) - copy the file /files/file.txt from the remote host host1.com to the directory /files on the remote host host2.com

scp -3 user1@host1.com:/files/file.txt [user2@host2.com:/files](mailto:user2@host2.com:/files) - To route the traffic through the machine on which the command is issued, use the -3 option

scp -l 400 Label.pdf [mrarianto@202.x.x.x](mailto:mrarianto@202.x.x.x):. - The “-l” parameter will limit the total bandwidth to use. It will be useful if you made an automation script to copy a lot of files, but you don’t want the bandwidth to be drained by the SCP process.

#### scp install.txt index.html jdk-linux-x64\_bin.rpm [root@172.20.10.8:/mnt](mailto:root@172.20.10.8:/mnt) - Transfer multiple files

**yes** command in linux is used to print a continuous output stream of given STRING. If STRING is not mentioned then it prints ‘y’;**Question:** Where it is used ?

**Ans:**Lets say that we want to delete all the .txt file present in the current directory. Instead of writing **rm -i \*.txt** and then typing y at the end for every file, what we can do is we can use **yes | rm -i \*.txt**.

**BC command:-bc** command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

 Arithmetic operators

 Increment or Decrement operators

 Assignment operators

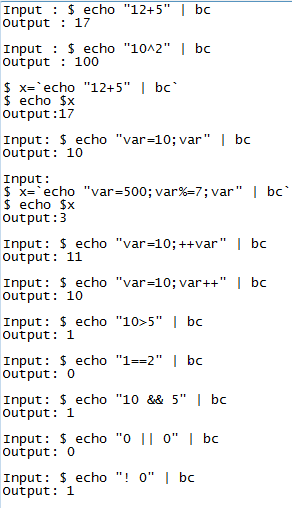
 Comparison or Relational operators

 Logical or Boolean operators

 Math functions

 Conditional statements

 Iterative statements

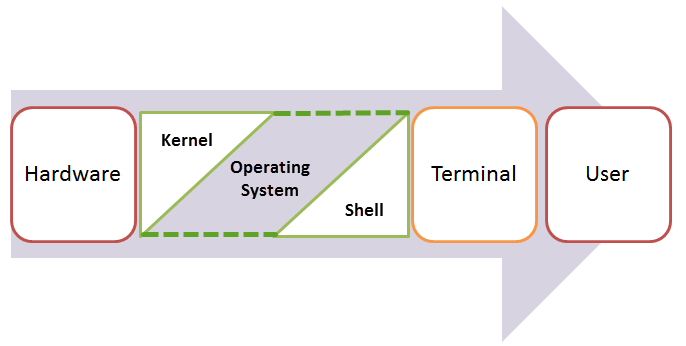


**Shell Scripting:-**

What is a Shell?

An Operating is made of many components, but its two prime components are -

**Kerne and Shell**



**SHELL** is a program which provides the interface between the user and an operating system. When the user logs in OS starts a shell for user. **Kernel** controls all essential computer operations, and provides the restriction to hardware access, coordinates all executing utilities, and manages Resources between process. Using kernel only user can access utilities provided by operating system.

**Types of Shells in Unix:-**

**The C Shell –** Denoted as **csh** /bin/csh

**The Bourne Shell –** Denoted as **sh** /bin/sh and /sbin/sh

**Korn Shell -** It is denoted as **ksh** /bin/ksh

**GNU Bourne-Again Shell –** Denoted as **bash** /bin/bash

**How to determine Shell :-**

**echo $SHELL**

****

The $ sign stands for a shell variable, echo will return the text whatever you typed in.

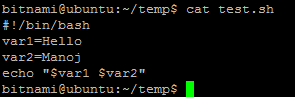
The sign #**!** is called **she-bang** and is written at top of the script. It passes instruction to program **/bin/sh.ch**

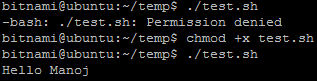
To run your script in a certain shell (shell should be supported by your system), start your script with **#!** followed by the shell name.

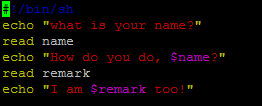
Any line starting with a hash (#) becomes comment.

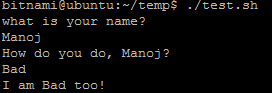
**Shell Scripting Variables:**

The name of a variable can contain only letters (a to z or A to Z), numbers ( 0 to 9) or the underscore character ( \_)









## Read-only Variables

Shell provides a way to mark variables as read-only by using the read-only command. After a variable is marked read-only, its value cannot be changed.

#!/bin/sh

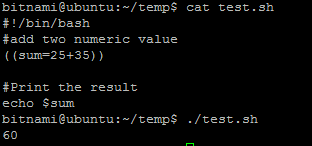
NAME="Zara Ali"

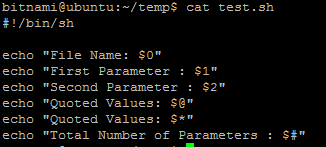
readonly NAME

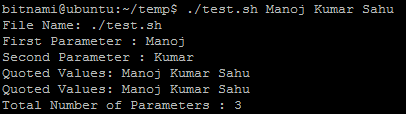
NAME="Qadiri"

Syntax to unset a defined variable using the **unset** command - unset variable\_name

**Sample shell script Examples:-**

****

****

****

**$0** - The filename of the current script.

**$n** - These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument

**$#** - The number of arguments supplied to a script.

**$\*** - All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2.

**$@** - All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.

**$?** - The exit status of the last command executed.

**$$** - The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

**$!** - The process number of the last background command.

Defining Array Values :-



## Arithmetic Operators:-

Assume variable **a** holds 10 and variable **b** holds 20 then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator | `expr $a + $b` will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand | `expr $a - $b` will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator | `expr $a \\* $b` will give 200 |
| / (Division) | Divides left hand operand by right hand operand | `expr $b / $a` will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder | `expr $b % $a` will give 0 |
| = (Assignment) | Assigns right operand in left operand | a = $b would assign value of b into a |
| == (Equality) | Compares two numbers, if both are same then returns true. | [ $a == $b ] would return false. |
| != (Not Equality) | Compares two numbers, if both are different then returns true. | [ $a != $b ] would return true. |

#!/bin/sh

val=`expr 2 + 2`

echo "Total value : $val"

## Relational Operators:-

Assume variable **a** holds 10 and variable **b** holds 20 then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
|  |  |  |
| **-eq** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a -eq $b ] is not true. |
| **-ne** | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true. | [ $a -ne $b ] is true. |
| **-gt** | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true. | [ $a -gt $b ] is not true. |
| **-lt** | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true. | [ $a -lt $b ] is true. |
| **-ge** | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -ge $b ] is not true. |
| **-le** | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -le $b ] is true. |

## String Operators:-

Assume variable **a** holds "abc" and variable **b** holds "efg" then −

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **=** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a = $b ] is not true. |
| **!=** | Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true. | [ $a != $b ] is true. |
| **-z** | Checks if the given string operand size is zero; if it is zero length, then it returns true. | [ -z $a ] is not true. |
| **-n** | Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true. | [ -n $a ] is not false. |
| **str** | Checks if **str** is not the empty string; if it is empty, then it returns false. | [ $a ] is not false. |

## File Test Operators :-

Assume a variable **file** holds an existing file name "test" the size of which is 100 bytes and has **read**, **write** and **execute** permission on –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **-b file** | Checks if file is a block special file; if yes, then the condition becomes true. | [ -b $file ] is false. |
| **-c file** | Checks if file is a character special file; if yes, then the condition becomes true. | [ -c $file ] is false. |
| **-d file** | Checks if file is a directory; if yes, then the condition becomes true. | [ -d $file ] is not true. |
| **-f file** | Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true. | [ -f $file ] is true. |
| **-g file** | Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true. | [ -g $file ] is false. |
| **-k file** | Checks if file has its sticky bit set; if yes, then the condition becomes true. | [ -k $file ] is false. |
| **-p file** | Checks if file is a named pipe; if yes, then the condition becomes true. | [ -p $file ] is false. |
| **-t file** | Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true. | [ -t $file ] is false. |
| **-u file** | Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true. | [ -u $file ] is false. |
| **-r file** | Checks if file is readable; if yes, then the condition becomes true. | [ -r $file ] is true. |
| **-w file** | Checks if file is writable; if yes, then the condition becomes true. | [ -w $file ] is true. |
| **-x file** | Checks if file is executable; if yes, then the condition becomes true. | [ -x $file ] is true. |
| **-s file** | Checks if file has size greater than 0; if yes, then condition becomes true. | [ -s $file ] is true. |
| **-e file** | Checks if file exists; is true even if file is a directory but exists. | [ -e $file ] is true. |

**Boolean Operators:-**

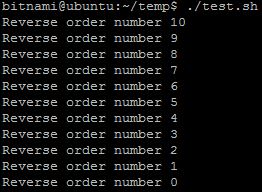
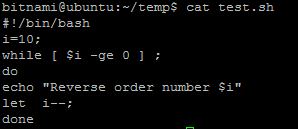
Assume variable **a** holds 10 and variable **b** holds 20 then –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **!** | This is logical negation. This inverts a true condition into false and vice versa. | [ ! false ] is true. |
| **-o** | This is logical **OR**. If one of the operands is true, then the condition becomes true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| **-a** | This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

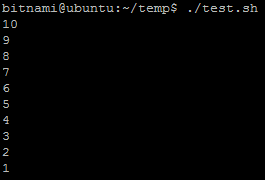
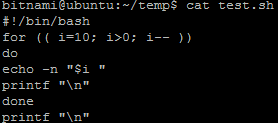
### Using While Loop:

#!/bin/bash  
valid=true  
count=1  
while [ $valid ]  
do  
echo $count  
if [ $count -eq 5 ];  
then  
break  
fi  
((count++))  
done

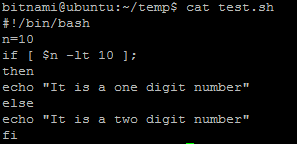




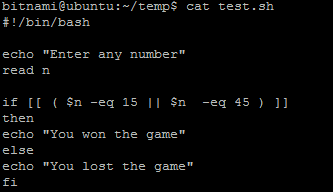
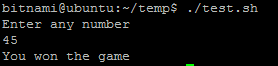
### Using For Loop:

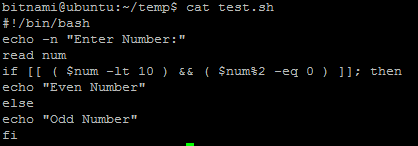


### Using if statement:

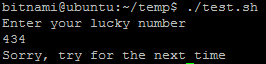
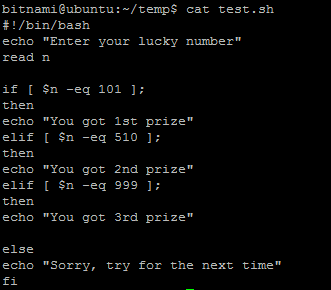
 

### Using if statement with AND / OR logic:

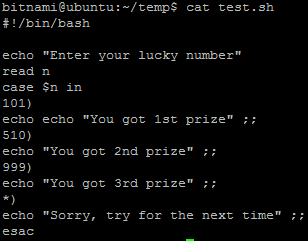
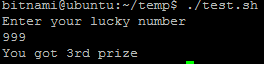
 

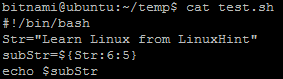
 

**Using else if statement:**

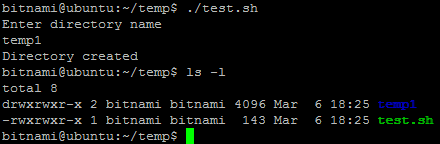
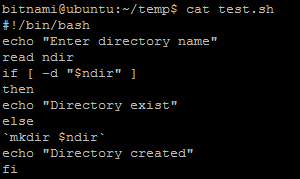
****

### Using Case Statement:

** Get substring of String:**

### Make directory by checking existence:



### Test if File Exist:

#!/bin/bash  
filename=$1  
if [ -f "$filename" ]; then  
echo "File exists"  
else  
echo "File does not exist"  
fi

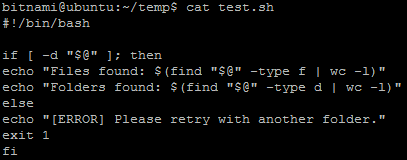
### Send Email:

#!/bin/bash  
Recipient=”admin@example.com”  
Subject=”Greeting”  
Message=”Welcome to our site”  
`mail -s $Subject $Recipient <<< $Message`

#### ****Displaying the Last Updated File****

ls -lrt | grep ^- | awk 'END{print $NF}'

#### ****Print Number of Files or Directories****





#### ****Backup Script Using Bash****

#!/bin/bash

BACKUPFILE=backup-$(date +%m-%d-%Y)

archive=${1:-$BACKUPFILE}

find . -mtime -1 -type f -print0 | xargs -0 tar rvf "$archive.tar"

echo "Directory $PWD backed up in archive file \"$archive.tar.gz\"."

exit 0

#### ****Removing Duplicate Lines from Files****

#! /bin/sh

echo -n "Enter Filename-> "

read filename

if [ -f "$filename" ]; then

sort $filename | uniq | tee sorted.txt

else

echo "No $filename in $pwd...try again"

fi

exit 0

<https://www.ubuntupit.com/simple-yet-effective-linux-shell-script-examples/>

**Chess Program:**

#!/bin/bash

for i in $(seq 1 8)

do

for j in $(seq 1 8)

do

S=$(((i+j)%2))

if [ $S -eq 0 ]

then

echo -n "\033[47m " # white

else

echo -n "\033[40m " # black

fi

done

echo -n "\033[40m" # black, ensure it exists normally

echo "" # new line

done

-----------------------------------------------------------------------------------------------------------------------------

for (( i = 1; i <= 9; i++ )) ### Outer for loop ###

do

for (( j = 1 ; j <= 9; j++ )) ### Inner for loop ###

do

tot=`expr $i + $j`

tmp=`expr $tot % 2`

if [ $tmp -eq 0 ]; then

echo -e -n "\033[47m "

else

echo -e -n "\033[40m "

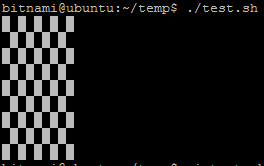
fi

done

echo -e -n "\033[40m" #### set back background colour to black

echo "" #### print the new line ###

done



**Sample Pyramids:-**

#!/bin/bash

a=0

while [ "$a" -lt 10 ] # this is loop1

do

b="$a"

while [ "$b" -ge 0 ] # this is loop2

do

echo -n "$b "

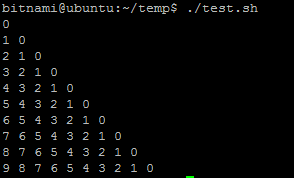
b=`expr $b - 1`

done

echo

a=`expr $a + 1`

done



## Bash Shell Script to print half pyramid using \*

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

|  |
| --- |
| #Bash Shell Script to print half pyramid using \*  rows=4  for((i=1; i<=rows; i++))  do    for((j=1; j<=i; j++))    do      echo -n "\* "    done    echo  done |

## Bash Shell Script to print half pyramid using numbers

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

|  |
| --- |
| #Bash Shell Script to print half pyramid using numbers  number=1  rows=5  for((i=1; i<=rows; i++))  do    for((j=1; j<=i; j++))    do      echo -n "$number "      number=$((number + 1))    done    number=1    echo  done |

## Bash Shell Script to print inverted half pyramid using \*

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

|  |
| --- |
| #Bash Shell Script to print inverted half pyramid using \*  rows=5  for((i=rows; i>=1; i--))  do    for((j=1; j<=i; j++))    do      echo -n "\* "    done    echo  done |

## Bash Shell Script to print inverted half pyramid using numbers

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

|  |
| --- |
| #Bash Shell Script to print inverted half pyramid using numbers  number=1  rows=5  for((i=rows; i>=1; i--))  do    for((j=1; j<=i; j++))    do      echo -n "$number "      number=$((number + 1))    done    number=1    echo  done |

## Bash Shell Script to print full pyramid using \*

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

|  |
| --- |
| #Bash Shell Script to print full pyramid using \*  rows=5  for((i=1; i<=rows; i++))  do    for((j=1; j<=rows - i; j++))    do      echo -n "  "    done    for((j=1; j<=2\*i - 1; j++))    do      echo -n "\* "    done    echo  done |

## Bash Shell Script to print full pyramid using numbers

1

2 3 2

3 4 5 4 3

4 5 6 7 6 5 4

5 6 7 8 9 8 7 6 5

|  |
| --- |
| #Bash Shell Script to print full pyramid using numbers  rows=5  number=1  for((i=1; i<=rows; i++))  do    for((j=1; j<=rows - i; j++))    do      echo -n "  "    done    number=$i    k=1    for((j=1; j<=2\*i - 1; j++))    do      if [ $j -lt $i ];      then        echo -n "$number "        number=$((number + 1))      elif [ $j -eq $i ];      then        echo -n "$number "        number=$((number - 1))      else        echo -n "$number "        number=$((number - 1))      fi    done    echo  done |

## Bash Shell Script to print Floyd's Triangle

1

2 3

4 5 6

7 8 9 10

|  |
| --- |
| #Bash Shell Script to print Floyd's Triangle  number=1  rows=4  for((i=1; i<=rows; i++))  do    for((j=1; j<=i; j++))    do      echo -n "$number "      number=$((number + 1))    done    echo  done |

<https://www.tutorialsandyou.com/bash-shell-scripting/pyramid-and-pattern-10.html>

**# Program for Fibonacci**

# Series

# Static input fo N

N=6

a=0

b=1

echo "The Fibonacci series is : "

for (( i=0; i<N; i++ ))

do

echo -n "$a "

fn=$((a + b))

a=$b

b=$fn

done

# End of for loop

**Basic Calculator :**

# !/bin/bash

# Take user Input

echo "Enter Two numbers : "

read a

read b

# Input type of operation

echo "Enter Choice :"

echo "1. Addition"

echo "2. Subtraction"

echo "3. Multiplication"

echo "4. Division"

read ch

# Switch Case to perform

# calulator operations

case $ch in

1)res=`echo $a + $b | bc`

;;

2)res=`echo $a - $b | bc`

;;

3)res=`echo $a \\* $b | bc`

;;

4)res=`echo "scale=2; $a / $b" | bc`

;;

esac

echo "Result : $res"

**Swap 2 numbers**

first=5

second=10

temp=$first

first=$second

second=$temp

echo "After swapping, numbers are:"

echo "first = $first, second = $second"

**KUBERNETES**

**Cheatsheet: Kubernetes commands**

kubectl get pod: Get information about all running pods

kubectl describe pod <pod>: Describe one pod

kubectl expose pod <pod> --port=444 --name=frontend: Expose the port of a pod (creates a new service)

kubectl port-forward <pod> 8080: Port forward the exposed pod port to your local machine

kubectl attach <podname> -i: Attach to the pod

kubectl exec <pod> -- command: Execute a command on the pod

kubectl label pods <pod> mylabel=awesome: Add a new label to a pod

kubectl run -i --tty busybox --image=busybox --restart=Never -- sh: Run a shell in a pod - very useful for debugging

kubectl get deployments: Get information on current deployments

kubectl get rs: Get information about the replica sets

kubectl get pods --show-labels: get pods, and also show labels attached to those pods

kubectl rollout status deployment/helloworld-deployment: Get deployment status

kubectl set image deployment/helloworld-deployment k8s-demo=k8s-demo:2: Run k8s-demo with the image label version 2

kubectl edit deployment/helloworld-deployment: Edit the deployment object

kubectl rollout status deployment/helloworld-deployment: Get the status of the rollout

kubectl rollout history deployment/helloworld-deployment: Get the rollout history

kubectl rollout undo deployment/helloworld-deployment: Rollback to previous version

kubectl rollout undo deployment/helloworld-deployment --to-revision=n: Rollback to any version version

## What is Kubernetes?

Kubernetes is an open source [Orchestration](https://www.edureka.co/blog/kubernetes-tutorial/#Challenges%20Without%20Container%20Orchestration) system for Docker Containers. Kubernetes is a platform that eliminates the manual processes involved in deploying containerized applications

It is a container management system developed in the Google platform. It helps you to manage a containerized application in various types of Physical, virtual, and cloud environments.

* It lets you to schedule containers on a cluster of machines
* You can run multiple containers on one machine.
* You can run long running services (like web applications)

Kubernetes will manage the state of these containers

* Can start the container on specific nodes
* Will restart a container when it get killed
* Can move containers from one node to another node.

Instead of just running a few docker containers on one host manually, Kubernetes is a platform that will manage the containers for you.

Kubernetes clusters can start with one node until thousands of nodes.

Popular docker orchestrators are – Docker Swarm , Mesos

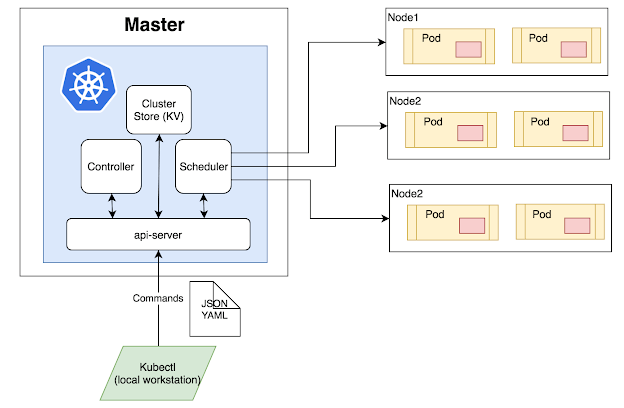
## Why use Kubernetes?

* Kubernetes can run on-premises bare metal, OpenStack, public clouds Google, Azure, AWS, etc.
* Helps you to avoid vendor lock issues as it can use any vendor-specific APIs or services except where Kubernetes provides an abstraction, e.g., load balancer and storage.
* Containerization using kubernetes allows package software to serve these goals. It will enable applications that need to be released and updated without any downtime.
* Kubernetes allows you to assure those containerized applications run where and when you want and helps you to find resources and tools which you want to work.

## Features of Kubernetes

* Automated Scheduling
* Self-Healing Capabilities
* Automated rollouts & rollback
* Horizontal Scaling & Load Balancing
* Offers environment consistency for development, testing, and production
* Infrastructure is loosely coupled to each component can act as a separate unit
* Provides a higher density of resource utilization
* Offers enterprise-ready features
* Application-centric management
* Auto-scalable infrastructure
* You can create predictable infrastructure
* **Automated Scheduling:** Kubernetes provides advanced scheduler to launch container on cluster nodes based on their resource requirements and other constraints, while not sacrificing availability.
* **Self Healing Capabilities:**Kubernetes allows to replaces and reschedules containers when nodes die. It also kills containers that don’t respond to user-defined health check and doesn’t advertise them to clients until they are ready to serve.
* **Automated rollouts & rollback:**Kubernetes rolls out changes to the application or its configuration while monitoring application health to ensure it doesn’t kill all your instances at the same time. If something goes wrong, with Kubernetes you can rollback the change.
* **Horizontal Scaling & Load Balancing:**Kubernetes can scale up and scale down the application as per the requirements with a simple command, using a UI, or automatically based on CPU usage

## Kubernetes Architecture:

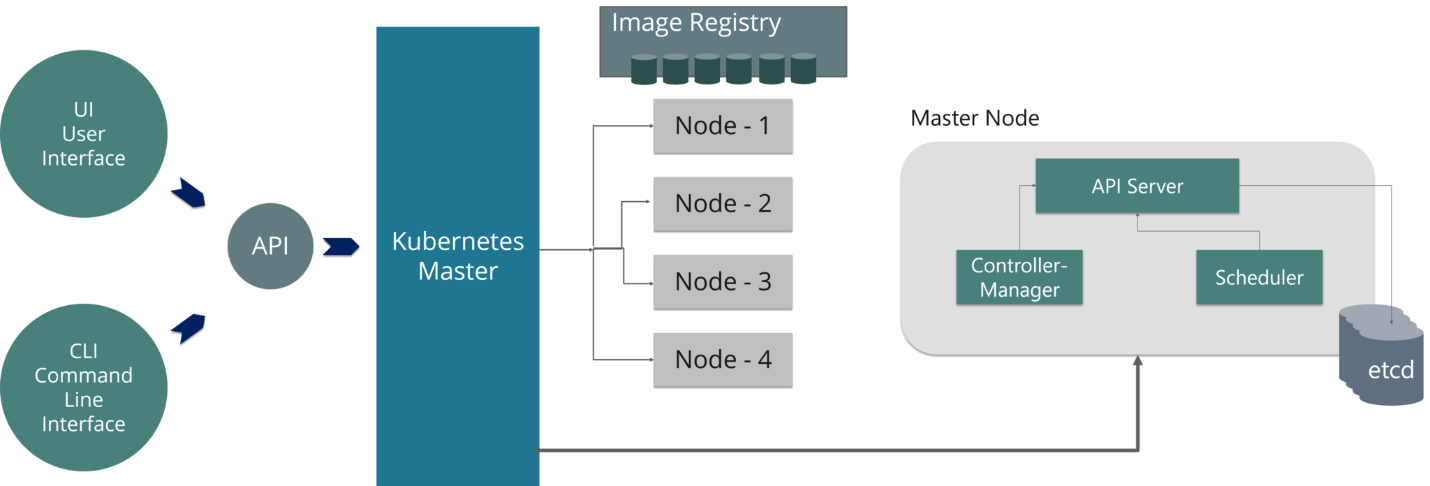


Kubernetes Architecture has the following main components:

* Master nodes
* Worker/Slave nodes

### **Master Node**

The master node is responsible for the management of Kubernetes cluster. It is mainly the entry point for all administrative tasks. There can be more than one master node in the cluster to check for fault tolerance.



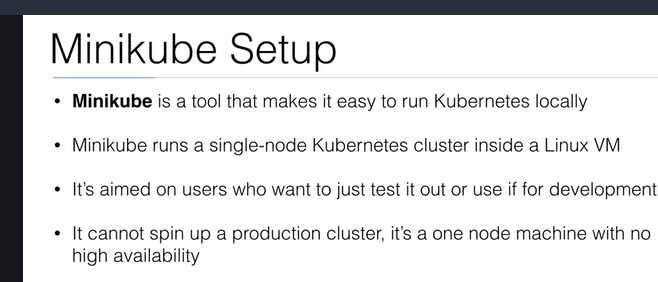
The master node has various components like API Server, Controller Manager, Scheduler, and ETCD.

* **API Server:**The API server is the entry point for all the REST commands used to control the cluster. **Kubeconfig** is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.
* **Controller Manager:**Is a daemon that regulates the Kubernetes cluster, and manages different non-terminating control loops. It is responsible for collecting and sending information to API server. It works toward getting the shared state of cluster and then make changes to bring the current status of the server to the desired state.
* **Scheduler:**The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node. It is responsible for distributing the workload. It also helps you to track how the working load is used on cluster nodes. It helps you to place the workload on resources, which are available and accept the workload.
* **ETCD:**It stores the configuration information, which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server as it may have some sensitive information. It is a distributed key value Store which is accessible to all.

### **Worker/Slave nodes**

Worker nodes contain all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the scheduled containers. The worker node has various components as below

* **Docker Container:** Docker runs on each of the worker nodes, and runs the configured pods
* **Kubelet:** Kubelet gets the configuration of a Pod from the API server and ensures that the described containers are up and running. It interacts with **etcd** store to read configuration details and wright values. This communicates with the master component to receive commands and work. The **kubelet** process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.
* **Kube-proxy:**Kube-proxy acts as a network proxy and a load balancer for a service on a single worker node. This is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time, it is isolated as well. It manages pods on node, volumes, secrets, creating new containers’ health checkup, etc.
* **Pods:** A pod is one or more containers that logically run together on nodes.



## Installation of Minikube on EC2 Ubuntu1. Run a public EC2 Server with the following setup

|  |  |
| --- | --- |
| **AMI** | Ubuntu Server 18.04 LTS (HVM), SSD Volume Type |
| **Instance Type** | t3.micro (2 vCPU, 1GB Memory) |
| **Storage** | 8 GB (gp2) |
| **Tags** | – Key: Name – Value: Minikube |
| **Security Group** | Name: Minikube Security Group – SSH, 0.0.0.0/0 Later we will be editing this. |
| **Key Pair** | Create your own keypair. You will need this to SSH to your EC2 Instance |

***Update:*** I changed the Instance Type from ***t2.micro***(1 vCPU) to ***t3.micro***(2 vCPU). An update to Minikube required a minimum of 2 vCPUs. The error when running with t2.micro was *Requested cpu count 1 is less than the minimum allowed of 2*.

t3.micro is no longer in the Free Tier, make sure to stop or terminate the instance after you are done testing to avoid a huge AWS bill.

Thank you to everyone in the comments section who pointed this change.

### 2. SSH into your created EC2 Instance using your keypair.

ssh ubuntu@<ipv4\_public\_ip> -i <keypair>.pem

### 3. Install kubectl

curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin/kubectl

### 4. Install Docker

sudo apt-get update && \

sudo apt-get install docker.io -y

Minikube requires Docker.

### 5. Install Minikube

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

### 6. Check Minikube Version

minikube version

We have now successfully installed Minikube!

Let’s test it!

## Running Minikube on EC2 Ubuntu

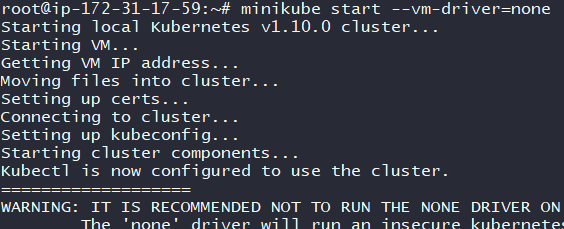
Become a root user.

sudo -i

If you are not comfortable running commands as root, you must always add sudo before the commands minikube and kubectl.

### 2. Start Minikube

minikube start --vm-driver=none

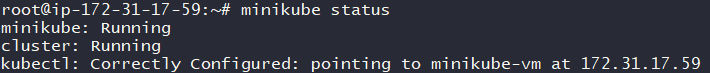


Do not worry about the warning. As long as you see the message ‘Kubectl is now configured to use the cluster.’ you have successfully ran Minikube.

**Note:** In the [Install Minikube documentation from Kubernetes.io](https://kubernetes.io/docs/tasks/tools/install-minikube/) it says that you need to enable virtualization by accessing the computer’s BIOS. For EC2 Instances we do not have access to the BIOS since AWS EC2 instance is a Virtual Machine. Thus we are using the --vm-driver=none tag. No need to install a Hypervisor (VirtualBox or KVM)

### 3. Check the status of Minikube

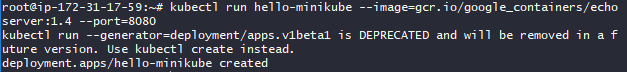
minikube status



If you see the status as ‘running’ then we can now run kubectl commands.

### 4. Let us run our first container

kubectl run hello-minikube --image=gcr.io/google\_containers/echoserver:1.4 --port=8080



### 5. Expose the container ports so that we can access it.

kubectl expose deployment hello-minikube --type=NodePort

https://www.radishlogic.com/wp-content/uploads/2018/10/expose-node-port.png

### 6. Find where port 8080 in container exposed in EC2 Instance port.

kubectl get services

https://www.radishlogic.com/wp-content/uploads/2018/10/get-the-exposed-ports.png

**Note:**Port 30263 is the EC2 Instance Port where the Port 8080 of the container is exposed.

The EC2 Instance Port changes each time you expose a port, you may have been given a different value than what I have.

kubectl get services command shows the list of services and their exposed ports.

Let us check by accessing this via a web browser on our local computer. But first we need to edit our EC2 Security Group.

### 7. Edit Security Group of the EC2 Instance to be access

The goal is for us to be able to access the EC2 Instance Port (30263 for me) via the internet.

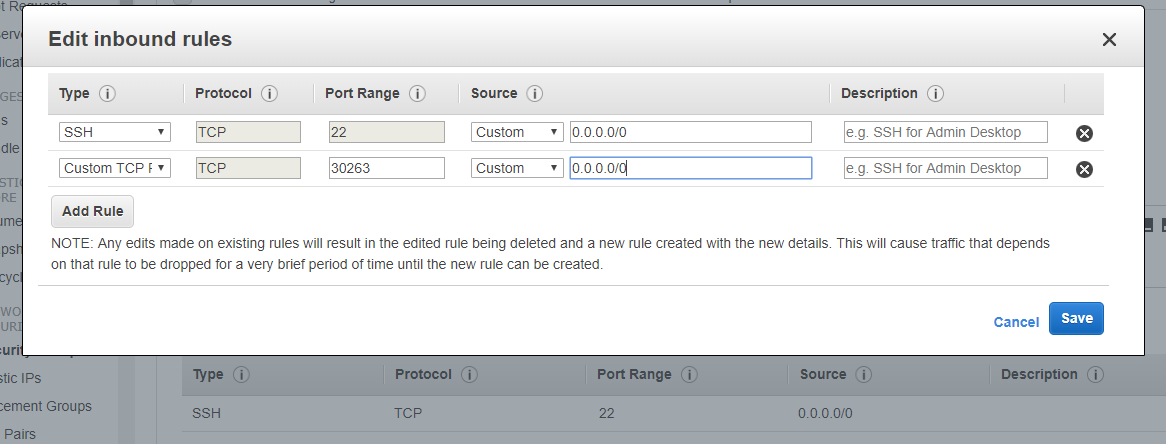
EC2 >> (Network & Security) Security Groups >> Minikube Security Group >> Ingress

Press **Edit**. Then **Add Rule**.

Add the following.

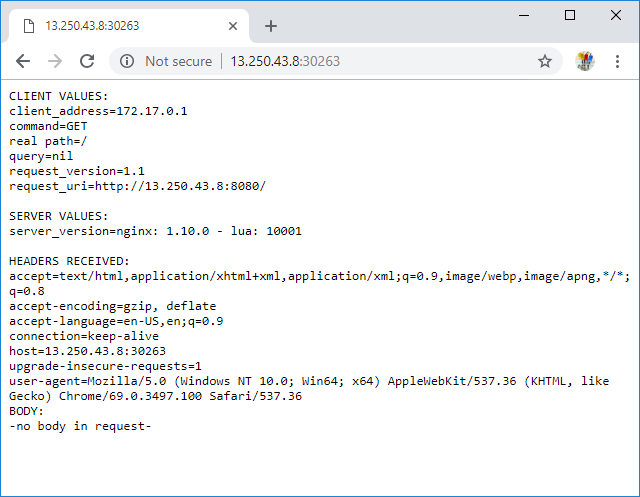
|  |  |
| --- | --- |
| **Type** | Custom TCP Rule |
| **Protocol** | TCP |
| **Port Range** | 30263 (the port given to you by the kubectl get services command) |
| **Source** | Custom 0.0.0.0/0 (Accessible via the internet) |

Click **Save**.



### 8. Access the our container via the EC2 Instance Port on a web browser.

The address is <ipv4\_public\_ip>:<ec2\_port>.



My EC2 Instance has an IPv4 Public IP of 13.250.43.8. And since my hello-minikube port 8080 is exposed on port 30263, the address that I placed on my browser is 13.250.43.8:30263.

See the request\_uri of the page displayed by the web browser, it says that I am accessing via port 8080.

I accessed my container deployment using Chrome on my laptop. You can use any web browser you like (Safari, Internet Explorer, Edge, Firefox, etc.)

Now that we know that we can access our container, let us finish this and clean up.

### 9. Delete the exposed service (port)

kubectl delete services hello-minikube

https://www.radishlogic.com/wp-content/uploads/2018/10/Delete-Service-Port.png

**10. Delete the deployed container (hello-minikube)**

kubectl delete deployment hello-minikube

https://www.radishlogic.com/wp-content/uploads/2018/10/Delete-Deployment-hello-minikube.png

**11.Stopping Minikube/Shutting Down the Cluster**

minikube stop

<https://dreamcloud.artark.ca/docker-hands-on-guide-docker-and-minikube-on-aws-ec2/>

**Install Minikube in Linux:**

Minikube is a free and open source tool that enables you to set up single node Kubernetes cluster inside your Linux system. Minikube can be installed on Linux, MacOS and Windows Operating system. Minikube also supports various Kubernetes features such as NodePorts, DNS, Container Network Interface, Ingress, ConfigMaps, Secrets and much more.

**Step 1 - Update system**

apt-get update -y

apt-get upgrade -y

**Step 2 - install some required packages with the following command**

apt-get install curl wget apt-transport-https -y

**Step 3 - Install VirtualBox Hypervisor**

Minikube supports both KVM and VirtualBox hypervisor. So, you will need to install VirtualBox or KVM to your system.

apt-get install virtualbox virtualbox-ext-pack

**Step 4 - Install Minikube**

wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

Once the download is completed, copy the downloaded file under /usr/local/bin with the following command:

**cp minikube-linux-amd64 /usr/local/bin/minikube**

Next, give execution permission to the minikube with the following command

**chmod 755 /usr/local/bin/minikube**

Next, check the version of Minikube with the following command

**minikube version**

****

## Install Kubectl :-

Kubectl is a tool to deploy and manage applications on Kubernetes. By default, Kubectl is not available in the Ubuntu 18.04 default repository.

First, download and add the GPG key with the following command

**curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -**

add Kubernetes apt repository with the following command

**echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | tee /etc/apt/sources.list.d/kubernetes.list**

Next, update the repository and install Kubectl with the following command

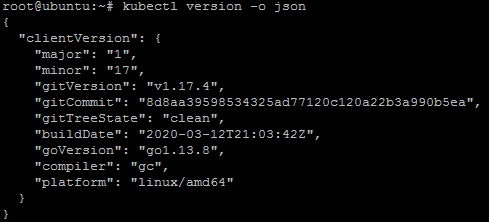
**apt-get update -y**

**apt-get install kubectl –y**

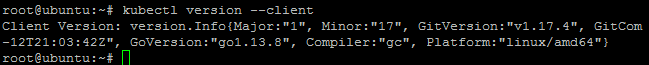
****

Once the Kubectl has been installed, you can check the version using the following command

**kubectl version -o json**

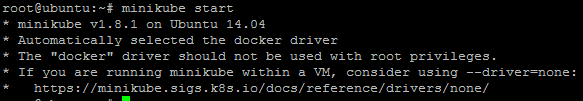


**kubectl version –client**

****

**Start Minikube**

*minikube start*



You can now check the cluster status with the following command:

**kubectl cluster-info**

You should see the following output:

Kubernetes master is running at https://192.168.99.100:8443

KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

You can also check the Kubectl default configuration with the following command:

**kubectl config view**

You should see the following output:

apiVersion: v1

clusters:

- cluster:

certificate-authority: /root/.minikube/ca.crt

server: https://192.168.99.100:8443

name: minikube

contexts:

- context:

cluster: minikube

user: minikube

name: minikube

current-context: minikube

kind: Config

preferences: {}

users:

- name: minikube

user:

client-certificate: /root/.minikube/client.crt

client-key: /root/.minikube/client.key

To check the running nodes, run the following command:

**kubectl get nodes**

Output:

NAME STATUS ROLES AGE VERSION

minikube Ready master 2m45s v1.13.3

You can also access the Minikube Virtualbox with the following command:

**minikube ssh**

You should see the following output:

\_\_\_ \_\_\_ (\_) \_\_\_ (\_)| |/') \_ \_ | |\_ \_\_

/' \_ ` \_ `\| |/' \_ `\| || , < ( ) ( )| '\_`\ /'\_\_`\

| ( ) ( ) || || ( ) || || |\`\ | (\_) || |\_) )( \_\_\_/

(\_) (\_) (\_)(\_)(\_) (\_)(\_)(\_) (\_)`\\_\_\_/'(\_,\_\_/'`\\_\_\_\_)

Now, exit from the Virtualbox shell:

$exit

You can also stop and delete kubernetes cluster anytime with the following command:

**minikube stop  
 minikube delete**

You can check the status of Minikube with the following command:

**minikube status**

Yo should see the following output:

host: Running

kubelet: Running

apiserver: Running

kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100

<https://www.howtoforge.com/how-to-install-kubernetes-with-minikube-on-ubuntu-1804-lts/>

<https://computingforgeeks.com/how-to-install-minikube-on-ubuntu-18-04/>

# How To Install Kubernetes Cluster On Ubuntu 16.04:-

## ****Pre-requisites To Install Kubernetes****

Since we are dealing with VMs, we recommend the following settings for the VMs:-

Master:

* 2 GB RAM
* 2 Cores of CPU

Slave/ Node:

* 1 GB RAM
* 1 Core of CPU

By this point of time, I have assumed you have 2 plain Ubuntu VMs imported onto your Oracle Virtual Box. So, I’l just get along with the installation process.

## ****Pre-Installation Steps On Both Master & Slave (To Install Kubernetes)****

The following steps have to be executed on both the master and node machines. Let’s call the **the master as ‘kmaster‘ and node as ‘knode‘.**

First, login as ‘sudo’ user because the following set of commands need to be executed with ‘sudo’ permissions. Then, update your ‘apt-get’ repository.

$ sudo su

# apt-get update

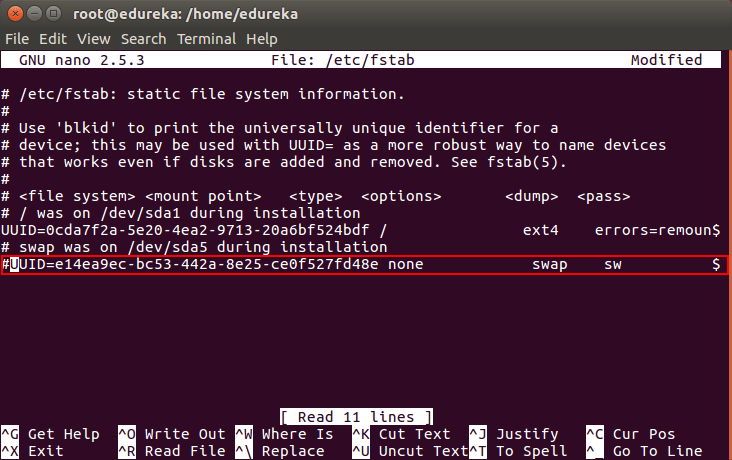
**Note**: After logging-in as ‘sudo’ user, note that your shell symbol will change to ‘#’ from ‘$’.

### ****Turn Off Swap Space****

Next, we have to turn off the swap space because Kubernetes will start throwing random errors otherwise. After that you need to open the ‘fstab’ file and comment out the line which has mention of swap partition.

# swapoff -a

# nano /etc/fstab



Then press ‘Ctrl+X’, then press ‘Y’ and then press ‘Enter’ to Save the file.

### ****Update The Hostnames****

To change the hostname of both machines, run the below command to open the file and subsequently rename the master machine to ‘kmaster’ and your node machine to ‘knode’.

# nano /etc/hostname

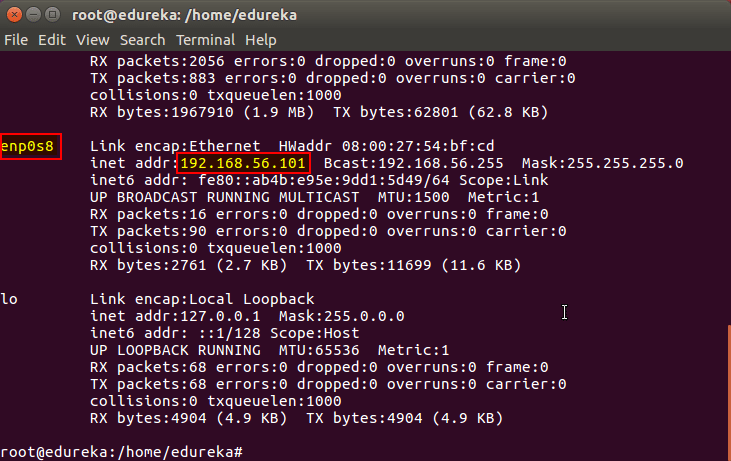


Then press ‘Ctrl+X’, then press ‘Y’ and then press ‘Enter’ to Save the file.

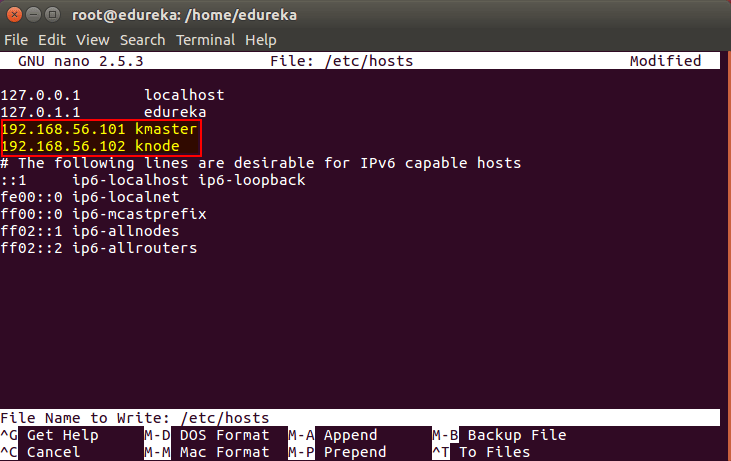
### ****Update The Hosts File With IPs Of Master & Node****

Run the following command on both machines to note the IP addresses of each.

# ifconfig

Make a note of the IP address from the output of the above command. The IP address which has to be copied should be under “enp0s8”, as shown in the screenshot below.

Now go to the ‘hosts’ file on both the master and node and add an entry specifying their respective IP addresses along with their names ‘kmaster’ and ‘knode’. This is used for referencing them in the cluster. It should look like the below screenshot on both the machines.

# nano /etc/hosts

Then press ‘Ctrl+X’, then press ‘Y’ and then press ‘Enter’ to Save the file.

### ****Setting Static IP Addresses****

Next, we will make the IP addresses used above, static for the VMs. We can do that by modifying the network interfaces file. Run the following command to open the file:

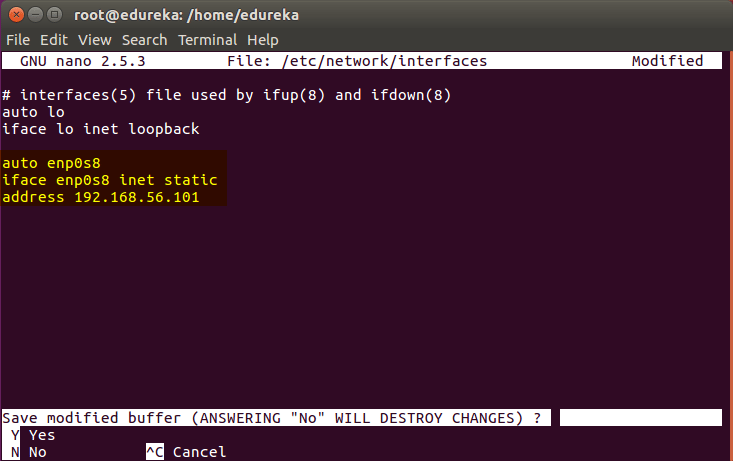
# nano /etc/network/interfaces

Now enter the following lines in the file.

auto enp0s8

iface enp0s8 inet static

address <IP-Address-Of-VM>

It will look something like the below screenshot.

Then press ‘Ctrl+X’, then press ‘Y’ and then press ‘Enter’ to Save the file.

After this, restart your machine(s).

### ****Install OpenSSH-Server****

Now we have to install openshh-server. Run the following command:

# **sudo apt-get install openssh-server**

### ****Install Docker****

Now we have to install Docker because Docker images will be used for managing the containers in the cluster. Run the following commands:

**# sudo su**

**# apt-get update**

**# apt-get install -y docker.io**

Next we have to install these 3 essential components for setting up Kubernetes environment: kubeadm, kubectl, and kubelet.

Run the following commands before installing the Kubernetes environment.

# apt-get update && apt-get install -y apt-transport-https curl

# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list

deb http://apt.kubernetes.io/ kubernetes-xenial main

EOF

# **apt-get update**

## ****Install kubeadm, Kubelet And Kubectl****

Now its time to install the 3 essential components. **Kubelet** is the lowest level component in Kubernetes. It’s responsible for what’s running on an individual machine. ***Kuebadm*** is used for administrating the Kubernetes cluster. ***Kubectl*** is used for controlling the configurations on various nodes inside the cluster.

# **apt-get install -y kubelet kubeadm kubectl**

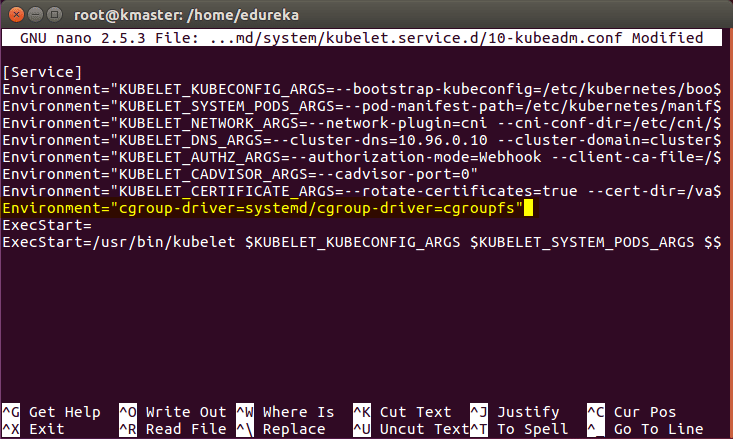
### ****Updating Kubernetes Configuration****

Next, we will change the configuration file of Kubernetes. Run the following command:

# **nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf**

This will open a text editor, enter the following line after the last “Environment Variable”:

Environment=”cgroup-driver=systemd/cgroup-driver=cgroupfs”



Now press Ctrl+X, then press Y, and then press Enter to Save.

***Voila!*** You have successfully installed Kubernetes on both the machines now!

As of now, only the Kubernetes environment has been setup. But now, it is time to install Kubernetes completely, by moving onto the next 2 phases, where we will individually set the configurations in both machines.

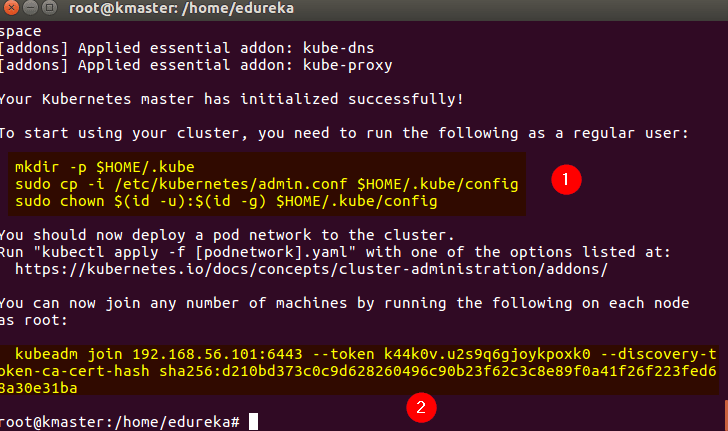
## **Steps Only For Kubernetes Master VM (kmaster)**

***Note***: **These steps will only be executed on the master node (kmaster VM).**

**Step 1**: We will now start our Kubernetes cluster from the master’s machine. Run the following command:

# **kubeadm init --apiserver-advertise-address=<ip-address-of-kmaster-vm> --pod-network-cidr=192.168.0.0/16**

1. You will get the below output. The commands marked as (1), execute them as a non-root user. This will enable you to use kubectl from the CLI
2. The command marked as (2) should also be saved for future. This will be used to join nodes to your cluster

****

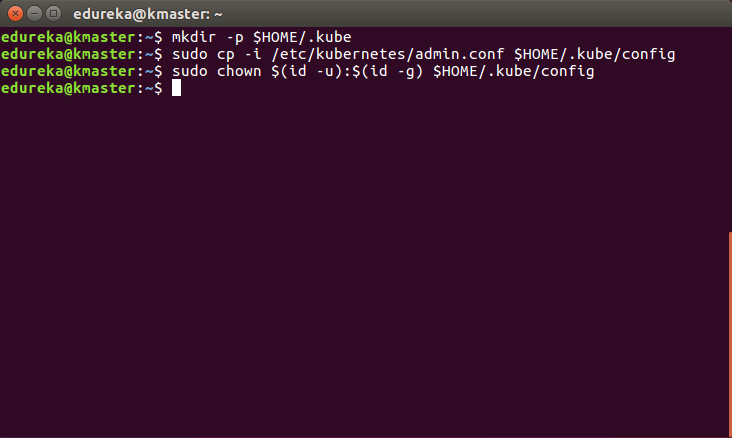
**Step 2**: As mentioned before, run the commands from the above output as a non-root user

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

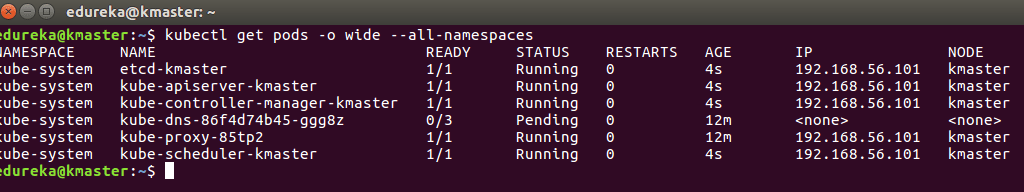
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

It should look like this:



To verify, if kubectl is working or not, run the following command:

$ **kubectl get pods -o wide --all-namespaces**

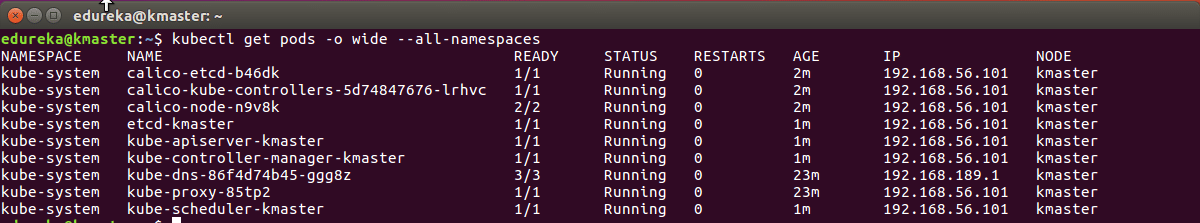
****

Next

**Step 3**: You will notice from the previous command, that all the pods are running except one: ‘kube-dns’. For resolving this we will install a pod network. To install the CALICO pod network, run the following command:

$ **kubectl apply -f https://docs.projectcalico.org/v3.0/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml**

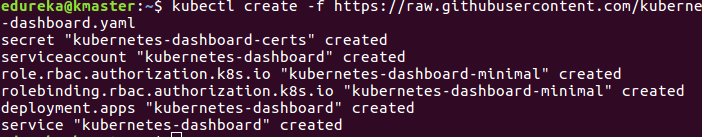
After some time, you will notice that all pods shift to the running state

****

**Step 4**: Next, we will install the dashboard. To install the Dashboard, run the following command:

$**kubectl create -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml**

It will look something like this:

****

**Step 5**: Your dashboard is now ready with it’s the pod in the running state.

****

**Step 6**: By default dashboard will not be visible on the Master VM. Run the following command in the command line:

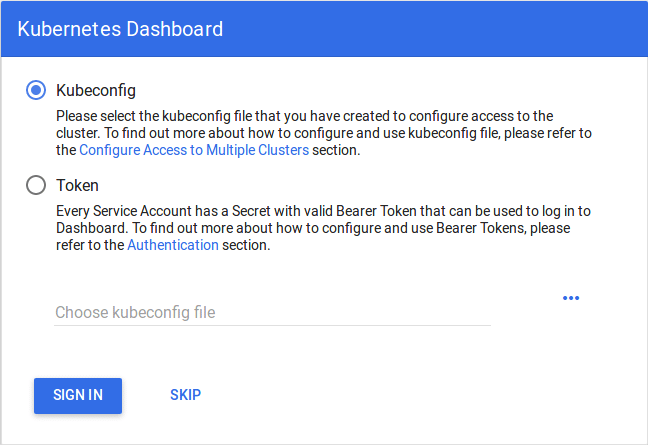
$ **kubectl proxy**

Then you will get something like this:

kubectl proxy - install kubernetes - edureka

To view the dashboard in the browser, navigate to the following address in the browser of your Master VM: **http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/**

You will then be prompted with this page, to enter the credentials:

****

**Step 7**: In this step, we will create the service account for the dashboard and get it’s credentials.  
**Note**: Run all these commands in a new terminal, or your kubectl proxy command will stop.

Run the following commands:

1. This command will create a service account for dashboard in the default namespace

**$ kubectl create serviceaccount dashboard -n default**

2. This command will add the cluster binding rules to your dashboard account

**$ kubectl create clusterrolebinding dashboard-admin -n default**

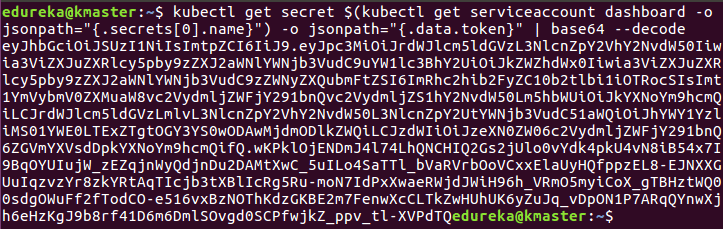
**--clusterrole=cluster-admin**

**--serviceaccount=default:dashboard**

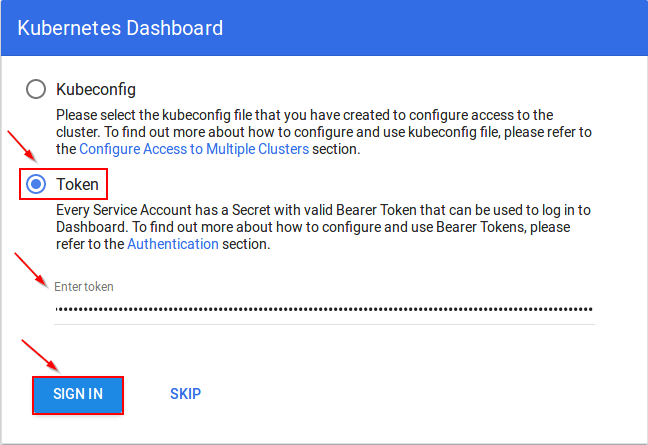
3. This command will give you the token required for your dashboard login

$ **kubectl get secret $(kubectl get serviceaccount dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 --decode**

You should get the token like this:



4. Copy this token and paste it in Dashboard Login Page, by selecting token option



5. You have successfully logged into your dashboard!

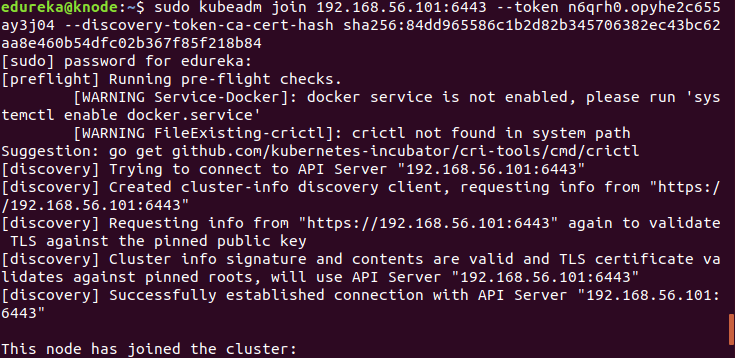
****

## **Steps For Only Kubernetes Node VM (knode)**

It is time to get your node, to join the cluster! This is probably the only step that you will be doing on the node, after installing kubernetes on it.

Run the join command that you saved, when you ran ‘kubeadm init’ command on the master.**Note**: Run this command with “sudo”.

**sudo kubeadm join --apiserver-advertise-address=<ip-address-of-the master> --pod-network-cidr=192.168.0.0/16**

******

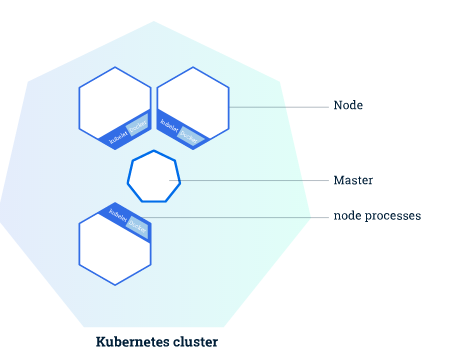
**Using kubectl to Create a Deployment :-**

### Kubernetes Clusters

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit. The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized. Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way. Kubernetes is an open-source platform and is production-ready.

A Kubernetes cluster consists of two types of resources:

* The **Master**coordinates the cluster
* **Nodes** are the workers that run applications



The following are typical use cases for Deployments:

* [Create a Deployment to rollout a ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#creating-a-deployment). The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.
* [Declare the new state of the Pods](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#updating-a-deployment) by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment.
* [Rollback to an earlier Deployment revision](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#rolling-back-a-deployment) if the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.
* [Scale up the Deployment to facilitate more load](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#scaling-a-deployment).
* [Pause the Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#pausing-and-resuming-a-deployment) to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.
* [Use the status of the Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#deployment-status) as an indicator that a rollout has stuck.
* [Clean up older ReplicaSets](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#clean-up-policy) that you don’t need anymore.

## Creating a Deployment

The following is an example of a Deployment. It creates a ReplicaSet to bring up three **nginx** Pods:

**controllers/nginx-deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.9

ports:

- containerPort: 80

In this example:

* A Deployment named nginx-deployment is created, indicated by the .metadata.name field.
* The Deployment creates three replicated Pods, indicated by the replicas field.
* The selector field defines how the Deployment finds which Pods to manage. In this case, you simply select a label that is defined in the Pod template (app: nginx). However, more sophisticated selection rules are possible, as long as the Pod template itself satisfies the rule.

The template field contains the following sub-fields:

* The Pods are labeled app: nginxusing the labels field.
* The Pod template’s specification, or .template.spec field, indicates that the Pods run one container, nginx, which runs the nginx [Docker Hub](https://hub.docker.com/) image at version 1.7.9.
* Create one container and name it nginx using the name field.

1 - Create the Deployment by running the following command:

**kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml**

2 - Run **kubectl get deployments** to check if the Deployment was created. If the Deployment is still being created, the output is similar to the following:

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 0/3 0 0 1s

When you inspect the Deployments in your cluster, the following fields are displayed:

* NAME lists the names of the Deployments in the cluster.
* DESIRED displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
* CURRENT displays how many replicas are currently running.
* UP-TO-DATE displays the number of replicas that have been updated to achieve the desired state.
* AVAILABLE displays how many replicas of the application are available to your users.
* AGE displays the amount of time that the application has been running.

Notice how the number of desired replicas is 3 according to **.spec.replicas** field.

3 - To see the Deployment rollout status, run **kubectl rollout status deployment.v1.apps/nginx-deployment**. The output is similar to this:

Waiting **for** rollout to finish: 2 out of 3 new replicas have been updated...

deployment.apps/nginx-deployment successfully rolled out

4 - Run the kubectl get deployments again a few seconds later. The output is similar to this:

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployment 3/3 3 3 18s

Notice that the Deployment has created all three replicas, and all replicas are up-to-date (they contain the latest Pod template) and available.

5 - To see the ReplicaSet (rs) created by the Deployment, run **kubectl get rs**. The output is similar to this:

NAME DESIRED CURRENT READY AGE

nginx-deployment-75675f5897 3 3 3 18s

6 - To see the labels automatically generated for each Pod, run **kubectl get pods --show-labels**. The following output is returned:

NAME READY STATUS RESTARTS AGE LABELS

nginx-deployment-75675f5897-7ci7o 1/1 Running 0 18s app=nginx,pod-template-hash=3123191453

nginx-deployment-75675f5897-kzszj 1/1 Running 0 18s app=nginx,pod-template-hash=3123191453

nginx-deployment-75675f5897-qqcnn 1/1 Running 0 18s app=nginx,pod-template-hash=3123191453

**Updating a Deployment : -**

1 - Let’s update the nginx Pods to use the nginx:1.9.1 image instead of the nginx:1.7.9 image.

**kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1**

or simply use the following command:

**kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1 --record**

The output is similar to this:

deployment.apps/nginx-deployment image updated

Alternatively, you can edit the Deployment and change

**.spec.template.spec.containers[0].image from nginx:1.7.9 to nginx:1.9.1:**

**kubectl edit deployment.v1.apps/nginx-deployment**

The output is similar to this:

deployment.apps/nginx-deployment edited

To see the rollout status, run:

**kubectl rollout status deployment.v1.apps/nginx-deployment**

The output is similar to this:

Waiting for rollout to finish: 2 out of 3 new replicas have been updated...

Or

deployment.apps/nginx-deployment successfully rolled out

* Run **kubectl get rs** to see that the Deployment updated the Pods by creating a new ReplicaSet and scaling it up to 3 replicas, as well as scaling down the old ReplicaSet to 0 replicas.

**kubectl get rs**

The output is similar to this:

NAME DESIRED CURRENT READY AGE

nginx-deployment-1564180365 3 3 3 6s

nginx-deployment-2035384211 0 0 0 36s

* Running get pods should now show only the new Pods:

**kubectl get pods**

The output is similar to this:

NAME READY STATUS RESTARTS AGE

nginx-deployment-1564180365-khku8 1/1 Running 0 14s

nginx-deployment-1564180365-nacti 1/1 Running 0 14s

nginx-deployment-1564180365-z9gth 1/1 Running 0 14s

**Get details of your Deployment:**

**kubectl describe deployments**

Name: nginx-deployment

Namespace: default

CreationTimestamp: Thu, 30 Nov 2017 10:56:25 +0000

Labels: app=nginx

Annotations: deployment.kubernetes.io/revision=2

Selector: app=nginx

Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable

StrategyType: RollingUpdate

MinReadySeconds: 0

RollingUpdateStrategy: 25% max unavailable, 25% max surge

Pod Template:

Labels: app=nginx

Containers:

nginx:

Image: nginx:1.9.1

Port: 80/TCP

Environment: <none>

Mounts: <none>

Volumes: <none>

Conditions:

Type Status Reason

---- ------ ------

Available True MinimumReplicasAvailable

Progressing True NewReplicaSetAvailable

OldReplicaSets: <none>

NewReplicaSet: nginx-deployment-1564180365 (3/3 replicas created)

**Rolling Back a Deployment :-**

Suppose that you made a typo while updating the Deployment, by putting the image name as nginx:1.91 instead of nginx:1.9.1:

kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.91 --record=true

* The output is similar to this:
* deployment.apps/nginx-deployment image updated
* The rollout gets stuck. You can verify it by checking the rollout status:

kubectl rollout status deployment.v1.apps/nginx-deployment

The output is similar to this:

Waiting for rollout to finish: 1 out of 3 new replicas have been updated...

**kubectl get rs**

The output is similar to this:

NAME DESIRED CURRENT READY AGE

nginx-deployment-1564180365 3 3 3 25s

nginx-deployment-2035384211 0 0 0 36s

nginx-deployment-3066724191 1 1 0 6s

Looking at the Pods created, you see that 1 Pod created by new ReplicaSet is stuck in an image pull loop.

**kubectl get pods**

The output is similar to this:

NAME READY STATUS RESTARTS AGE

nginx-deployment-1564180365-70iae 1/1 Running 0 25s

nginx-deployment-1564180365-jbqqo 1/1 Running 0 25s

nginx-deployment-1564180365-hysrc 1/1 Running 0 25s

nginx-deployment-3066724191-08mng 0/1 ImagePullBackOff 0 6s

**Note:** The Deployment controller stops the bad rollout automatically, and stops scaling up the new ReplicaSet. This depends on the rollingUpdate parameters (maxUnavailable specifically) that you have specified. Kubernetes by default sets the value to 25%.

**Rolling Back to a Previous Revision –**

kubectl rollout undo deployment.v1.apps/nginx-deployment

**Scaling a Deployment :-**

You can scale a Deployment by using the following command:

**kubectl scale deployment.v1.apps/nginx-deployment --replicas=10**

The output is similar to this:

**deployment.apps/nginx-deployment scaled**

Assuming [horizontal Pod autoscaling](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/) is enabled in your cluster, you can setup an autoscaler for your Deployment and choose the minimum and maximum number of Pods you want to run based on the CPU utilization of your existing Pods.

**kubectl autoscale deployment.v1.apps/nginx-deployment --min=10 --max=15 --cpu-percent=80**

**Pausing and Resuming a Deployment :-**

You can pause a Deployment before triggering one or more updates and then resume it. This allows you to apply multiple fixes in between pausing and resuming without triggering unnecessary rollouts.

Pause by running the following command:

**kubectl rollout pause deployment.v1.apps/nginx-deployment**

The output is similar to this:

**deployment.apps/nginx-deployment paused**

Then update the image of the Deployment:

**kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1**

The output is similar to this:

**deployment.apps/nginx-deployment image updated**

Eventually, resume the Deployment and observe a new ReplicaSet coming up with all the new updates:

**kubectl rollout resume deployment.v1.apps/nginx-deployment**

The output is similar to this:

**deployment.apps/nginx-deployment resumed**

Expose the Pod to the public internet using the kubectl expose command:

kubectl expose deployment hello-node --type=LoadBalancer --port=8080

**delete a deployment in Kubernetes-**

1. If you’ve created your deployment from a file, you can use **kubectl delete -f deployment.yaml** to delete your deployment
2. If you’ve created your deployment from the command line, you can use **kubectl delete deployment my-deployment** to delete your deployment

**CREATE EC2 INSTANCES USING ANSIBLE**

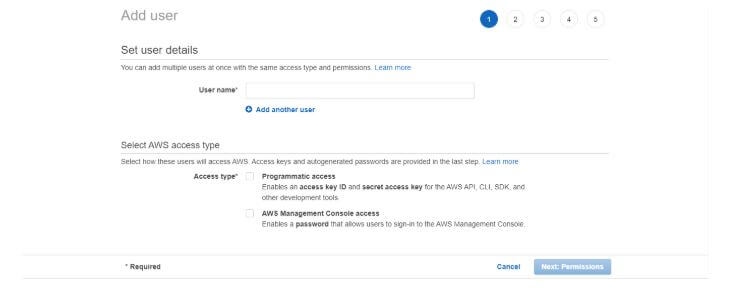
For working on Ansible we need to first set up a few things,

* **AWS user account**
* **Ansible**
* **Python**
* **Boto**

For creating the AWS account just go to the [Amazon AWS](https://aws.amazon.com/account/) server and follow the signup process.

Once the AWS account gets created you need to create the IAM user (As we will need a secret key and secret ID).

Open the [AWS Console](https://console.aws.amazon.com/), search for IAM (Identity and Access Management) and follow these [steps](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html#id_users_create_console) to create a user and take note of the Access Key and Secret Key that will be used by Ansible to set up the instances. (For account access just give Programmatic access as of now.)



Once you are done with the AWS account and the User creation, you can move forward and install the required things.

1. **Ansible:**
   1. Install Ansible on a RHEL/CentOS Linux based system
      1. $ sudo yum install Ansible
   2. Install Ansible on a Debian/Ubuntu Linux based system
      1. $ sudo apt-get install software-properties-common
      2. $ sudo apt-add-repository ppa:Ansible/Ansible
      3. $ sudo apt-get update
      4. $ sudo apt-get install Ansible
   3. Install Ansible using pip
      1. $ sudo pip install Ansible
      2. Once installed you can verify by Ansible –version this command.
2. **Python:**
   1. $ sudo apt-get update
   2. $ sudo apt-get install python3.6
   3. You can follow this [link](https://docs.python-guide.org/starting/install3/linux/) for more details.
3. **Boto:**(Boto is a Python package which provides an interface to AWS.)
   1. First, install pip
      1. $ sudo apt install python3-pip or
      2. $ yum install python-pip
   2. Now install boto
      1. $ pip install boto

Now, we are done with the package installation, we can move ahead and start writing our Ansible playbook.

Now open a terminal and create a file with the extension .yml or .ymal, add below script and save it.

**# Basic provisioning example**

**- name: Ansible test**

**hosts: localhost**

**tasks:**

**- name: launching AWS instance using Ansible**

**ec2:**

**key\_name: aws\_instance\_Ansible**

**instance\_type: t2.micro**

**image: ami-0dacb0c129b49f529**

**region: us-east-2**

**wait: yes**

**group: Ansible**

**count: 1**

**vpc\_subnet\_id: default**

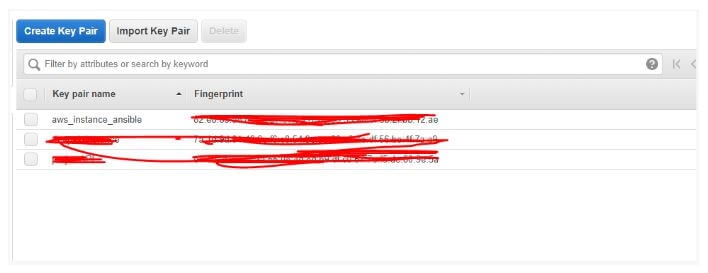
**assign\_public\_ip: yes**

**aws\_access\_key: \*\*\*\*\*\*\*\*\*\*\*xxxxxxxx**

**Aws\_secret\_key: \*\*\*\*\*\*\*\*\*\*\*xxxxxxxx**

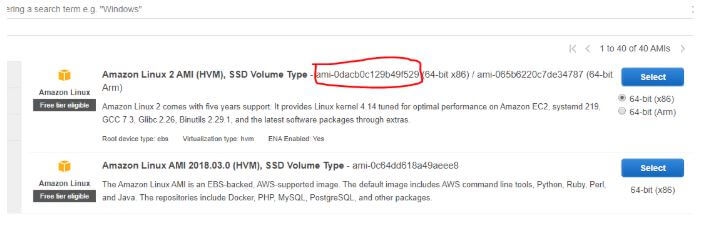
**Hosts:** add [webserver] localhost in /etc/Ansible/hosts file as my internet is running on the local server. If the file does not exist create one at the same location then add.

**Key\_name:** Go to EC2 dashboard -> Key pairs -> Create key pair -> Copy key pair name



**Instance\_type:** You can select the instance type whichever you want to launch. Go to EC2 dashboard -> Launch instance -> Check instance type.

**Image:** Go to EC2 dashboard -> Launch instance -> ami id (Image id)



**Vpc\_subnet\_id:** I made it default as I don’t any VPC configuration.

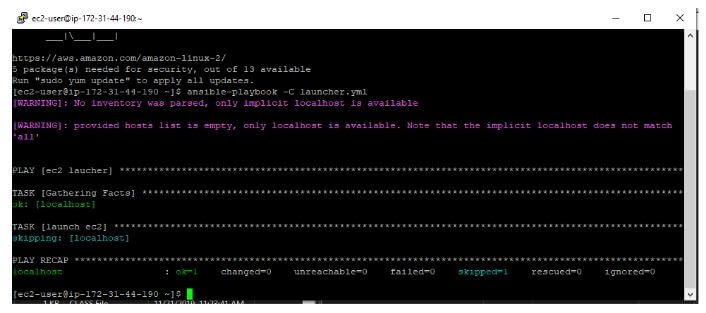
Add your **aws\_access\_key** and **aws\_secret\_key** which you got from IAM user creation. The rest are the basic details. If you want more details you can visit the [Ansible official website](https://docs.ansible.com/ansible/latest/modules/ec2_module.html).

Now our Ansible file is ready.

Run below command to check whether Ansible is ready to launch EC2 or not.

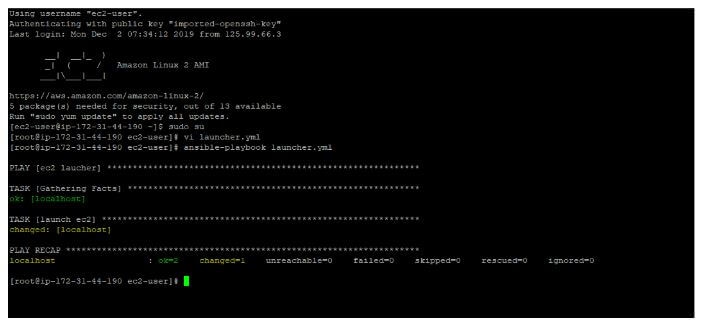
Ansible-playbook -C filename.yml

Where -C will check if everything is ready or not.



Once everything looks good, run below command and within a minute your EC2 server will be launched.

**Ansible-playbook filename.yml**



Now if you go to Amazon console you will see the server is launched successfully.

<https://www.tothenew.com/blog/launching-and-configuring-an-aws-ec2-instance-using-ansible/>

<https://medium.com/datadriveninvestor/devops-using-ansible-to-provision-aws-ec2-instances-3d70a1cb155f>

**Continous Monitoring**

DevOps lifecycle is a continuous loop of several stages, continuous monitoring is the last stage of this loop.

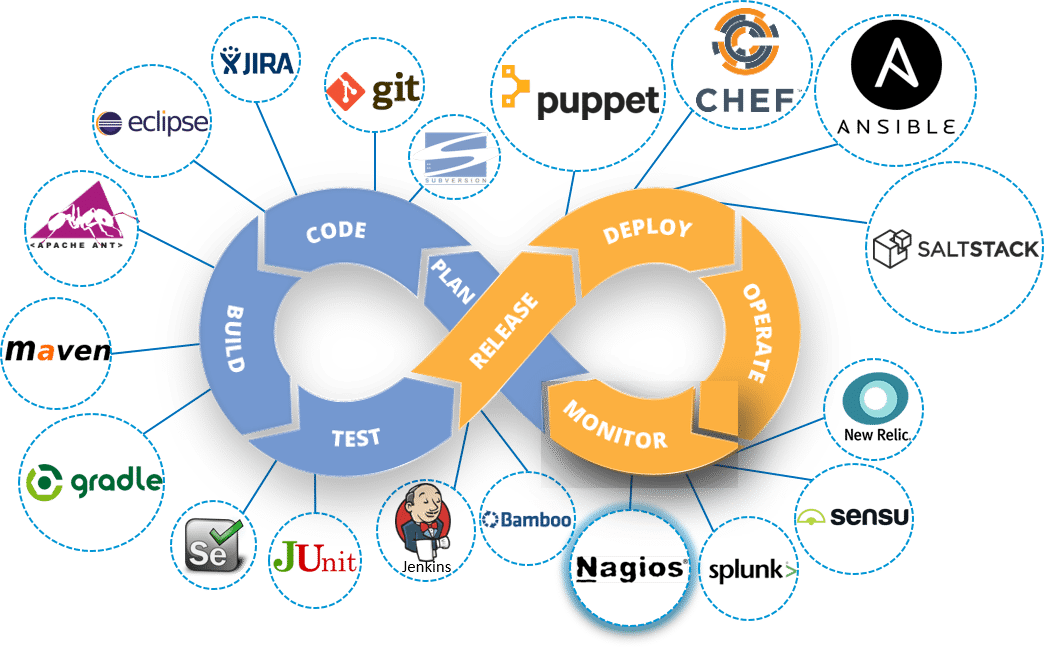
## What is Continuous Monitoring?

Continuous monitoring is a process to detect, report, respond all the attacks which occur in its infrastructure. Once the application is deployed into the server, the role of continuous monitoring comes in to play. The entire process is all about taking care of the company's infrastructure and respond appropriately.

Continuous monitoring starts when the deployment is done on the production servers. From then on, this stage is responsible to monitor everything happening. This stage is very crucial for the business productivity.

Important reasons to use a monitoring tool are:

* It detects all the server and network problems.
* It finds the root cause of the failure.
* It helps in reducing the maintenance cost.
* It helps in troubleshooting the performance issues.
* It helps in updating infrastructure before it gets outdated.
* It can fix problems automatically when detected.
* It makes sure the servers, services, applications, network is always up and running.
* It monitors complete infrastructure every second.



## What is Nagios?

Nagio is a free to use open source software tool for continuous monitoring. It helps you to monitor system, network, and infrastructure. It is used for continuous monitoring of systems, applications, service and business process in a DevOps culture.

Nagios runs plugins stored on the same server. It plugin's connects with a host or another server on your network or the Internet. Therefore, in the case of failure Nagios core can alert the technical staff about the issues. So that, your technical team performs the recovery process before outage in the business processes.

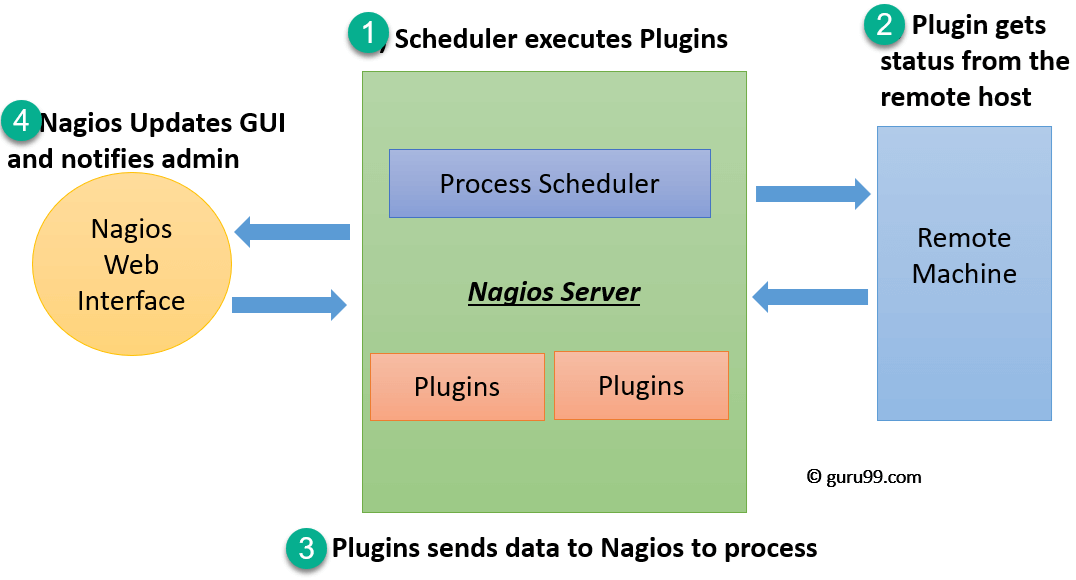
## Why We Need Nagios?

Here, are Important reasons to use Nagios monitoring tool are:

* Detects all types of network or server issues
* Helps you to find the root cause of the problem which allows you to get the permanent solution to the problem
* Active monitoring of your entire infrastructure and business processes
* Allows you to monitors and troubleshoot server performance issues
* Helps you to plan for infrastructure upgrades before outdated systems create failures
* You can maintain the security and availability of the service
* Automatically fix problems in a panic situation

## ****Nagios Architecture:****

* Nagios is built on a server/agents architecture.
* Usually, on a network, a Nagios server is running on a host, and Plugins interact with local and all the remote hosts that need to be monitored.
* These plugins will send information to the Scheduler, which displays that in a GUI.



1. The scheduler is a component of server part of Nagios. It sends a signal to execute the plugins at the remote host.
2. The plugin gets the status from the remote host
3. The plugin sends the data to the process scheduler
4. The process scheduler updates the GUI and notifications are sent to admins

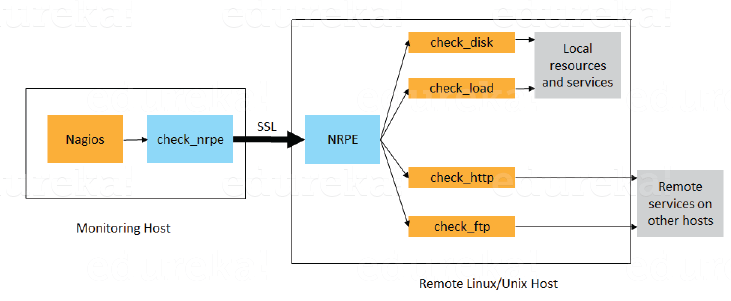
**The following points are worth notable about Nagios architecture −**

* Nagios has server-agent architecture.
* Nagios server is installed on the host and plugins are installed on the remote hosts/servers which are to be monitored.
* Nagios sends a signal through a process scheduler to run the plugins on the local/remote hosts/servers.
* Plugins collect the data (CPU usage, memory usage etc.) and sends it back to the scheduler.
* Then the process schedules send the notifications to the admin/s and updates Nagios GUI.

-----------------------------------------------------------------------------------------------------------------------------

The NRPE addon is designed to allow you to execute Nagios plugins on remote Linux/Unix machines. The main reason for doing this is to allow Nagios to monitor “local” resources (like CPU load, memory usage, etc.) on remote machines. Since these public resources are not usually exposed to external machines, an agent like NRPE must be installed on the remote Linux/Unix machines.

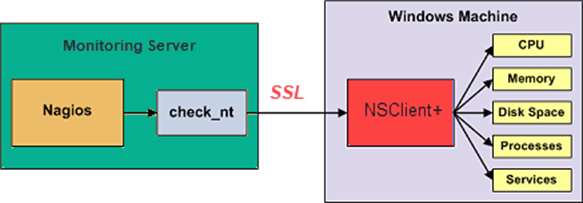
Consider the diagram below:



* The check\_nrpe plugin, resides on the local monitoring machine.
* The NRPE daemon, runs on the remote Linux/Unix machine.
* There is a SSL (Secure Socket Layer) connection between monitoring host and remote host as shown in the diagram above.

**Plugins:**

* Nagios plugins provide low-level intelligence on how to monitor anything and everything with Nagios Core. Plugins operate acts as a standalone application, but they are designed to be executed by Nagios Core. It connects to Apache that is controlled by CGI to display the result. Moreover, a database connected to Nagios to keep a log file.
* How do plugins work?



Consider the above example-

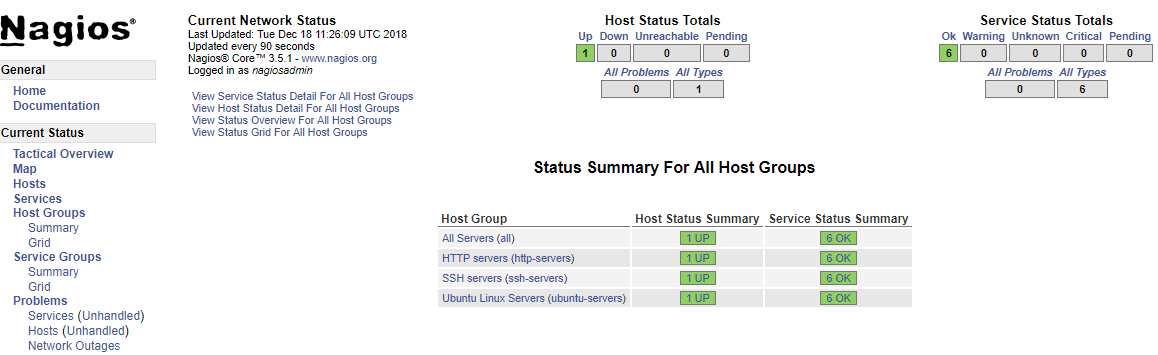
* Check\_nt is a plugin to monitor a windows machine which is mostly available in the monitoring server
* NSClinet++ should be installed in every Windows machine that you wants to monitor
* There is an SSL connection between the server and the host which continuously exchange information with each other

Likewise, NRPE(Nagios Remote plug-in Executor) and NSCA plugins are used to monitor Linux and Mac OS X respectively.

**GUI:**

An interface of Nagios is used to display in web pages generated by CGI. It can be buttons to green or red, sound, graph, etc.

When the soft alert is raised many times, a hard alert is raised, then the Nagios server sends a notification to the administrator.



**Nagios – Installation**

Before you install Nagios, some packages such as Apache, PHP, building packages etc., are required to be present on your Ubuntu system. Hence, let us install them first.

**Step 1** − Run the following command to install pre-required packages −

sudo apt-get install wget build-essential apache2 php apache2-mod-php7.0 php-gd

libgd-dev sendmail unzip

**Step 2** − Next, create user and group for Nagios and add them to Apache www-data user.

sudo useradd nagios

sudo groupadd nagcmd

sudo usermod -a -G nagcmd nagios

sudo usermod -a -G nagios,nagcmd www-data

**Step 3** − Download the latest Nagios package.

wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-

4.4.3.tar.gz

**Step 4** − Extract the tarball file.

tar -xzf nagios-4.4.3.tar.gz

cd nagios-4.4.3/

**Step 5** − Run the following command to compile Nagios from source.

./configure --with-nagios-group=nagios --with-command-group=nagcmd

**Step 6** − Run the following command to build Nagios files.

make all

**Step 7** − Run the command shown below to install all the Nagios files.

sudo make install

**Step 8** − Run the following commands to install init and external command configuration files.

sudo make install-commandmode

sudo make install-init

sudo make install-config

sudo /usr/bin/install -c -m 644 sample-config/httpd.conf /etc/apache2/sitesavailable/

nagios.conf

**Step 9** − Now copy the event handler directory to Nagios directory.

sudo cp -R contrib/eventhandlers/ /usr/local/nagios/libexec/

sudo chown -R nagios:nagios /usr/local/nagios/libexec/eventhandlers

**Step 10** − Download and extract Nagios plugins.

cd

wget https://nagios-plugins.org/download/nagiosplugins-

2.2.1.tar.gz

tar -xzf nagios-plugins\*.tar.gz

cd nagios-plugins-2.2.1/

**Step 11** − Install Nagios plugins using the below command.

./configure --with-nagios-user=nagios --with-nagios-group=nagios --with-openssl

make

sudo make install

**Step 12** − Now edit the Nagios configuration file and uncomment line number 51 → cfg\_dir=/usr/local/nagios/etc/servers

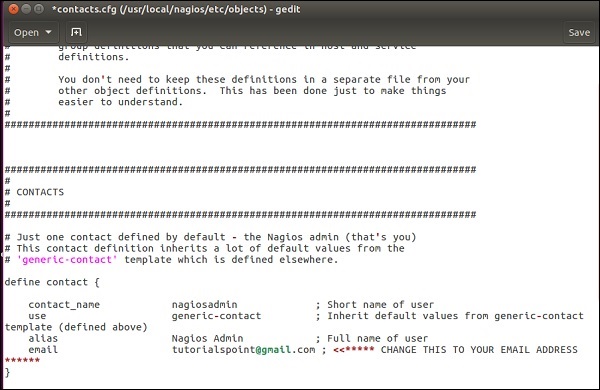
sudo gedit /usr/local/nagios/etc/nagios.cfg

**Step 13** − Now, create a server directory.

sudo mkdir -p /usr/local/nagios/etc/servers

**Step 14** − Edit contacts configuration file.

sudo gedit /usr/local/nagios/etc/objects/contacts.cfg



**Step 15** − Now enable the Apache modules and configure a user nagiosadmin.

sudo a2enmod rewrite

sudo a2enmod cgi

sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

sudo ln -s /etc/apache2/sites-available/nagios.conf /etc/apache2/sites-enabled/

**Step 16** − Now, restart Apache and Nagios.

service apache2 restart

service nagios start

cd /etc/init.d/

sudo cp /etc/init.d/skeleton /etc/init.d/Nagios

**Step 17** − Edit the Nagios file.

sudo gedit /etc/init.d/Nagios

DESC = "Nagios"

NAME = nagios

DAEMON = /usr/local/nagios/bin/$NAME

DAEMON\_ARGS = "-d /usr/local/nagios/etc/nagios.cfg"

PIDFILE = /usr/local/nagios/var/$NAME.lock

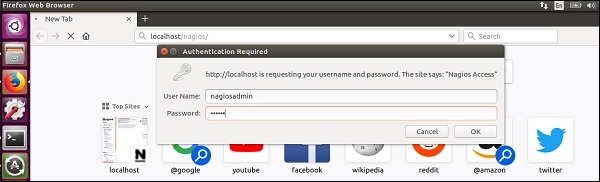
**Step 18** − Make the Nagios file executable and start Nagios.

sudo chmod +x /etc/init.d/nagios

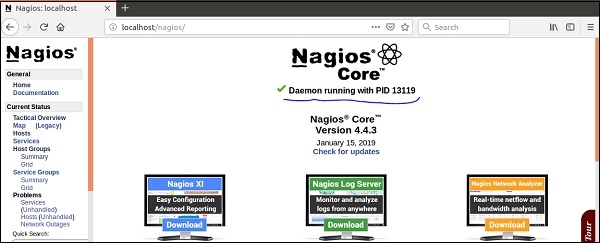
service apache2 restart

service nagios start

**Step 19** − Now go to your browser and open url → **http://localhost/nagios**. Now login to Nagios with username nagiosadmin and use the password which you had set earlier. The login screen of Nagios is as shown in the screenshot given below −

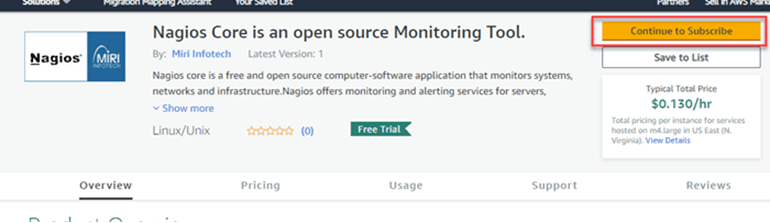


If you have followed all the steps correctly, you Nagios web interface will show up. You can find the Nagios dashboard as shown below −

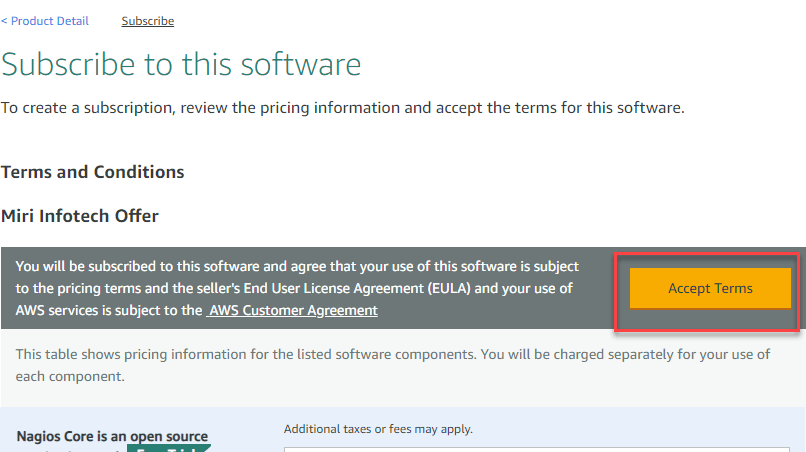


## Install Nagios at AWS

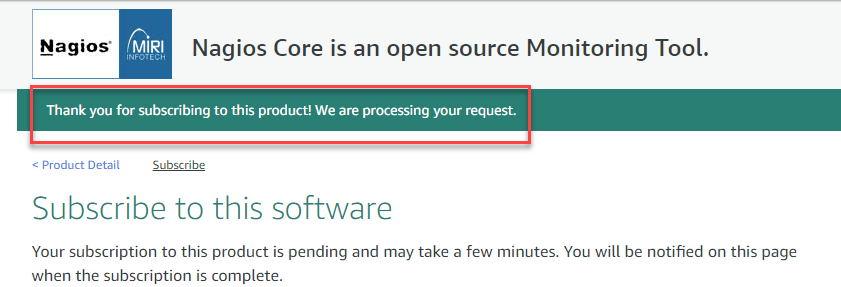
**Step 1)**Got to <https://aws.amazon.com/marketplace/pp/B0773T3529> and click Continue to Subscribe

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor4.png)

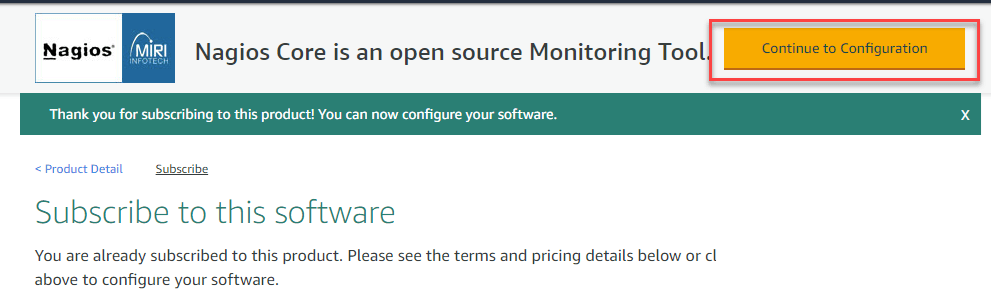
**Step 2)**Accept Terms

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor5.png)

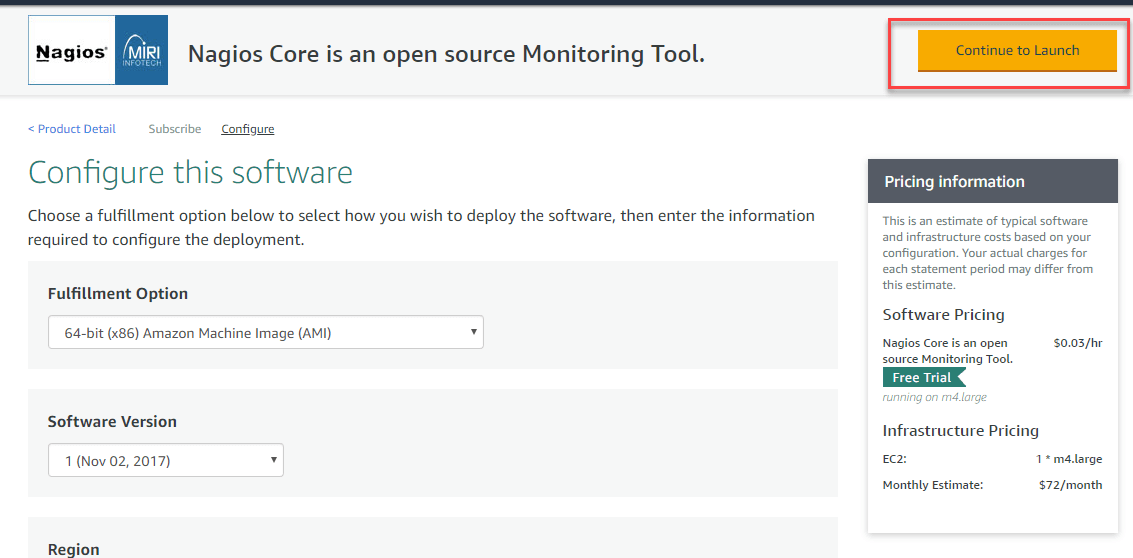
**Step 3)**You will see subscription pending message

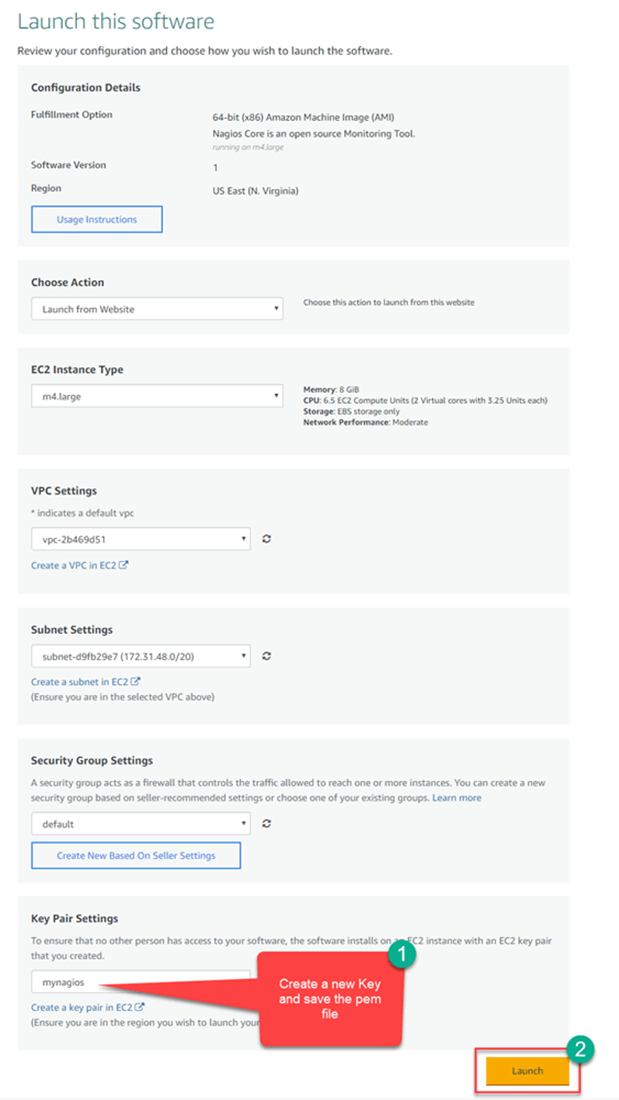
[](https://www.guru99.com/images/1/122118_0620_NagiosTutor6.png)

**Step 4)**Refresh the same page after a few minutes and click "Continue to Configuration

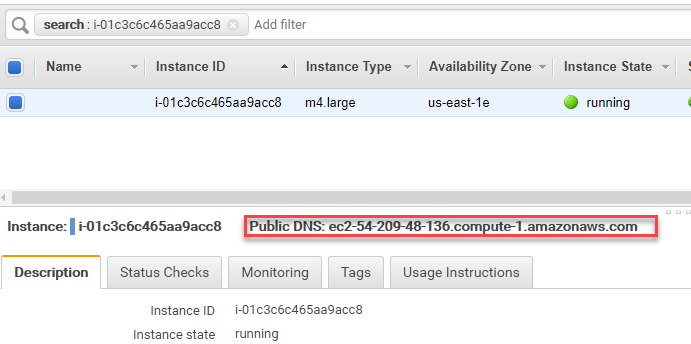
[](https://www.guru99.com/images/1/122118_0620_NagiosTutor7.png)

**Step 5)**Keep the settings default and click Continue to Launch

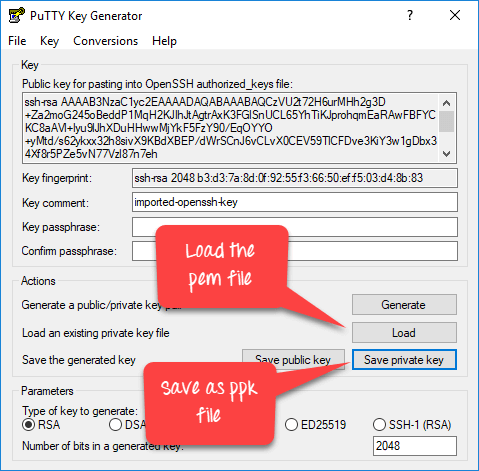
[](https://www.guru99.com/images/1/122118_0620_NagiosTutor8.png)

**Step 6)**Review the settings. Create a new Key and click launch[](https://www.guru99.com/images/1/122118_0620_NagiosTutor9.png)

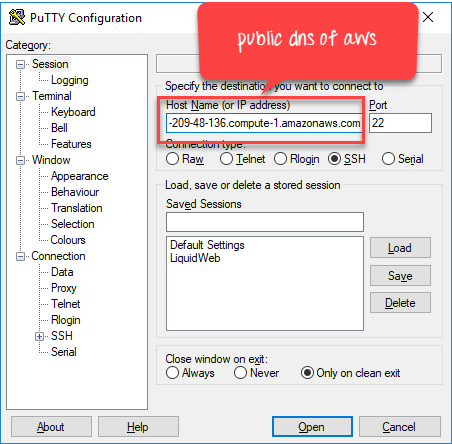
**Step 7)**Note the public DNS of your instance

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor10.png)

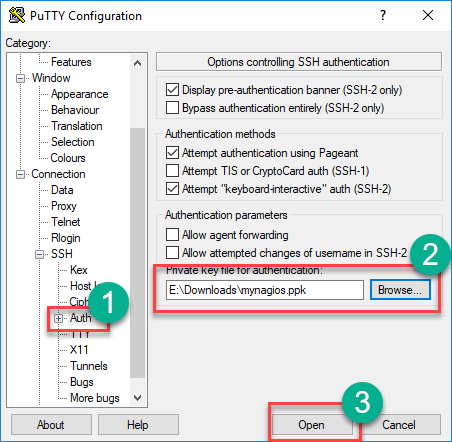
**Step 8)**In your windows machine, use the tool putty generator to convert pem file to ppk

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor11.png)

**Step 9)**In putty, enter the public DNS

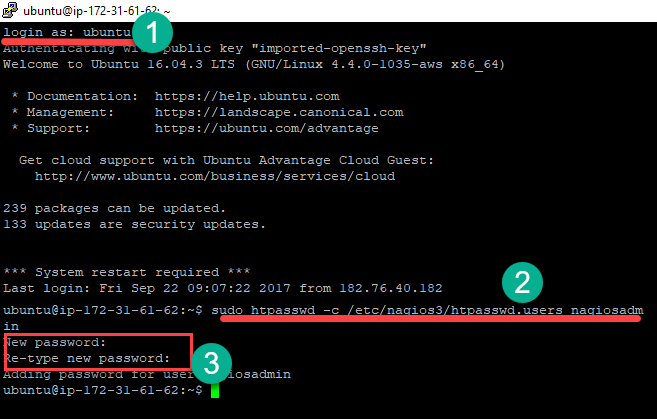
[](https://www.guru99.com/images/1/122118_0620_NagiosTutor12.png)

**Step 10)**In Auth section, enter the ppk key and click open

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor13.png)

**Step 11)**In terminal,

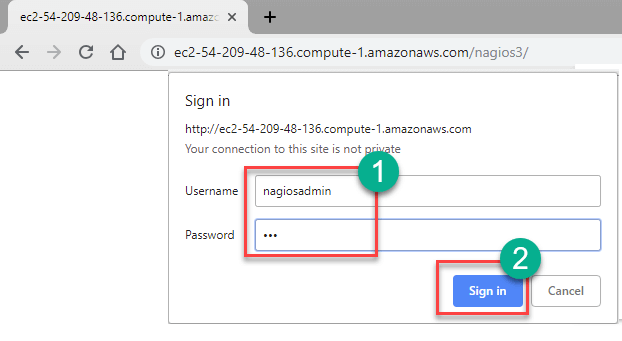
1. **Enter login name as ubuntu**
2. **Run this command**sudo htpasswd -c /etc/nagios3/htpasswd.users nagiosadmin
3. **Enter a new password of your choice**

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor14.png)

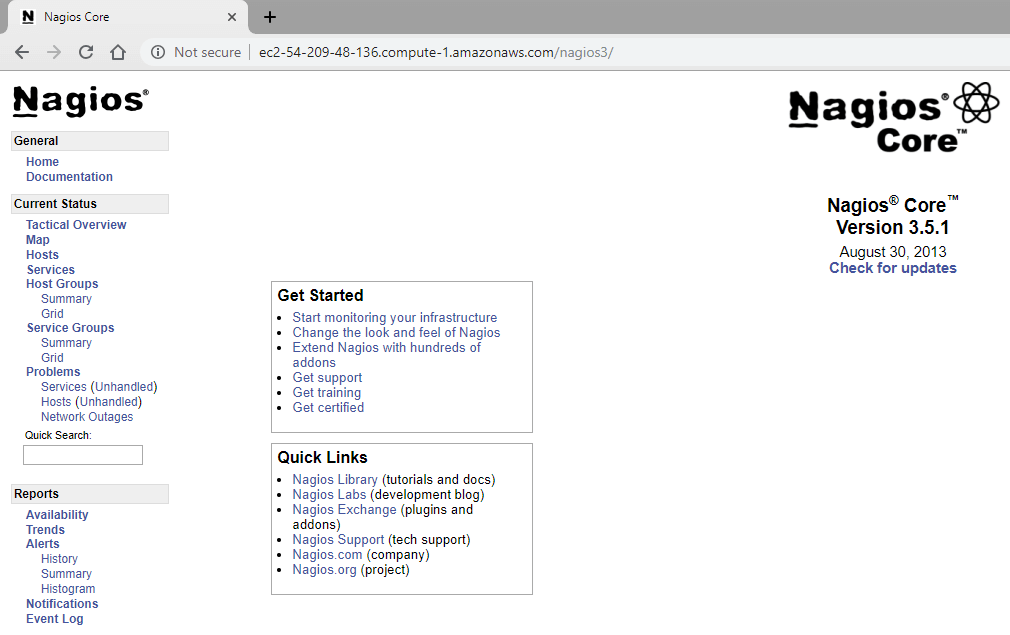
**Step 12) In your browser, Go to location http://<Public DNS>/nagios3 in my case**[**http://ec2-54-209-48-136.compute-1.amazonaws.com/nagios3/**](http://ec2-54-209-48-136.compute-1.amazonaws.com/nagios3/)**.**

**Enter Username: nagiosadmin**

**pass: set in the previous step**

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor15.png)

**Step 13)**Nagios Loads

[](https://www.guru99.com/images/1/122118_0620_NagiosTutor16.png)

Nagios can be applicable to a wide range of applications. They are given here −

* Monitor host resources such as disk space, system logs etc.
* Monitor network resources – http, ftp, smtp, ssh etc.
* Monitor log files continuously to identify infra-issue.
* Monitor windows/linux/unix/web applications and its state.
* Nagios Remote Plugin Executer (NRPE) can monitor services remotely.
* Run service checks in parallel.
* SSH or SSL tunnels can also be used for remote monitoring.
* Send alerts/notifications
* via email, sms, pager of any issue on infrastructure
* Recommending when to upgrade the IT infrastructure.

## Ports

The Default ports used by common Nagios Plugins are as given under −

* Butcheck\_nt (nsclient++) 12489
* NRPE 5666
* NSCA 5667
* NCPA 5693
* MSSQL 1433
* MySQL 3306
* PostgreSQL 5432
* MongoDB 27017, 27018
* OracleDB 1521
* Email (SMTP) 25, 465, 587
* WMI 135, 445 / additionaldynamically-assigned ports in 1024-1034 range

<https://www.edureka.co/blog/nagios-tutorial/>

**What is the Linux kernel?**

The Linux® kernel is the main component of a [Linux operating system](https://www.redhat.com/en/topics/linux/what-is-linux) (OS) and is the core interface between a computer’s hardware and its processes. It communicates between the two, managing resources as efficiently as possible.

The kernel is so named because—like a seed inside a hard shell—it exists within the OS and controls all the major functions of the hardware, whether it’s a phone, laptop, server, or any other kind of computer.

The kernel has 4 jobs:

1. **Memory management:** Keep track of how much memory is used to store what, and where
2. **Process management:** Determine which processes can use the CPU, when, and for how long
3. **Device drivers:** Act as mediator/interpreter between the hardware and processes
4. **System calls and security:** Receive requests for service from the processes

**DEVOPS**