

LINUX & DevOps

BY CAREER FOCUS ACADEMY

PDP_Connectz

+91-9493575893

What is Linux:



Linux

- **Linux is Operating system.**
- **OS is system software that directly runs on physical machines.**

Linux Features

- Most widely used operating system for servers.
- Open-source Operating system or Free os.
- 32/64 Bit support
- Multiuser, Multi-task
- Can Support Multiple Hardware platforms
- Source code os open
- OS Customization is Possible
- X Windows is Gui

Windows VS Linux

Windows	Linux
License based	Free
Secured	More secured
Downtime required	No Downtime required
Virus effect	No Virus effect
Preferred for desktops	Preferred for servers
Not open	Source code is Open
No OS Customization	Os Customization

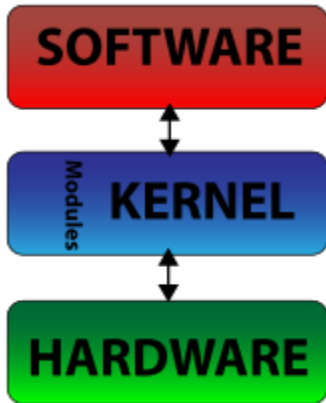
Linux Distributions

- Distribution is a set of packages that build OS
- Various Distributions are available
 - Redhat
 - Centos
 - Debian
 - Mandriva
 - Slackware
 - SuSE
 - Caldera

- Ubuntu etc..
- Fedora

KERNEL

- Core component of Operating System
- Interface between Software and Hardware
- Responsible to manage system resources (CPU, Memory Etc..)

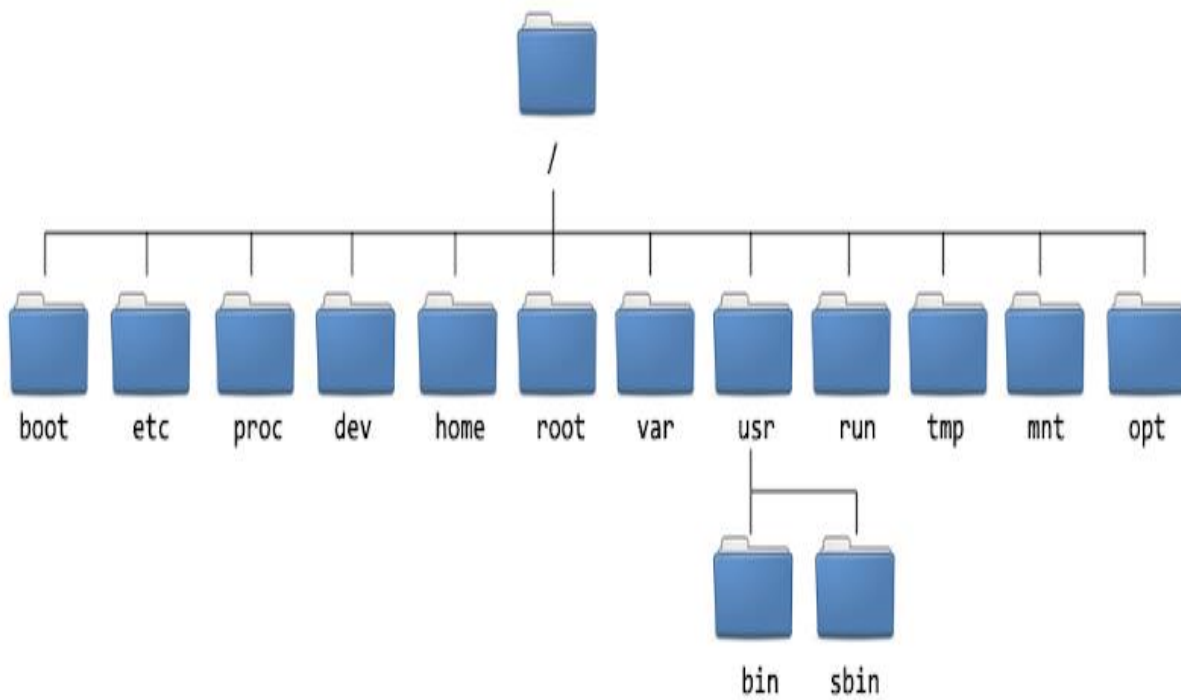


Shell

- Shell is a command interpreter that execute commands. Read from the standard i/p device (keyboard)
- Responsible for finding commands and execution
- Several Shells are available, Bash is popular.

Linux File system

- Linux has Hierarchical File System Structure
- All directories, files, devices etc reside under “ / “
- Linux OS creates some default directories under /.



/bin: This directory holds the commands. Those commands used by all users.

/sbin: This directory holds the commands. These commands using by Super user/root/
- It contains commands. Which can be by super user.

/boot: It contains programs which are used to boot the system.

/proc: It contains the system processors information.

/usr:(unix system resources)

- It contains system libraries and Man Pages

/var:(Variable)

- It contains system logs (Messages)

/lib: (Libraries)

- It contains system libraries which accessed dynamically.

/etc: It contains the configuration files

/opt: (Optional) Here We can store the third party software.

/dev: The directory contains the special device files. The device files are created during installation

/home: It can used as Home directory for Normal Users.

/root: Home Directory for Root.

/mnt: It can be used as Mount point for the removable storages.

Basic Commands:

logname: It displays logged-in user name

logname

- O/P:

```
[root@localhost ~]# logname  
root
```

Whoami:

Who executing the commands or operating

pwd: It displays current working directory

pwd

- O/P :

```
[root@localhost Desktop]# pwd  
/root/Desktop
```

echo \$SHELL: It contains default shell of system. (in shell default shell is bash)

- O/P:

```
[root@localhost Desktop]# echo $shell  
[root@localhost Desktop]# echo $SHELL  
/bin/bash
```

echo \$0 : It contains current shell of system. (we have different shells like cshell, kshell so it will display the current shell name)

- O/P :

```
[root@localhost Desktop]# echo $0  
-bash
```

I.Q : Difference between echo \$SHELL and echo \$0

Clear: To clear the screen. (It will only clear the screen what you are able to see in your screen, but history will save in top of the screen)

O/P:

```
[root@localhost Desktop]#
```

uname -a: It displays the system information.

O/P :

```
[root@localhost Desktop]# uname -a  
Linux localhost.localdomain 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC  
2013 x86_64 x86_64 x86_64 GNU/Linux  
[root@localhost Desktop]#
```

uname: It displays only "OS" in the system or Machine.

- O/P :

```
[root@localhost Desktop]# uname  
Linux
```

uname -n or hostname: It displays hostname of system

O/P:

```
[root@localhost Desktop]# uname -n
localhost.localdomain
[root@localhost Desktop]# hostname
localhost.localdomain
[root@localhost Desktop]#
```

uname -i: To Display the machine architecture.

O/P:

```
[root@localhost Desktop]# uname -i
x86_64
```

who -b: It display last booted time.

O/P:

```
[root@localhost Desktop]# who -b
system boot 2020-02-11 02:13
```

uptime: It display the system running time.

- O/P :

```
[root@localhost Desktop]# uptime
03:04:57 up 52 min, 3 users, load average: 0.00, 0.02, 0.07
```

ls: List of files and directories in current directory

O/P:

```
[root@localhost Desktop]# ls
apache-maven-3.6.3      apache-tomcat-8.0.32  aswi  file2  index2.html  jdk-8u231-linux-x64.tar.gz  myGit  prct2
apache-maven-3.6.3-bin.tar.gz  apache-tomcat-8.0.32.tar.gz  demo_1  images  jdk1.8.0_231  jyofile  practicer  test.txt
```

ll or ls -l: Long List of files and directories in current directory

O/P:

```
[root@localhost Desktop]# ll
total 207888
drwxr-xr-x. 6 root root    4096 Dec 17 19:23 apache-maven-3.6.3
-rw-r--r--. 1 root root 9506321 Nov 19 13:50 apache-maven-3.6.3-bin.
drwxr-xr-x. 9 root root    4096 Dec 19 00:03 apache-tomcat-8.0.32
-rw-r--r--. 1 root root 9169108 Dec 17 18:56 apache-tomcat-8.0.32.ta
-rw-r--r--. 1 root root      0 Dec 19 21:50 aswi
drwxr-xr-x. 3 root root    4096 Dec 20 02:59 demo_1
-rw-r--r--. 1 root root      0 Feb  7 01:54 file2
drwxr-xr-x. 2 root root    4096 Feb  7 02:09 images
-rw-r--r--. 1 root root    39 Feb  7 02:06 index2.html
drwxr-xr-x. 7 uucp 143    4096 Oct  5 03:13 jdk1.8.0_231
-rw-r--r--. 1 root root 194151339 Dec 13 19:09 jdk-8u231-linux-x64.tar
-rw-r--r--. 1 root root    17 Dec 19 22:04 jyofile
drwxr-xr-x. 3 root root    4096 Dec 16 19:10 myGit
drwxr-xr-x. 2 root root    4096 Dec 19 21:54 practicer
drwxr-xr-x. 2 root root    4096 Dec 19 21:55 prct2
-rw-r--r--. 1 root root    30 Dec 20 02:36 test.txt
```

ls -a: It will display the all files in pwd (Including hidden files)

O/P:

```
sbacpe-walaw-3'e'3 sbacpe-romcsc-8'0'35'fex'ds qemo'j tw9dea jdx-8u231-jtunx-xe4'fex'ds bxcscfcesj 'romcsc-naeka'xwjt'amb
.. sbacpe-romcsc-8'0'35 'samtuj 'dte jdxj'8'0'33j wAetf cear'fxf
sbacpe-walaw-3'e'3-ptu'fex'ds samj tTteS twqexS'pawj jlofTte bxcrg
[root@localhost Desktop]# ls -a
```

ls -r : Display the files in reverse order

O/P:

```
[root@localhost Desktop]# ls -lr
test.txt  practicer  jyofile      jdk1.8.0_231  images  demo_1  apache-tomcat-8.0.32.tar.gz  apache-maven-3.6.3-bin.tar.
prct2     myGit       jdk-8u231-linux-x64.tar.gz  index2.html  file2    aswi     apache-tomcat-8.0.32        apache-maven-3.6.3
```

ls -i : It will display the inode numbers of files or dirs.

O/P:

```
[root@localhost Desktop]# ls -i
525906 apache-maven-3.6.3          271806 apache-tomcat-8.0.32.tar.gz  269934 file2          269949 jdk1.8.0_231          271785 myGit          271886 test.t
271874 apache-maven-3.6.3-bin.tar.gz 271888 aswi          526084 images          269943 jdk-8u231-linux-x64.tar.gz 271894 practicer
271808 apache-tomcat-8.0.32          271885 demo_1          271907 index2.html  271891 jyofile          391610 prct2
```

- inode is an entry in inode table. It contains the information like meta data about regular file & dirs

ls -s : To display the files sizes

O/P:

```
[root@localhost Desktop]# ls -s
total 207888
  4 apache-maven-3.6.3          4 index2.html
9284 apache-maven-3.6.3-bin.tar.gz  4 jdk1.8.0_231
  4 apache-tomcat-8.0.32      189604 jdk-8u231-linux-x64.tar.gz
8956 apache-tomcat-8.0.32.tar.gz  4 jyofile
  0 aswi          4 myGit
  4 demo_1          4 practicer
  0 file2          4 prct2
  4 images          4 test.txt
```

ls -????: Display the file name only four characters in name

O/P:

```
[root@localhost Desktop]# ls
apache-maven-3.6.3      aswi      index2.html      myGit
apache-maven-3.6.3-bin.tar.gz  demo_1    jdk1.8.0_231     practice
apache-tomcat-8.0.32    file2     jdk-8u231-linux-x64.tar.gz  prct2
apache-tomcat-8.0.32.tar.gz  images    jyofile          test.txt
[root@localhost Desktop]# ls - ?????
ls: cannot access -: No such file or directory
file2

myGit:

prct2:
```

ls -d ?????: Display the dir name only four characters in name

O/P:

```
[root@localhost Desktop]# ls
apache-maven-3.6.3      aswi      index2.html      myGit
apache-maven-3.6.3-bin.tar.gz  demo_1    jdk1.8.0_231     practicer
apache-tomcat-8.0.32    file2     jdk-8u231-linux-x64.tar.gz  prct2
apache-tomcat-8.0.32.tar.gz  images    jyofile          test.txt
You have new mail in /var/spool/mail/root
[root@localhost Desktop]# ls -d ???
ls: cannot access ???: No such file or directory
[root@localhost Desktop]# ls -d ????
aswi
```

ls -ltr: IT Will display the list of files in long format along with time and reverse format.

O/P:


```
[root@localhost Desktop]# ls -ltr
total 207888
drwxr-xr-x. 7 uucp 143      4096 Oct  5 03:13 jdk1.8.0_231
-rw-r--r--. 1 root root 9506321 Nov 19 13:50 apache-maven-3.6.3-bin.tar.gz
-rw-r--r--. 1 root root 194151339 Dec 13 19:09 jdk-8u231-linux-x64.tar.gz
drwxr-xr-x. 3 root root      4096 Dec 16 19:10 myGit
-rw-r--r--. 1 root root 9169108 Dec 17 18:56 apache-tomcat-8.0.32.tar.gz
drwxr-xr-x. 6 root root      4096 Dec 17 19:23 apache-maven-3.6.3
drwxr-xr-x. 9 root root      4096 Dec 19 00:03 apache-tomcat-8.0.32
-rw-r--r--. 1 root root      0 Dec 19 21:50 aswi
drwxr-xr-x. 2 root root      4096 Dec 19 21:54 practicer
drwxr-xr-x. 2 root root      4096 Dec 19 21:55 prct2
-rw-r--r--. 1 root root      17 Dec 19 22:04 jyofile
-rw-r--r--. 1 root root      30 Dec 20 02:36 test.txt
drwxr-xr-x. 3 root root      4096 Dec 20 02:59 demo_1
-rw-r--r--. 1 root root      0 Feb  7 01:54 file2
-rw-r--r--. 1 root root      39 Feb  7 02:06 index2.html
drwxr-xr-x. 2 root root      4096 Feb  7 02:09 images
```

Working with files

In linux we have different categories of files

- i. Regular files
- ii. Directories

- **File type:** regular file, directory, pipe etc.
- **Permissions to that file:** read, write, execute
- **Size of file**
- **Time stamp**

stat<filename>

You can display the inode data on a file or directory by using stat command

Cat (concatenate)

By using cat we can create files. We can append the data to existing files and we can list the content of the file.

Create a filename with devops

cat > devops1 (File created)

Welcome to DevOps

Ctrl D (Save the file)

Now View the content of the file

cat devops1 or cat < devops1

```
[root@localhost Desktop]# cat > devops1
Welcome to devops class1
[root@localhost Desktop]# cat devops1
Welcome to devops class1
```

Now Append the data to existing file

cat >> devops1

We are discussing about cat command

Ctrl d

O/P:


```
[root@localhost Desktop]# cat >>devops1
we are discussing about cat command
[root@localhost Desktop]# cat devops1
Welcome to devops class1
we are discussing about cat command
[root@localhost Desktop]#
```

Note: The data to be added at the end of the file

NOTE :if file is already existed cat command override it

FOR EX:

```
[root@localhost Desktop]# cat devops1
Welcome to devops class1
we are discussing about cat command
[root@localhost Desktop]# cat > devops1
hello devops world
[root@localhost Desktop]# cat devops1
hello devops world
```

To Display the data through line number

cat -n devops1

```
[root@localhost Desktop]# cat >> devops1
hii

hello

hw r u

wr r u
[root@localhost Desktop]# cat -n devops1
 1 hello devops world
 2 hii
 3
 4 hello
 5
 6 hw r u
 7
 8
 9 wr r u
```

Shows the content of the files at a time

cat file1 file2

```
[root@localhost Desktop]# cat devops2
hello devops2
we will learn devops here
it is latest technology
[root@localhost Desktop]# cat devops1 devops2
hello devops world
hii

hello

hw r u

wr r u
hello devops2
we will learn devops here
it is latest technology
```

To skip the empty lines in file

cat -b myfile

```
[root@localhost Desktop]# cat -b devops1
 1  hello devops world
 2  hii

 3  hello

 4  hw r u

 5  wr r u
```

NOTE: IF YOU WANT TO KNOW MORE COMMANDS ABOUT CAT RUN THE BELOW COMMAND

man cat

Same like for lscommand also:

#man ls

Touch

Touch command is used to create empty files or create one or more files at time

touch devfile1

```
[root@localhost Desktop]# touch devfile1
[root@localhost Desktop]# ls
apache-maven-3.6.3          devfile1  index2.html          practice1
apache-maven-3.6.3-bin.tar.gz devops    index.html           prct2
apache-tomcat-8.0.32       devops1  jdk1.8.0_231        test.txt
apache-tomcat-8.0.32.tar.gz devops2  jdk-8u231-linux-x64.tar.gz
aswi                       file2    jyofile
demo_1                    images   myGit
[root@localhost Desktop]#
```

or

touch myfile1 myfile2

```
[root@localhost Desktop]# touch devops4 devops5
[root@localhost Desktop]# ls
apache-maven-3.6.3          devfile1  file2                jyofile
apache-maven-3.6.3-bin.tar.gz devops    images               myGit
apache-tomcat-8.0.32       devops1  index2.html          practice
apache-tomcat-8.0.32.tar.gz devops2  index.html           prct2
aswi                       devops4  jdk1.8.0_231        test.txt
demo_1                    devops5  jdk-8u231-linux-x64.tar.gz
```

To create files in the range

touch {1..10} or touch {a..z}

```
[root@localhost Desktop]# touch file{1..10}
[root@localhost Desktop]# ls
apache-maven-3.6.3      {devops6..10}      images
apache-maven-3.6.3-bin.tar.gz {devops6..devops10} index2.html
apache-tomcat-8.0.32    file1               index.html
apache-tomcat-8.0.32.tar.gz file10              jdk1.8.0_231
aswi                    file2               jdk-8u231-linux-x64.tar.gz
demo_1                  file3               jyofile
devfile1                file4               myGit
devops                   file5               practicel
devops1                  file6               prct2
devops2                  file7               test.txt
devops4                  file8
devops5                  file9
```

Note:

If file is created already then it is not override the existing file. It only changes the time stamp of the existing file.

I.Q: DIFFERENCE BETWEEN CAT AND TOUCH COMMAND?

Day-2

Creating directory

mkdir dir1

```
[root@localhost Desktop]# mkdir dir1
[root@localhost Desktop]# ls
apache-maven-3.6.3      {devops6..10}      file9
apache-maven-3.6.3-bin.tar.gz {devops6..devops10} images
apache-tomcat-8.0.32    dir1               index2.html
apache-tomcat-8.0.32.tar.gz file1               index.html
aswi                     file10              jdk1.8.0_231
demo_1                   file2               jdk-8u231-linux-x64.tar.gz
devfile1                 file3               jyofile
devops                    file4               myGit
devops1                   file5               practicel
devops2                   file6               prct2
devops4                   file7               test.txt
devops5                   file8
```

command to create parent directory and sub directory at a time

mkdir -p dir3/images1

```
[root@localhost dir2]# mkdir -p dir3/images1
[root@localhost dir2]# cd dir3
[root@localhost dir3]# ls
images1
```

To create multiple directories at a time

mkdir img3 img4

```
[root@localhost dir1]# mkdir img3 img4
[root@localhost dir1]# ls
img1 img2 img3 img4
```

If you want to create several directories under parent directory

mkdir -p names/ {img1,img2,img3} (with out space)

```
[root@localhost names]# mkdir -p names/{img1,img2,img3}
[root@localhost names]# cd names
[root@localhost names]# ls
img1  img2  img3
```

Copy:

cp stands for copy. *cp* is a Linux shell command to *copy* files and directories.

Syntax:

cp [OPTION] Source Destination

cp file1 file2

Here copying the file1 to file2

To copy the directories

cp -r dir1 dir2

Move

- Move command is used to move the files and directories

- It rename a file or folder

syntax:

mv [options] source dest

mv file1 file2 - Moving the file

mv dir1 dir2 - Moving the directories

rm

Removing the files and directories

rm file or directory (It will ask you permission)

rm -rf file or directory (Forcefully delete the files without asking)

rmdir empty_directory_name (It will delete the empty directory)

cd

Change directory from one location to another location

cd /etc

cd - (change to previous directory)

File permissions:

When we create a file or directory it is having default permissions. The permissions will applied on three attributes.

i. Owner

ii. Group

iii. Others

Permission modes are three types

1. read (r=4)

ii. write (w=2)

iii. execute (x=1)

The default permission of file "644" and directory "755".

Here "umask" value determine the default permission on a file (or) directory.

Note: The default umask value is "022"

To checking the umask value

```
# umask
```

To change permission of file or directory use the command "chmod"

- Only owner (or) root user can change the file permission

- The permissions can be applied in two ways

- i. Absolute mode (Numeric values)

- ii. Symbolic Mode (Absolute Mode)

Symbolic	Absolute
r	4
w	2
x	1

u - owner or user

g - group

o - others

a - all

+ - Add permissions

- - Remove permissions

= Assign permissions

Examples:

```
#chmod 644 filename
```

Set the permissions of **file** to "owner can read and write; group can read only; others can read only".

```
#chmod 666 file
```

Set the full permissions for all users

```
# chmod a=rw filename (Absolute Mode)
```

Set the full permissions for all users

```
# chmod u-r file
```

Removing the write permission owner/user

```
# chmod -R 777 dir1
```

Recursively (**-R**) Change the permissions of the directory **myfiles**, and all folders and files it contains, to mode **777**: User can read, write, and execute; group members and other users also can read, execute and write.

pipe (|) :

To combine two or more commands with pipe command

Pipe command will take command1 output as input to command2

```
# ls -l | head
```

Wc

wc stands for word count.

wc [options] filenames

wc -l : Prints the number of lines in a file.

wc -w : prints the number of words in a file.

wc -c : Displays the count of bytes in a file

Example:

ls -l | wc -l

It will display number of files and folder in present working directory

wc -l filename

It will display number of lines in a given file (filename)

Head

The head command outputs the first part (the head) of a file or files.

Note:

head, by default, prints the first 10 lines of each FILE to standard output.

cat file | head

Tail

Tail is a command which prints the last few number of lines (**10 lines by default**) of a certain file, then terminates.

cat file | tail

Note: It give the dynamic changes at any time. When code is changing the automatically it gives last "10" lines

Grep:

Grep command in Unix/Linux is a powerful tool that searches for matching a regular expression against text in a file, multiple files or a stream of input. It searches for the pattern of text that you specify on the command line and prints output for you.

Examples:

grep email nagi.txt

It will match the email pattern against file

or

cat nagi.txt | grep email

grep -i email nagi.txt (i-ignore case/Insensitive)

It will match the email pattern against file (It maybe Capital/Small letters)

grep -i -n email greptest (i-ignore case)

Display the email lines with their line numbers

grep -v email greptest

Print the lines excluding the pattern called email

```
# grep -vi email greptest (Ignore case)
```

Print the lines excluding the pattern called email with all cases of character

```
# grep -ino email greptest
```

Display the just word with line number

```
# grep -r linux /etc/
```

Search the pattern recursively using -r option

```
# grep [0-9] 1.txt
```

Display the lines, those lines are numbers contains with 0 to 9

Anchors

^ - Cap- Beginning of the line

\$- Dollar- End of the line

```
# ls -l | grep ^-
```

Display the all files

```
# ls -l | grep ^- | wc -l
```

Display all the files with count

```
# ls -l | grep ^d
```

Display all the directories

```
# cat 1.txt | grep spam$
```

Display the lines end with spam

```
# cat 1.txt | grep ^Hello
```

Display the lines start with Hello

Find

Find command is used to search and locate the list of files and directories based on conditions you specify for files that match the arguments.

Find can be used in a variety of conditions like you can find files by permissions, users, groups, file type, date, size, and other possible criteria.

Example

```
# find . -name data.txt
```

It find the data.txt in current location

```
# find . -i -name data.txt
```

- I = Ignore case

```
# find / -name test -type f
```

Search for file in entire file system based on condition

```
# find / -name testdata -type d
```

Search for directory in entire file system


```
# find . -type f -name "*.php"
```

Display all the files end with ".php"

```
# find /tmp -type f -empty
```

Display the all empty files under /tmp directory

Search by permissions

```
# find . -type d -perm 777
```

Display all directories having the 777 permissions

```
# find . -type d ! -perm 777
```

Find directories Without 777 Permissions

To search by size

```
# find / -size +10M
```

Display the files and directories more than 10MB

```
# find / -size +50M -size -100M
```

To find all the files which are greater than 50MB and less than 100MB

```
# find . -cmin -60
```

To find all the files which are changed in last 1 hour.

```
# find . -mtime -15
```

Days wise

```
# find / -type f -name "find.txt" -exec rm -f {} \;
```

To find a single file called find.txt and remove it.

```
# find / -type f -name "nagi.txt" -print -exec rm -f {} \;
```

Print the deleted file

Vi (Visual Editor)

Vi is a text editor used to create or modify the files.

Vi has three modes

i. command mode

ii. insert mode

iii. Colon mode

command mode

we can execute commands to navigate curser and perform deletion, copy, and paste etc operations

Insert Mode

Here we can perform insert operations

Colon Mode (:)

To perform save, quit, search etc operations

Note: default mode is Command mode

Moving within a file

Keystroke	Use
K	Move cursor up
J	Move cursor down
H	Move cursor left
L	Move cursor right

Saving

- x or w = save
- q = quit
- q! = quit forcefully
- wq = save and quit
- wq! = save and quit forcefully

- s = search
- 1 = Means from first line
- \$ = Means to last line
- g= globally (If pattern is occurrence multiple times it will change all occurrence)
- se nu = set numbers

g/patter1/s//pattern2/g To Replace string word for Global level

Keystrokes	Action
i	Insert at cursor (goes into insert mode)
A	Write after cursor (goes into insert mode)
a	Write at the end of line (goes into insert mode)
ESC	Terminate insert mode
u	Undo last change

O	Open a new line (goes into insert mode)
ndd 3dd	Delete Delete 3 lines. line
D	Delete contents of line after the cursor
dw 4dw	Delete Delete 4 words word
Cw	Change word
X	Delete character at the cursor
R	Replace character
R	Overwrite characters from cursor onward
G	Go to end of file
gg	Beginning of the file
\$	End of the cursor present line

Day-6

User management

User management includes everything from creating a user to deleting a user on your system. And also modify the users, locking users etc.

```
# useradd demo
```

To Create new demo user

```
# passwd demo
```

Adding password to demo user

```
# userdel demo
```

Deleted user demo

```
# cat /etc/passwd
```

To check the all the available users in file system

```
# cat /etc/shadow
```

To see the all the available user's password information in file system

```
# usermod -u (newuserid) username
```

To modify the userid for particular user

```
# usermod -l (newusername) (oldusername)
```

changing the user name

Package Management

Package is nothing but collections of files and directories grouped in defined structure or format.

Packages on different Platforms

In Windows- >.exe - It's a software (vlc.exe)

In Linux - .rpm - package (vlc.rpm)

Here in Linux we have two utilities to install packages in Linux Machine

1. rpm - RedHat Package Manager

2. yum - Yellow dog updater Modifier

q - query for packages

a - all packages

l - Display files list particular a packages

i - install

u - update

e - remove

h - Display the hashes # while installing the package

v - Verbose

--force - To install packages forcefully

-- nodeps - It doesn't check for dependencies before installing packages

Examples:

rpm -qa

It shows the which packages are installed in the system

rpm -qa | grep tree

Query for particular package

rpm -i tree-1.5.3-3.el6.x86_64.rpm

To install a tree package

rpm -e tree

To uninstall/remove tree package

rpm -iv tree-1.5.3-3.el6.x86_64.rpm

It shows the tree packages installation process which progress bar(#)

Yum

Yum will install rpm packages along with their dependencies. To achieve this yum maintains packages repository

yum configuration directory /etc/yum.repo.d (Repositories list)

yum update

To keep your system up-to-date with all security and binary package updates, run the following command. It will install all latest patches and security updates to your system

yum install tree

To install package called tree

```
# yum install firefox
```

To install firefox

```
# yum install -y tree
```

It will install the tree package without asking permission

```
# yum remove tree
```

It will remove the package called tree

```
# yum remove -y tree
```

It will uninstall the tree package without asking permission

```
# yum update tree
```

Let's say you have outdated version of **tree** package and you want to update it to the latest stable version

```
# yum info tree
```

To show the all information about Zsh

Service management

service is the command to manage services in linux

Syntax: service (servicename) (action means Status, start, stop, restart)

Status: To verify service status

start: To start service

stop: To stop the service

restart: To restart the services

Examples:

```
# service httpd status    -checking the status of httpd service
```

```
# service httpd start    - start the httpd service
```

```
# service httpd stop    - stop the httpd service
```

```
# service httpd restart    - restart the httpd service
```

crontab

cron is the utility to schedule the jobs which can run at specific intervals of time

```
# service crond status
```

To check the crond service

```
# crontab -l
```

It display the scheduled jobs (By default shows the root jobs)

```
# crontab -l username
```

It display the scheduled jobs for specific user

```
# crontab -e
```

To create jobs and edit jobs

```
# crontab -r
```

To remove the jobs

Note: crond all related files located under /etc/cron.d

```
# cat /etc/crontab
```

configuration file of cron

Example of job definition:

```
* .----- minute (0 - 59)
* | .----- hour (0 - 23)
* | | .----- day of month (1 - 31)
* | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
* | | | | .---- day of week (0 - 6) (Sunday=1 or 7) OR sun,mon,tue,wed,thu,fri,sat
* | | | | |
```

Examples:

```
# * * * * * cp/etc/passwd /root/Desktop/Linux/passwd.bkp
```

For every minute it will take the backup of passwd file

```
# 0 5,17 * * * sh /opt/logs.sh
```

It will run the job twice a day

```
# */10 * * * * /scripts/monitor.sh
```

For every ten minutes it will run the job

```
# 0 17 * * sun,fri /scripts/script.sh
```

It will run job on selected days at 5 PM

```
# 0 */4 * * * /opt/myjobs/backup.sh
```

It will run the job for every four hours

```
# @yearly /opt/myjobs/backup.sh
```

It will execute the job on the first minute of every year

```
# @daily /opt/myjobs/backup.sh - daily
```

```
# @hourly /opt/myjobs/backup.sh
```

Note: The location of schedule cron job files are located under /var/spool/cron

SSH(Secure shell)

ssh is utility to connect the remote machine

The ssh command provides a secure encrypted connection between two hosts over an insecure network.

The configuration file /etc/ssh/sshd_config

The ssh port number is "22". With help of port number we can identify the service

```
# ssh <ip_remote_Machine>
```

To connect remote Machine

```
# ssh username@<ip_remote_Machine>
```

To connect remote machine with specific user

```
# service sshd status
```

Checking the sshd status

Trust Relation

Without give credentials (user name and password) login the remote machines

SSH trust between two servers so that the two servers share the same SSH keys and can log into each other.

To establish trust relation between two servers follow the below steps

Step 1: Generate the ssh key

```
# ssh-keygen -t rsa
```

Note: Two types of keys are available one is rsa other hand is dsa. rsa keys more secure because of encrypt mode. Dsa keys are less secure so any one can hack it.

For above command we get the two keys i. public key ii. private key

We can find the keys the home directory of user for example

location: /root/.ssh/ id_rsa, id_rsa.pub

Step:2

Copy the id_rsa.pub key into remote machine the location /root/.ssh/authorized_keys

Step3:

Restart the sshd service

Step4:

Now check the trust relation

ssh Machine

SCP(Secure copy)

scp command is used to copy the files and directories from one machine to another machine (Either local or remote)

Syntax:

```
Scp <source_path> <Destination_path>
```

```
# scp /opt/mydata.txt 192.168.249.144:/root/Desktop/
```

Copy file from one machine to another machine

```
# scp filename 192.168.249.160:/root/Desktop/file /opt
```

Copy the file from remote to local machine

```
# scp -r /opt/dir1 192.168.249.144:/root/Desktop/
```

Copy the directory from local to remote machine

Tar (tape archive) To perform backups

c- to create backups

v- verbose

f- To specify backup filename

x- Extract

t- To display table of content

Note: Basically all softwares are in tar files

```
# tar cvf mydata.tar /root/Desktop/linux/data
```

To create a backup file with name of mydata.tar

```
# tar tvf mydata.tar
```

To view the content of the tar file

```
# tar xvf mydata.tar
```

To extract the tar file

System monitoring

du(Disk usage)

du

To find out the disk usage summary of a current directory tree and each of its sub directories.

du -h

To show the human readable like bytes, kilobytes, megabytes, gigabytes.

du -sh

To get the summary of a grand total disk usage size of an directory use the option “-s”.

du -h /opt

To find out the disk usage summary of opt directory tree and each of its sub directories.

df -h

To display the space utilization of the system with human readable format

TOP

Display system performance (Memory and cpu utilization information)

-d = To specify delay time

-n = To specify count

-o = To sort the o/p by specifying field

-m = Sort the o/p memory usage

-p = To specify process id

Shift o - for shorting utilization data

Note: By default it will refresh every 3 seconds

top -d5 -n10

For every 5 seconds it will refresh and show the info 10 times quit it

top -u username

Display the specific user process details

Note:

Press c after fire the "top" command Then it will show absolute path of running process

z - Highlighting the running the process which may help you to identified running process easily.

k - we can kill the process after finding the process id

shift + p - It will sort the cpu utilization

Process

Whenever a command is issued in unix/linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.

Process is nothing but entity of application. Every process running in the system will have a process ID. The first process start in the linux machine "init". Which will have the process id "1"

When you start a process (run a command), there are two ways you can run it –

- Foreground Processes
- Background Processes

it reads this information from the virtual files in /proc in filesystem

ps

It is easy to see your own processes by running the **ps** (process status) command

ps -f

which provides more information about your process

ps -fU root (username)

To display a user's processes by real user ID (RUID) or name, use the -U flag

ps -p 23

It display the complete information about process id 23

Kill

kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. kill command sends a signal to a process which terminates the process.

kill process id (Kill the process)

kill -9 process id (Forcefully delete the process id including any dependency)

SED- Stream editor

It is mainly used for text substitution, find & replace but it can also perform other text manipulations like insertion, deletion, search etc..

Options:

-n = To suppress the entire file out put

-e = To specify the expression

q = quit after reading the lines

p = print the lines

d = delete the lines

i = insert the lines

a = append the lines

c = change the lines

s = search the pattern and to replace it

Eg: s/string1/string2/

It replaces the first occurrence of string1 with place of string2 in all lines

s/string1/string2/g'

It replaces all the occurrence of string1 with place of string2

i - ignore case

^ - beginning of the lines

\$ - End of the line

sed -n '1,10p' filename

It Just display the lines from 1 to 10

sed -n '50,\$p' filename

It Just display the lines from 50 to last line of file

sed -n -e '10,20p' -e '40,50p' filename

It will display the line numbers from 10 to 20 and 40 to 50

sed '40 50d' filename

It will delete lines from 40 to 50 in a file

Note: It delete temporarily means skip them

sed '\$=' filename

or

sed -n '\$=' filename

It will count the line numbers

sed -n '/usr/p' filename

Display the line numbers with particular word

sed -n '/usr/=' filename

Display the line numbers of word

sed 's/usr/rsa/g' filename

Replace the word in place of usrrsa will appear

#sed 's/usr//g' anaconda-ks.cfg

It will delete the usrword globally in file

sed 's/.\$//g' anaconda-ks.cfg

It will delete the last character of every end of line.

sed 's/[^]*\$//g' anaconda-ks.cfg

It will delete the last word of every line in file

GIT & GITHUB:



git



Git init

“git init” initializes a repository for you to work in on your computer.

After git init we can see some default structure in .git

HEAD: It tells you where you are means on which branch you are working

hooks

Hooks are customization scripts used by various Git commands. A handful of sample hooks are installed when *git init* is run, but all of them are disabled by default.

This directory contains shell scripts that are invoked after the corresponding Git commands. For example, after you run a commit, Git will try to execute the post-commit script.

info/exclude

This file, by convention among Porcelains, stores the exclude pattern list. .gitignore is the per-directory ignore file

refs folder

References are stored in subdirectories of this directory.

refs/heads

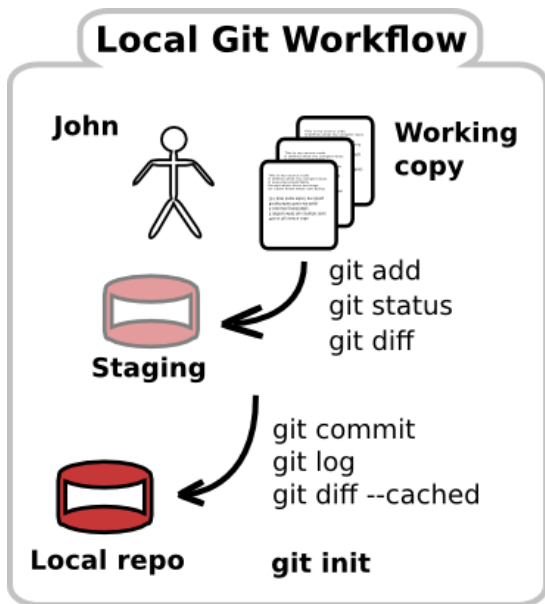
Contains commit objects.

refs/tags

Contains the tags

refs/remotes

Contains commit objects of branches copied from a remote repository.



Phase in git

1. working dir (When you create a new file it is under working dir/area)
2. stage/cache (When you add file to goes to stage/cached)
3. local_repo (When you commit a file it goes to local repo)

Adding a file to local repository

```
# touch sample (create a file called sample)
```

```
# git add sample
```

```
# git commit -m "Sample file added"
```

It will create a one version/revision

Note: "git log" will display all the logs with sha id and commit message.

Note: git add means file move from working area to staging area

Note: git reset HEAD or git reset Filename Means move back to working area from staging area

How to import code to repository (Existing code)

Step1: go to source code location

Step2: Run git init

Step3: git add . or git add filename

Step4: git commit -m "Commit message" (Move to local repository)

Step5: Run git remote add origin <remote repository URL>

Step6: git push origin master --force (Push to remote repository)

How to work with New repository

Step1: Clone the repository from remote server

git clone <Remote repo URL>

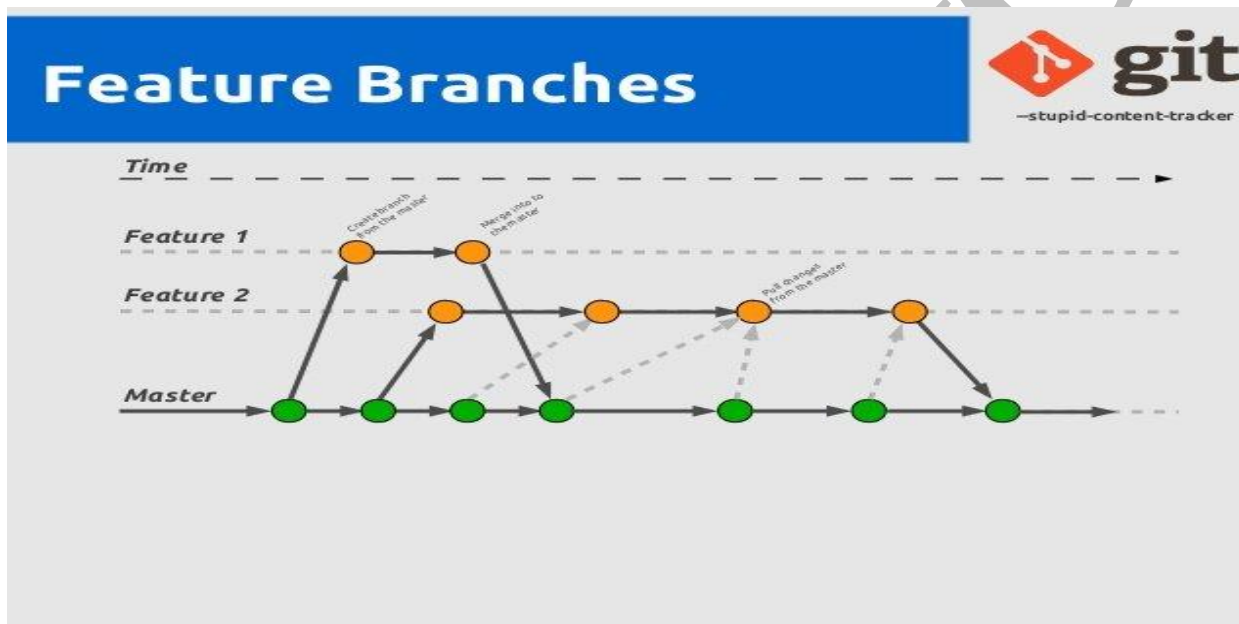
Step2: Can Change the code or create a New files

Step3: git add

Step4: git commit

Step: git push origin master

Branch strategy



Branch

Branch is collection of commits or series of commits along with the files/dir

How to create a branch

git branch <branch name>

(or)

git checkout -b <branch name> (branch will create and switch to branch)

How to upload the branch to remote repo

git push origin (branch name)

How to upload multiple branches at a time into remote repo

```
# git push origin --all
```

(or)

```
# git push <url of repo>/origin -all
```

How to delete branch

```
# git branch -D <branch name>
```

```
# git push origin :Deleted branch name
```

COMMIT AND LOGS

After committing a file we can see sha1 id or commit id with help of git log

git show <commit id> we can see content of the file

what is sha1 number ?

It is a random and unique 40 character hexa decimal value (0-9, a-f)

- Revisions will be maintained based on sha1
- Git never track empty directory

LOGS

git log --oneline (it shows all logs in one line)

```
git log --since=2018-07-16
```

```
git log --until =today date
```

Search by author

```
git log --author="nag"
```

How to check logs based on pattern

```
git log --grep="index"
```

Checking the diff b/w two commits

```
git dif 12sdwdw..sadj1w233
```

Adding and commit a file or dir in a single step

```
git commit -am "<commit name>"
```

NOTE: applicable for existing files only

```
# rm file1 ( it will stay in stage, local repo)
```

Git checkout file1 to get back

```
# git rm file1 ( it will deleted all areas)
```

Ignore file

To ignore the unnecessary files in a project.

A *.gitignore* file should be committed into your repository, in order to share the ignore rules with any other users that clone the repository.

Project level

```
pwd vi .gitignore
```

```
*.log
```

```
*.class
```

```
*.obj
```

```
!index.php
```

```
[abc].php
```

```
# git add .gitignore
```

```
# git commit -m " Message"
```

```
# git push origin master
```

Git Squash: It means Squash the several Git commits into a single commit

The first thing to do is to invoke git to start an interactive rebase session:

```
git rebase --interactive HEAD~[N]
```

Or, shorter:

```
git rebase -i HEAD~[N]
```

Example:

```
# git rebase -i HEAD~4
```

OUTPUT:

Fire below command

Note: git rebase -i --abort (If you get like this "interactive rebase already started")

```
pick 59d2a23 Initial implementation
```

```
pick 752ae4c Add more features
```

```
pick cd302a3 Fix something
```

```
pick 5410de3 Fix something else
```

Change below like this

```
pick 59d2a23 Initial implementation
```

squash 752ae4c Add more features

squash cd302a3 Fix something

squash 5410de3 Fix something else and then save file

Push to git hub like below

git push origin {BranchName} or **git push origin {BranchName} --force**

Git cherrypick

Cherry picking in Git means to choose a specific commit id from one branch and apply it onto another branch

Step1: Pick the particular commit id with help of git log

git log

Step2: Checkout the branch where you want merge

git checkout branch name

Step3: Execute the cherry pick command

git cherry-pick <commit id>

Git fetch:

It gets the updates from remote repo into working directory, but it will not merge directly into your working copy. After you have to use merge command or rebase command to merge to your working copy

Steps for Fetch example

Step1: Add one file or directly in remote repository

Step2: Do commit

Step3: Come to local Machine/repo

Step4: Fire git fetch

Step5: git log origin/master (Here we can see the diff of remote repo)

Step6: git merge origin/master (To merge into local repository)

Git pull:

Get the updates of remote repo into local repo and merge directly

pull = fetch + merge

Git Merge

Merge command is used to combine the two branches

Example:

Start a new feature

```
# git checkout -b new-feature master
```

Edit some files and add it

```
# git add <file>
```

```
# git commit -m "Start a feature" (committing a file)
```

```
# Merge in the new-feature branch
```

```
# git checkout master
```

```
# git merge <branch name>
```

```
# git push origin <branch name>
```

```
# git branch -d <branch name>
```

Git Stash

This is a feature of git which is used for leaving unfinished work and start a new functionality related coding.

In simple terms save the work copy before commit to local repository

Step1: Create a one file and staged it (Means add to git repo)

Step2: (To create stash)

Step3: git stash list (To show the stash list)

Step4: git stash apply (Stash list will not remove from local repo)

or

Step4: git stash pop (Apply the latest stash to working copy and remove from the stash list)

Step: git stash pop stash@{2} (To apply specific stash from the stash list and remove from the stash list)

Note: The stash is local to your Git repository; stashes are not transferred to the server when you push.

Git tags

Tags are ref's that point to specific points in Git history. Tagging is generally used to capture a point in history that is used for a marked version release (i.e. v1.2.3)

Two types Tags are available

i. Lightweight tags

ii. Annotated tags

Lightweight tags

Lightweight tags are essentially 'bookmarks' to a commit, they are just a name and a pointer to a commit, useful for creating quick links to relevant commits.

Lightweight tags as private.

Create a Lightweight Tags

```
# git tag <tag-name> (v1.4.0)
```

checking git tags

```
# git tag
```

Annotated Tags

Annotated tags store extra meta data such as: the tagger name, email, and date. This is important data for a public release.

Annotated tags as public

Create an Annotated Tag:

```
# git tag -a v1.1.1
```

or

```
# git tag -a <tag-name> -m <tag-message>
```

```
# git tag -a 'Release_1_0' -m 'Tagged basic string operation code' HEAD
```

```
# git tag -a 'Release_1_0' -m 'Tagged basic string operation code' Commit ID ( Git log then pick particular CommitID )
```

Create a tag for a specific commit:

```
# git tag -a <tag-name> <commit-checksome>
```

Push a specific tag to remote:

```
# git push origin <tag-name>
```

Push all the tags to remote:

```
# git push origin --tags
```

To know about the details of tag

```
# git show <tagname>
```

To delete tag

```
# git tag -d <tagname>
```

or

```
# git tag --delete tagname
```

Share the deleted tags to remote repo

```
git push origin :Deleted tagname
```

How can i check branch has been already merged into master

```
# git branch --merged : List out branches merges into master
```

What is origin

origin is the default **alias** to the URL of your remote repository.

There's no requirement to name the remote repository origin: in fact the same repository could have a different alias for another developer.

This alias name is not hard coded and could be changed using following command prompt:

```
# git remote rename origin my_new_alias_name
```

```
# git config --global --edit It will open the configuration file
```

```
# git ls-files -s -> Display the files with sha ID and Size of files
```

How to unstage a staged file

```
# git rm --cached README.md at Initial stage of Head
```

```
# git reset HEAD <file>..." to unstage when head having more than one commits
```

How to undo the most recent commits in Git?

```
# git reset --hard HEAD~1 (Here head is going to previous commit id)
```

Note: Completely remove the file content and it say nothing to commit on particular branch

```
# git reset HEAD~1
```

It will send to staging area where we can see like "untracked file"

How to modify last commit message ?

Git commit --amend (It is open text editor you can modify your committee msgs)

How to modify specified commit id?

```
# git rebase --interactive 'Commitid^'
```

Example:

```
# git rebase --interactive 'bbc643cd^'
```

How to clone a specific branch in git

```
# git clone -b <branch> <remote_repo>
```

Example: `git clone -b develop git@github.com:user/myproject.git`

file from How to download a single git/github

```
# wget url of file/path of file ( Url from the github)
```

How to modify specific commit message?

Step1: Get the commit id with help of “git log”

Step2: select particular commit id

Step3: fire below command

```
# git rebase --interactive 'commitID^'
```

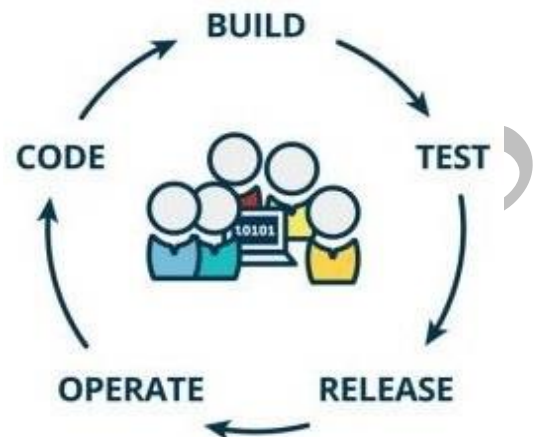
Step4: In the default editor, modify “pick to edit” in the line mentioning 'CommitID'

Step5: `git commit --amend`

Step6: `git rebase --continue`

JENKINS:

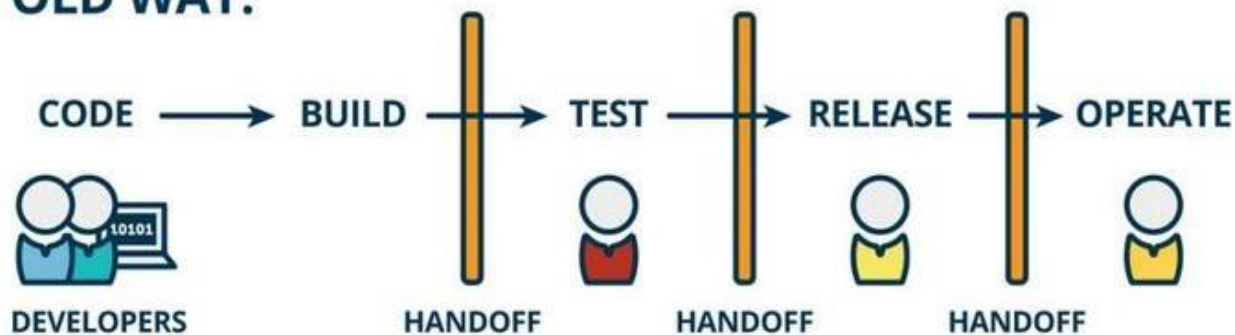
Continuous Integration and Tools:



Situation if there is no Continuous Integration tool is used

- Integration of code happens manually
- Time factor - it takes time to integration the code of developers
- “last minute rush” activities in project
- Project delays

OLD WAY:



CI Flow



CI: Continues integration

It's a development practice or software engineering practice where members of team integrate their work frequently.

Usually, each person integrates at least once in a daily and leading to multiple integrations per day. Each integration is verified by an automated build to detect integration errors as early as possible

Improve the code quality and reduce integration problems

Here, the fresh changes are immediately tested and reported.

In traditional software development process integration is generally done at the end of day when every developer had completed their work.

As a result of this integration took lots of time and was a very painful process

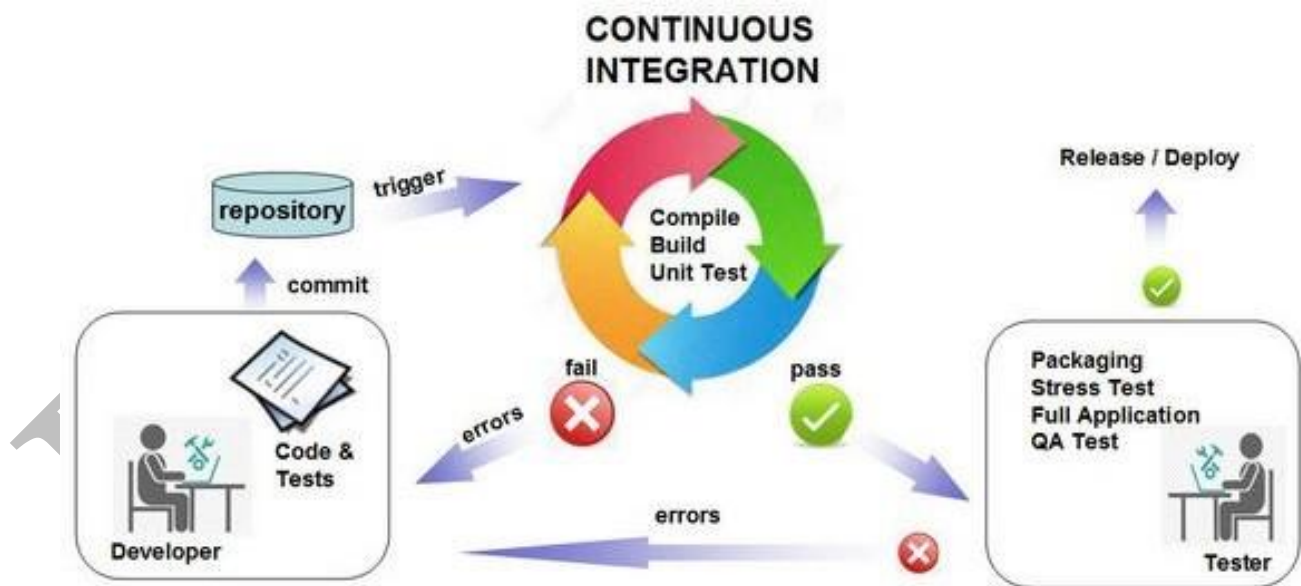
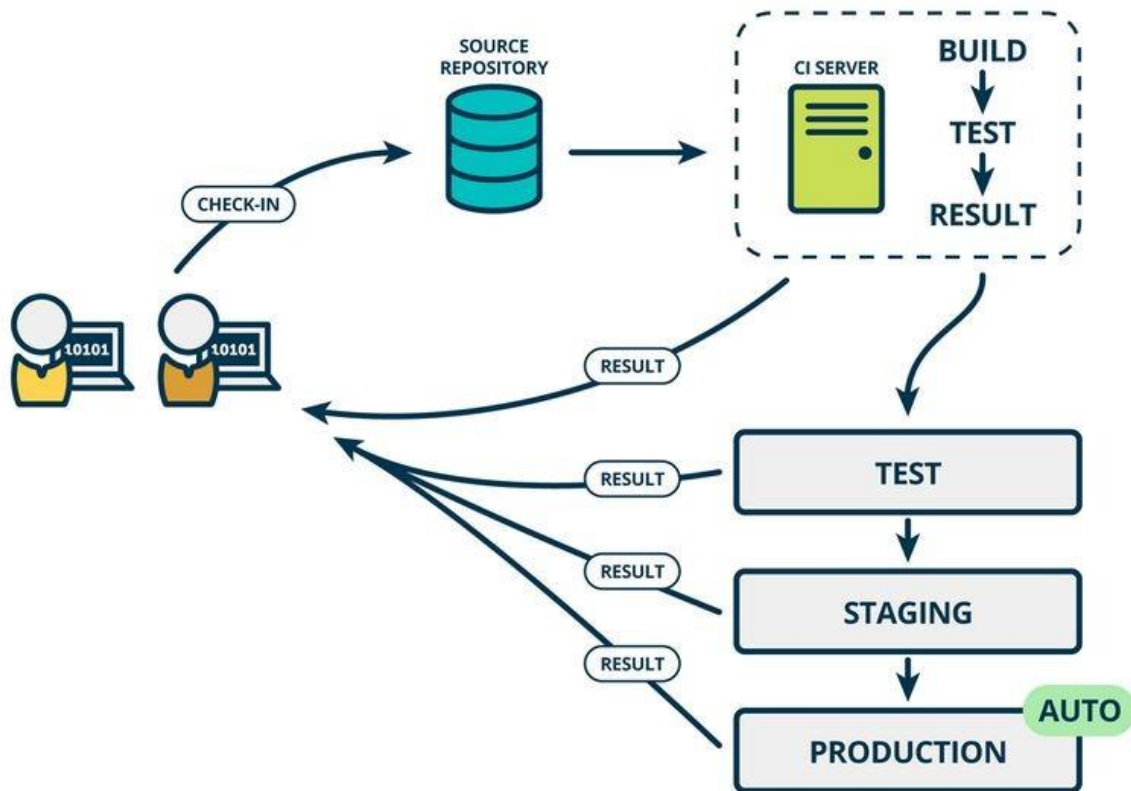
Developers generally use a tool called CI Server to do the building and the integration for code

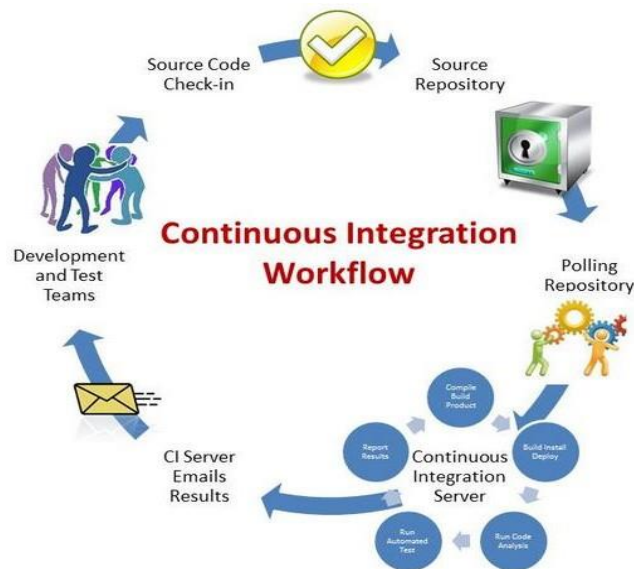
This code runs a self-test to ensure that it is working as expected

Benefits of CI

- ✓ Faster product releases
- ✓ Reduce repetitive manual processes
- ✓ Enable better project visibility
- ✓ Less bugs
- ✓ Good quality code
- ✓ Improves team communication
- ✓ Reduce risks

GENERIC ARCHITECTURE MODEL OF C.I





Features of Jenkins

- Open Source
- Easy installation
- Easy configuration
- Rich plugin ecosystem
- Extensibility
- Distributed builds
- Platform: Cross-platform

Features :-

- Free and Paid
- Gated Commits (prevents developers from breaking sources in a version control system by running the build remotely for local changes prior to commit)
- Build Grid. Allows running multiple builds and tests under different platforms and environments simultaneously
- Integrated code coverage, inspections and duplicates search
- Integration with IDEs: Eclipse, IntelliJ IDEA, Visual Studio
- Platforms supported: Java, .NET and Ruby
- Supports cloud integration

Other Notable CI Tools



Travis CI

- Open source
- Supports pull request and branch build flow
- Parallel test runs
- Easily synchronize with GitHub
- Flexible plans for every size project
- Platforms: Hosted
- Supports Many Languages like Node js, php, Xcode, python and many more.



Visual Studio Team Foundation Server

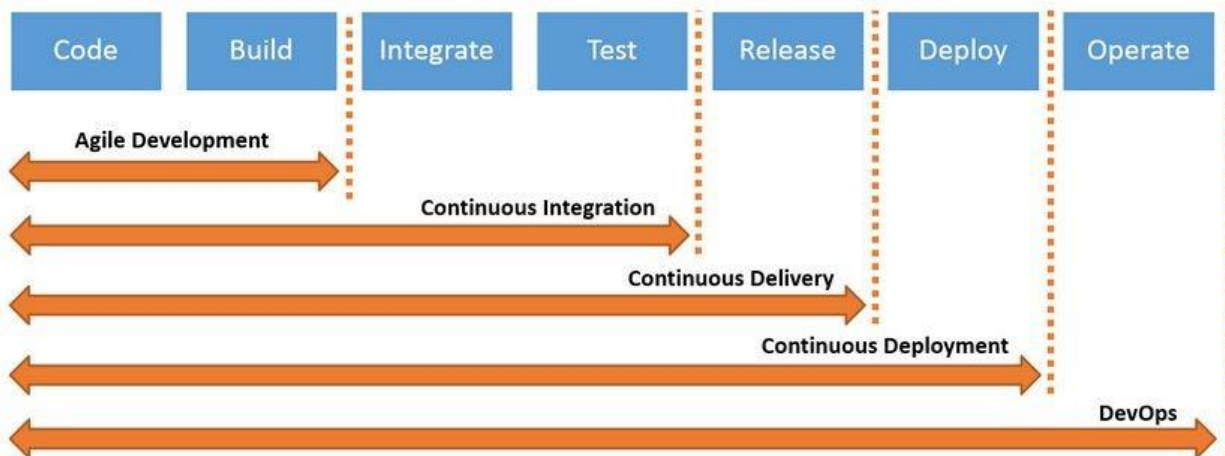
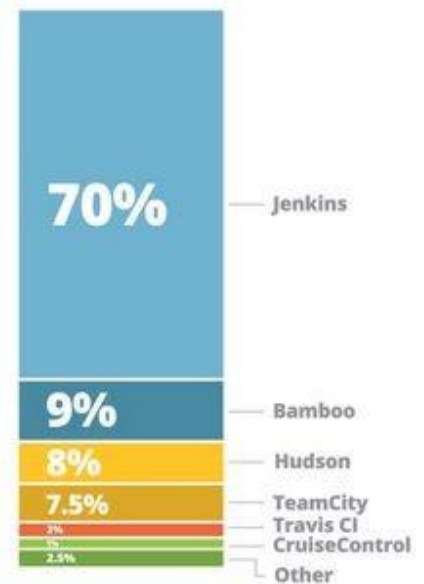
- Trial-ware
- Supports many languages like Python, C#, HTML, Java and various others
- Work in any environment like Visual Studio, Xcode, Eclipse, or any Git client
- Extensible tool can work effectively for all shapes and sizes

Atlassian



Bamboo

- Paid and Free trial
- Cross platform
- Allow to Import data from Jenkins
- Works with JIRA and Bitbucket
- Works with others tools like CodeDeploy, Docker, Maven, Git, SVN, Mercurial, Ant, AWS, Amazon S3 buckets
- Support many languages
- Can run multiple builds parerally
- customization of triggers and variables
- Very fast and easy to use



JENKINS -Technical Features

- Build Server
- Distributed build Support
- Gets the code from Repository
- Trigger builds - Manual, Periodically and Automatically
- Automatic build and tests etc
- Open Source C.I tool written in JAVA language
- Jenkins is used in variety of Projects like in Java, .net, Ruby, PHP etc

JENKINS - RECOMMENDED HARDWARE PREREQUISITES

1 GB RAM
50 GB + HDD

JENKINS - RECOMMENDED SOFTWARE PREREQUISITES

Need Java

App Server (Tomcat, Weblogic, Glassfish)

JENKINS - Supported Platforms

Windows

Linux

Mac

JENKINS SETUP USING GENERIC WAR FILE

Step 1 - Verify if Java is installed on the PC or not

Step 2 - If not installed, install the Java Environment variables

Step 4 - Install Jenkins using war file, download from jenkins web site

Types of Installing

i. Stand alone

ii. Deploy war

iii. As a service

Stand-alone Method

WAR file

The Web application ARchive (WAR) file version of Jenkins can be installed on any operating system or platform that supports Java.

- To download and run the WAR file version of Jenkins:

Download the [latest stable Jenkins WAR file](#) to an appropriate directory on your machine.

Open up a terminal/command prompt window to the download directory.

Run the command `java -jar jenkins.war`.

Browse to `http://localhost:8080` and wait until the Unlock Jenkins page appears.

Continue on with the [Post-installation setup wizard](#) below.

Post-installation setup wizard

After downloading, installing and running Jenkins using one of the procedures above, the post installation setup wizard begins.

This setup wizard takes you through a few quick "one-off" steps to unlock Jenkins, customize it with plugins and create the first administrator user through which you can continue accessing Jenkins.

Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically generated

1. Browse to `localhost:8080` (or whichever port you configured for Jenkins when installing it) and wait until the **Unlock Jenkins** page appears.

Unlock Jenkins

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

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

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

ERROR: The password entered is incorrect, please check the file for the correct password

Administrator password

Then

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

From the Jenkins console log output, copy the automatically-generated alphanumeric password

On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**

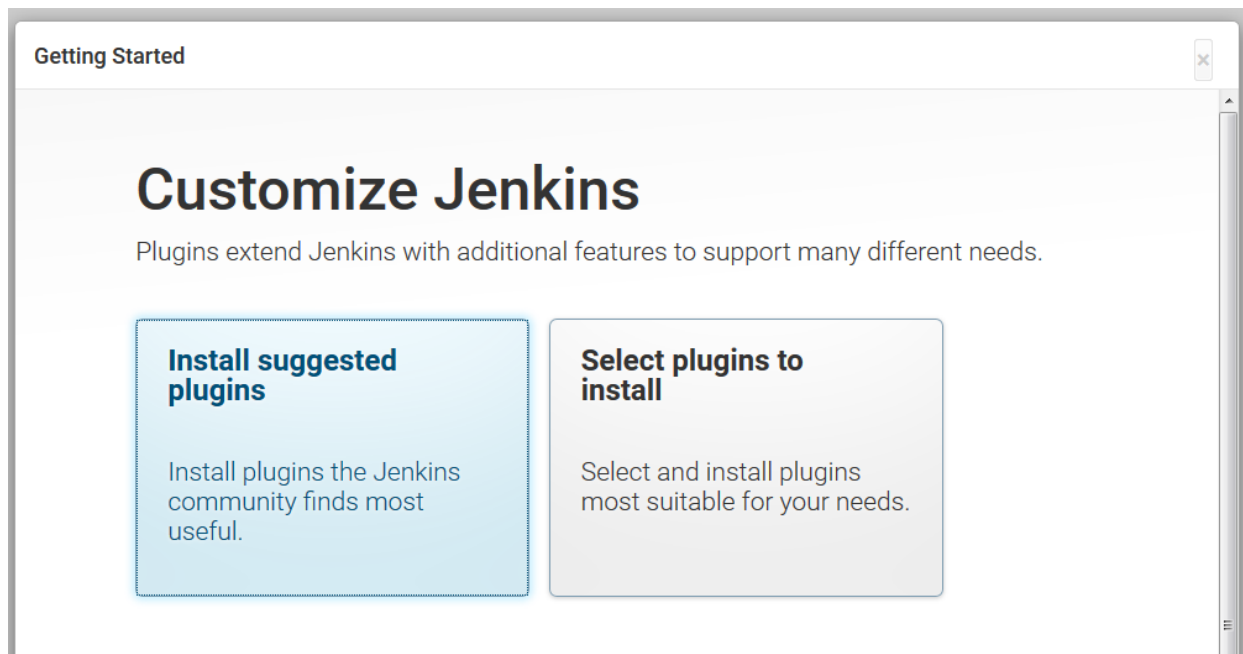
Customizing Jenkins with plugins

After unlocking Jenkins, the **Customize Jenkins** page appears. Here you can install any number of useful plugins as part of your initial setup.

Click one of the two options shown:

Install suggested plugins - to install the recommended set of plugins, which are based on most common use cases

Select plugins to install - to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.



Click on Install Suggested Plug-ins

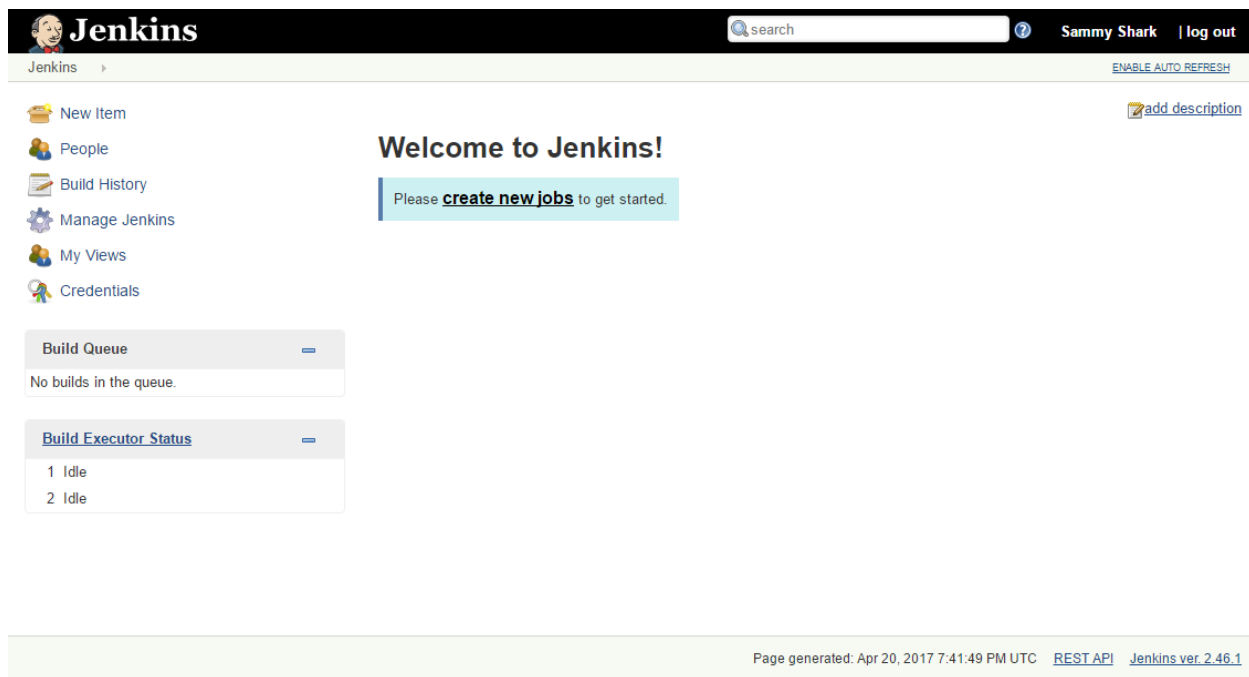
After That we see Admin account page

Creating the first administrator user

Finally, after customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify the details for your administrator user in the respective fields and click **Save and Finish**.
2. When the **Jenkins is ready** page appears, click **Start using Jenkins**.
3. **Notes:**
 - This page may indicate **Jenkins is almost ready!** instead and if so, click **Restart**.
 - If the page does not automatically refresh after a minute, use your web browser to refresh the page manually.
4. If required, log in to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

Jenkins Dashboard will appear after all the steps



The Dashboard Left Panel

New Item

- This is where you'll add projects, folders and pipelines.
- The core of Jenkins functionality is here.

People

- This is where you can see a list of users and their latest activities.

Build History

- You'll see an overview display of build history for all projects in graphical form.

Manage Jenkins

- Where Jenkins is managed

My Views

- For configuring custom views for projects for the logged in user.

Credentials

- Lists credentials that have been configured for Jenkins

Build Queue

- Jobs waiting for an executor are listed here.

Build Executor Status

- The status of projects associated with Jenkins executors
- An executor runs projects dictated by Jenkins.
- They can run in parallel.
- The default number of executors on the master is 2.

Build History

- Shows the history of builds
- Stable is blue
- Red is broken

Configure System

This is where you manage paths to the various tools you use in your builds, such as JDKs, and versions of Ant and Maven, as well as security options, email servers.

Jenkins search [DISABLE AUTO REFRESH](#)

[Jenkins](#)

[New Job](#)

[Manage Jenkins](#)

[People](#)

[Build History](#)

Build Queue
No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Manage Jenkins

- [Configure System](#)
Configure global settings and paths.
- [Reload Configuration from Disk](#)
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.
- [Manage Plugins](#)
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- [System Information](#)
Displays various environmental information to assist trouble-shooting.
- [System Log](#)
System log captures output from java.util.logging output related to Jenkins.
- [Load Statistics](#)
Check your resource utilization and see if you need more computers for your builds.
- [Jenkins CLI](#)
Access/manage Jenkins from your shell, or from your script.
- [Script Console](#)
Executes arbitrary script for administration/trouble-shooting/diagnostics.
- [Manage Nodes](#)
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
- [About Jenkins](#)
See the version and license information
- [Prepare for Shutdown](#)
Stops executing new builds, so that the system can be eventually shut down safely.

Page generated: May 5, 2011 7:24:07 AM [Jenkins ver. 1.410](#)

Click on Configure System
Change your Executors and Remaining settings all

Next click Apply and Save.

Configure System

This is where one can manage paths to the various tools to use in builds, such as the JDKs, the version of Ant and Maven.

Reload Configuration from Disk

Jenkins stores all its system and build job configuration details as XML files stored in the Jenkins home directory. It also stores all of the build history in the same directory. If you are migrating build jobs from one Jenkins instance to another, or archiving old build jobs, you will need to add or remove the corresponding build job directories to Jenkins's *builds* directory. You don't need to take Jenkins offline to do this—you can simply use the "Reload Configuration from Disk" option to reload the Jenkins system and build job configurations

directly. This process can be a little slow if there is a lot of build history, and Jenkins loads the build configurations.

Manage Plugins

One of the best features of Jenkins is its extensible architecture. Enabling you to add extra features to your build server, from support for different SCM tools.

Plugins can be installed, updated and removed through the Manage Plugins screen.

Note: Removing plugins needs to be done with some care, as it can sometimes affect the stability of your Jenkins instance.

System Information

This screen displays a list of all the current Java system properties and system environment variables. Here, you can check exactly what version of Java Jenkins is running in, what user it is running under, and so forth. You can also check that Jenkins is using the correct environment variable settings. Its main use is for troubleshooting, so that you can make sure that your server is running with the system properties and variables you think it is.

Load Statistics

Jenkins keeps track of how busy your server is in terms of the number of concurrent builds and the length of the build queue (which gives an idea of how long your builds need to wait before being executed). These statistics can give you an idea of whether you need to add extra capacity or extra build nodes to your infrastructure.

Script Console

This screen lets you run Groovy scripts on the server.

Manage Nodes

Jenkins handles parallel and distributed builds well. You can configure how many builds you want. Jenkins runs simultaneously, and, if you are using distributed builds, set up build nodes. A build node is another machine that Jenkins can use to execute its builds.

Prepare for Shutdown

If you need to shut down Jenkins, or the server Jenkins is running on, it is best not to do so when a build is being executed. To shut down Jenkins cleanly, you can use the Prepare for Shutdown link, which prevents any new builds from being started. Eventually, when all of the current builds have finished, you will be able to shut down Jenkins cleanly.

Setting Up Your Build Jobs

Build jobs are the basic currency of a Continuous Integration server.

A build job is a particular way of compiling, testing, packaging, deploying or otherwise doing something with your project. Build jobs come in a variety of forms; you may want to compile and unit test your application, report on code quality metrics related to the source code, generate documentation, bundle up an application for a release, deploy it to production, run an automated smoke test, or do any number of other similar tasks.

Jenkins Build Jobs

Creating a new build job in Jenkins is simple: just click on the “New Job” menu item on the Jenkins dashboard. Jenkins supports several different types of build jobs, which are presented to you when you choose to create a new job.

Freestyle software project

Freestyle build jobs are general-purpose build jobs, which provides maximum of flexibility.

Creating a Freestyle Build Job

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.

Steps to create a sample job For Executing a sample shell script

Step1: Click on New Item

Step2: Give the project Name and Select Freestyle Project, Press "OK"

Step3: Go to Build Section then go to Execute Shell and write your logic

Step4: Click on apply and save it

Step5: Run the job.

Continuous Build

Step1: Click on New Item

Step2: Give the Job name (Development Job) and select Free style Job

Step2: Select Version Control Tool (Git)

Step3: Give the Git url and Credentials

Step4: Go to Build sections

Step5: Click on Invoke Maven Top level goals

Step6: Enter a goal name "package"

Step7: Click on Build Now

Note: Above Job will create final artifacts

Continuous Deployment

Step1: Click on New Item

Step2: Give the Job name (Deployment Job) and select Free style Job

Step2: Select Version Control Tool (Git)

Step3: Give the Git url and Credentials

Step4: Go to Build sections

Step5: Click on Invoke Maven Top level goals

Step6: Enter a goal name "package"

Step7: Go to Post Build Actions

Step8: Click on Deploy to container

Step9: Give the all details as per tomcat Configuration

Step10: Click on Build NOW

How To install Plug-in

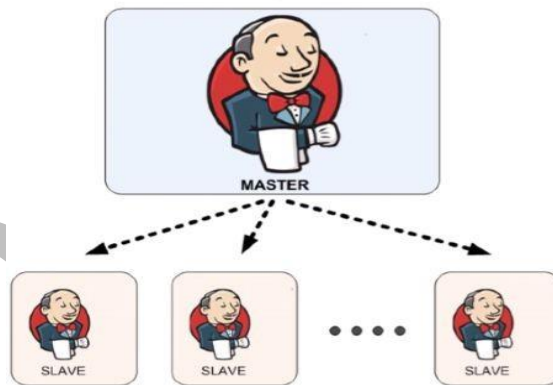
Step1: Go to Jenkins Dash Board

Step2: Manage Jenkins
Step3: Manage Plugins
Step4: Go available section
Step5: Search for required Plug-in
Step6: Install it

How to install Third Party Plug-ins

Step1: Go to Jenkins Dash Board
Step2: Manage Jenkins
Step3: Manage Plugins
Step4: Advanced Section
Step5: Upload your own plug-in

JENKINS MASTER-SLAVE ARCHITECTURE



Master and Slave

Sometimes you might need several environments to test your builds. This cannot be done by a single server.

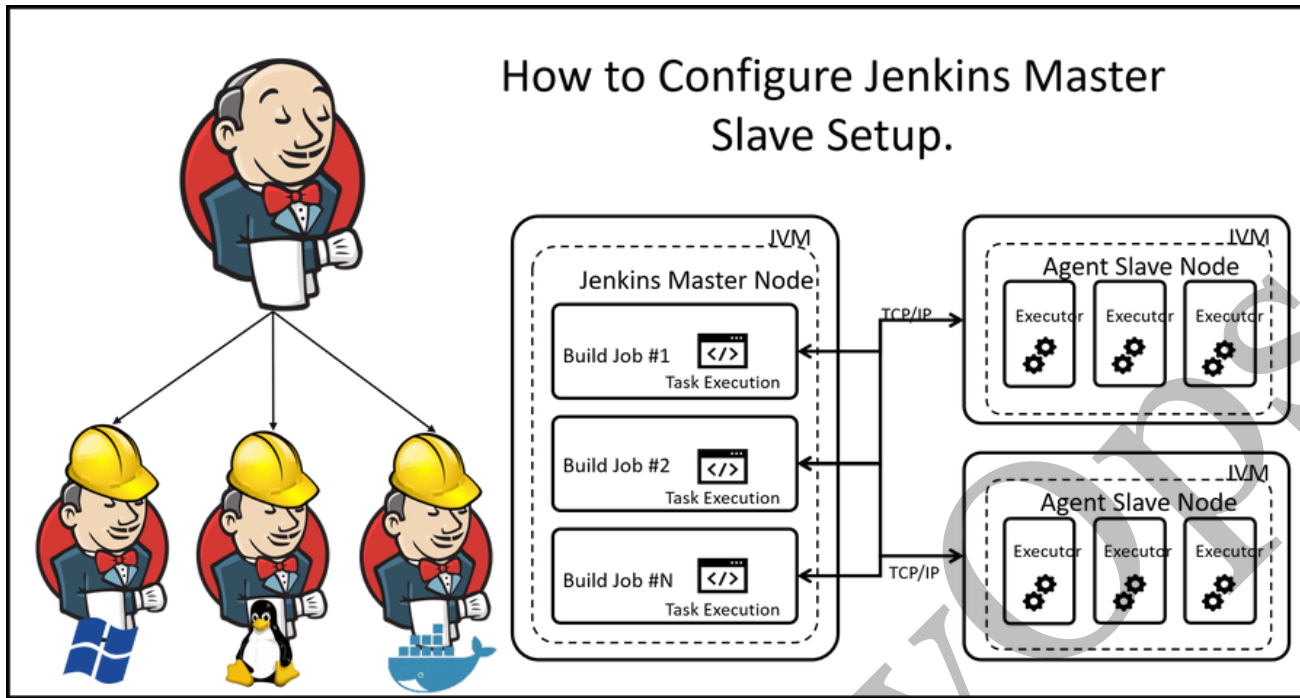
If you have a bunch of jobs get build on a regular basis then a single Jenkins server can not handle the entire all builds.

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

Jenkins uses a Master-slave architecture to manage distributed builds.

Master:

- Schedule build jobs.
- Dispatch builds to the slaves for the actual job execution.
- Monitor the slaves and record the build results.
- Can also execute build jobs directly.



The communication between Master and Jenkins slave nodes is bi-directional and it is happening with TCP/IP connection protocol.

* - asterisk

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 execution.
- Monitor Slaves (Possibly taking them online and offline as required)
- Recording and presenting the build results.
- A Master server of Jenkins can also execute build jobs directly.

Jenkins Slave

- It takes the requests from Jenkins Master.
- Slaves can run on a variety of operating systems.
- Slaves involve executing the build jobs dispatched by master.
- Slaves Machine Need to have like JDK, VCS and Build Tools

Note: Trust Relation should be completed.

Configure the Slave on Master

Step1: Open Jenkins home Page

Step2: Go to Manage Nodes

Step3: Click on create New node

Step4: Give Name of Node Machine
Step5: Modify the Executors number if required.
Step6: Specify the Remote root directory on Node Machine
Step7: Give the label name
Step8: Launch method (Launch agent agents via ssh)
Step9: Enter Host Details like IP Address
Step10: Enter Node Machines credentials
Step11: Specify the Java Location (where Java Installed on Node Machines)
Step12: Save and go slave1 Machine then Launch agent

Run a Job On Slave Machine

Step1: Click on New Item
Step2: Give the Job name (Deployment Job) and select Free style Job
Step3: Click on Restrict where this project can be run and Select Slave Machine where we can see list of all available slave Machines
Step4: Specify the all required Fields
Step5: Run the job and check the result on Master

Back Up

There are different ways to take the backup of Jenkins

Manual Way:

All the settings, build logs, artifact archives are stored under the JENKINS_HOME directory. Simply archive this directory to make a back up. Similarly, restoring the data is just replacing the contents of the **JENKINS_HOME** directory from a back up.
Backups can be taken without stopping the server, but when you restore, please do stop the server.

Note: For consistent backups it is good practise to keep JENKINS_HOME directory under Git repository.

Plug-in way

Jenkins has a backup plug-in which can backup configurations settings related to Jenkins.

Step1: Go to manage Jenkins and Click manage plug-ins
Step2: In Available section search for Backup Plug-in. Click on Install without restart

Restrict Jenkins Project Access to Users and Groups using Roles

Manage Roles:

Role-Based Authorization Strategy Plug-in is Needed.

NOTE: In Jenkins, by default you can create users, but not groups.

So, if you want groups in Jenkins, you have the following few options:

- Use “Role-based authorization strategy” plugin for Jenkins
- Use OpenLDAP with Jenkins
- Use Active Directory with Jenkins
- Use Unix user/group database. This will use PAM library to integrate with Jenkins.

Now, Start with user creation

Creating a user

Step1: Go to Manage Jenkins

Step2: Go to Manage users

Step3: Click on Create a user

Step4: Give info of user (user1)

Step5: Add one more user

Step6: And then log in as normal user (user1). After login as user1 on right side top corner we will be able to see settings regarding the logged user. There we can change password and find Api Token.

Up to Now successful Added users and Assign users to roles. For creation of roles we need install "Role based authentication strategy"

After install Plug-in

Step6: Go to configure global security

Step7: check role base strategy and save

Step8: And again go for Manage Jenkins

There we will be able to see Manage and Roles and Click on it



Manage Roles

Manage Roles



Assign Roles

Assign Roles



Role Strategy Macros

Provides info about macro usage and available macros

Step9: Go to Manage Roles

Step10: Add Global roles and Project Roles. Save it



Manage Roles

Global roles

Role	Overall	Credentials				Agent				Job				Run		View		SCM													
	Administer	Read	Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag
 admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 secondadm	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Role to add

Project roles

Role	Pattern	Credentials				Job								Run		SCM Lockable Resources				
		Create	Delete	ManageDomains	Update	View	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Tag	Reserve
 developer	"Dev.*"	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
production	"Production.*"	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 tester	"Test.*"	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Role to add

Slave roles

Role	Pattern	Credentials				Agent				Lockable Resources			
		Create	Delete	ManageDomains	Update	View	Build	Configure	Connect	Delete	Disconnect	Provision	Reserve

Step11: Go to Assign Roles

Step12: Add Global Roles and Item Roles. Save it

Jenkins > Manage and Assign Roles

New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Lockable Resources

Credentials

New View

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Assign Roles

Global roles

User/group	admin	secondadm
jenkins	<input checked="" type="checkbox"/>	<input type="checkbox"/>
user1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
user2	<input type="checkbox"/>	<input checked="" type="checkbox"/>
user3	<input type="checkbox"/>	<input checked="" type="checkbox"/>
user4	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add

Item roles

User/group	developer	production	tester
user1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
user2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
user3	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
user4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Now log-in as normal user called user1 and Notice the all difference

Changing home directory

Step1: Find out jenkins home dir

step2: create a new folder for example /opt/jenkinshome

step3: copy all files from .jenkins to new jenkins home dir

step4: export path jenkins path

step5: export JENKINS_HOME=path of dir (/etc/profile)

step6: restart the server

Email Notification

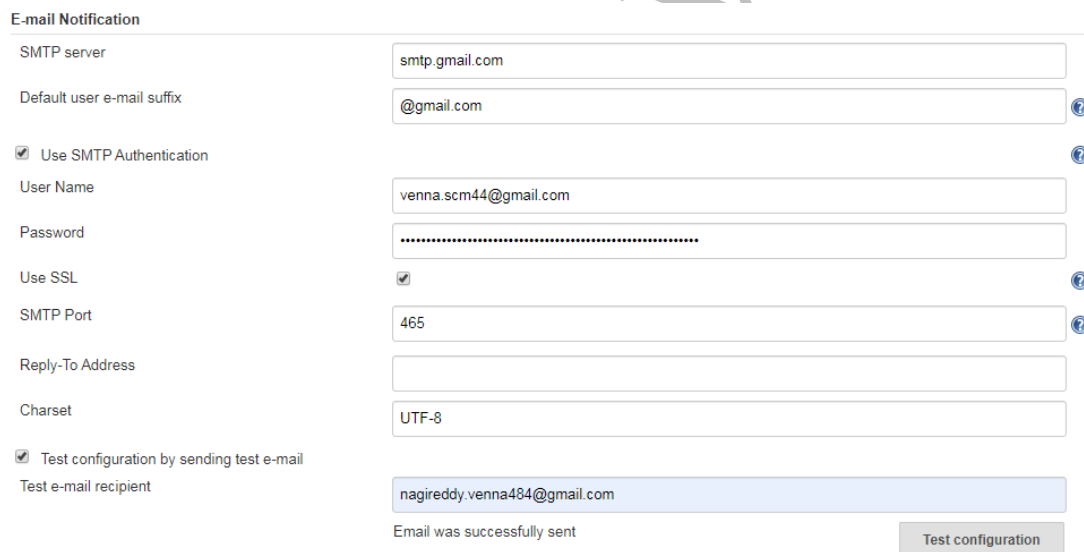
Jenkins is one open-source tool that can be used to perform Continuous Integration and build automation.

The resulting artifacts are automatically created and tested, and as a result, the process of identification of errors becomes faster.

How can an email service be integrated with Jenkins?

1. Default email notifier. This is what comes default with Jenkins. It contains a default message containing the build number and its status.

Go to configure Section



The screenshot displays the 'E-mail Notification' configuration interface in Jenkins. It includes fields for SMTP server (smtp.gmail.com), default user e-mail suffix (@gmail.com), a checked box for 'Use SMTP Authentication', user name (venna.scm44@gmail.com), a masked password field, a checked box for 'Use SSL', SMTP port (465), a reply-to address field, and a charset dropdown set to UTF-8. At the bottom, there is a checked box for 'Test configuration by sending test e-mail' and a 'Test e-mail recipient' field containing nagireddy.venna484@gmail.com. A 'Test configuration' button is located at the bottom right, and a status message 'Email was successfully sent' is visible above it.

Go to jobs section and pick one job then select configure. Now go to post build actions invoke email notification

2. Email extension plugin. This plugin allows you to configure every aspect of email notifications. You can customize when an email is sent, who should receive it, and what the email says.

Note: To send log as attachment format

Extended E-mail Notification

SMTP server: smtp.gmail.com

Default user E-mail suffix: @gmail.com

☒ Use SMTP Authentication

User Name: venna.scm44@gmail.com

Password:

Advanced Email Properties:

Use SSL: ☒

SMTP port: 465

Charset: UTF-8

Additional accounts: Add

Default Content Type: Plain Text (text/plain)

☐ Use List-ID E-mail Header

Save Apply

Go to jobs section and pick one job then select configure. Now go to post build actions invoke Editable Email notification

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Triggers

Failure - Any

Send To: Recipient List

Add

Recipient List: nagireddy.venna484@gmail.com

Reply-To List: \$PROJECT_DEFAULT_REPLYTO

Content Type: Project Content Type

Subject: \$PROJECT_DEFAULT_SUBJECT

Content: \$PROJECT_DEFAULT_CONTENT

Attachments:

Can use wildcards like 'module/dist/**/*.zip'. See the [@includes of Ant fileset](#) for the exact format. The base directory is [the workspace](#).

Attach Build Log: Attach Build Log

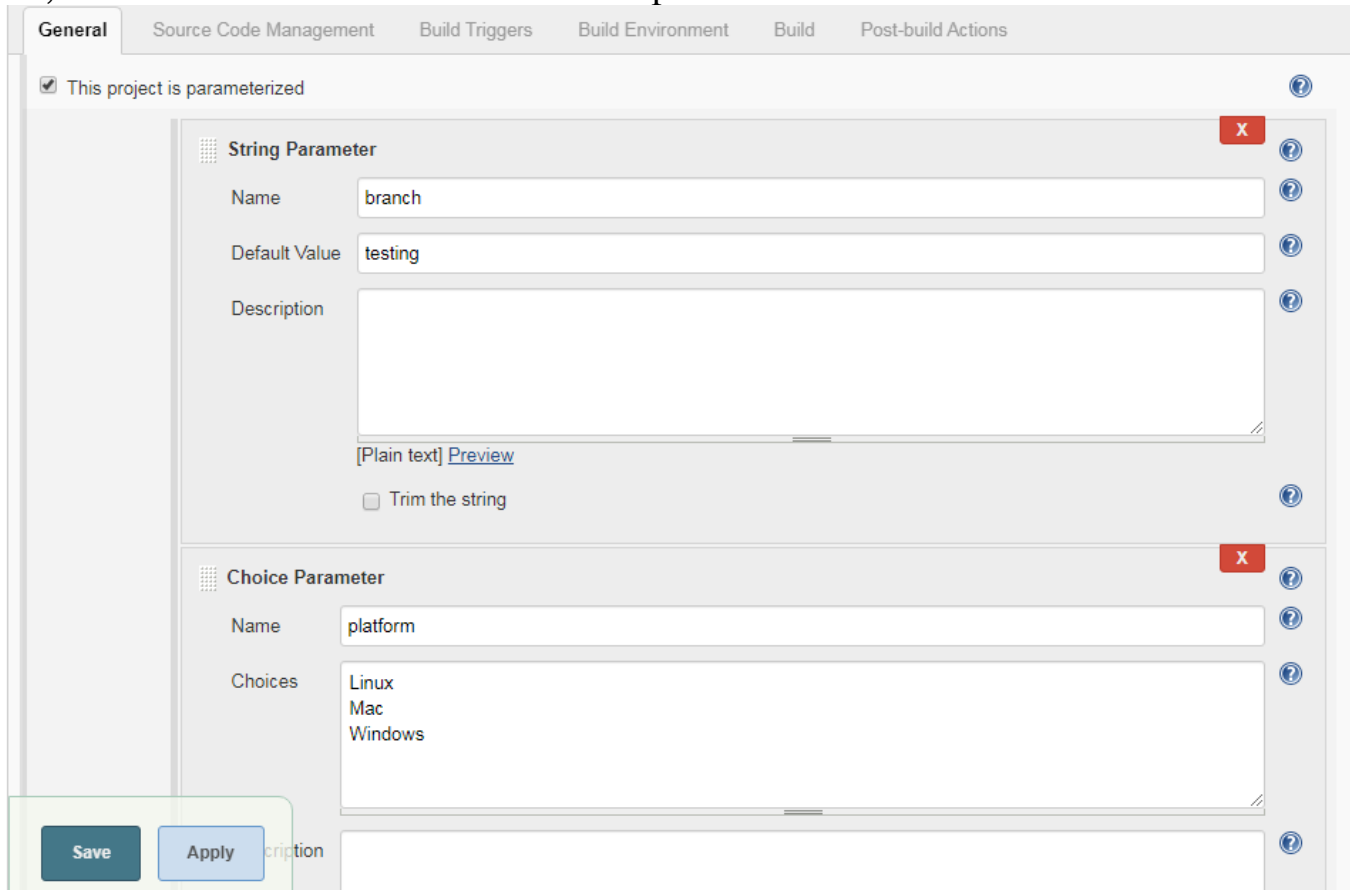
Save Apply

Add Trigger

Parameterized Jenkins Job Scenario:

want to use same job in different machine. But I don't want to change the configuration of the job each time. Can I pass the machine name label as parameter and run the job in different machine?

Yes, we achieve the above scenario with help of Parameterized Jobs.



The screenshot shows the Jenkins Job Configuration interface, specifically the 'Parameters' tab. The 'General' tab is selected, and the checkbox 'This project is parameterized' is checked. Below this, there are two parameter sections: 'String Parameter' and 'Choice Parameter'. The 'String Parameter' section has fields for 'Name' (branch), 'Default Value' (testing), and 'Description' (empty). The 'Choice Parameter' section has fields for 'Name' (platform) and 'Choices' (Linux, Mac, Windows). At the bottom, there are 'Save' and 'Apply' buttons.

General Source Code Management Build Triggers Build Environment Build Post-build Actions

☒ This project is parameterized

String Parameter

Name: branch

Default Value: testing

Description: [Plain text] [Preview](#)

☐ Trim the string

Choice Parameter

Name: platform

Choices: Linux, Mac, Windows

Save Apply

Publish Junit Test Cases:

Allows JUnit-format test results to be published.

The JUnit plugin provides a publisher that consumes XML test reports generated during the builds and provides some graphical visualization of the historical test result.

Go to Post build actions and Select Junit Publish Option.

Configuration below

Publish JUnit test result report

Test report XMLs: test-reports/*.xml

test-reports/*.xml' doesn't match anything: 'test-reports' exists but not 'test-reports/*.xml'

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is [the workspace root](#).

☐ Retain long standard output/error

Health report amplification factor: 1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Allow empty results: ☐ Do not fail the build on empty test results

Add post-build action ▼

Upstream and Down Stream

Upstream and downstream jobs help you to configure the sequence of execution for different operations and hence you can orchestrate the flow of execution. We can configure one or more projects as downstream jobs in Jenkins.

Upstream Versus Downstream Projects

- A project that is triggered by another project is considered to be downstream of that project.
- A project that triggers another project is considered to be upstream of that project.

Note: Build Pipeline Plug-in for Nice framework

After installation Plug-in We will be able to see Build Pipeline view
After All We can see the "+"

Jenkins

View name

☐ Build Pipeline View
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

☐ List View
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

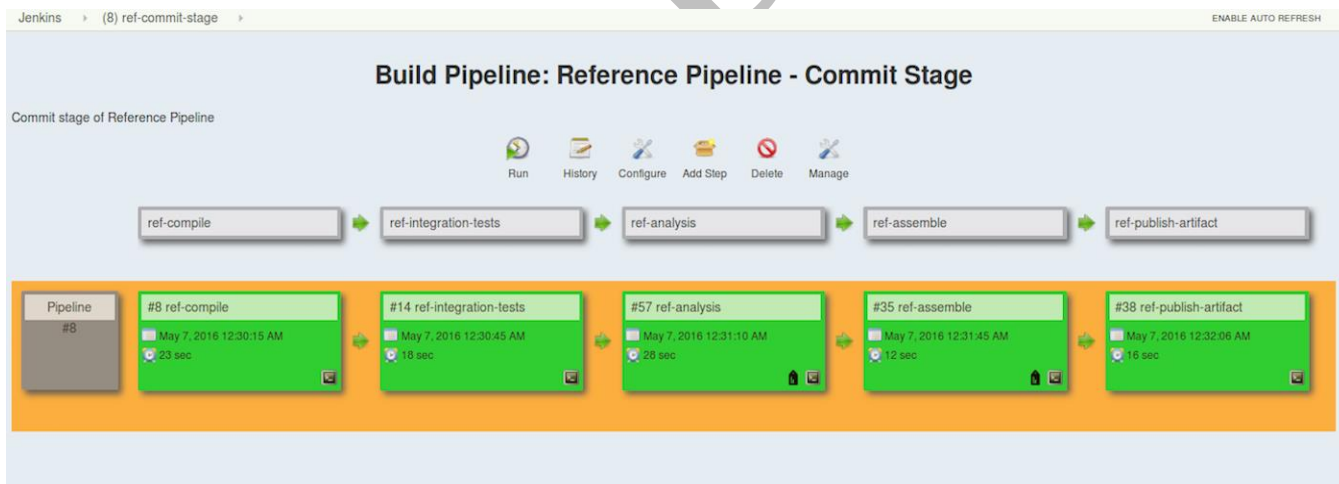
☐ My View
This view automatically displays all the jobs that the current user has an access to.

OK

Build Queue
No builds in the queue.

Build Executor Status
1 Idle
2 Idle

Page generated: Aug 6, 2019 5:01:09 AM PDT [REST API](#) Jenkins ver. 2.187



Executing the build the parallel

There is an option called “Execute concurrent builds if necessary”, which allows you to run multiple builds in parallel.

Use case: Multiple developer are working on the same stream. Before they deliver their changes they would like to trigger a personal build. In a huge project the build takes approx. 1 h. If every single developer is requisition a personal build with the same build definition there will be a large queue on jenkins side. That means, they have to wait a long time until their build is finished.

- ☐ Throttle builds
- ☐ Disable this project
- ☒ Execute concurrent builds if necessary

Discard old builds

It controls the disk consumption of Jenkins by managing how long you’d like to keep records of the builds (such as console output, build artifacts, and so on.) Jenkins offers two criteria: driven by age and driven by number.

☒ Discard Old Builds

Strategy

Log Rotation

Days to keep builds

15

if not empty, build records are only kept up to this number of days

Max # of builds to keep

15

if not empty, only up to this number of build records are kept

Days to keep artifacts

if not empty, artifacts from builds older than this number of days will be deleted,

Max # of builds to keep with artifacts

1

if not empty, only up to this number of builds have their artifacts retained

Jenkins Admin password

Recovering the Jenkins admin password

Step1: Go to Jenkins Home directory

Step2: Edit the config file and go to Use security. Change from true to false, restart the Tomcat server (It Means disabling the security)

Then after It won't ask credentials to login into Jenkins. It means any one can access it.

Note: We can't find Manage users Option

Step3: Go to Manage Jenkins and Configure Global Security.

Step4: Click on Enable Security, Click Jenkins Own user data base and Click on Logged user can do anything

And Save It

Note: Now we can able see Manage users Option

Now we are able see Signup page. sign up a user by default he get the Admin permission.

Step5: Again go to Configure Global Security and Click Role-Based Strategy save it.

Step6: Go to People find the admin user and change the admin password save it

Step7: Go to the Manage and Assign Roles and click on Assign Roles if admin user registered assign to Admin access to admin. If not there Create admin user and then assign admin access.

Configuring the Github web hooks

Jenkins

Pipeline

Jenkins Pipeline is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

Note: Pipeline start with "P"

A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the "Pipeline domain-specific language syntax." **DSL**

The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository.

This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

- Code review/iteration on the Pipeline (along with the remaining source code).
- Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

Why Pipeline

Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline

Jenkins Pipeline Advantages

- It models simple to complex pipelines as code by using **Groovy DSL** (Domain Specific Language)
- The code is stored in a text file called the Jenkinsfile which can be **checked into a SCM** (Source Code Management)
- Improves user interface by incorporating **user input** within the pipeline
- It is durable in terms of unplanned restart of the Jenkins master
- It supports complex pipelines by incorporating conditional loops, fork or join operations and allowing tasks to be performed in parallel
- It can integrate with several other plugins

What is a Jenkinsfile?

A Jenkinsfile is a text file that stores the entire workflow as code and it can be checked into a SCM.

How is this advantageous?

This enables the developers to **access, edit and check the code at all times**.

The Jenkinsfile is written using the Groovy DSL and it can be created through a text/groovy editor or through the configuration page on the Jenkins instance. It is written based on two syntaxes, namely:

1. **Declarative pipeline syntax**
2. **Scripted pipeline syntax**

Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.

Whereas, the scripted pipeline is a traditional way of writing the code. In this pipeline, the Jenkinsfile is **written on the Jenkins UI instance**.

Though both these pipelines are based on the groovy DSL, the scripted pipeline uses stricter groovy based syntaxes because it was the first pipeline to be built on the groovy foundation. Since this Groovy script was not typically desirable to all the users, the declarative pipeline was introduced to offer a simpler and more optioned Groovy syntax.

The declarative pipeline is defined within a block labelled '**pipeline**'

The scripted pipeline is defined within a '**node**'.

This will be explained below with an example.

Declarative

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.

```
pipeline {
}
```

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.

Any

Runs the pipeline/stage on any available agent.

```
pipeline {
  agent any
}
```

Specify Particular Node Machine

```
pipeline{
  agent {
    label "windows"
  }
}
```

Label

Executes the pipeline/stage on the labelled agent.

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.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        //
      }
    }
    stage('Test') {
      steps {
        //
      }
    }
    stage('Deploy') {
```

```

    steps {
      //
    }
  }
}

```

Examples (<https://jenkins.io/doc/book/pipeline/>)

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.

Example

```

pipeline {
  agent any
  stages {
    stage ('Build') {
      steps {
        echo 'Running build phase...'
      }
    }
  }
}

```

Installing Jenkins As a service

If you want install jenkins as a service for that we need to add jenkins repo on your machine Follow the below steps

Now we have to add Jenkins repository like below

To use this repository, run the following command:

```

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key

```

You will need to explicitly install a Java runtime environment, because Oracle's Java RPMs are incorrect and fail to register as providing a java dependency. Thus, adding an explicit dependency requirement on Java would force installation of the OpenJDK JVM.

```

yum install java-1.8.0-openjdk -y

```

Now Install Jenkins

```
yum install jenkins-2.187
```

Start the Jenkins server

```
# service jenkins start
```

Check in the browser like <http://ipaddress:8080>

NOTE: <https://pkg.jenkins.io/redhat/> --- All jenkins rpms available here

ANSIBLE:



Ansible is an open-source platform for CM, orchestration, provisioning and deployment of compute resources.

It manages resources with the use of SSH (Paramiko, a Python SSH2 implementation, or standard SSH).

Ansible supports and is available for CentOS and Red Hat Enterprise Linux, and it is also available as a commercial product by Ansible Inc.

It is built on the popular Python language. It is possible to install Ansible by using the Git repository clone of a master server.

Ansible is agentless. Its design goals are consistent, secure, minimal in nature and highly reliable, and it is easy to learn.

Its main features are:

Resources are added to the Ansible configuration.

SSH authorized keys or sudo credentials (root access is not needed) are needed for each managed compute resource based on the user.

Ansible master server communicates with the compute resources using SSH and performs all the necessary tasks.

Ansible deploys modules to compute resources.

Playbooks are configuration files in Ansible, which use YAML syntax. Ansible has a collection of modules to manage resources on various cloud platforms such as Amazon EC2 and OpenStack.

Ansible supports deployments on various virtualisation platforms as well as public and private cloud environments such as KVM, AWS, VMware, Eucalyptus Cloud, OpenStack and CloudStack.

It also supports deployment of Big Data and analytics environments such as Hadoop, Riak and Aerospike.

What is that Ansible can do?

Ansible can do the following for us:

- Configuration management
- Application deployment
- Task automation
- IT orchestration

CONFIGURATION MANAGMENT

Configuration management is everything that you need to manage in terms of a project. This includes software, hardware, tests, documentation, release management, and more.

Infrastructure as Code

Infrastructure as code is the act of describing what you want your servers to look like *once*, and using that to *provision* many machines to look the same

Why is Ansible better than shell scripting?

Parallel execution across multiple machines. Using the ad-hoc mode to run shell commands across many machines in parallel.

Automatic step-by-step reporting. Ansible encourages you to name each 'task' in your provisioning script, and then reports whether or not that task succeeded with-or-without changes, or failed, and any error messages. All colour coded. This is nice.

Tagging. You can tag your commands, making it easy to execute a subset of a provisioning script without extracting that section or commenting everything else out.

Composability:

Ansible tries to solve the problem of code re-use by formalising a set of conventions for building-blocks called roles, and putting together a sort of 'github for deployment-patterns' called Ansible Galaxy.

Types of Configuration management tools

Two types of configuration management tools available

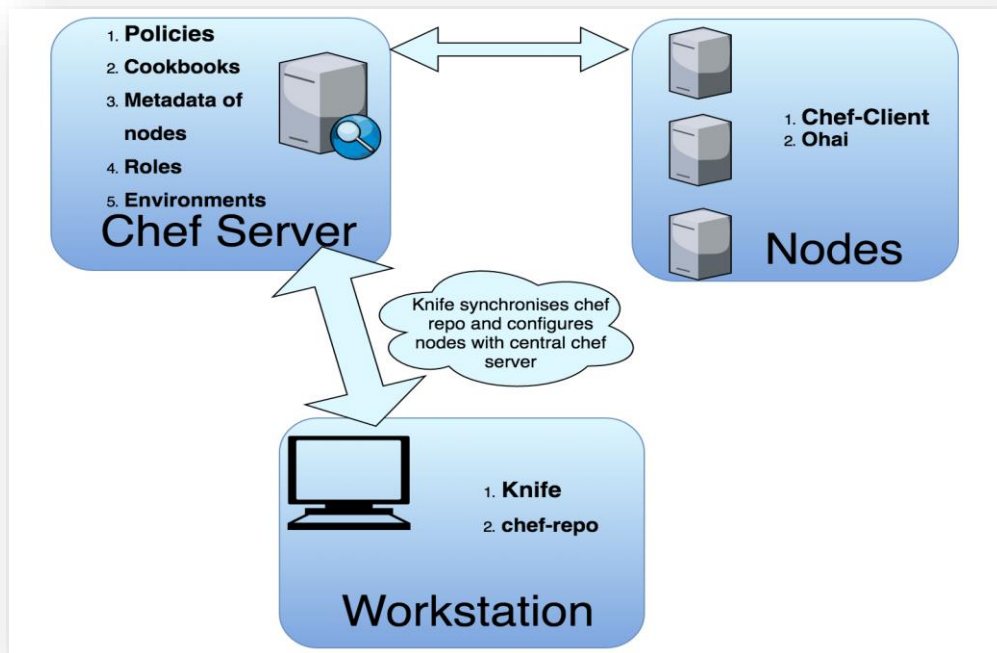
- Pull Model
- Push Model

Pull Model

The server being provisioned (node) runs an agent (daemon) that asks a central authority (master) if/when it has any updates that it should run.

Requires a daemon to be installed on all machines *and* a central authority to be setup.

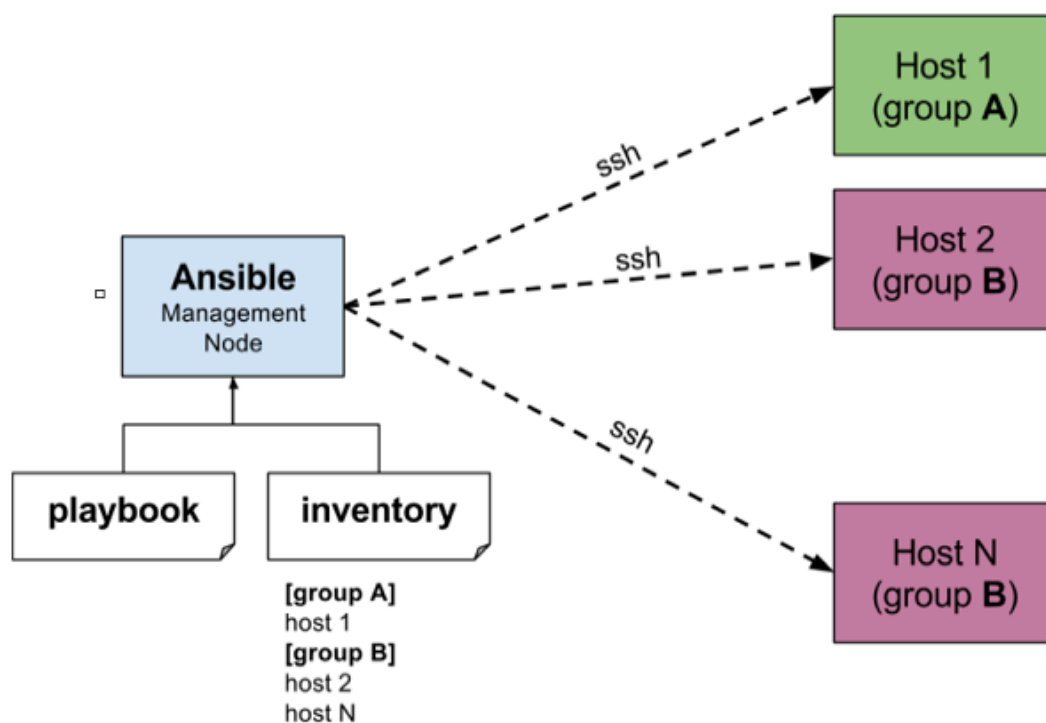
Examples: Chef, Puppet
Chef Workflow



Push Model

A central server contacts the nodes and sends updates as they are needed. When a change is made to the infrastructure (code) each node is alerted of this and they run the changes.

Examples: Ansible and Salt



Ansible Installation:

Please follow the below link for installations

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-ansible-on-rhel-centos-or-fedora

Checking ansible version

```
# Ansible --version
```

How Ansible works

Inventory:

The Ansible inventory file defines the hosts and groups of hosts. The default location for the inventory file is **/etc/ansible/hosts**. If necessary, you can also create project-specific inventory files in alternate locations.

In hosts file we will specify Node Ip Address or DNS Name like below

```
# vi /etc/ansible/hosts
[webserver]
192.168.249.131

[appserver]
192.168.249.139
```

We can group the group of groups like below

```
[production:children]
webserver
appserver
```

Types of Ansible Inventories:

- Static Inventory
- Dynamic Inventory

Static inventory is default inventory and is defined in the `/etc/ansible/ansible.cfg` file. Default file can be changed inside the `ansible.cfg` file.

If you want to use the custom file as inventory input can specify it using

"`-i /path/to/file`" with Ansible command line.

Static inventories are described. They don't change unless you make changes to them.

Dynamic Inventory

If you have the setup where you add and remove the hosts very frequently, then keeping your inventory always up-to-date become a little bit problematic. In such case Dynamic inventory comes into picture, generally are scripts (Python/Shell) for dynamic environments (for example cloud environments) With Ansible, as aforementioned, can use "`-i`" to specify the custom inventory file.

For example, if you use AWS cloud and you manage EC2 inventory using its Query API, or through command-line tools such as `awscli`, then you can make use of dynamic inventory,

Dynamic inventory got benefits over static inventories:

- Reduces human error, as information is collected by scripts.
- Very less manual efforts for managing the inventories.

Ansible has inventory collection scripts for the below platforms

- AWS EC2 External Inventory Script, Collber, OpenStack, BSD Jails, Google Compute Engine, and Spacewalk.

NOTE: Should be transfer ssh keys of where ansible installed to node machines because ansible completly rely on ssh keys only

After done the all above stuff it's time to check ansible master will successfully communicating with nodes machine for fire the below commands

```
# ansible -m ping "webserver"
192.168.249.139 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

The output would be like that, that means successfully communicating with nodes machines

NOTE: In Ansible, nothing happens without an inventory. Even ad hoc actions performed on localhost require an inventory, even if that inventory consists just of the Localhost.

The Inventory is the most basic building block of ansible Architecture.

When executing ansible or ansible-playbook an inventory must be referenced.

Ad-Hoc commands

What is Ad-hoc command

An ad-hoc command is something that you might type in to do something really quick, but don't want to save for later.

Managing Services

Ensure a service is started on all web servers:

```
# ansible webserver -m service -a "name=httpd state=started"
```

Alternatively, restart a service on all web servers:

```
# ansible webserver -m service -a "name=httpd state=restarted"
```

Ensure a service is stopped:

```
# ansible webserver -m service -a "name=httpd state=stopped"
```

Create a Directory

```
# ansible localhost -m file -a "dest=/opt/optional state=directory"
```

Create a file

```
# ansible localhost -m file -a "dest=/opt/optional.txt state=touch"
```

Checking the Load average

```
# ansible webserver -m shell -a "uptime"
```

Managing Packages

There are modules available for yum and apt. Here are some examples with yum.

Ensure a package is installed, but don't update it

```
# ansible webserver -m yum -a "name=httpd state=present"
```

Ensure a package is installed to a specific version

```
# ansible webserver -m yum -a "name=acme-1.5 state=present"
```

Ensure a package is at the latest version

```
# ansible webserver -m yum -a "name=acme state=latest"
```

Ensure a package is not installed

```
# ansible webserver -m yum -a "name=acme state=absent"
```

Transferring file to many servers/machines

```
# Ansible webserver -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"
```

NOTE:

Ansible is the executable for doing ad-hoc one-task executions

Ansible-playbook is the executable that will process playbooks for orchestrating many tasks.

SETUP

Gathers facts about remote hosts (Ip address of Machine, ansible_architecture, ansible_bios_date, ansible_distribution, ansible_distribution_version, domain, ansible_env (What you export the paths in hosts machine like tomcat, java paths, if they already installed)

Gathering Facts ansible will use "setup" Module.

- This module is automatically called by playbooks to gather useful variables about remote hosts that can be used in playbooks. It can also be executed directly by /usr/bin/ansible to check what variables are available to a host. Ansible provides many *facts* about the system, automatically.
- This module is also supported for Windows targets.

If you specify "gather_facts: no" it will skip the setup module So that the execution of playbook is very speed (Setup Module Takes more time to gathering facts about hosts)

How to run setup Module

```
# ansible -i hosts db -m setup
```

(or)

```
# ansible web -m setup
```

Playbook

Playbooks are Ansible's configuration, deployment, and orchestration language
(or)

A playbook contains place, a play contains different tasks and a tasks contains modules and when you run a playbook it's actually the module that gets executed on your target machines.

Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. YAML stands for Yet Another Markup Language. Playbooks are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks. Playbooks contain the steps which the user wants to execute on a particular machine. **Playbooks are run sequentially.**

Playbook is divided into 3 Sections

1. Target Section: It Defines on which node/nodes will be executed.
2. Variable section: Define variables which can be used from the playbooks. It's optional based on project.
3. Tasks Section: List all the modules that you intend to run in the order.

Module

Bits of code copied to the target system. Executed to satisfy the task declaration.
Customizable.

(or)

It is kind of action EX: yum, apt-get, service

Playbook Structure

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.

The function of a play is to map a set of instructions defined against a particular host.

Let's look at a basic playbook:

Sample Playbook for Creating Directory

```
# vi dir.yml
```

```
- hosts: app
  tasks:
  - name: ansible create directory example
    file:
      path: /tmp/devops_directory
      state: directory
```

Now it is going to create a devops_directory in app node machine. Let's see how to execute this playbook

```
# ansible-playbook dir.yml      (out put given below)
PLAY [app] *****
TASK [Gathering Facts] *****
ok: [192.168.249.139]
TASK [ansible create directory example] *****
changed: [192.168.249.139]
PLAY RECAP *****
192.168.249.139      : ok=2  changed=1  unreachable=0  failed=0
```

In the above task, the directory will be created with the default permission. We can set the permissions using the 'mode' parameter. We can give it in two ways.

- the symbolic form like 'u=rw,g=rw,o=rw' -> This gives read and write permission to everyone.
- Octal numbers like '0777' -> Read, write and execute permission to everyone

The following task sets the permission of already created devops_directory to 'u=rw,g=wx,o=rwx.'

```
- hosts: all
  tasks:
  - name: ansible create directory with mode setting example
    file:
      path: /tmp/devops_directory
      state: directory
      mode: "u=rwx,g=wx,o=rwx"
```

output

```
drw--wxrwx. 2 root  root  4096 Aug 18 19:54 devops_directory
```

You can see the permissions of the devops_directory has changed as given on the task. But the files created inside that folder has the default permissions set.

Changing the Permissions for Directory and the files

You can modify the permissions of a directory and all the files inside also recursively. You can use the above task itself. You just have to add the recurse parameter and set it to 'yes.'

```
# touch /tmp/devops_directory/test1
# touch /tmp/devops_directory/test2
output
-rw-rw-r-- 1 mdtutorials2 mdtutorials2 o Oct  4 09:35 /tmp/devops_directory/test1
-rw-rw-r-- 1 mdtutorials2 mdtutorials2 o Oct  4 09:35 /tmp/devops_directory/test2
```

```
# vi dir.yml
```

```
- hosts: all
  tasks:
    - name: ansible set permission recursively for a directory
      file:
        path: /tmp/devops_directory
        state: directory
        mode: "u=rw,g=wx,o=rwx"
        recurse: yes
```

output:

```
# ls -lrt /tmp/devops_directory/test2
-rw--wxrwx 1 root root o Oct  4 09:48 /tmp/devops_directory/test1
-rw--wxrwx 1 root root o Oct  4 09:48 /tmp/devops_directory/test2
```

Create multiple directories using "with_items"

You can also create multiple directories using the with_items statement in Ansible.

For example, to create three directories, devops_system1, devops_system2, and devops_system3, you can execute the following task

```
- hosts: all
  tasks:
    - name: ansible create multiple directory example
      file:
        path: "{{ item }}"
        state: directory
      with_items:
        - '/tmp/devops_system1'
        - '/tmp/devops_system2'
        - '/tmp/devops_system3'
```

output

=====

```
mdtutorials2@system01:~$ ls -lrt /tmp
```

total 16

```
drwxr-xr-x 2 root    root    4096 Oct  4 09:59 devops_system1
drwxr-xr-x 2 root    root    4096 Oct  4 09:59 devops_system2
drwxr-xr-x 2 root    root    4096 Oct  4 09:59 devops_system3
```

But what if you need to set the permission differently for each directory while using "with_items"

In the following task, I am independently setting the modes for each directory.

```
- hosts: all
tasks:
- name: ansible create directory with _items example
  file:
    path: "{{ item.dest }}"
    mode: "{{ item.mode }}"
    state: directory
  with_items:
    - { dest: '/tmp/devops_system1', mode: '0777' }
    - { dest: '/tmp/devops_system2', mode: '0707' }
    - { dest: '/tmp/devops_system3', mode: '0575' }
```

output

=====

mdtutorials2@system01:~\$ ls -lrt /tmp/

total 16

drwxrwxrwx 2 root root 4096 Oct 4 09:59 devops_system1

drwx---rwx 2 root root 4096 Oct 4 09:59 devops_system2

dr-xrwxr-x 2 root root 4096 Oct 4 09:59 devops_system3

Installing Multiple applications at a time for that with _items. Let's See below example

```
---
- hosts: web
  tasks:
  - name: Install Multiple applications at a Time
    package:
      name: "{{ item }}"
      state: present
    with_items:
      - git
      - tree
      - java
      - httpd
```

Creating a local directory using local_action statement

You can also create a local directory in ansible using the 'local_action' statement along with the given examples.

For example, to create a directory 'local_folder' in the Ansible control machine.

```
- hosts: all
tasks:
- name: ansible create local directory example
  local_action:
    module: file
    path: /tmp/local_file
    state: directory
```

Deleting a Directory in Ansible

You can delete a directory by setting the state parameter to absent. This will remove the directory and all its contents.

For example, to remove the '/tmp/devops/' directory, you can execute the following task.

```
dtutorials2@system01:~$ ls -lrt /tmp/devops_directory/
total 0
-rw--wxrwx 1 mdtutorials2 mdtutorials2 0 Oct  4 09:35 test1
-rw--wxrwx 1 mdtutorials2 mdtutorials2 0 Oct  4 09:48 test2
```

```
- hosts: all
tasks:
- name: ansible remove directory example
  file:
    path: /tmp/devops_directory
    state: absent
```

output

=====

```
mdtutorials2@system01:~$ ls -lrt /tmp/devops_directory/
ls: cannot access '/tmp/devops_directory/': No such file or directory
```

Apache Playbook (Installing) Installing Apache Webserver

vi apache.yml

```
---
- hosts: webserver
  tasks:
    - name: Installing Apache Webserve
      yum: name=httpd state=present

    - name: Enable Apache on System Boot
      service: name=httpd enabled=yes

    - name: Start The Apache Server
      service: name=httpd state=started
```

Note: yml or yaml both are same.

Executing the ansible playbook

ansible-playbook apache.yml (playbook name)

or

ansible-playbook -i (Path of Inventory file) playbook name

Template

Templates are simple text files that we can use in Ansible. Most of the time you will use them to replace configuration files or place some documents on the server.

Let's say that we want to change the index.html of Apache. We will use simplest way and we will just replace whole index.html file. Inside playbook directory create a file and name it for instance index.html.j2. J2 is extension of Jinja2 templating language that Ansible is using.

Syntax:

- name: Template a file to /etc/files.conf

template:

src: /mytemplates/foo.j2

dest: /etc/file.conf

Scenario

Nginx Installation, change the port number and Change the root documentary (Need to Get Customized Welcome Page)

To Install Nginx

```
---
- hosts: webserver
  tasks:

    - name: Install Epel-repo
      yum: name=epel-release state=present

    - name: Installing Nginx Webserver
      yum: name=nginx state=present

    - name: Start The Nginx Server
      service: name=nginx state=started

    - name: changing the port Number
      template:
        src: /root/Desktop/ansible/default.conf.j2
        dest: /etc/nginx/conf.d/default.conf

    - name: Restart the Nginx
      service: name=nginx state=restarted

    - name: Changing the Root Documentary
      template:
        src: /root/Desktop/ansible/index.html
        dest: /usr/share/nginx/html/

    - name: Restart the Nginx
      service: name=nginx state=restarted
```

Let's break this down in sections so we can understand how these files are built and what each piece means

This is a requirement for YAML to interpret the file as a proper document. YAML allows multiple "documents" to exist in one file, each separated by ---, but Ansible only wants one per file, so this should only be present at the top of the file.

YAML is very sensitive to white-space, and uses that to group different pieces of information together. You should use only spaces and not tabs and you must use consistent spacing for your file to be read correctly. Items at the same level of indentation are considered sibling elements.

Items that begin with a - are considered list items. Items that have the format of key:

Each playbook is composed of one or more 'plays' in a list. **means host info**

Handlers:

Handlers are just like regular tasks in an Ansible playbook.

But are only run if the Task contains a notify directive and also indicates that it changed something.

```
---
- hosts: webserver
  tasks:
    - name: Installing Nginx Webserver
      yum: name=nginx state=present
      notify:
        - Start Nginx

    - name: changing the port Number
      template:
        src: /root/Desktop/ansible/default.conf.j2
        dest: /etc/nginx/conf.d/default.conf
      notify:
        - Start Nginx

    - name: Changing the Root Documentory
      template:
        src: /root/Desktop/ansible/index.html.j2
        dest: /usr/share/nginx/html/index.html
      notify:
        - Start Nginx

  handlers:
    - name: Start Nginx
      service: name=nginx state=started
```

The "notify" item contains a list with one item, which is called "start nginx". This is not an internal Ansible command, it is a reference to a handler, which can perform certain functions when it is called from within a task. We will define the "start nginx" handler below.

The "handlers" section exists at the same level as the "hosts" and "tasks". Handlers are just like tasks, but they only run when they have been told by a task that changes have occurred on the client system.

TAGS

Controlling on an execution of playbook. Let's say you want run a portion of playbook that can be done by tags

Tags are great way to test a bunch of tasks without executing the complete playbook.

Ex:

```
---
- hosts: webserver
  tasks:

  - name: Installing Nginx Webserver
    yum: name=nginx state=present
    tags: install

  - name: Start The Apache Server
    service: name=nginx state=started
    tags: started

  - name: changing the port Number
    template:
      src: /root/Desktop/ansible/default.conf
      dest: /etc/nginx/conf.d/
    tags: port
```

Now how to control the palybook

```
# ansible-playbook playbook.yml --tags install
```

It will just install nginx only

```
# ansible-playbook playbook.yml --tags "install, start"
```

It will install and start the nginx server

```
# ansible-playbook playbook.yml --skip-tags "install"
```

It will skip the install portion

NOTE: (Troubleshoot/Debug)

Debugging purpose or verbose we should use **"-vvv"**

```
# ansible-playbook playbook_name -vvv
```

It gives information step by step what is going on remote machine (or) host.

NOTE:

When we execute a playbook, ansible first check the syntax of the playbook.

Syntax checking is done only when you use "ansible-playbook" command. It's not a ad-hoc command.

Ex:

```
# Ansible-playbook playbook name --syntax-check
```

Another Mode (--check)

Check mode it's like a dry run, it will not apply anything it will nearly show you what will happen if you apply

EX:

```
# ansible-playbook playbook_name --check
```

SAFELY LIMITING ANSIBLE PLAYBOOK TO A SINGLE MACHINE

My inventory file would be like below

```
[appserver]
```

```
Linux-1.local_Machine
```

```
Linux-2.local_Machine
```

```
Linux-3.local_Machine
```

To run a playbook on signal machine in appserver group

EX: # ansible-playbook --limit Linux-2.local_Machine dir.yml

INSTALLING APACHE ON MULTIPLE FLAVOURS OF LINUX WITH SINGLE PLAYBOOK

Here we can use loop conditions. The Example playbook given below. This playbook reference to install apache

```
---
- hosts: web
  gather_facts: yes
  tasks:
    - name: Install apache when OS_Family=Redhat
      yum:
        name: httpd
        state: present
      when: ansible_os_family == "RedHat"

    - name: install Apache2 when OS_Family=Debian
      apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"
```

SPECIFYING THE VARS IN PLAYBOOK.

Let's see how will specify in playbook

```
---
- hosts: web
  vars:
    package_name: httpd

  tasks:
    - name: installing the variable value
      yum:
        name: "{{ package_name }}"
        state: present
```

Passing extra variable while running the playbook

```
# ansible-playbook playbook_name --extra-vars "package_name=tree"
```

Or

```
# ansible-playbook playbook_name -e "package_name=tree"
```

Create ansible playbook installing apache and tree each application on different machines at a time

```

---
- name: installing apache
  hosts: 192.168.249.156
  tasks:
    - name: installing apache
      yum:
        name: httpd
        state: present

- name: Installing Tree
  hosts: 192.168.249.157
  tasks:
    - name: installing tree
      yum:
        name: tree
        state: present

```

Vault

The “Vault” is a feature of Ansible that allows you to keep sensitive data such as passwords or keys protected at rest, rather than as plaintext in playbooks or roles.

There are 2 types of vaulted content and each has their own uses and limitations:

Vaulted files

- The full file is encrypted in the vault, this can contain Ansible variables or any other type of content.
- It can be used for inventory, anything that loads variables (i.e vars_files, group_vars, host_vars, include_vars, etc)

Single encrypted variable:

- Only specific variables are encrypted inside a normal ‘variable file’.
- Decrypted on demand, so you can have vaulted variables with different vault secrets and only provide those needed.
- You can mix vaulted and non vaulted variables in the same file, even inline in a play or role.

Example:

Creating Encrypted Files

To create a new encrypted data file, run the following command:

```
# ansible-vault create foo.yml (foo.yml=playbook name)
```

First you will be prompted for a password. The password used with vault currently must be the same for all files you wish to use together at the same time.

After providing a password, the tool will launch whatever editor you have defined with \$EDITOR, and defaults to vi (before 2.1 the default was vim). Once you are done with the editor session, the file will be saved as encrypted data.

Running a Playbook with Vault

To run a playbook that contains vault-encrypted data files, you must provide the vault password.

To specify the vault-password interactively:

```
# ansible-playbook site.yml --ask-vault-pass
```

It prompted for password then enter password. Now it will execute

Passing the password within the file instead of passing on CLI

Using "--vault-password-file"

Before run, save your vault password in a file and the playbook again

```
# cat >> vault-passwd  
padma
```

Now vault password is stored in a file called vault-passwd

```
# ansible-playbook users.yml --vault-password-file <path of vault-passwd>
```

This time vault password will be taken from the file you have provided hence it won't prompt you to enter the vault passwd

Decrypting Encrypted Files:

If you have existing files that you no longer want to keep encrypted, you can permanently decrypt them by running the ansible-vault decrypt command.

```
# ansible-vault decrypt apache.yml
```

Create vault for already existing file

```
# ansible-vault encrypt foo.yml (If already playbook written)
```

Viewing Encrypted Files

If you want to view the contents of an encrypted file without editing it, you can use the ansible-vault view command:

```
# ansible-vault view apache.yml
```

Editing Encrypted Files

When you need to edit an encrypted file, use the ansible-vault edit command:

```
# ansible-vault edit foo.yml
```

You will be prompted for the file's password. After entering it, Ansible will open the file in an editing window, where you can make any necessary changes.

Upon saving, the new contents will be encrypted using the file's encryption password again and written to disk.

Changing the Password of Encrypted Files

If you need to change the password of an encrypted file, use the `ansible-vault rekey` command:

```
# ansible-vault rekey foo.yml
```

Ask you for old Password After you need to enter a new password, confirm it again

Encrypting specific variables

Best practice while using Ansible Vault is to encrypt only the sensitive data. In the example explained above, the development team does not want to share their password with the production and the staging team but they might need access to certain data to carry out their own task. In such cases you should only be encrypting the data you do not want to share with others, leaving the rest as it is.

Ansible Vault allows you to encrypt only specific variables. You can use the **`ansible-vault encrypt_string`** command for this.

```
# ansible-vault encrypt_string <string>
```

You'll be prompted to insert and then confirm the vault password. You can then start inserting the string value that you wish to encrypt. Press `ctrl-d` to end input. Now you can assign this encrypted value to a string in the playbook.

Register

Ansible register is a way to capture the output from task execution and store it in a variable. This is an important feature, as this output is different for each remote host, and the basis on that we can use conditions loops to do some other tasks. Also, each register value is valid throughout the playbook execution.

So, we can make use of `set_fact` to manipulate the data and provide input to other tasks accordingly

Example:

```
--
- hosts: webserver
  tasks:
    - name: System Uptime
      command: uptime
      register: out_put
    - debug: var=out_put
or
    - debug: var=uptime.stdout
if you have multiple registries then you can include loop here. Check the below example
- debug: var={{ item }}
  loop:
    - uptime.stdout
    - host or host.stdout
```

lineinfile

Ansible lineinfile module can be used to insert a line, modify an existing line, remove an existing line or to replace a line.

The line 'Happy Independence day' to the file 'myfile_server.tx'. The new line will be added to the EOF. If the line already exists, then it won't be added.

```
hosts: webserver
```

```
tasks:
```

```
- name: Ansible insert lineinfile example
```

```
  lineinfile:
```

```
    dest: /root/myfile_server.txt
```

```
    line: Happy Independence Day.
```

```
    state: present
```

```
    create: yes
```

We have also set the create parameter, which says if the file is not present then create a new file. The default value for the state is present. But I am adding it anyway for clarity.

Insert a line before a pattern

If you need to insert a line before a pattern, you can use the insertbefore parameter. The following example will insert the line before the pattern '#library' in ansible.cfg.

```
- name: Ansible lineinfile insertbefore example
```

```
  lineinfile:
```

```
    dest: /etc/ansible/ansible.cfg
```

```
    line: 'inventory = /home/mdtutorials/inventory.ini'
```

```
    insertbefore: '#library'
```

Removing a line using Ansible regexp

You can also specify a regexp to remove a line. So you can say remove all lines that start with the word 'Helloworld' etc.

We give the regular expression using lineinfile regexp parameter. The following example will remove all lines starting with Helloworld.

```
- hosts: loc
```

```
tasks:
```

```
- name: Ansible lineinfile regexp example
```

```
  lineinfile:
```

```
    dest: /room/devops_server.txt
```

```
    regexp: '^DevOps'
```

```
    state: absent
```

Note: groups (This command helps you find user's group who logged in machine)

update cache=yes

```
---
```

```
- hosts: apache
```

```
tasks:
```

```
- name: install apache2
```

```
  apt: name=apache2 state=latest update_cache=yes
```

"apt or yum" maintains a local list of packages; that's how it "knows" what packages are available, their dependencies etc. apt or yum update updates these lists of packages by retrieving them from the repositories; it doesn't upgrade any package.

Downloading the packages using below module

get_url:

Downloads files from HTTP, HTTPS, or FTP to node.

- hosts: webserver

tasks:

- name: Download the package from the web

get_url:

url: https://storage.googleapis.com/golang/go1.8.4.linux-amd64.tar.gz

dest: /var/tmp/

NOTE

We could simply run the command `ansible-doc -l` on your ansible system.

`ansible-doc -l` (To Show the all available Module)

To show particular module

`ansible-doc file`

To show all nodes machines fire the below command

\$ `ansible-playbook playbook_name --list-host`

(By default it will goes to `/etc/ansible/hosts` file bcz default inventory file).

It will not perform anything on Node machines except counting the servers

Let's assume inventory file present in your working directory

\$ `ansible -i (Host address file name) --list-host all`

How to connect node machine with root password

\$ `ansible -m ping "testserver" -u user -k`

How can we list all tasks of a playbook ?

Run `ansible-playbook` using the `--list-tasks` flag and Ansible will list all its tasks

`ansible-playbook (playbook name) --list-tasks`

or

`ansible-playbook -i hosts apache.yml --list-tasks`

ROLES

Roles are ways of automatically loading certain `vars_files`, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.

or

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

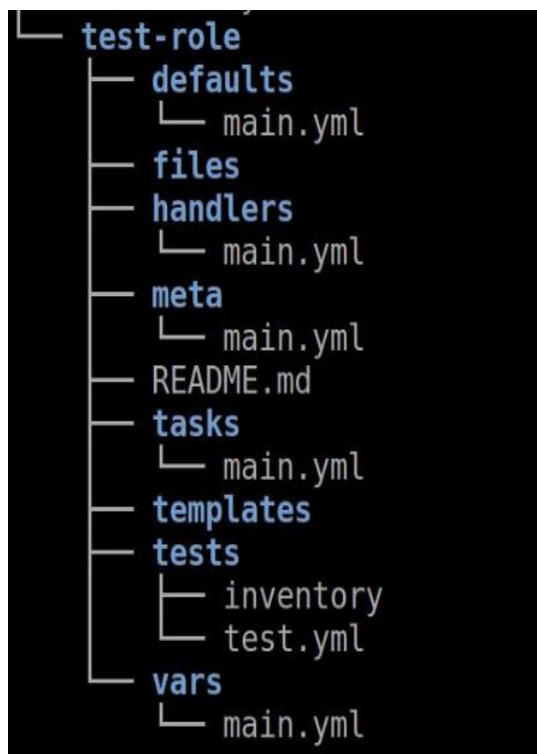
The role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

To create Role

ansible-galaxy init <Role name> or ansible-galaxy init test-role



Let's have a look into directory structure

defaults: contains default variables for the role. Variables in default have the lowest priority so they are easy to override.

vars: contains variables for the role. Variables in vars have higher priority than variables in defaults directory.

tasks: contains the main list of steps to be executed by the role.

files: contains files which we want to be copied to the remote host. We don't need to specify a path of resources stored in this directory.

templates: contains file template which supports modifications from the role. We use the Jinja2 templating language for creating templates.

meta: contains metadata of role like an author, support platforms, dependencies.

handlers: contains handlers which can be invoked by “notify” directives and are associated with service.

Ansible Galaxy

Ansible Galaxy refers to the Galaxy website where users can share roles, and to a command line tool for installing, creating and managing roles.

Galaxy, is a free site for finding, downloading, and sharing community developed roles. Downloading roles from Galaxy is a great way to jumpstart your automation projects.

The command line tool

The `ansible-galaxy` command comes bundled with Ansible, and you can use it to install roles from Galaxy or directly from a git based SCM. You can also use it to create a new role, remove roles, or perform tasks on the Galaxy website.

By default Ansible downloads roles to the first writable directory in the default list of paths `~/.ansible/roles`

This will install roles in the home directory of the user running "`ansible-galaxy`"

You can override this by setting the environment variable **ANSIBLE_ROLES_PATH** in your session, defining `roles_path` in an `ansible.cfg` file, or by using the `--roles-path` option.

The following provides an example of using `--roles-path` to install the role into the current working directory:

```
$ ansible-galaxy install --roles-path . geerlingguy.apache
```

version

You can install a specific version of a role from Galaxy by appending a comma and the value of a GitHub release tag. For example:

```
$ ansible-galaxy install geerlingguy.apache,v1.0.0
```

It's also possible to point directly to the git repository and specify a branch name or commit hash as the version. For example, the following will install a specific commit:

```
$ ansible-galaxy install git+https://github.com/geerlingguy/ansible-role-apache.git
```

List installed roles

Use `list` to show the name and version of each role installed in the `roles_path`.

```
$ ansible-galaxy list
```

Remove an installed role

Use `remove` to delete a role from `roles_path`:

```
$ ansible-galaxy remove username.role_name
```

What is the difference between ansible playbook and roles?

Ansible playbook is a script file which contains all the tasks that need to be performed along with all the ingredients required to perform these tasks.

Roles are ways of automatically certain var files, tasks, and handlers based on the known file structure.

where are ansible logs stored

Ansible doesn't create it's own logs by default - you have to tell it to do so, using an **ansible.cfg** file. Ansible does do *some* logging to `syslog` by default:

[defaults]

log_path = ./ansible.log

Error handling

When return code is “0” of specific task ignoring that specific task

Hosts: localhost

Tasks:

-name: list of files

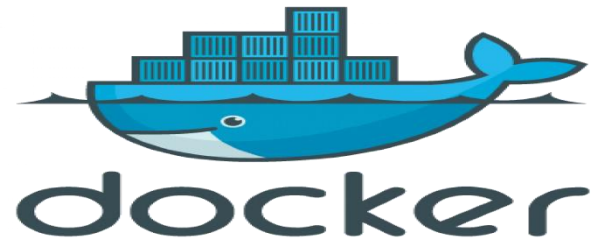
- command: “ls /opt”

register: home_out

- debug: var=home_out

ignore_errors: yes Add second below

Docker:



Docker is a containerization platform that packages your application and all its dependencies together in the form of Containers to ensure that your application works seamlessly in any environment.

- Each application will run on a separate container and will have its own set of libraries and dependencies.
- It also ensures that there is process level isolation, meaning each application is independent of other applications, giving developers the surety that they can build applications that will not interfere with one another.

This is a containerization platform which can be used for creating the development environment, testing environment and production environment....

Docker use a concept called containers. This is the next step in virtualization.

What is the use of Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Difference between Virtualization vs Containerization Virtualization:

In virtualization we can create multiple VM's on one host o/s. This is done by using a software called hypervisor.

Virtualization is the technique of importing a Guest operating system on top of a Host operating system.

In virtualization we have bear metal on this we install the host o/s. on the host we install a software called as hypervisor.

Ex: VMware and oracle vm, etc.

Advantages:

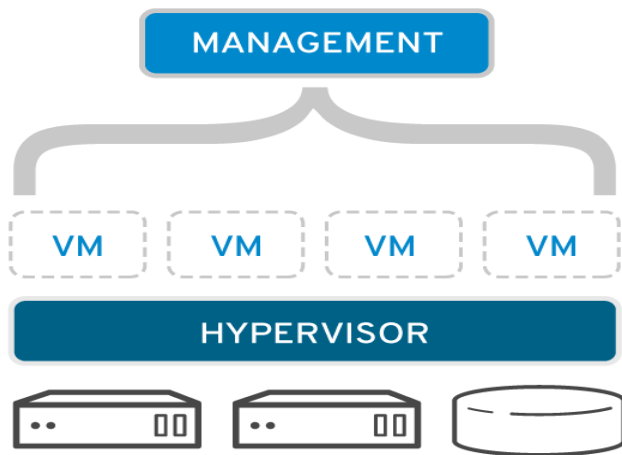
- Multiple operating systems can run on the same machine
- Maintenance and Recovery were easy in case of failure conditions

Disadvantages:

- Running multiple Virtual Machines leads to unstable performance because of the guest OS running on top of the host OS, which will have its own kernel and set of libraries and

dependencies. This takes up a large chunk of system resources, i.e. hard disk, processor and especially RAM.

- Boot up process is long and takes time



Containerization:

Containerization is the technique of bringing virtualization to the operating system level

Containerization is more efficient because there is no guest OS here and utilizes a host's operating system, share relevant libraries & resources as and when needed, unlike virtual machines.

Application specific binaries and libraries of containers run on the host kernel, which makes processing and execution very fast.

They are lightweight and faster than Virtual Machines.

We have a docker engine and on this docker engine the container run on separate process.

These containers not to be allocated and fixed amount of H/W resources.

During the runtime based on the usage of each container docker engine automatically assign the necessary amount H/W resources.

The allocation hardware and o/s resources to these containers is automatically done by docker engine.

Ex: if docker container is using less amount of RAM and another docker container is running multiple container and it requires more RAM.

The docker engine will automatically reduce the amount of RAM for the first container and increase the RAM of the 2nd the RAM

Since container are individual process, they consume less amount of memory, and cpus
And creating and remover containers can also be done very quickly .
they are two current editions of docker

Docker solves this problem by using containers. A container is an indivual process running separately in a user space.

- 1) Community edition
- 2) Enterprise edition

Note: containers are not to be an individual process. Which is run in user spaces.

Docker uses Kernel features such as Cgroups and namespace to allow an independent container to run on single os instance.

Installation

For Installation better to official Docker website. Please find the below link.

<https://docs.docker.com/engine/install/>

Docker Basic commands

docker --version

This will display version of docker installed on our machine

docker info

This will display the information about docker engine and also information about system like below

How many images available,

Stopped, paused, running, volume, swarm enabled or not? Architecture, Total memory.

docker help

This will give the list of all the commands that can be used in docker

The help can also be used for finding information about application docker commands

Eg:1 # docker search --help

Eg:2 # docker rm --help

Images and containers:

Image:

A Docker image is a file, comprised of multiple layers, used to execute code in a Docker container. An image is essentially built from the instructions for a complete and executable version of an application, which relies on the host OS kernel. When the Docker user runs an image, it becomes one or multiple instances of that container.

Container:

Running instances of on image is called “**container**”.

All the docker images present in a site called hub.docker.com

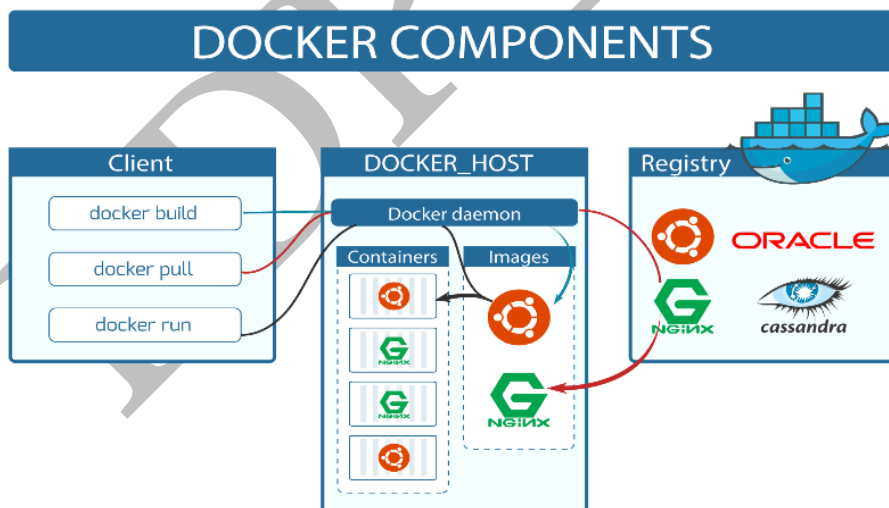
We can download the images & customize them according to our requirement.

Components of docker

Docker contains 3 main components

- 1) Docker client [DC]
- 2) Docker daemon
- 3) Docker registry (or) hub

Docker Architecture



DC (docker client): It is a location where we execute docker commands.

DD (Docker daemon): It contains is docker engine and it also contains the containers and images.

All the commands we execute through docker client will be received by docker engine.

The docker engine sends these commands to respective images or containers and perform the action.

DH (or) DR: this is the cloud site of docker where docker community has uploaded different images.

The docker engine can download the image from this registry.

Important docker commands:

1) To pull the image

docker pull image:tag

2) To view all images

docker images or docker image ls

3) To remove/delete image

docker rmi image:tag

4) To delete multiple images

docker rmi \$(docker images -q)

5) To start a container

docker run image_name/Id

6) To stop the container

docker stop container-name/Id

Or

docker kill container-name

7) To see all running container

docker container ls

or

docker ps -a

8) To restart the container

docker restart container-name

9) To restart after 20 seconds

docker restart -t 20 container-name

10) To delete container

docker rm container-name/container id

11) To delete multiple containers at a time

docker rm \$(docker ps -a -q)

12) To get into a container which is already running

docker exec -it container-name/id /bin/bash

13) To see the ports of a containers

docker port container_name/container id

14) To get info about container like Ip address etc..

docker inspect containerid/container name

15) To get info about image

docker inspect image id/image name

8) To find all available networks

docker network ls

9) To inspect

docker network inspect network-name

10) To inspect a specific container

docker inspect container-name

11) To create a new N/W

docker network create my_network

12) To connect to a N/W

docker network connect network_id container_id

13) To disconnect from N/W

docker network disconnect n/w_id container_id

14) To display logs container-name

docker logs container_name

Default location

(/var/lib/**docker**/containers/[container-id]/[container-id]-json.**log**)

15) To display last 100 logs of a container

docker logs tail -100 container_name

17. To Display volumes of docker

```
# docker volumes ls
```

18. Pause the container

```
# docker pause container name
```

and for unpause

```
# docker unpause container name
```

Section-I:

Build commands

docker build—The docker build command builds Docker images from a Dockerfile and a “context”. A build’s context is the set of files located in the specified PATH or URL. Use the -t flag to label the image,

for example

```
# docker build -t my_container . ( t= Tag Name )
```

with the . at the end signalling to build using the currently directory.

2) To show list of all images in docker engine

```
# docker images
```

3) Delete images from the local images

```
# docker rmi image_name
```

- Use them together, for example to clean up all your docker images and containers:
- kill all running containers with

```
# docker kill $(docker ps -q)
```
- Delete all stopped containers with

```
# docker rm $(docker ps -a -q)
```
- Delete all images with

```
# docker rmi $(docker images -q)
```

4) Rename a local image

```
# docker tag <original image_name> <new_image_name>
```

Section-II(ship commands)

1) pull on image from the registry

```
# docker pull image_name
```

or

```
# docker pull image_name: latest or Specify version Ex(6.0)
```

Run commands uses is multiple options

- a) `-rm` :remove the containers after we exit from it.
- b) `-it`: connects the containers to an interactive terminal window. We can execute Linux commands.
- c) `--name`: to give a customized name to our containers.
- d) `-p`: `-p 5000:5000` this will expose the containers internal port 5000 to host machine port 5000.
- e) `-v` : this is used for mounting volumes
- f) `-d` : to run detached mode `-detach`
- g) `-P`:publish the port numbers, it will assign port numbers in randomly
- h) `-e` : to pass environment variables to containers (settings)
- i) `-a`: to attach the container to STDIN or STDOUT

19. What is docker attach?

Attach local standard input, output, and error streams to a running container

Attach to and detach from a running container

Example:

```
# docker run -d --name topdemo ubuntu /usr/bin/top -b  
  
# docker attach topdemo
```

Get the exit code of the container's command

When containers are run with the interactive option, you can connect to the container and enter commands as if you are on the terminal

```
$ docker run -itd --name busybox busybox  
dcaecf3335f9142e8c70a2ae05a386395b49d610be345b3a12d2961fccab1478  
  
$ docker attach busybox  
/ # echo hello world  
hello world
```

Pushing Docker images to Dockerhub

- Create Dockerfile

vi Dockerfile

```
#This is a sample Image
ARG VERSION=latest
FROM ubuntu: $VERSION
MAINTAINER demousr@gmail.com
RUN apt-get update
RUN apt-get install -y nginx
CMD ["echo","Image created"]
```

FROM

FROM instructions support variables that are declared by any ARG instructions that occur before the first FROM.

It tells docker, from which base image you want to base your image from. In our example, we are creating an image from the **ubuntu** image

MAINTAINER

The next command is the person who is going to maintain this image. Here you specify the **MAINTAINER** keyword and just mention the email ID.

RUN

The RUN instruction will execute any commands in a new layer on top of the current image.

RUN has 2 forms:

- RUN <command> (*shell* form, the command is run in a shell, which by default is /bin/sh - c on Linux or cmd /S /C on Windows)
- RUN ["executable", "param1", "param2"] (*exec* form)

ENV

The ENV instruction sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be **replaced inline** in many as well.

The ENV instruction has two forms. The first form, ENV <key> <value>, will set a single variable to a value. The entire string after the first space will be treated as the <value> - including

whitespace characters. The value will be interpreted for other environment variables, so quote characters will be removed if they are not escaped.

The second form, `ENV <key>=<value> ...`, allows for multiple variables to be set at one time. Notice that the second form uses the equals sign (=) in the syntax, while the first form does not. Like command line parsing, quotes and backslashes can be used to include spaces within values.

For example:

```
ENV myName="John Doe" myDog=Rex\ The\ Dog \  
myCat=fluffy
```

and

```
ENV myName John Doe  
ENV myDog Rex The Dog  
ENV myCat fluffy
```

ADD

The ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

```
ADD hom* /mydir/      # adds all files starting with "hom"  
ADD hom?.txt /mydir/  # ? is replaced with any single character, e.g., "home.txt"
```

When adding files or directories that contain special characters (such as [and]), you need to escape those paths following the Golang rules to prevent them from being treated as a matching pattern. For example, to add a file named `arr[o].txt`, use the following;

```
ADD arr[[o].txt /mydir/  # copy a file named "arr[o].txt" to /mydir/
```

COPY

The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.

Example

```
# copy index.html /usr/share/nginx/html
```

CMD

CMD sets default instruction and/or parameters, which is executed when we run a container out of image. which can be overwritten from command line when docker container runs.

Ex:

```
# vi Dockerfile
```

```
FROM ubuntu:16.04
```

```
CMD echo 'welcome to docker'
```

```
# docker build -t app .
```

Now if you run the docker container we will the output as "Welcome to docker"

Now I want override the CMD from command line/ While running the container

```
# docker run -it app Imageid echo "welcome to AWS"
```

OP: Welcome to Aws.

The CMD instruction has three forms:

- CMD ["executable","param1","param2"] (*exec form*, this is the preferred form)
- CMD ["param1","param2"] (as *default parameters to ENTRYPOINT*)
- CMD command param1 param2 (*shell form*)

There can only be one CMD instruction in a Dockerfile. If you list more than one CMD then only the last CMD will take effect.

ENTRYPOINT

Entry points also run time instruction and command line instruction. ENTRYPOINT command and parameters will not be overwritten from command line. Instead, all command line arguments will be added after ENTRYPOINT parameters.

For Reference: <https://www.youtube.com/watch?v=6lcYy09e7-o>

Note:

Both CMD and ENTRYPOINT instructions define what command gets executed when running a container.

USER

the USER instruction sets the user's name (or UID) and optionally the user group (or GID) to use when running the image.

Step1 login to the docker registry

```
# docker login my_registry.com.8080
```

push on image to docker registry

```
# docker push name_of_our_repository/image_name
```

Docker RUN vs CMD vs ENTRYPOINT

- RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.
- CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.
- ENTRYPOINT command and parameters will not be overwritten from command line. Instead, all command line arguments will be added after ENTRYPOINT parameters.

NOTE: In Dockerfile should specify at least one CMD or ENTRYPOINT commands. when we use both CMD and ENTRYPOINT in dockerfile CMD act as argument to the ENTRYPOINT.

EXample: Vi Dockerfile

```
# CMD ["apache2"] (Here Argument can change in place of apache2 tomcat  
will chnage )
```

```
# ENTRYPOINT ["service start"]
```

NOTE: Docker communicate with daemon with help of http or sockets.

Reference: <http://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/>

Use cases

1) Run nginx on external port 80: and internal port 80 name it as webserver

- **# docker run -d -p 81:80 --name webser httpd**
- The above command will start httpd. to see the home page of the httpd
- Open another session or terminal give the command
Port 81 from outside world, port 80 is internal world
- **For testing in browser <http://localhost:80>**
Note: in the above command run initial checks for that image in the local image cache.
If the image is present here it will started

- If it is not present it will download from docker hub and start the container.
- It creates a virtual ip on the docker private network in the docker engine.
- Opens port 80 on the host machine and communicates with port 80 on the docker container.

-p, --publish list

Publish a container's port(s) to the host

-P, --publish-all

Publish all exposed ports to random ports

2) Run httpd in detached mode on external port 8080 and internal port 80 name it apache webserver

- # docker container run -d -p 8080:80 --name apachewebserver httpd

<http://localhost:8080>

3) Run MySQL on external port 3307 and internal port 3306 ... pass environment variable MYSQL_ROOT_PASSWORD=yes

- # docker container run -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=yes --name mydbmysql
- <http://localhost:3307>

4) Run Ubuntu container and install git on it. Commit the image. Run the image with a new name and check if git is still present

- # Docker container run -it --name myubuntuUbuntu
- In Ubuntu shell execute the below commands
- # apt-get update
- # apt-get install git
- # exit
- Open hub.docker.com -> create a free account
- To commit the image
- # docker commit container_id[name] repository_id/image_name[syntax]
- # docker commit myubuntu/git-ubuntu
- # docker run -it nagt/git-ubuntu
- We will find git installed in it

5) Push the above image to docker hub

- The docker image created in the previous step is present with in the docker daemon it can be uploaded into the docker hub from where any can download or it can be uploaded into the local registry
- To upload into docker hub
 - Create an account in hub.docker.com
 - Docker login
 - Enter username and password
 - # docker push image_name

Committing images to the local registry

- Local registry is similar to docker hub but it is specific for particular organization.
- which ever images are committed in to the local registry can be access with in that network
- To create local registry we should download and run the registry image.

6) Download alpine Linux and push into the local registry

- # docker pull alpine
- # docker container run -d -p 5000:5000 --name local registry
- The above will download an image called registry run it on external and internal port 5000
- To commit any image in to the local registry we should first tag it [tag mean giving a name]
- # docker tag tag_name localhost:5000/image_name[syntax]
- # docker tag alpine localhost:5000/alpine
- # dicker push localhost:5000/alpine

Image layers

- Docker images are always downloaded in form of layers the same layer will not get downloaded multiple times that is if we download Ubuntu docker image
- Next we download Ubuntu with tomcat install on it
- One more Ubuntu with git install
- Another tomcat with web application deployed on it.'
- It will download them one time

Data volumes

Whenever we exit from a container the data that has been created with in the container will be losses.

To preserve the data that has been created by container we can use volumes.

They are two types.

- 1) Data volumecontainer
- 2) Data volumes container

Note: volumes can be mounted using the docker run command `-v` options.

Use cases:

7.Run Ubuntu container with name myUbuntu... Mount it on a data volume called 'data' and create some files in this 'data' folder ...check if the files remain intact after restart of the container.

```
# docker container run -it --name myubuntu -v /data Ubuntu
In the Ubuntu container.
# cd /data
# touch file{1..3}
# $ exit
# $ docker restart myubuntu
# $ docker attach myubuntu
# $ cd data
# We will find all the previously created files
```

Data volume containers:

The volume which is used by one container can be shared with other containers using data volumes containers.

8. Create a Ubuntu container and name it container1 mount a volume called /data in this container create some files in this /data folder come out of the container without exiting share this container volume with container2 and share container2 volumes with container3.

- # docker container run -it --name container1 -v /data Ubuntu
- In the container cd to data volume
- Create files using `$touch file{1..5}`
- Come out without existing (`ctrl+p+q`)

- # docker container run -it -volumes-from container1 -name container2 Ubuntu
- # docker container run -it -volumes-from container2 -name container3 Ubuntu
- Stopping multiple container
- # docker stop \$(docker ps -aq)
- Remove the containers
- # docker rm \$(docker ps -aq)

Linking's

Multiple can take can be using --link option

This will enable us to allow commutation b/w containers

9. Run a busy box container and name it source run another busy box container and name it target link target with source and check if we can ping b/w the containers.

- # docker container run -it -name source busybox
- Come out of container using ctrl+p+q
- # docker container run -it --link source:source_alias -name target busybox. --
- Here source is source container name
- Source_alias is alias name

Note: the network establish using --link option will work only for communicating b/w containers it can't be used for communicating for the host machine.

Use case

10) Runmysql container and name nag-mysql run wordpress container and name it nag-wordpress and link to the mysql container

- # docker run -name nag -e MYSQL_ROOT_PASSWORD=password1 -d mysql
- # docker run -name nag --link nag:mysql_alias -p 8080:80 -d wordpress
- Open any browser and navigate to localhost:8080
- Create a wordpress website.

Docker compose

- This is tool of docker which is used for executing multiple docker commands from one point of control.
- This file is created using yaml.
- Yaml always takes the data in the format of key and values.

Devops:

- traniers:
- nag:
 - devops: 1400
 - selenium: 5000
- ram:
 - devops: 12000
 - aws: 5000
- receptionist
- one

...

To validated our yamil syntax <http://www.yamllint.com>

- To install docker compose on Linux
- <https://docs.docker.com/compose/install/#install-compose>
- `sudo curl -L https://github.com/docker/compose/releases/download/1.16.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose`
- `sudo chmod +x /usr/local/bin/docker-compose`
- `Docker-compose --version`

Use case

1) Create a docker compose file for starting a word press container and mysql container

- Create a file called docker-compose.yml
- `# vim docker-compose.yml`
- Go into insertion mode by pressing "I"
 - Version: '3'
 - Services:
 - Mysql:
 - Image: mysql
 - Environment:
 - MYSQL_ROOT_PASSWORD: mypassword
 - Wordpress:
 - Images: wordpress
 - Ports:
 - 8080:80
- **Save and quit**
- **Escape : wq enter**
- **To run the above file**
- `# docker-compose up`

- To stop the services using our compose file
- # docker-compose down
- # docker –compose –f filename up

Building docker images using docker file

Docker file is a simple text file which uses specific commands using which it is possible to create our own images.

It uses the following keywords to create or modify images

- FROM
- MAINTAINER
- CMD
- ENTRYPOINT
- RUN
- COPY
- EXPOSE
- ADD
- USER
- VOLUME
- WORKDIR
- ENV
- LABEL

To create images via docker file we should perform the below two steps

- 1) Create the docker file with above commands
- 2) Build an image using that file

Use case

1) Create a docker file using the base image Ubuntu and specify the name of the author

- # vim dockerfile
 - Go into insert mode by pressing 'i'
- ```
FROM ubuntu
MAINTAINER Sai
```
- Save and quit(:wq)

To build an image using the above t.....

```
docker build -t newdocker .
```

**Note:** -t is used for specifying a name for our image. represents current working directories i.e it will build an image based on the docker file present in our working directory.

2) Create a docker file based on alpine Linux image and executes some Linux commands in it

```
vim dockerfile
Go into insert mode by pressing 'I'
FROM alpine
MAINTAINER nag
CMD ["date"]
CMD ["ls", "-la"]
:wq
dockerbuild -t newalpine .
```

## **ENTEYPOINT**

This command is used for taking which comes from CMD as arguments

### **Use case**

Create a docker file from busybox image set the ENTEYPOINT as cat command and open a file called /etc/passwd".

```
vim dockerfile
FROM busybox
MAINTAINER nag
ENTERYPPOINT ["/bin/cat"]
CMD ["/etc/passwd"]
docker build -t newbusybox .
docker run newbusybox
```

### **Use case**

1. Download Ubuntu image and then install git and maven ping and curl init. Perform the above action thorough a docker files.

```
vim dockerfile
FROM ubuntu
RUN apt-get update && apt-get install -y git \
 Maven \
 Oputils* \
 Curl \
:wq
docker build -t myUbuntu .
Run the myUbuntu image created using the docker file
docker run -it myUbuntu
git -version
mvn -version
```

2. Create 5 docker images using the same dockerfile that we have created in the previous use case.

- Create a shell script with the name 'myscript.sh'
- # vim myscript.sh
- Go Into insert mode by pressing 'I'
- For I in {1..5}
- do

- `docker build -t myUbuntu$1 .`
- `:wq`
- Give execute permissions on the above shell script
- `# chmod u+xmyscript.sh`
- Run the shell script using
- `./myscripts.sh`

## Docker Networking

To see the list of networks available

```
docker network ls
```

To create new network

```
docker network create network_name
```

To remove/delete a network

```
docker network rm network_name
```

To find the information about the network

```
docker network inspect network_name
```

To attach a container to a network

```
docker network connect network_name container_name
```

To disconnect the network to container

```
docker network disconnect network_name container_name
```

### Use case:

Create a network called nag1 another network called nag2

Create 3 busy box containers container1 container2 container3

Start container1 and container2 on nag1 network and check if they are pinging not

Start container3 is nag2 network and check that it cannot ping to container1 and container2

Now container1 should be able to communicate with container2 but cannot communicate with container3 similarly container3 should be able to communicate with container2 but not with container1

```
docker network create nag1
```

```
docker network create nag2
```

```
docker run -itd --name container1 --network nag1 busybox
```

```
docker run -it --name container2 --network nag1 busybox in container2
```

**Ping container1: it should successfully ping**

**Ctrl+p+q to come out of container2**

**# docker run -it --name container3 --network nag2 busybox**

**In container3**

**Ping container1**

**Ping container2**

**It should not be able to ping because container1 and container2 are running nag network**

**# docker network connect nag2 container2**

**The above command will attach container2 to nag2 network...container2 is now running on both nag1 and nag2 network**

**Docker container attach container3**

**Ping container2: it will ping successfully**

**Ping container1: it cannot ping**

### **Docker swarm:**

This is a feature of docker using which we can perform container orchestration. Imagine a scenario where we run 100s of containers in distributed network we can use docker swarm. Important activates of docker swarm

- 1) Launching fix number of containers for particular docker image
- 2) Performing the health checks on containers
- 3) Scaling the number of container are up and down
- 4) Performing rolling updates
- 5) On services in these containers

### **Setup of Docker swarm**

Create a 3 Linux vms and install docker on all of them. One machine will be called as manager the other machine will be worker one and two.

On all the 3 machine install docker

Create the manager machine as swarm manger

**# docker swarm init --advertise-addr 192.168.61.10**

Note: --advertise-addr configure the manager node to publish 192.168.61.10

As its ip address using which other nodes can connect

To find the list of the nodes attached to our docker swarm

**# docker node ls**

To create the remaining nodes as workers and assign them to the manager

**# docker swarm join --token tokenid**

Note: this command can be copied from the console when we start the docker manager.

Creating services on swarm

This can be done using docker service create command



Create a nginx service on the docker swarm also create 5 replicas of it.

```
docker service create --replicas 5 --p 80:80 --name webserver nginx
```

To find the list of the docker services currently running

```
docker service inspect --pretty webserver
```

To scale the services up or down

```
docker service scale webserver=8
```

To find the list of containers running on these services on different nodes

```
docker service ps webserver
```

```
docker service create --replica 3 --name myredis redis:3.0.6
```

```
docker service update --image redis:3.0.7 myredis
```

To roll back redis to the previously installed version

```
docker service update --rollback myredis
```

To remove service from docker swarm

```
docker service rm myresids
```

```
docker node ls
```

The docker swarm manager can remove a node availability drain worker1

Command to drain worker1 from the swarm

```
docker node update --availability drain worker1
```

To reactivate the drain worker

```
docker node update --availability active worker1
```

For worker node to leave the swarm go to the worker node

```
docker swarm leave
```

To start nginx with 3 replicas on overlay network

Create overlay network

```
docker network create --driver overlay my-network
```

```
docker network create --driver overlay mynetwork
```

```
docker service create --replicas 3 --network mynetwork --name webserver nginx
```

To attach a network to an already existing docker swarm services

```
docker service update --network-add mynetwork webserver
```

To remove a network from a service

```
docker service update --network-rm mynetwork webserver
```

## Different Types of Volumes

There are three types of volumes: *host*, *anonymous*, and *named*:

- A **host volume** lives on the Docker host's filesystem and can be accessed from within the container. To create a host volume:

- `docker run -v /path/on/host:/path/in/container ...`

- An **anonymous volume** is useful for when you would rather have Docker handle where the files are stored. It can be difficult, however, to refer to the same volume over time when it is an anonymous volumes. To create an anonymous volume:

- `docker run -v /path/in/container ...`

- A **named volume** is similar to an anonymous volume. Docker manages where on disk the volume is created, but you give it a volume name. To create a named volume:

- `docker volume create somevolumename`
- `docker run -v name:/path/in/container ...`

What is Docker stack ?

Diff between Docker stack and Docker Swarm ?

#### NOTE:

Docker images is broken down into layers, Those storage layers are Read only layers.

Containers are read/write layers. Basically every container will have read layers and one read write layer

#### Layers history

Every run Instruction will create a layer

Every layer will contain hash id

#### Example

Since you have a tomcat:7 image again when you are trying to pull the tomcat8 first it will check the other images layers if the layers are matched it will not download based layers sha id it simply reuse the layers

All layers are stored location called show below

```
/var/lib/docker/image/overlay2/imagedb/content/sha256
```

```
docker history ImageID
```

It will display the what happend into image what are diffrent commands used in image

docker multistage build

**Vertical Scaling:** Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine.

**Horizontal Scaling:** Horizontal scaling means that you scale by adding more machines into your pool of resources.

## Manage data in Docker

By default all files created inside a container are stored on a writable container layer. This means that:

The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.

To list out existing volumes

```
docker volume ls
```

## *to create a container for stateless applications*

```
docker run -it --name vtuatjenkins01
```

Docker has 3 options for containers to store files in the host machine, so that the files are persisted even after the container stops:

## docker volume types:

1. anonymous volumes
2. named volumes
3. host volume or bind volumes

## Anonymous Volumes

Create a container with an anonymous volume which is mounted as /data01 on container. In this case we mention container directory name. On host system it maps to a random-hash directory under /var/lib/docker directory.

```
docker run -it --name vtwebuat01 -v /data01 nginx /bin/bash
```

On Host to verify volume

```
docker volume ls
```

```
docker inspect <volume_name>
```

## Named Volumes

Create a container with a named volume name which is mounted as /data01 on container. You can see volume name as vtwebuat02\_data01\_val

```
docker run -it --name vtwebuat02 -v vtwebuat02_data01_val:/data01 nginx /bin/bash
```

Create a named volume then attach volume to a container

```
docker volume create vtwebuat03_data01_vol
```

```
docker run -it --name vtwebuat03 -v vtwebuat03_data01_vol:/data01 nginx /bin/bash
```

Create a named volume with size

```
docker volume create --opt o=size=100m --opt device=/data3 --opt type=btrfs vtwebuat03_data3 // to create volume with size
```

## Host Volumes or Bind Volume

Create a host volume

```
mkdir /opt/data02
```

```
docker run -it --name vtwebuat03 -v /opt/data02:/data02 nginx /bin/bash
```

# KUBERNETES:



# kubernetes

Kubernetes Installation

**Step1:** apt-get update

Need to install docker

**Step2:** apt-get install docker.io

or

Follow below link (<https://docs.docker.com/engine/install/ubuntu/>)

To check the Docker version

```
docker --version
```

Need to enable the docker

```
systemctl enable docker
```

To verify the docker status

```
systemctl status docker
```

If docker is not running

```
systemctl start docker
```

**step3--** Installing Kubernetes

Enter the following to add a signing key:

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

If you get an error that curl is not installed, install it with:

```
apt-get install curl
```

**Step4--**

Kubernetes is not included in the default repositories. To add them, enter the following:

```
apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
```

1. Install Kubernetes tools with the command:

```
apt-get install -qy kubeadm=1.18.20-00 kubelet=1.18.20-00 kubectl=1.18.20-00
```

Its install latest version, if we need specific version indicate version

```
apt-mark hold kubeadm kubelet kubectl
```

for kublete version  
kubeadm version  
for kubelete enable  
systemctl enable kubelet  
for kubelete start  
systemctl start kubelet

### **step5 --**

Kubernetes Deployment  
swapoff -a

### **step6--**

Initialize Kubernetes on Master Node

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

this step is for only master node, if incase using in worker node it act's also master node.

Once the command finishes it will display a kubeadm join message

this is run in worker nodes

### **step7-- pod network**

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

### **step8--**

once completed all the above steps

run kubectl get nodes in master node it displays the all worker nodes

kubeadm token create --print-join-command = To recreate the token

<https://computingforgeeks.com/join-new-kubernetes-worker-node-to-existing-cluster/>

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 172.31.21.38:6443 --token 34ms8c.412awoibhe3d2bk5 \
--discovery-token-ca-cert-hash
sha256:2069c588d356c2228d2b1c5152dod12cc712f493a7531fecf931ed8d8b0591b3
```