DevOps

1.Introduction to DevOps & Agile Methodology

Myths about DevOps :

1.Programming Language Knowledge is Required
2.Linux Experience is Must
3.Prior IT Experience is Required
4.Non-Technical background cannot do

Why Organisations Needs DevOps Specialists ??

1.Fast Delivery
2.High Availability
3.Less Capital Expense & Operational Expense
4.Reduced Outages

Software Development Life Cycle(SDLC) Phases :

1.Planning and Preparing Design Documents
2.Development : Developer develops code using Programming Languages like Java,Python etc and uploads the code to Github or Gitlab
3.Build : Maven
4.Testing : Selinium
5.Quality Assurance
6.Deploy : chef ...Ansible....Docker....Puppet
7.Maintenance
8.Monitoring : Nagios....AWS Cloudwatch

Popular SDLC Models : Waterfall....Agile...Spiral....V-Model.....Incremental Model.....Big Bang Model etc...

DevOps : Implementing automation at each and every stage

DevOps Stages :

1.Version Control : Maintain different version of the code --→ Git
2.Continuous Integration : Compile,Validate,Code Review,Unit Testing,Integration Testing -→ Jenkins
3.Continuous Delivery : Deploying the build app to test server -→ Maven
4.Continuous Deployment : Deploying the test app on the production server for release -→

Chef,Ansible,Puppet,Docker,Kubernetes

• The term DevOps is a combination of two words i.e Development and Operation
• DevOps is a Methodology that allows a single team to manage the entire application development life cycle,that is development,testing,deployment and operation
• The objective of devops is to shorten the systems development life cycle
• Devops is a software development approach through which superior quality software can be developed quickly and with more reliability

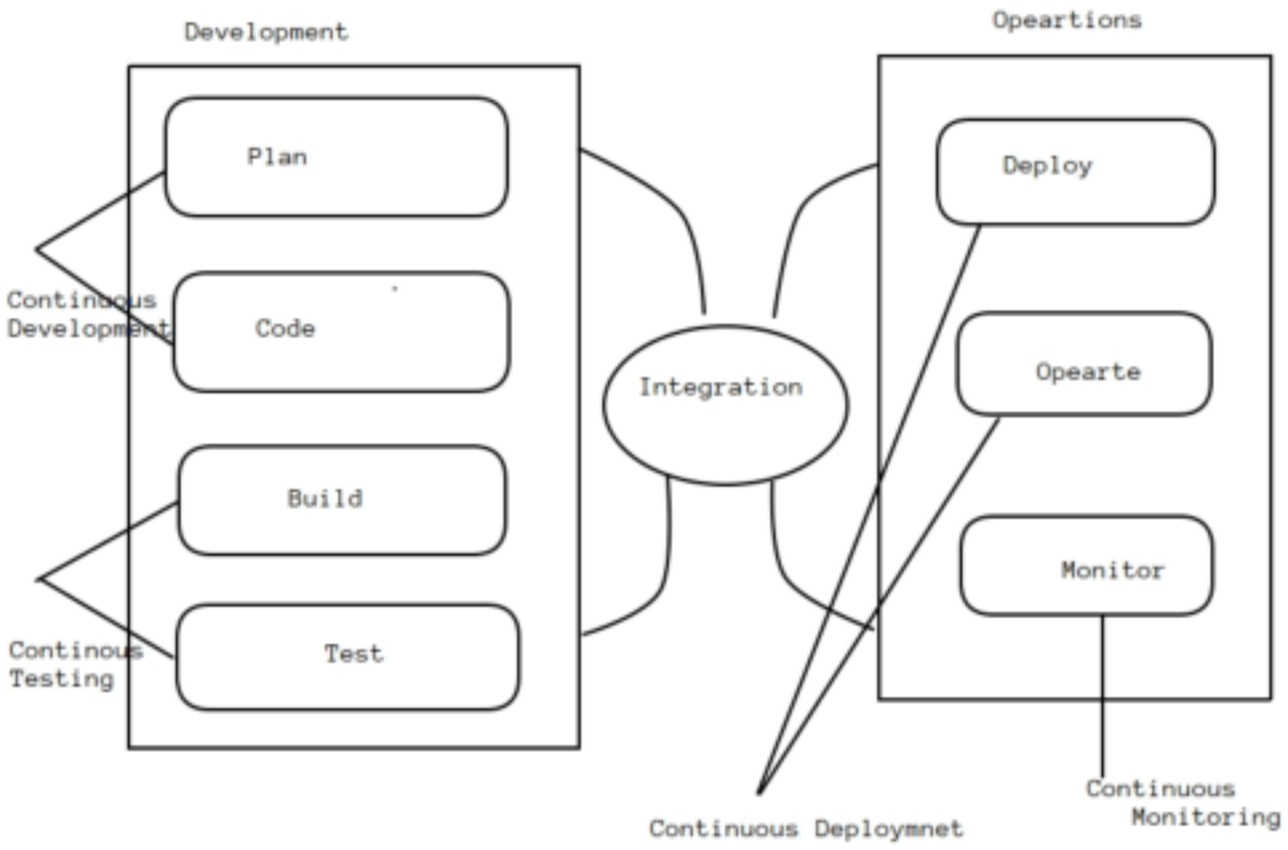
Agile Methodology : Build Short,Build Often

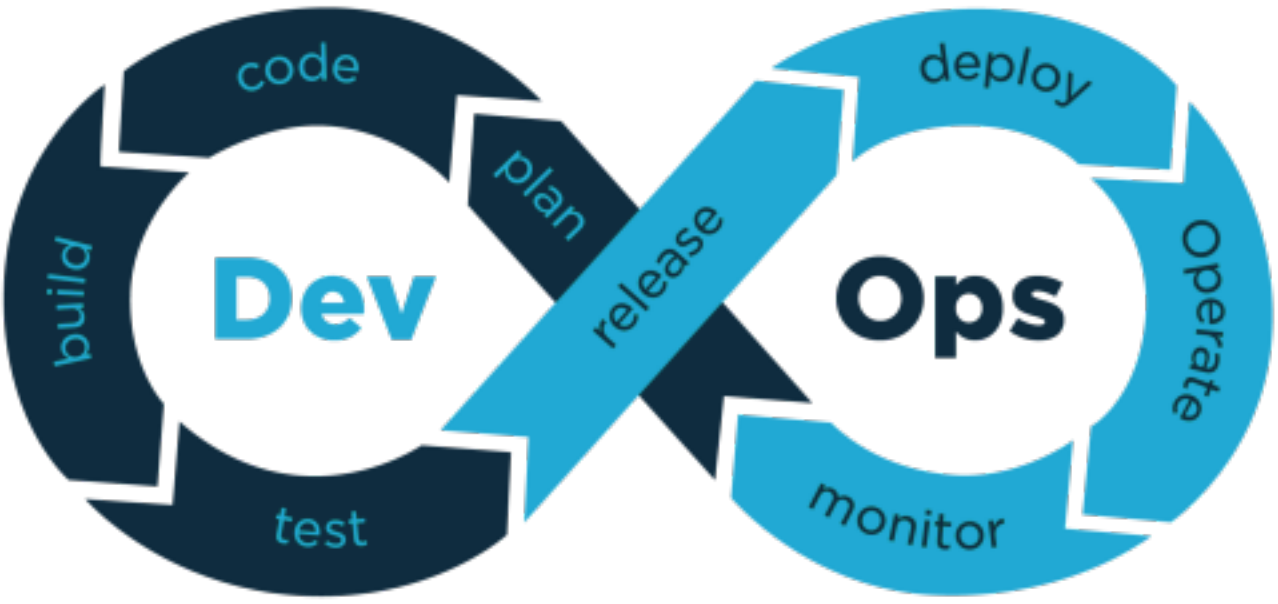
Scrum : Scrum is a framework used to manage a product development.With scrum a project is built in a series of itterations called as Sprints.

Sprints : Sprint is a short.time-boxed period where planned amount of work is completed and made available for review.

Note : Agile is a improvement of Waterfall SDLC Method where DevOps is improvement of Agile Methodology.

DevOps Stages :





2.Linux from Scratch

• Create AWS Free Tire Account and Create Linux Machine using AWS EC2 service

Introduction to Linux :

• Bell laborartory in 1964 started a project to create O.S which supports multi tasking and Multi User
• They shutdow the above project in 1969
• Dennis Ritchie and Ken Thompson started working on the said Project with the name UNICS(Uniplexed Information and Computing Services) which is free and made source code available to all
• In 1975 new version released UNIX V6
• With UNIX code many vendors created their flavour of UNIX,like IBM-AIX,Sun-Solaris,Apple-MAC OS,HP-UX etc and started selling them
• Andrew Tanenbaum created an O.S named MINIX to teach students which insipres Linus Torvalds to create his Linux Kernel
• Linus Torvalds in 1991 created an Kernel named Linux
• Many GNU Softwares were added to LInux Kernel to create entire O.S....
• Linux is Just a Kernel and not an entire Operating System....SO Flavours of Linux like Redhat,Debian,Arch are actually GNU/Linux not just Linux
• Popular Flavours of Linux are Redhat,Debian,Fedora,Arch,Manjaro,Suse Linux,Kali Linux,CentOs,Ubuntu,Deepin etc..

• O.S had two Types of Interfaces.One is Command Line Interface(CLI) and another one is Graphical User Interface(GUI)

Linux Keypoints :

• Linux is Kernel not entire O.S
• Linux is not a UNIX derivative.It was wriiten from scratch
• A Linux distribution is the Linux kernel and a collection of softwares that together,creates an entire O.S

Linux Features :

• Open Source
• Secure
• Simplified updates for an installed software
• Light Weight
• Multiuser-MultiTask
• Multiple distribution-Redhat.Debian,Fedora,Arch,Deepin etc

Linux VS Windows :

Linux System Architecture : User-→ Shell-→ Kernel -→ Hardwrae

Windows System Architecture : User -→ Shell -→ O.S -→ Hardware

Some Terminology Difference Between Windows & Linux :

•Folder in Windows & Directory in Linux
• Administrator in Windows & Root user in Linux
• Software in Windows & Package in Linux

File System Hierarchy :

Windows -→ c:\ -→ Programfiles   Users   ProgramFiles(x86)   Preflogs

Linux :

/	: Top level root Directory
/home	: home directory of other users
/root	: Home directory of root user
/boot	: It contains bootable files for linux.Eg : initrd
/etc	: It contains all configuration files
/usr	: By default software are installed in this directly
/bin	: Contains commands used by all users
/sbin	: Contains commands used by only root user
/opt	: Optional application software packages
/dev	: Essential device files.This includes terminal devices or any device attached to the system

Linux Commands and thier Use Cases :

1.Creating a File : cat,touch,vi/vim,nano

A.Cat Command : The cat command is one of the most universal tools,yet all it does is copy standard input to standard output

cat can :
- create files : creates single file
- concatinate files : to add more than one file into a single file
- copy files : to copy the content of x into y
- tac : to see the content from bottom to top

Using cat :

☐ cat >file1 -→ hit enter --→ add content --→ hit ctrl+d
ls to view file1
cat file1 to view content

☐ cat >file2 -→ hit enter -→ add content -→ hit ctrl+d
ls to view file1 & file2
cat file2 to view file2 content

☐ cat >>file2 --→ hit enter -→ add new content(append) -→ hit ctrl+d
cat file2

☐ cat file1 file2 >all -→ hit enter
ls to view file1 file2 all
cat all

☐ cat file1 >file2 -→ hit enter
cat file2 to view updated content

☐ tac file1 -→ hit enter
this prints content from bottom to top

B.Touch Command :

- Create and empty file
- create multiple files
- change all timestamps of a file
- update only access time of file,modified time of file

Timestamp : stat filename will show following info
☐ Access time(last time when a file was accessed)....touch -a
☐ Modify time(last time when a files was modified)....touch -m
☐ Change time(ast time when a file metadata is changed)

Using Touch :

☐ touch file3 file4 file5 -→ hit enter
ls to view newly created files
stat file1 -→ hit enter -→ this prints timestamp
touch file1
stat file1 -→ hit enter -→ prints updated timestamp

☐ touch -a file2 -→ hit enter

stat file2 -→ hit enter -→ this only changes access time
touch -m file2 -→ hit enter
stat file2 -→ hit enter -→ this only access modified time

C.vi Editor :

• A programmer text editor
• It can be used to edit all kind of plain text,it is specially useful for editing programs mainly used for unix programs

Note :
:w -→ To save
:wq or :x -→ to save & quit
:q -→ quit
:q! -→ force quit,no save

• 'Vi' is a standard whereas 'nano' has to be available depending on the linux we use.

Using vi :

☐ vi file5 -→ hit enter
hit i to enter insert mode-→ add content -→ hit esc to enter in command mode -→ hit :wq to save & quit
cat file5 to view content

☐ vi file5 -→ hit enter
hit i to enter insert mode -→ add new content -→ hit esc to enter in command mode -→ hit :wq to save & quit
cat file5 to view updated content

D.nano Editor :

☐ nano filename -→ add content -→ ctrl+x -→ y
☐ ls to view file and cat to view file content

Using nano :

☐ nano file6 -→ hit enter -→ add content -→ hit ctrl+x -→ hit y to save
cat file6 to view file6 content

☐ nano file6 -→ hit enter -→ add new content -→ ctrl+o -→ hit enter or (change file name say file7 then hit y) to save -→ hit ctrl+x
cat file6 or file7 to view content

2.Creating & Removing Directories, Change Directories,pwd,creating hidden files & Dir's :

Creating Directory :

A.mkdir command :

☐ mkdir dir1----creates dir1
☐ mkdir -p dir2/dir3/dir4 ----creates dir2 and dir3 under dir2 and dir4 under dir3

B.cd command -→ change directory :

☐ cd dir2 ----changes working directory to dir2
☐ cd dir3 ----changes working directory to dir3 from dir2
☐ cd .. ----changes working directory to parent directory
☐ cd ../.. ----changes working directory to parents parent directory

C. pwd....print working directory

D.Creating hidden file or directory

☐ touch .file1 ----Creates hidden file1
☐ mkdir .dir5 ----creates hidden dir1
☐ ls -a ------shows hidden files and directories

E.Removing File & Directories

☐ rmdir ----Remove specified directory(Only empty directory)
☐ rmdir -p -----Remove both the parenta nd child Directories
☐ rmdir -pv ----Removes all the parent & subdirectories along with the verbose
☐ rm -rf ----Removes even non-empty file & directories
☐ rm -rp -----Removes non-empty directories including parent & Subdirectories
☐ rm -d -----Removes empty directories

3.Copying,Moving & Renaming Files & Dir's

A.cp command : copy

☐ cp file1 file2 -----Copies & Creates file2 in same directory with file1 contents
☐ cp /home/user/file1 /home/user/test/file1 -----Copies file1 from /home/user to /home/user/test
☐ cp -R /home/user/dir1 /home/user/test/dir1 ----Copies dir1 from /home/user to /home/user/test

B.mv command : moving & renaming

☐ mv file1 file2 ------Renames file1 as file2 in same directory
☐ mv /home/user/file1 /home/user/test/file1 ----Moves file1 from /home/user to /home/user/test
☐ mv /home/user/dir1 /home/user/test/dir1 ----Copies dir1 from /home/user to /home/user/test

4.Head & Tail

A. head command ----Displays first 10 lines of file
☐ head file1

B.tail command ----Displays last 10 lines of file
☐ tail file1

5.Less & More

A. less and more Commands -----Displays files contents in pager.....Use q to exit

6.hostname & ifconfig

A.hostname Command ---- Prints Name of the Linux machine
☐ hostname -i ---- prints ip of linux machine

B. ifconfig Command ---- Prints network interfaces and corrosponding Ip addresses and Netmask etc

C. cat /etc/os-release --- prints Opertaing System related info like Name,Build ID,Release Version etc

7. yum --- Yellowdog updater,Modified

A. yum ----yellowdog updater,Modified------Uses to install,update,remove packages

☐ yum install httpd ----- Installs apache2
☐ yum remove httpd ---- Removes apache2
☐ yum update httpd ---- Updates apache2 to latest version
☐ yum list installed --- lists all installed packages

8. Service & chkconfig

☐ service httpd start ---- Puts apache2 in active state
☐ service httpd status ----- Checks the curent status of apache2

☐ chkconfig httpd on --- Auto start httpd service at every boot
☐ chkconfig httpd off ---- Disables apache2 service to start at boot

9.which,whoami,echo

A.which ---- prints the path of the executable file of installed package
☐ which ls

B.whoami ----- Prints the currently loggen in user like root......username....

C. echo ------ Prints the output of the message typed along

☐ echo “Hello” ---- Prints echo on terminal
☐ echo can be used in scripts and be used to provide messages to users
☐ echo “Hello” > file1 ---- Creates file1 and redirects echo command output to file1
☐ echo > file1 ----- Empties file1

10.grep & sort

A. grep --- global regualr expression print ---- used to search the strings or characters
☐ grep root /etc/passwd ---- Searches and prints root in /etc/paswd

B. sort ---- Sorts the contents accordingly like alphabatically

11.useradd,groupadd,gpasswd

A.useradd ---- Used to create new users

☐ useradd bhupinder
☐ cat /etc/passwd | grep bhupinder

B.groupadd ---- Used to create new groups

☐ groupadd techguftgu
☐ cat /etc/group | grep techguftugu

C.gpasswd -a / -M ---- To add user to group,to add multiple users

☐ useradd ajay
☐ useradd vikas
☐ gpasswd -a bhupinder techguftgu
☐ gpasswd -M ajay,vikas techguftgu
☐ cat /etc/group

12.ln --- Hard & Softlinks

A.ln -s ---- Softlink

☐ cat >>file1 -----and add content then ctrl+d
☐ ln -s file1 softlinkfile1
☐ ls -l
☐ cat softlinkfile1
☐ cat >>softlinkfile1 ----add content then ctrl+d
☐ cat file1 ----- verify new content added to file1
☐ rm file1
☐ rm -rf softlinkfile1

B.ln ---- Hardlink

☐ cat >>file2 ----add content and then ctrl+d
☐ ln file2 hardlinkfile2
☐ cat >>hardlinkfile2 ----add content then ctrl+d
☐ cat file2 ----verify content added
☐ cat >>file2 ----add content then ctrl+d
☐ cat hardlinkfile2 ----verify content added
☐ rm file2

13.tar & gzip

A. tar ---- It is a archiver used to combine multiple files into one

☐ tar -cvf dirx.tar dirx

B.gzip ----gzip is a compression tool used to reduce the size of the archive/file

☐ gzip dirx.tar
☐ gunzip dirx.tar.gz

☐ tar -xvf dirx.tar
☐ rm -rf dirx

14.wget

A.wget ---- It is a non-interactive network downloader

☐ wget <"paste url"> then hit enter
☐ ls ---- to check downloaded file
☐ yum install downloaded file.rpm
☐ rm -rf * ------ removes everything in directory

15.Changing Permissions of a File or Directory :chmod,chown,chgrp

• Access Mode/Permissions :

Access Mode		**File**	**Directory**
r	4	To display the content	To list the content
w	2	To Modify	To create or remove
x	1	To execute the file	To enter into directory

A.chmod ---- used to change the access mode of file

☐ sudo su
☐ touch file1
☐ mkdir dir1
☐ ls
☐ useradd bhupinder
☐ groupadd linux
☐ ls -l
☐ chmod 777 dir1
☐ chmod 700 dir1
☐ ls -l
☐ chmod 755 file1
☐ ls -l
☐ chmod g=r,o=rw dir1
☐ ls -l
☐ chmod u=r,g=rwx,o=x dir1
☐ ls -l
☐ chmod u+w,g-w,o+r file1
☐ ls -l
☐ chmod 000 file1
☐ ls -l

B.chown -----used to change the owner of the directory/file

☐ chown bhupinder dir1

C.chgrp ----- used to change the group of dir/file

☐ chgrp linux file1

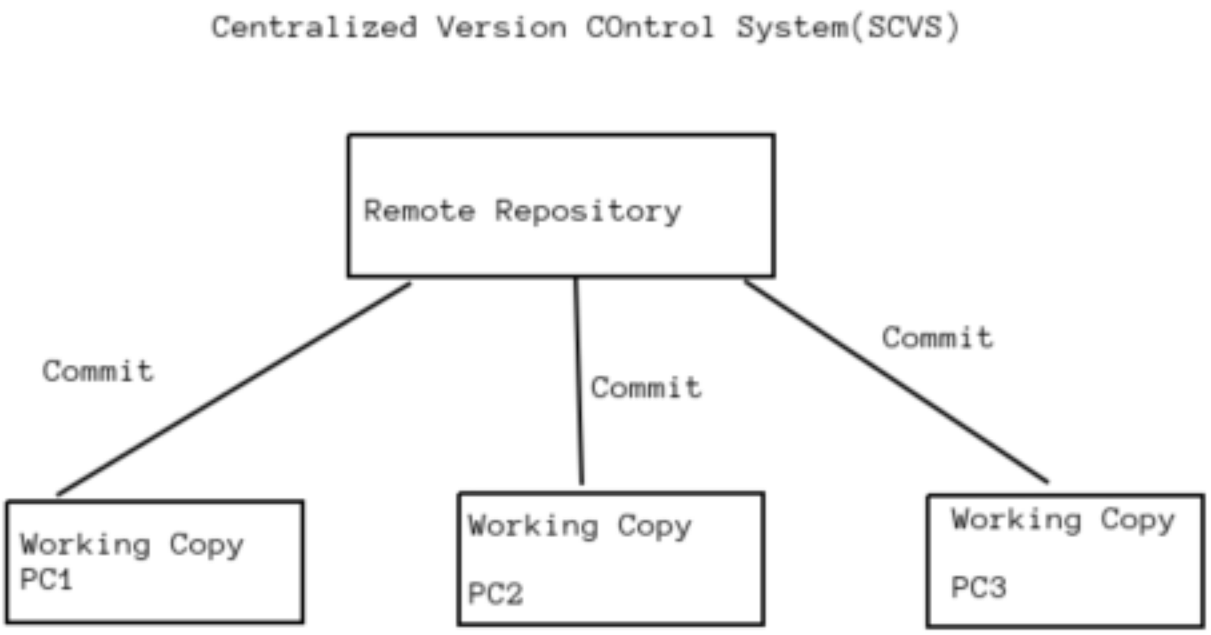
3.Git

Introduction to GIT :

• **Git** is a version control system that lets us manage and keep track of our source code history
• Git is created by Linux Torvalds
• Github & Gitlab are cloud-based hosting services that lets us manage **Git** repositories.

Source Code Management (OR) Software Configuration Management :

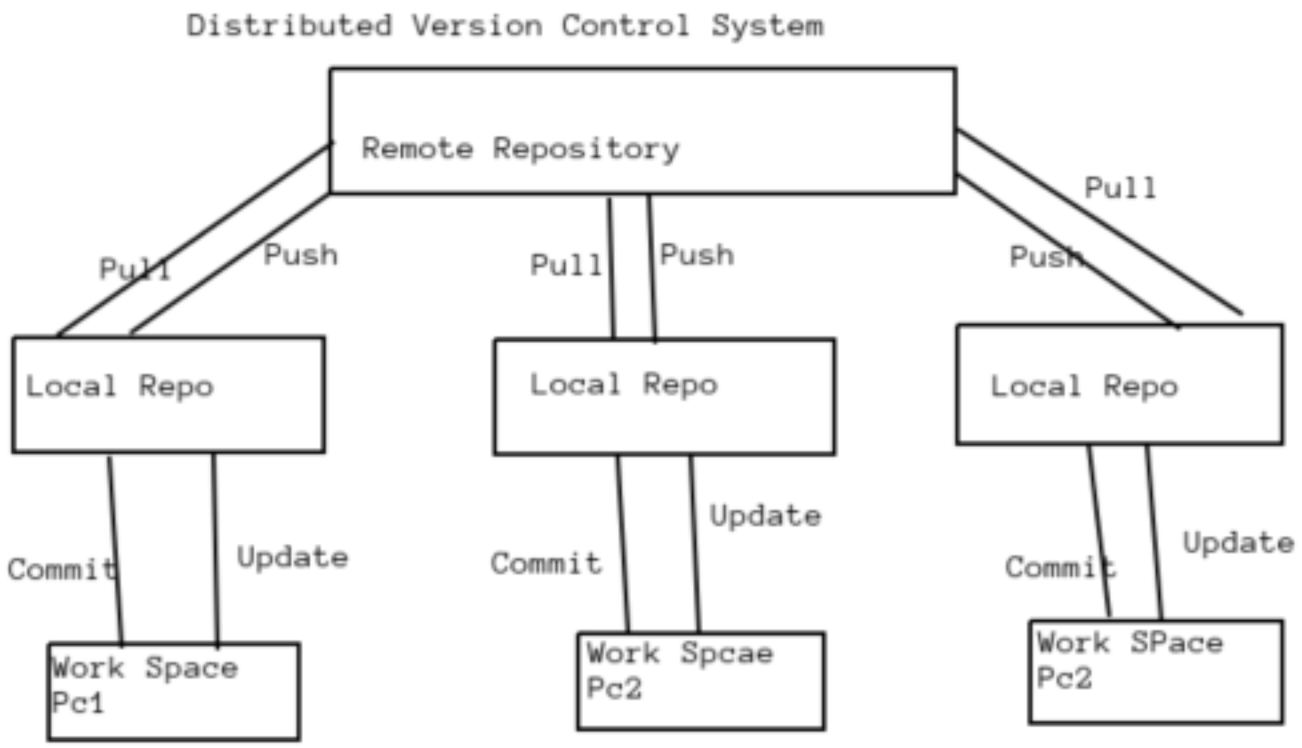
• Centralized Version Control System : Eg : SVN Tool



Drawbacks :

☐ It is not locally available,meaning we always need to be connected to a network to perform any action
☐ As everthing is centralized,if central server gets failed,we will loose entire data

• Distributed Version Control System :



☐ In distributed version control system,every contributor has a local clone of main repository i.e everyone maintains a repository of their own which contains all the files & metadata present in main repository

• CVCS VS DVCS

CVCS	**DVCS**
Client need to get local copy of source from the server,do the changes and commit those changes to the central source on the server	Each client need to pus repository
These are easy to learn and setup	Difficult for
Working on branches are difficult.Developers often face merge conflicts	Working on
These do not provide offline access	DVCS syste repository c
CVCS is slow because every command needs to be communicated with central server	DVCS is fas
If CVCS server is down,developer cannot work	If DVCS ser

Stages of Git & Its Terminology

Repository :

• Repository is a place where we have all our codes or kind of folder on server
• It is a kind of folder related to one product
• Changes are personal to that particular repository

Server :

• It stores all repositories
• It contains metadata also

Working Directory :

• Where we see files physically and do modification
• At a time,we can work on particular branch
• In other CVCS,developers generally makes modifications and commit their changes directly to the repository.But git uses a different strategy.Git does not track each and every modified file.Whenever we do commit an operation git looks for the files present in the staging area.Only those files present in the staging area are considered for commit and not all the modified files

Working Directory → Staging Area → Local Repository → Github

Commit :

• Store changes in repository.We will get one commit-ID
• It is 40 alpha-numeric characters
• It uses SHA-1 checksum concept
• Even if we change one dot,commit-ID will get change
• It actually helps us to track the changes
• Commit is also named as SHA-1 hash

Commit-ID/Version-ID/Version :

• Reference to identify each change
• TO identify who changed the file

Tags :

• Tags assign a meaningful name with specific version in the repository.Once a tag is created for a particular save,even if we create a new commit,it will not be updated

Snapshots :

• Represents same data of particular time
• It is always incremental i.e It stores the changes (appended data) only,not entire copy

Push :

• Push operations copies changes from a local repository instances to a remote or central repo.This is used to store the changes permanently into the git repository

Pull :

• Pull operation copies the changes from the remote repository to a local machine.The pull opeartion is used for synchronisation between two repos

Branch :

• Product is same,so one repository but different task
• Each task has one separate branch
• Finally merges (code) all branches
• Useful when we want to work parallelly
• Can create one branch on the basis of another branch
• Changes are personal to that particular branch
• Default branch is ‘Master’
• FIle created in workspace will be visible in any of the branch workspace untill we commit.Once commit,then that file belongs to that particular branch

Advantages of Git :

• Free and Open Source
• Fast and small as most of the operations are performed locally,therefore it is fast
• Security : Git uses a common cryptographic hash function called Secure hash function(SHA-1) to name and identify objects within its databases
• No need of powerful hardware
• Easier branching : If we create a new branch,it will copy all the codes to the new branch

Types of Repositories :

• *Bare Repositories (Central Repo)*
☐ Store and share only
☐ All central repos are Bare repos

• *Nonbare repositories(Local Repo)*
☐ Where we can modify the files
☐ All local repos are non-bare repos

Create Linux mahines one in Mumbai and another one in Singapore region using AWS EC2 Instances

Install Git in Linux Machine

• Command to Use :(Do in both Machines)
☐ sudo su
☐ yum update -y
☐ yum install git -y
☐ git --version
☐ git config --global user.name “Username”
☐ git config --global user.email “example@gmail.com”

• *Create and Verify Github Account*

Commit,Push & Pull from Github :

• *Commands to use in* Mumbai Region :
☐ Login into Mumbai EC2 Instance

☐ Create one directory and go inside it → mkdir mumbaigit
☐ git init
☐ touch myfile → add some data
☐ git status
☐ git add .
☐ git commit -m “1st commit from Mumbai”
☐ git status
☐ git log
☐ git show <commit-id>
☐ git remote add origin <central git url>
☐ git push -u origin master(enter username & password)
☐ cat >>myfile → modify content
☐ git status
☐ git add .
☐ git commit -m “2nd commit from Mumbai”
☐ git status
☐ git log
☐ git show <commit-id>
☐ git push origin master(enter username & password)

• Commands to Use in Singapore EC2 Instance :
☐ Create one directory and go inside it → mkdir singaporegit
☐ git init
☐ git remote add origin <central git url>
☐ git pull -u origin master
☐ git log
☐ git show <commit-id>
☐ cat >>myfile → add some content
☐ git status
☐ git add .
☐ git status
☐ git commit -m “Singapore update 1”
☐ git status
☐ git log
☐ git show <commit -id>
☐ git push origin master(enter username & password)

• *Commands to use after Above steps in* Mumbai Instance :
☐ cd mumbaigit
☐ git pull origin master
☐ git log
☐ git show <commit id>

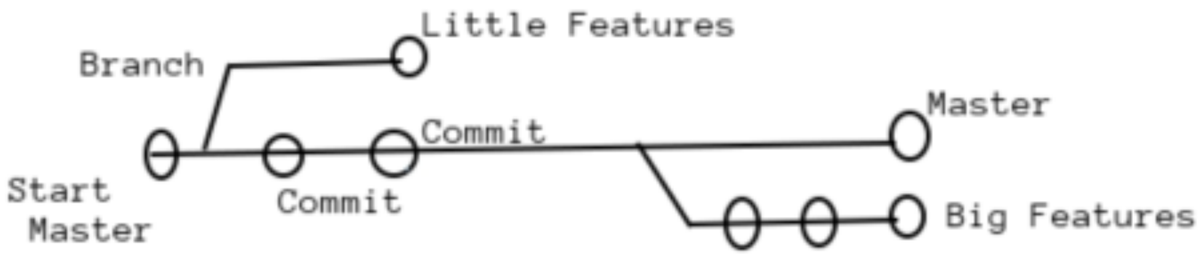
Gitignore : .gitignore

• To ignore some files while committing
• Create one hiddenfile .gitignore and enter file format which we want to ignore
• Ex : vi .gitignore → enter *.css *.java

• Commands to create .gitignore and add to Centralrepo :
☐ cd mumbaigit
☐ vi .gitignore → enter *.css *.java
☐ git add .gitignore

☐ git commit -m “added .gitignore”
☐ git status
☐ touch file1.text file2.txt file3.java file4.css file5.java
☐ git status
☐ git add .
☐ git status
☐ git commit -m “Latest updated excluded css & java files”
☐ git log
☐ git show <commit id>
☐ touch file6.java
☐ git status
☐ touch file7.txt
☐ git status
☐ git log -1
☐ git log -2
☐ git log --oneline
☐ git log --grep “ignore”
☐ git log --grep “commit”

Git Branches :



• The diagram above visualizes a repository with two isolated lines of development.
• One for a little features,and one for a longer-running features.
• By developing them with Branches,its not only possible to work on both of them in parallel,but it also keeps the main branch free from error

• Each task has one separate branch
• After done with code,merge other branches with master
• This concept is useful for parallel development
• We can create any no of branches
• Changes are personal to that Particular Branch
• Default branch is “Master”
• File created in workspace will be visible in any of the branch workspace untill we commit.Once we commmit,then that files belongs to that particular branch

• When created new branch,data of existing branch is copied to new branch.

Important Commands of Git Branches :

☐ git branch → To see list of available branches
☐ git branch <new branch name> → To Create a New Branch
☐ git checkout <branch name> → To switch to another branch
☐ git branch -d <branch name> → To delete branch

Commands to be used in Lab :

☐ sudo su
☐ cd singaporegit
☐ git log --oneline
☐ git branch
☐ git branch branch1
☐ git branch
☐ ls
☐ git checkout branch1
☐ git branch
☐ ls
☐ git log --oneline
☐ cat >branch1file → add content and save
☐ ls
☐ git add .
☐ git commit -m “branch1 first commit”
☐ git log --oneline
☐ git checkout master
☐ git log --online
☐ ls
☐ git branch
☐ git checkout branch1
☐ cat >secondfile → add content
☐ git checkout master
☐ ls
☐ git checkout branch1
☐ git add .
☐ git commit -m “branch1 2nd commit”
☐ git log --oneline
☐ git checkout master
☐ ls
☐ git log --oneline

Git Merge :

• We can not merge branches of different repositories
• We use pulling mechanism to merge Branches

Commands for Merge in Lab :

☐ git branch
☐ ls
☐ git log --online

☐ git merge branch1
☐ ls
☐ git log --oneline
☐ git push origin master

Git Conflict :

• When same file having different content in different branches,if we do merge,conflict occurs(Reslove COnflict then add and Commit)
• Conflict occurs when we merge branches.

Commands for git conflict in lab :

☐ git branch
☐ ls
☐ cat >rajputfile → add content
☐ git add .
☐ git commit -m “1st commit before conflict”
☐ git checkout branch1
☐ cat >rajputfile → add content
☐ git add .
☐ git commit -m “Commit from branch1”
☐ git branch
☐ git checkout master
☐ git merge branch1
☐ vi rajputfile → press i → modify content → press esc → enter :wq
☐ git status
☐ git add .
☐ git commit -m “commit after resloving conflict”
☐ git log --oneline

Git Stash :

• Suppose we are implementing a new feature for our product,our code is in progress and suddenly a customer escalation comes because of this,we have to keep aside our new feature work for few hours
• We cannot commit our partial code and also cannot throw away our changes.So we need some temporary storage,where we can store our partial changes and later on commit it
• To stash an item (only applies to the modified files not new file)

Commands for Stashing :

☐ git stash → To stash an item
☐ git stash list → To see stashed items list
☐ git stash apply stash@{0} → To apply stashed items
☐ Then we can add and commit
☐ git stash clear → To clear the stash items

Commands of git stash in Lab :

☐ touch demofile
☐ git add .

☐ git commit -m “demofile”
☐ git branch
☐ vi demofile → press i → add content → press esc → :wq
☐ git stash
☐ vi demofile → returns empty file
☐ cat demofile
☐ git stash
☐ git stash list
☐ vi demofile → press i → add content → press esc → :wq
☐ cat demofile
☐ git stash
☐ git stash list
☐ cat demofile
☐ git stash apply stash@{1}
☐ cat demofile
☐ git add .
☐ git commit -m “First code done”
☐ git stash apply stash@{0}
☐ vi demofile → press i → modify content → press esc → :wq
☐ git add .
☐ git commit -m “Second code done”
☐ git status
☐ git log --oneline
☐ git stash list
☐ git stash clear
☐ git stash list

Git Reset :

• Git reset is a powerful command that is used to undo local changes to the state of a git repo

Commands for git reset :

☐ git reset <file name>
☐ git reset .
☐ git reset --hard → To reset the changes from both staging area and working directory at a time....deletes files from workspace also

Commands for git reset in Lab :

☐ git branch
☐ cat >testfile → add content
☐ git add .
☐ git status
☐ git reset .
☐ git status
☐ git add .
☐ git status
☐ git reset --hard
☐ git status

Git Revert :

• The revert command helps us undo an existing commit
• It does not delete any data in this process.Instead, git creates a new commit with the included files reverted to their previous state.So our version control history moves forward while the state of our file moves backward

Commands of git revert in Lab :

☐ sudo su
☐ cd singaporegit
☐ ls
☐ git status
☐ git log
☐ cat >revertfile → add content and save
☐ git add .
☐ git commit -m “Code1”
☐ cat >>revertfile → add content and save
☐git add .
☐ git commit -m “Code2”
☐ git log --oneline
☐ git revert <Commit-Id>
☐ modify commit message using vi and save
☐ ls
☐ cat revertfile
☐ git log --oneline

How to Remove Untracked File :

• git clean -n → (dry run)
• git clean -f → (forcefully)

Commands to remove untracked file in lab :

☐ touch filex filey filez filexx filexy filexz
☐ git status
☐ git clean -n
☐ git clean -f
☐ ls

Tags :

• Tag operation allows giving meaningful names to a specific version in the repository

To apply Tags :
• git tag -a <tag-name> -m <message> <commit-id>

To see list of Tags :
• git tag

To see particular commit content by using Tag :
• git show <tag-name>

To delete a tag :
• git tag -d <tag-name>

Commands to use in Lab :

☐ git log --oneline
☐ git tag -a important -m “This is Very imp commit” <commid-id>
☐ git tag
☐ git show imporatnt
☐ git tag -d important
☐ git tag
☐ git log --online
☑ git tag -a important -m “This is Very imp commit” <Different commid-id>
☐ git tag
☐ git log --online

Github Clone :

• Open github website
• Login and choose existing repository
• Now,go to our linux machine,and run command :
☐ git clone <url of github repo>
• It creates a local repo automatically in linux machine with the same name as in github account

Commands to use in Lab :

☐ git clone <url of github repo>
☐ ls
☐ cd <name of cloned repo>
☐ touch fileyy
☐ git add .
☐ git commit -m “New commit”
☐ git push origin master

4.CHEF

CHEF - Configuration(Each and every minute detail of machine like Server,Storage etc) Management (Like Delete,Update,Create etc..) Tool

Two Types of Configuration Management Tools
• Push Based : Server pushes configuration to the nodes(Ansible,Salt Stack)
• Pull Based : Nodes check with the server.Periodically and fetches the configuration from it.(Chef,Puppet)

CHEF :

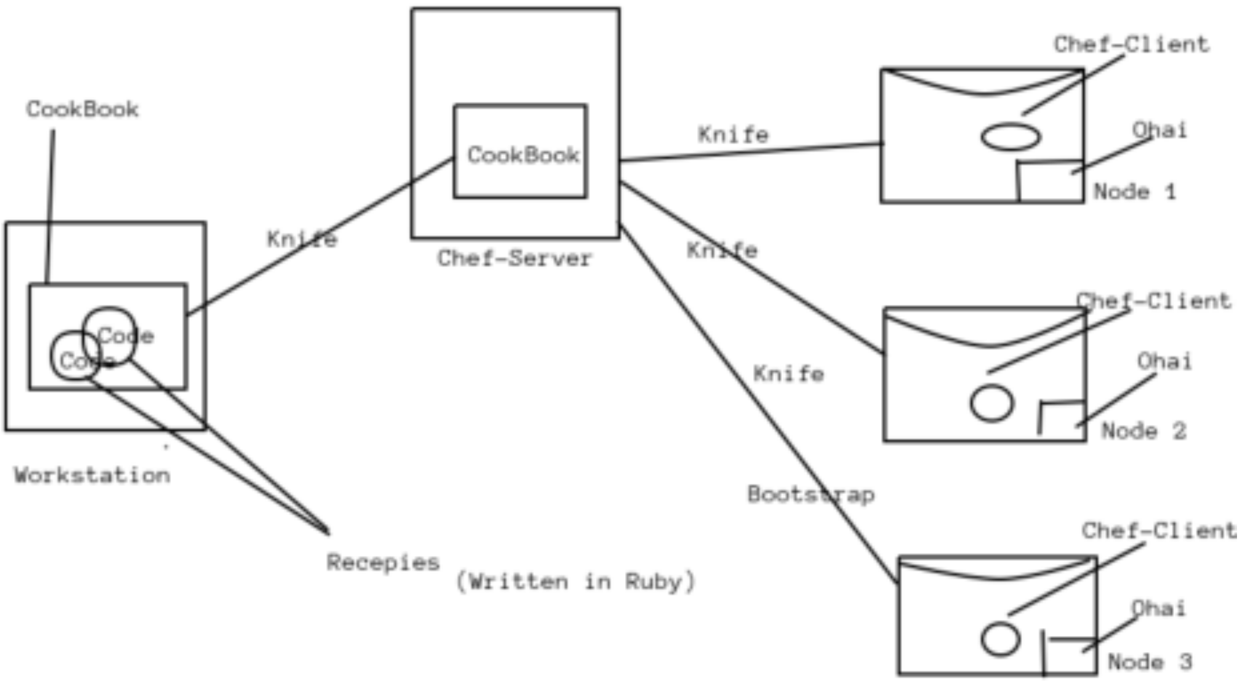
• Chef is a company and the name of a configuration management tool written in Ruby and Erlang
• Founded by Adam Jacobs in year 2009
• Actual name was “Marionette” later renamed to Chef
• On April 2,2019 the company announced that all their products are now open source under the Apache 2.0 License
• Chef is used by Facebook,AWS Opsworks,Hp Public Cloud etc
• Chef is a Admin. tool whatever system admins use to do manually,now we are automating all those tasks by using chef

Configuration Management : It is a method through which we automate admin tasks
• Config management tool turns our code into Infrastructure
• So our code would be repeatable testable and versionable

Advantages of CM Tools:

• Complete Automation
• Increase Uptime
• Improve Performance
• Ensure Compliance
• Prevent Errors
• Reduce Costs

Chef-Architecture or Process :



Components of Chef :

Workstation :Where we write code

• Workstations are personal computers or Virtual Servers where all configuration code is created,tested or changed
• Devops engineer actually sits here and write codes.This code is called recipe.A collection of recepies are called CookBooks
• Workstation communicate with the chef server using knife
• Knife is a command line tool that uploads the cookbook to the server

Chef-Server: Where we store code

• The chef-server is a middle-man between workstation and the nodes
• All cookbooks are stored here
• Server may be hosted locally or remote

Node : Where we apply code

• Nodes are the systems that require the configuration
• Ohai fetches the currents state of the node its located in
• Node communicates with the chef-server using the chef-client
• Each node can have a different configuartion required
• Chef-client is installed on every node

Knife : Tool to establish communication among workstation,server and node knife is a command-line tool that runs on workstation

Chef-Client : Tools runs on every chef node to pull code from chef server
• Chef client will
☐ gather current system configuration
☐ download the desired system configuration from the chef server
☐ Configure the node such that it adhere to the policy

Ohai : Maintain Current state information of the chef-node

Idempitency : Tracking the state of system resources to ensure that the changes should not reapply repeatedly

Chef-SuperMarket : Where we get custom Code

Creating Cookbooks and Recipes:

• First of all create one linux machine in AWS
• Now use putty and access the machine
☐ Login as -- ec2 user
☐ sudo su
☐ yum update -y
• Now go to google & search www.chef.io
• Go to downloads→ chef workstation
• Enter name,email,company -- It automatically starts downloading → go to Downloads & copy the URL
• Now in linux-machine
☐ wget <url>

☐ ls → it shows chef .rpm package
☐ yum install <chef-workstation> -y
☐ which chef
☐ chef --version

Cookbook : It is a collection of recipes and some other files and folders

Inside Cookbook :

chefignore : Like .gitignore
kitchen.yml : for testing cookbook
Metadata.rb : name,version,author etc of cookbook
readme.md : information about usgae of cookbook
recipe : where we write code
spec : for unit test
test : for integration test

Lab :

☐ which chef
☐ mkdir cookbooks
☐ ls
☐ cd cookbooks/
☐ chef generate cookbook test-cookbook
☐ yum install tree -y
☐ tree
☐ cd test-cookbook
☐ chef generate recipe <name>
☐ tree
☐ cd ..
☐ vi test-cookbook/recipes/test-recipe.rb
☐ add following code :

```
    File '/myfile' do
    Content 'Welcome to TG'
     action :create
      end
```

☐ chef spec ruby -c test-cookbook/recipes/test-recipe.rb
☐ chef-client -zr "recipe[test-cookbook:test-recipe]"
☐ ls /

Creating and Writing Second Recipe :

☐ cd test-cookbook
☐ chef generate recipe recipe2
☐ cd ..
☐ vi test-cookbook/recipes/recipe2.rb
☐ add following code :

```
    package 'tree' do
    action install
    end

     File '/myfile2' do
    Content ‘second project code’
     action :create
     owner :root
     group:root
      end
```
☐ chef-client -zr “recipe[test-cookbook::recipe2]”
☐ cat /myfile2
☐ yum remove tree -y
☐ chef-client -zr “recipe[test-cookbook::recipe2]”

Deploying an Apache Server

☐ ls
☐ chef generate cookbook apache-cookbook
☐ ls
☐ cd apache-cookbook
☐ chef generate recipe apache-recipe
☐ tree
☐ cd ..
☐ ls
☐ vi apache-cookbook/recipes/apache-recipe.rb
☐ add following :
```
       package “httpd” do
       action install
       end

        File ‘/var/www/html/index.html’ do
        content ‘Welcome to TG’
        option : create
        end

        Service ‘httpd’ do
        action [:enable,:start]
        end
```

☐ chef exec ruby -c apache-cookbook/recipes/apache-recipe.rb
☐ chef-client -zr “recipe[apache-cookbook::apache-recipe]”

Resource : It is the basic components of a recipe used to manage the infrastructure with different kind of states.There can be multiple resources in a recipe,which will help infrastructure
Ex:
Package : Manages the package on a node
Service : Manages the services on a node
User : Manages the users on the node
Group : Manages groups
Template : Manages the files with embedded ruby template
Cookbook-File : Transfers the files from the files subdirectory in the cookbook to a location on the node
File : Manages the content of a file on the node
Execute : Executes a command on the node

Cron : Edits an existing cron file on the node
Directory : Manages the directory on the node

CHEF Attributes :

Attributes : It is a Key value pair which represents a specific details about a node

• Attributes are used by chef-client

• Attribute are used to determine :
☐ The current state of the node
☐ What the state of the node was at the end of the previous chef-client run
☐ What the state of the node should be at the end of the current chef-client

• Types of Attributes :
☐ Default
☐ Force_Default
☐ Normal
☐ Override
☐ Force_Override
☐ Automatic

• Attributes are defined by
☐ Node(Collected by ohai at the start of each chef-client run)
☐ Cookbooks(Attribute Files)
☐ Roles
☐ Environment

Note:Attributes defined by Ohai have the highest priority,followed by attributes defined in a recipe then attributes defined in an attribute files

Lab :

☐ login into Aws Linux Machine
☐ sudo su
☐ ls
☐ ll
☐ ohai
☐ ohai ipaddress
☐ ohai memory/total
☐ ohai cpu/0/mhz
☐ ls
☐ cd Cookbooks/
☐ ls
☐ cd apache-cookbook/
☐ tree
☐ chef generate recipe recipe-new
☐ cd ..
☐ vi apache-cookbook/recipes/recipes-new.rb
☐ add following :

```
    File ‘/basicinfo’ do
```

```
      Content "This is to get Attributes
      HOSTNAME:  #{node['hostname']}
      IPADDRESS:  #{node['ipadress']}
      CPU:  #{node['cpu']['0']['mhz']}
      MEMORY:  #{node['memory']['total']}"
      owner ‘root’
      group ‘root’
      action  :'create
      end
```
☐ chef-client -zr “ ”recipe[apache-cookbook::recipe3]"
☐ ls /
☐ cat /basicinfo

• Executing Linux Commands:

☐ Login to AWS Linux Machine
☐ sudo su
☐ cd cookbooks
☐ ls
☐ vi test-cookbook/recipes/test-recipes.rb
☐ add following
```
     execute "run a script" do
        command <<-EOH
        mkdir /rajputdir
        touch /rajputfile
        EOH
      end
```
☐ chef exec ruby -c test cookbook/recippes/test-recipe.rb
☐ chef-client -zr “recipe[test-cookbook::test-recipe]”
☐ ls /
☐ vi test-cookbook/recipes/test-recipe.rb
☐ add following
```
        user “rajput” do
        action :create
        end
```
☐ chef-client -zr “recipe[test-cookbook::test-recipe]”
☐ vi test-cookbook/recipes/test-recipe.rb
☐ add following
```
        group “technicalguftugu” do
        action :create
        members ‘rajput’
        append true
        end
```
☐ chef-client -zr “recipe[test-cookbook::test-recipe]”
☐ cat /etc/group

• We run chef client to apply recipe to bring node into desired state.This process is known as Convergence

Runlist :

• To run the recipes in a sequence order that we mention in a run list
• With this process,we can run multiple recipes,but the condition is,there must be only one recipe from one cookbook
☐ chef-client -zr “recipe[test-cookbook::test-recipe],recipe[apache-cookbook::apache-recipe]”

How to Include Recipes:
• To call recipe/recipes from another recipe with in same cookbook
• To run multiple recipes from same cookbook
• Here comes the default recipe into action(we can use any recipe)
• We can run any no. of recipes with this command,but all must be from same cookbook
☐ vi test-cookbook/recipes/default.rb
☐ add following

```
include-recipe “test-cookbook::test-recipe”
include-recipe “test-cookbook::recipe2”
```

☐ chef-client -zr “recipe[test-cookbook::default]"

• Combining previous 2 concepts to run multiple recipes from multiple cookbooks simultaneously

☐ chef-client -zr “recipe[test-cookbook::default],recipe[apache-cookbook::default]”
(OR)
☐ chef-client -zr “recipe[test-cookbook],recipe[apache-cookbook]”

Chef Server and Node:

• Chef server is going to be a mediator for the code or cookbooks
• Firstly create one account in chef-server
• Then,attach our workstation to the chef-server
• Now upload cookbooks from workstation to chef server
• Now attach nodes to chef server via bootstrap process
• Apply cookbooks from chef server to Node

Lab :
☐ Login to amazon linux machine using putty
☐ ls
☐ cd cookbooks/
☐ ls

☐ Open google chrome→ search manage.chef.io
☐ Create one account
☐ Go to chef account → click on organisation→ starter kit→ download starter kit
☐ Open the downloaded content → unzip → chef-repo
☐ Now download winscp → login with ec2 credentials
☐ Now drag & drop chef folder from window to linux

☐ Now open workstation in AWS again
☐ ls
☐ cd ..
☐ ls

☐ cd chef-repo/
☐ ls -a
☐ cd .chef/
☐ ls
☐ cat config.rb
☐ cd ..
☐ knife ssl check

Bootstrap a Node:
• Attaching a node to chef server is called Bootstrapping(Both workstation and node should be in same az)
• Now onwards,we have to be inside chef-repo directory to run any command
• Two actions will be done while bootstrapping:
☐ Adding node to chef-server
☐ Installing chef package

• Create one linux machine(node 1) ,launch in same AZ
• Advance Details

```
#!/bin/bash
sudo su
yum update -y
```

• Now go to chef-workstation
• cd chef-repo
• Paste node-key.pem in chef-repo folder from local pc
• knife bootstrap <private ip of node> --ssh-user ec2-user --sudo -i node-key.pem -N node1
• knife node list

• under chef-repo do ls
• cd ..
• ls
• mv cookbook/test-cookbooks chef-repo/cookbooks
• mv cookbook/apache-cookbooks chef-repo/cookbooks
• rm -rf cookbooks/ → inside ec2-user
• cd chef-repo
• ls
• ls cookbooks/

Uploading apache-cookbook into chef-server:

☐ knife cookbook upload apache-ccokbook
☐ knife cookbook list
☐ knife node run_list set node1 “recipe[apache-cookbook::apache-recipe]”
☐ knife node show node1

Accessing Node1:

☐ sudo su
☐ chef-client

Inside chef-repo :

☐ vi cookbooks/apache-cookbbok/recipes/apache-recipe.rb → change some content and

save
☐ knife cookbook upload apache-cookbook
☐ Now run chef-client command inside node1 again

Automating chef-client
☐ inside node1
☐ vi /etc/crontab
☐ add following:
```
    * * * * * root chef-client
```
• Now in chef-repo
☐vi cookbooks/apache-cookbooks/recipes/apache-recipe.rb
☐ knife cookbook upload apache-ccokbook
☐ Open browser and check updated content

Managing Multiple Nodes:
• Now,create another linux machine(node2)
• Advance details
```
 !#/bin/bash
 sudo su
 yum update -y
 echo “* * * * * root chef-client” >> /etc/crontab
```
• Now in workstation run bootstrap command
• Now attach cookbook to node run list
• Now in browser check node 2 webpage

Commands to delete and clean chef-server:

☐ knife cookbook list
☐ knife cookbook delete <cookbook name> -y
☐ knife node list
☐ knife node delete <node name> -y
☐ knife client list
☐ knife client delete <client name> -y
☐ knife role list
☐ knife role delete <role name> -y

Lab :
• In Workstation
☐ sudo su
☐ ls
☐ cd chef-repo
☐ knife node list
☐ knife node delete node1 -y
☐ knife node delete node2 -y
☐ knife node list
☐ knife cookbook list
☐ knife cookbook delete apache-cookbook -y
☐ knife client list
☐ knife client delete node1 -y
☐ knife client delete node2 -y

Chef-role:

☐ inside chef-repo do ls
☐ cd roles/
☐ ls
☐ vi devops.rb
☐ add following :
    name “devops”
    description “web server role”
     run_list “recipe[apache-cookbook::apache-recipe]”
☐ now inside chef-repo
☐ knife role from file roles/devops.rb
☐ knife role list

• Now create 2 instances as node1 and node2 in same az as of workstation
☐ knife bootstrap <private ip of node> --ssh-user ec2-user --sudo -i node-key.pem -N node1
☐ knife bootstrap <private ip of node> --ssh-user ec2-user --sudo -i node-key.pem -N node2
• Now connect these nodes to role
☐ knife node list
☐ knife node run_list set node1 “role[devops]”
☐ knife node run_list set node2 “role[devops]”
☐ knife node show node1
☐ knife cookbook upload apache-cookbook
• check with public ip of any node
☐ vi cookbooks/apache-cookbook/recipes/apache-recipe.rb
☐ edit content and save
☐ knife cookbook upload apache-cookbook
☐ cat cookbooks/apache-cookbook/recipes/-recipe3.rb
☐ vi roles/devops.rb
☐ add following :
     name “devops”
     description “web server role”
     run_list “recipe[apache-cookbook::recipe3]”
☐ knife role from file roles/devops.rb

• Now access any node via putty & check

• Again in workstation
☐ vi roles/devops.rb
☐ add following :
     name “devops”
     description “web server role”
     run_list “recipe[apache-cookbook]”
☐ knife role from file roles/devops.rb
☐ vi cookbooks/apache-cookbook/recipes/-recipe3.rb
☐ add follwoing
 user “bhupinder”
 file"/bhupinderfile
☐ vi cookbooks/apache-cookbook/recipes/apache-recipe.rb
☐ edit content and save
☐ knife cookbook upload apache-cookbook
☐ vi roles/devops.rb

☐ add following :
```
      name “devops”
      description “web server role”
      run_list “recipe[apache-cookbook],recipe[test-cookbook]”
```
☐ knife role from file roles/devops.rb
☐ knife cookbook upload test-cookbook
• inside chef-repo
☐ vi cookbooks/test-cookbook/recipes/test-recipe.rb
☐ add following:
```
     %w(httpd mariadb-server unzip git vim) .each do |p|
      package p do
      action :install
       end
       end
```
☐ knife cookbook upload test-cookbook
• Now inside any node search git or vim(which git or which vim) etc to check it is working properly

~~~~~~

5.Docker

Docker :Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

• Docker is an open source centralised platform designed to create,deploy and run applications
• Docker uses container on the host O.S to run applications.It allows applications to use the same linux kernel as a system on the host computer,rather than creating a whole virtual os
• We can install docker on any O.S but docker engine runs natively on linux distribution
• Docker written in ‘go’ language
• Docker is a tool that performs O.S level virtualization also known as containerization
• Before Docker many users faces the problem that a particular code is running in developers system but not in the users systems
• Docker was first released in March 2013.It is developed by solemen Hykes and Sebastian Patil
• Docker is a set of platform as a service that uses O.S level virtualization whereas VMware uses hardware level virtualization

Advantages of Docker:

• No pre-allocation of RAM
• CI Efficency → Docker enables you to build a container image and use that same image across every step of the deployment process
~~~~~~

• Less cost
• It is light in weight
• It can run on physical H/W or Virtual H/W or on cloud
• We can re use the image
• It takes very less time to create container

Disadvantages:

• Docker is not a good solution for application that requires rich GUI
• Difficult to manage large amount of containers
• Docker does not provide cross-platform compatibility means if an application is designed to run in a docker container on windows,then it cant run on linux or vice-versa
• Docker is suitable when the development OS and testing OS are same i.e if dev oS is Centos then testing OS also need to be Centos but not Ubuntu or other linux distro... if the OS is different,we should use VM
• No solution for data recovery and backup

Architecture of Docker:

Docker Ecosystem: Docker client → Docker Daemon or Server or Docker Engine → Docker Hub or Registry → Docker Images → Docker Compose

Docker Daemon :
• Docker daemon runs on the host OS
• It is responsible for running containers to,manage docker services
• Docker Daemon can communicate with other daemons

Docker Client:
• Docker users can interact with docker daemon through a client
• Docker client uses commands and rest api to communicate with docker daemon
• When a client runs any server command on the docker client terminal,the client terminal sends these docker commands to the docker daemon
• It is possible for docker client to communicate with more than one daemon

Docker Host :
• DOcker host is used to provide on environment to execute and run applications.It contains the docker daemon,images,containers,networks and storages

Docker Hub/Registry:
• Dockers registry manages and stores the docker images
• There are two Types of registries in docker:
☐ Public Registry: Public registry is also called as docker hub
☐ Private Registry:It is used to share images within the enterprise

DOcker IMages:
• Docker Images are the read only binary template used to create docker containers
(OR)
• Single file with all dependencies and configuration required to run a program

☐ Ways to create an Images:
1. Take image from docker hub
2. Create image from docker file

3. Create image from existing docker container

Docker Container:
• Container holds the entire package that is needed to run the application
OR
• In other words we can say that,the image is a template and the conatiner is a copy of the template
• Container is like a Virtual Machine
• Images becomes container when they run on docker engine

LAB:

• docker images → List all images present in local machine
• docker search centos → find out images in docker hub
• docker pull centos → download image from docker hub to local machine
• docker run -it --name containername centos /bin/bash → Creating and giving name to container and i means interactive mode and t means terminal → run means create and start
• systemctl start/stop/status docker.service OR service docker start/stop/status
• docker start containername → Starting docker
• docker attach containername → go inside container
• docker ps -a → list all containers → ps means process status
• docker ps → list only running containers
• docker stop conatinername
• docker rm containername

Docker file Components & Diff Command:

Lab:
• docker run -it --name containername2 centos /bin/bash
• cd tmp/
• touch myfile
• docker diff containername
☐ C /root → C means Change
☐ A /root/.bash_histroy → A means Append
☐ C /tmp
☐ A /tmp/myfile
☐ D → means Delete
• docker commit newconatiner newimagename → Create image of new container
• docker images
• docker run -it --name containername newcontainerimagename /bin/bash
• ls
• cd tmp/
• ls
• myfile

Docker FIle :

• Dockerfile is basically a text file.It containes some set of instructions
• Automation of docker image creation
Dockerfile components:

☐ FROM : FOr base image this command must be on top of the docker file
☐ RUN : To execute commands,it will create a layer in image
☐ MAINTAINER : Author/Owner/Description
☐ COPY : Copy files from local system(docker vm) we need to provide source,destination...
(We cant download file from internet and any remote repo)
☐ ADD : Similar to copy but,it provides a feature to download files from internet,also we extract file at docker image side
☐ EXPOSE : To expose ports such as por 8080 for tomact,port 80 for nginx
☐ WORKDIR : TO set working directory for a container
☐ CMD : Execute commands but during container creation
☐ ENTRYPOINT : Similar to CMD,but has higher priority over CMD,first commands will be executed by ENTRYPOINT only
☐ ENV : Environment Variables
☐ ARG :

LAB:
• Create a file named DOckerfile
• Add instructions in Dockerfile
• Build Dockerfile to create image
• Run image to create Container

• vi Dockerfile
☐ FROM centos
☐ WORKDIR /tmp
☐ RUN echo “DarkRose” > testfile
☐ ENV myname Yaseen
☐ COPY testfile1 /tmp
☐ ADD test.tar.gz /tmp

• docker build -t myimg .
• docker ps -a
• docker images
• docker run -it --name name myimg /bin/bash
• cat /tmp/testfile

Docker Volume :
• Volume is simply a directory inside our container
• Firstly we have to declare the dir as a volume and then share volume
• Even if we stop container,still we can access volume
• We can declare a dir as a voume.Only while creating container
• We cant create volume from existing container
• We can share one volume across any number of containers
• Volume will not be included when we update an image
• We can map volume in two ways:
☐ Container <----→ Container
☐ Host <---→ Container

Benefits of Volume:
• Decoupling container from storage
• Share volume among diff containers
• Attach volume to containers
• On deleting container volume does not delete

Lab:

• vi Dockerfile
☐ FROM centos
☐ VOLUME [“/myvolume1”]
• docker build -t myimage .
• docker run -it --name containename myimage /bin/bash
• ls
• docker run -it --name conatiner2 --privileged=true --volumes-from container1 centos /bin/bash
• touch /myvolume1/exfile
• docker start container1
• docker attach container1
• ls /myvolume1

Creating volume using Command:
• docker run -it --name conatiner3 -v /volume2 centos /bin/bash
• cd /volume2
• touch cont3file
• docker run -it --name conatiner4 --privileged=true --volume-from container3 centos /bin/bash
• cd /volume2
• touch cont4file

VOlume sharing between HOST to Container

• docker run -it --name container6 -v /home/yaseen/Documents/dockervolume --privileged=true centos /bin/bash
• cd /dockervolume
• ls
• touch cont6file
• exit
• Verify files in host

SOme other commands
• docker volume ls
• docker volume create volumename
• docker volume rm volumename
• docker volume prune → Remove all unused docker volume
• docker volume inspect volumename
• docker container inspect containername

Docker Port Expose :

LAB:
• docker run -td --name container7 -p 80:80 ubuntu
• docker ps
• docker port container7
• docker exec -it container7 /bin/bash
• apt update && apt install apache2 -y

• cd /var/www/html
• echo “Darkrose” > index.html
• service apache2 start
• docker run -td --name myjenkins -p 8080:8080 jenkins

Difference between Docker attach and DOcker execute :
• Docker exex creates a new process in the containers environmnat while docker attach just conect the standard Input/Output of the main process inside the conatiner to corresponding standard input/output error of current terminal
• Docker exec is specifically for running new things in a already started container be it a shell or some other process

Diff bwteen expose and publish :
• Basically we have three options:
☐ Neither specify expose nor -p
☐ Only specify expose
☐ Specify expose and -p

• If we specify neither expose nor -p the service in the container will only be accessible from inside the container itself
• If we expose a port,the service in the container is not accessible from outside docker but from inside other docker containers,so this is good for inter-container communication
• If we expose and -p a port,the service in the container is accessible from anywhere,even outside docker
• If we use -p but do not expose docker does an implict expose.Thi is becaue if a port is open to the public it is automatically also open to the other docker containers.Hence -p includes expose

Docker Hub :

LAB:

• docker run -it --name container8 ubuntu /bin/bash
• Create some file inside container
• docker commit container8 myimg
• Create account in hub.docker.com
• docker login
• enter username and password
• docker tag myimg dockerid/newimage
• docker push dockerid/newimage
• docker pull dockerid/newimage
• docker run -it --name container9 dockerid/newimage /bin/bash

COmmands:

• docker stop $(docker ps -a -q) → stop all running containers
• docker rm $(docker ps -a -q) → Delete all stopped containers
• docker rmi -f $(docker images -q) → Delete all images

6.Ansible

Ansible is an open source It config mgt,deployment and orchestration tool.It aims to provide arge productivity gains to a wide variety of automation chaenge

Ansible History:
➢ Michael Dehaan developed ansible and the ansible project began in February 2012.
➢ Redhat acquired the ansible tool in 2015.
➢ Ansible is available for RHEL, Debian, cent OS and oracle Linux.
➢ We can use this tool whether your servers are in on-premises or in cloud.
➢ It turns your code into infrastructure i.e. your computing environment has some of the same

Advantages:
➢ Ansible is free to use by everyone.
➢ Ansible is very consistent and light weight and no constrains regarding the OS or underlying hardware are present.
➢ It is very secure due to its agentless capabilities and open SSH security features.
➢ Ansible doesn’t need any special system administrator skills to install and use it.
➢ It is push mechanism.

Disadvantages:
➢ Insufficient user interface, though ansible tower is GUI, but it is still in development stage.
➢ Cannot achieve full automation by ansible.
➢ New to the market, therefore limited support and document is available.

Terms used in Ansible:

a. Ansible Server: the machine where ansible is installed and from which all tasks and Playbooks will be run.
b. Module: basically, a module is a command or set of similar commands meant to be executed on the client side.
c. Task: a task is section that consist of a single procedure to be completed.
d. Role: a way of organizing tasks and related files to be later called playbook.
e. Fact: information fetched from the client form the global variables with the gather facts operation.
f. Inventory: file containing data about the ansible client servers.
g. Play: execution of playbook.
h. Handler: task which is called only if notifier is present.
i. Notifier: section attributed to a task which calls a handler if the output is changed.
j. Playbooks: it consists code in YAML format which describes tasks to be executed.

Go to AWS account- create 3 EC2 instances in same AZ.
Take access of all machines via putty.
Now go inside ansible server and download ansible package
# wgetd http://dl.fedoreaproject.org/pub/epel/epel-release-latest-7.noarch.rpm
Now do #ls
# yum install epel-rerlease-latest.7.noarch.rpm
#yum update -y
Now we have to install all the packages one by one
# yum install git python python-level python.pip openssl ansible -y

Now go to host file inside ansible server and paste private ip of node1 and node2
# vi /etc/ansible/hosts
Now this host file is only working after updating ansible.cfg file
# vi /etc/ansible/ansible.cfg
Uncommented
#inventory = /etc/ansible/hosts
# sudo-user = root
If you want to run any command then remove # from it, then it will work

Now create one user, in all the three instances
# adduser ansible
Now set password for this user
# passwd ansible
Now switch as ansible user
# su – ansible
This ansible user don’t have sudo priviledges right now. If you want to give sudo ppriviledge to ansible user

# visudo

Now go inside this file
Root ALL= (ALL) ALL
(ansible ALL= (ALL) NOPASSWD: ALL)
:wq!
Now do this thing in other nodes also.

Now go to ansible server and try to install httpd package as an ansible user.
# sudo yum install httpd -y
Now establish connection between server and node, go to ansible server
$ ssh 172.31.41.240
o/p- permission denied
now we have to do some changes in sshd-config file, go to ansible server
# vi /etc/ssh/sshd-config
Do some changes and saved the file.
Do this work in node1 and node2 also.
Now verify in ansible server
# su -ansible
# ssh 172.31.41.240
Now it asks for passwd, enter the password after that you will be inside node1.
Now go to ansible server and create keys.
Run this command as ansible user.
# ssh-keygen
# ls -a
o/p- .ssh
# cd .ssh/
Ls
o/p- id_rsa id_rsa_pub
now I need to copy public key in both the nodes.
# ssh-copy-id ansible(username)@172.31.41.240
Ask for password
# ssh-copy-id ansible@172.31.41.228
Ask for password

Now verify, go to ansible server
# cd ..
# ssh 172.31.41.240
Now you will enter into node1.

Host Patterns:
# vi /etc/ansible/hosts
“all” pattern refers to all the machines in an inventory
Ansible all –list-host
Ansible <group name> --list-hosts
Ansible <groupname>[0] –list-hosts

Groupname[0]- picks first machine of the group
Groupname[1]- picks second machine of the group
Groupname[-1]- picks last machine of the group
Groupname[0:1]- picks first two machines of the group
Groupname[2:5]- picks 3,4,5 and 6 machines of the group

Group separated by colon (:) can be used to use hosts from multiple groups.
Groupname1:groupname2

Ad-hoc Commands:
➢ Ad-hoc commands are commands which can be run individually to perform quick functions.
➢ These ad-hoc commands are not used for configuration management and deployment,
➢ The ansible ad-hoc commands uses the /usr/bin/ansible command line tool to automate a single task.

Go to ansible server
$ ansible demo -a “ls”
$ ansible demo[0] -a “touch filez”
$ ansible all -a “touchfile4”
$ ansible demo -a “ls-al”
$ ansible demo -a “sudo yum install httpd -y” or
$ ansible demo -ba “yum install httpd -y”
$ ansible demo -ba “yum remove httpd -y”

Ansible Modules:

➢ Ansible ships with a number of modules (called module library) that can be executed directly on remote hosts or through “playbooks”.
➢ Your library of modules can reside on any machine and there are no servers, daemons or databases required.

Q. where ansible modules are stored?
The default location of the inventory file is /etc/ansible/hosts.

$ ansible demo -b -m yum -a “pkg=httpd state=present”
$ ansible demo -b -m yum -a “pkg=httpd state=latest”
$ ansible demo -b -m yum -a “pkg=httpd state=absent”
$ ansible demo -b -m service -a “name=httpd state=started”

```
$ ansible demo -b -m user -a “name=raj”
$ ansible demo -b -m copy -a “src=file4 dest=/tmp”

$ ansible demo -m setup
$ ansible demo -m setup -a “filter= *ipv4* ”
```

Playbook:

➢ Playbooks in ansible are written in YAML format.
➢ It is human readable data serialization language and is commonly used for configuration files.
➢ Playbook is like a file where you write codes consists of variables, tasks, handlers, files, templates and roles.
➢ Each playbook is composed of one or more ‘modules’ in a list. Module is a collections of configuration files.
➢ Playbooks are divided into many sectors like

a. Target section: defines the host against which playbooks task has to be executed.
b. Variable: define variables
c. Task section: list of modules that we need to run in an order.

YAML (Yet Another Markup Language):

➢ For ansible nearly every YAML files starts with a list.
➢ Each item in the list is a list of key-value pairs commonly called as a dictionary.
➢ All YAML files have to begins with “---” and ends with “…”.
➢ All members of a list lines must begin with same indentation level starting with “-”.

For e.g:

```
--- # a list of fruits

       Fruits:
     - Mango
     - Strawberry
     - Banana
     - Grapes
     - Apple
…
```

A dictionary is represented in a sample key : value form

```
 --- # details of customer
Customer:

       Name: Nagarjuna

       Job: trainer

       Skills: devops

       Exp: fresher
...
```

Extension for playbook files is .yml.
Note: there should be space between : and value.

Go to ansible server.
Now create one playbook.

```
# vi target.yml
--- # target playbook
       -hosts: demo
        user: ansible
        become: yes
        connection: ssh
        gather_facts: yes

Esc- :wq!
```

```
$ ansible-playbook target.yml
```

Variables:

➢ Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles. They can be used to loop through a set of given values, access various information like the host name of a system and replace certain strings in templates with specific values.

➢ Put variable section above tasks so that we define it first and use it later.

Now go to ansible server and create one playbook.

```
$ vi vars.yml
--- # my variable playbook
- host: demo
  user: ansible
  become: yes
  connection: ssh

  vars:
        pkgname: httpd
  tasks:
        - name: install httpd server
          action: yum name= "{{pkgname}}" state=install
Esc - :wq!
```

Now execute playbook $ ansible-playbook vars.yml

Handlers Section:

A handler is exactly the same as a task, but it will run when called by another task.

Or

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.

DRY-RUN:

Check whether the playbook is formatted correctly or not.

Anible-playbook handlers.yml –-check

Go to ansible server

```
$ vi handler.yml
---# handlers playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
        - name: install httpd server
          action: yum name=httpd state=installed
          notify: restart HTTPD
   handlers:
          - name: restart HTTPD
            action: service name=httpd state=restarted

Esc- :wq!
```

Now execute this playbook

```
$ ansible-playbook handlers.yml
```

Loops: Sometimes you want to repeat a task multiple time. In computer programming this is called as loops. Common ansible loops include changing ownership on several files and/or directories with the file module, creating multiple users with the user module and repeating a polling step until certain result is reached.

Now go to ansible server.

```
$ vi loops.yml
---# my loops playbook
- host: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
        - name: add a list of users
          user: name= '{{item}}' state=present
                        With items:
                                  - bhupinder
                                  - nagarjuna
                                  - rajpal
                                  - kapil
                                  - dhoni

Esc- :wq!
$ansible-playbook loops.yml
```

To verify go inside node1

```
$cat /etc/passwd
```

Conditions: Whenever we have different different scenarios we put conditions to the scenario.

We put conditions in ansible by “when” statement.

```
---# condition palybook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
        - name: install apache on Debian
          command: apt-get -y install apache2
          when: ansible_os_family == “debian”
        - name: install apache for redhat
          command: yum -y install httpd
          when: ansible_os_family == “redhat”
```

Vault: Ansible allows keeping sensitive data such as passwords or key in encrypted files, rather that a plaintext in your playbooks.

Creating a new encrypted playbook:

```
$ ansible-vault create vault.yml
```

Edit the encrypted playbook:
```
$ ansible-vault edit vault.yml
```

To change the password:
```
$ ansible-vault rekey vault.yml
```

To encrypt on existing playbook:
```
$ ansible-vault encrypt target.yml
```

To decrypt an encrypted playbook:
```
$ ansible-vault decrypt target.yml
```

Roles:

➢ We can use two techniques for reusing a set of tasks : includes and role.
➢ Roles are good for organizing tasks and encapsulating data needed to accomplish those tasks.
➢ We can organize playbook into a directory structure called roles.
➢ Adding more and more functionality to the playbooks will make it difficult to maintain in a single file.

Ansible Roles:

a. Default: it stores the data about role/ application. Default variables e.g: if you want to run to port 80 or 8080 then variables need to define in this path.
b. Files: it contains files need to be transferred to the remote VM (static files).
c. Handles: they are triggers or task. We can segregate all the handlers required in playbook.
d. Meta: this directory contains files that establish roles dependencies. E.g: author name,supported platform, dependencies if any.

e. Templates:
f. Tasks: it contains all the tasks that is normally in the playbook. E.g: installing packages and copies files etc.
g. Vars: variables for the role can be specified in this directory used in your configuration files. Both vars and default stores variables.

```
$ mkdir -p playbooks/roles/webserver/tasks
$ tree -p playbook/roles/webserever/handler
$cd playbook/
$ tree
$ touch roles/webserver/tasks/main.yml
$ touch master.yml
$ vi roles/webserver/tasks/main.yml
```

```
 Inside main.yml
    - name: install apache
      yum: pkg=httpd state=latest
```

Esc- :wq!

```
$vi master.yml
- host: all
  user: ansible
  become: yes
  connection: ssh
  roles:
   - webserver
```

```
$ansible-playbook master.yml
```

7.Jenkins

• What is CI/CD Pipeline

☐ Continuous Integration/Continuous Delivery or Deployment

JENKINS

• Jenkins ia an opensource project written in Java that runs on windows,linux and Macos and other unix like OS.It is free community supported and might be your first choice tool for

CI
• Jenkins automate the entire SDLC
• Jenkins was originally developed by sun microsystems in 2004 under the name hudson
• The project was later named jenkins when oracle bought microsystems...Hudson is a enterprise edition
• It can run on any major platform without   any compatibility issues
• Whenever developers write code,we integrate all that code of all developers at that point of time and we build test ad deliver/deploy to the client .This process is called CI?CD
• Jenkins helps us to achieve the above goal
• Because of CI,now bugs will be reported fast and get rectified fast.So the entire software development happens fast

WorkFlow of Jenkins:

• We can attach git,Maven,Selenium and Artifactory plugins to Jenkins
• Once developers puts code in github,jenkins pull that code and send to Maven for buildf
• Once build is done,jenkins pulls that code and send to selenium for testing
• Once testing is done,then jenkins will pull that code and send to artifactory as per requirement and so on
• We can also deploy with Jenkins

Advantages of Jenkins:

• It has lots of Plugins
• We can write our own Plugins
• We can use Community Plugins
• Jenkins is not just a tool.It is a framework i.e we can do whatever we want .All we need is plugins
• We can attach slaves(nodes) to the master.It instruct other nodes(slaves) to do job.If slaves are not available,jenkins itself does the job
• Jenkins also behave as a cron server replacement i.e can do scheduled tasks
• It can create labels

Build(Means) : Compile -→ Code Review -→ Unit Testing -→ Integration Testing -→ Packaging(WAR,JAR)

Jenkins Plugins : Plugins are small libraries that add new abilities to jenkins and can provide integration points to other tools

MAVEN JOB,SCHEDULE TASK,POLL SCM

Lecture 4:

Lecture 5 : Maven and Why we use it??

Maven :

• Maven is an automation and project management tool developed by apache software foundation.It is based on POM(Project Object Model).xml(extensible markup language)
• Maven can build any number of projects into desired output such as .jar,.war,metadata
• Mostly used fot Java based projects
• It was initially released on 13 july 2004
• Maven is written in java
• Meaning of maven(in yiddish language) is “ Accumulator of Knowledge”
• Maven helps in getting the right jar file for each project as there may be different version of separate packages
• To download dependencies it is no more needed to visit the official website of each software.It could now be easily done by visiting “mvnrepository.com”

Dependencies : It refers to the java libraries that are needed for the project

Repositories : Refers to the directories of packaged jar files

Build Tools :

☐ C,C++ : Make File
☐ .net : Visual Studio
☐ Java : Ant,Maven,Gradel

Problems without Maven :

1. Adding set of jars in each project : In case of struts,spring,we need to add jar files in each project.It must include all the dependencies of jars also
2. Creating the right project structure : We must create the right project structure in servlet,struts etc otherwise it will not be executed
☐ Example : .war file layout
```
       new.jar → index.html
                        → web-INF
                                  → web.xml
                                  → weblogic.xml
                                  → classes
```
3.Building and deploying the project : We must have to build and deploy the project so that it may work

What Maven DOes ??

1. It makes a project easy to build
2. It provides project info(Ex: log documents,cross reference sources,mailig list,dependency list,unit tes)
3. Easy to add new dependencies
4. Therefore apache maven helps to manage:
• Build
• Dependencies
• Reports
• Releases
• Distribution

What is Build Tool??

• A build tool takecare of everything for building a process.It does following:
☐ Generate source code
☐ Generate documentation from source code
☐ Compiles source code
☐ Install the package code in local repo,server rpo or central repo

POM(Project object Model)

• POM refers to the XML files that have all the info regarding project and configuration details
• Main configuration file is pom.xml
• It has the description of the project,details regarding the versioning and config mgt of the project
• The XML file is in the project home directory

pom.xml COntents :
☐ Metadata
☐ Dependencies
☐ Kind of project
☐ Kind of Output(.jar,.war)
☐ Description

One Project → One Workspace → One pom.xml

Requirement to Build :

• Source code (Present in workspace)
• Compiler (Remote Repo → local repo → Workspace)
• Dependencies (Remote repo → Local repo → workspace)

Architecture of Maven :

• .m2 local repo
• pom.xml ,, goals(commands like mvn clean package) ,, source code(from git or github) → workspace
• remote or central repo like mvnrepository.com

Local repo : Local repo refers to the machine of the developer where all the project material is saved

Remote Repo : It refers to the repo present in webserver which is used when maven needs to download dependencies.This repo works same as the central repo whenever anything is needed from remote repo it is first downloaded to the local repo and then it is used

Central rpo : Central repo refers to the maven community that comes into action when there is a need of dependencies and those dependencies cannot be found in the local repo

Maven Build Life-cycle :

Goals :

1. Generate resource(Dependency)
2. Compile code
3. Unit test
4. Package(Build)
5. Install(into local repo and artifactory)
6. Deploy(to servers)
7. Clean(delete all run time files)

    Ex: mvn install ....mvn clean package

☐ 1 to 6 → default and sequence order
☐ 7 → not default and wont allow sequence

• Build lifecycle consists of a sequence of build phases and each build phase consist of a sequence of goals
• Each goal is responsible for a particular task
• When a phase is run, all the goals related to that phase and its plugins are also compiled

ANT :

• Ant does not has formal conventions, so we need to provide info of all the project structure in build.xml file
• Ant is procedural,we need to provide info about what to do and when to do through code
• There is no lifecycle in ant
• It is a toolbox
• It is mainly a build tool
• It is less prefered than maven

Maven :

• Maven has a convention to place source code,compiled codeetc so we dont need to provide info about the project structure is pom.xml file
• Maven is declarative ,everything we define in the pom.xml file
• There is a lifecycle in maven
• it is a framework
• It is mainly project mgt tool
• It is more preferred

Maven Directory Structure

☐ New project :
```
            → src
                → main
                → test
            → pom.xml
```

8.Nagios

Continuous Monitoring Tools - Nagios,Splunk,Prometheus,ELK,Sensu,Librate,Cloudwatch

• Nagios : It is an open source software for continuous monitoring of systems,networks and infrastructure.It runs plugins stroed on a server which is connected with a host or another server on our network,or the internet .In case of any failure,nagios alerts about the issues,so that the technical team can perform recovery process immediately

History of Nagios :

• In year 1999,ethan galstad developed it as part of Net Saint distribution
• 2002,ethan renames the project to “Nagios” because of trademark issues with the name “Netsaint”
• 204, nagios releases its first commercial version,Nagios xi
• In 212,nagios again renamed as nagios core
• It uses port number 5666,5667 and 5668 to monitor its client

Why Nagios :

• Detect all types of networks,or server issues
• Helps us to find the root cause of the problem which allows us to get the permanent solution to the problem
• Reduce downtime
• Active monitoring of entire infrastructure
• Allows us to monitor and troubleshoot server performance issues
• Automatically fix problems

Features of Nagios :

• Oldest and Latest
• Good log and database system
• Information and attractive web interface
• Automatically send alerts if condition changes
• helps us to detect network errors or server crashes
• We can monitor the entire business process and IT infrastructure with a single pass
• Monitor network services like http,smtp,snmp,ftp,ssh,pop,ldap,ipmi,dns etc

Phases of Continuous monitoring :

1. Define : Develop a monitoring strategy
2. Establish : How frequently we are going to monitor it
3. Implement
4. Analyze data and report finding
5. Respond
6. Renew and Update

Nagios Architecture :

• It is a client server architecture,usually on a network,a nagios server is running on a host and plugins are running on all the remote host which should we monitor

☐ How does it work ??

• Mention all details in configuration files
• Daemon reads those details what data to be collected
• Daemon use NRPE plug-in to collect data from nodes and store in its own database
• Finally shows everything in dashboard

Prerequisite :

• httpd (browser)
• php (dashboard)
• gcc & gd (compiler,to convert raw code into binaries)
• makefile (to build)
• perl (script)

☐ main configuration file .../usr/local/nagios/etc/nagios.cfg
☐ All monitoring things called as ‘service’
☐ Eg : 5 Servers → 4 checks each we have to monitor → 5X4=20services

Dashboard Overview :

• In dashboard,we can see

☐ host → down,unrecheable,up,recovery,none
☐ Services → warning,unknown,critical,recovery,pending`

• To start nagios core installation we must have a EC2 instance up and running and have already configured SSH access to the instance

1. Install pre-requisite software on our ec2 machine prior to nahios installation like apache,php,gcc compiler and gd development
☐ sudo su
☐ yum install httpd php
☐ yum install gcc glibc glibc-common
☐ yum install gd gd-devel

2.Create account info ....we need to setup a nagios user run following:
☐ adduser -m nagios
☐ passwd nagios
☐ groupadd nagioscmd
☐ usermod -aG nagioscmd nagios
☐ usermod -aG nagioscmd apache

3.Download nagios core and the plugins create a directory for storing the downloads:
☐ mkdir ~/Downloads

☐ cd ~/Downloads
☐ Download nagios core tar.gz
☐ Download nagios plugins from https://nagios-plugins.org/download/nagios-plugins-<version-number>.tar.gz

4.Compile and install nagios extract the nagios source code tarball

☐ tar zxvf nagios-<version num>.tar,gz
☐ cd nagios-<version num>
• run configuration script with the name of the group which we have created in above steps:
☐ ./configure --with-command-group=nagioscmd
☐ compile the nagios source code using command make all
• Install Binaries,init script,sample config files and set permissions on the external command directly
☐ make install
☐ make install-init
☐ make install-config
☐ make install-commandmode

5.Configure the web interface

☐ make install-webconf

6.Create a nagiosadmin account for login into the nagios web interface .Set password as weel

☐ htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
☐ service httpd restart

7.Compile and install the nagios plugins.Extract the nagios plugins source code tarball
☐ cd ~/downloads
☐ tar zxvf nagios-plugins-2.0.3.tar.gz
☐ cd nagios-plugins-2.0.3

• Compile and Install Plugins

☐ ./configure --with-nagios-user=nagios --with-nagios-group=nagios
☐ make
☐ make install

8. Start Nagios,add nagios to the list of system services and have it automatically start when the system boots
☐ chkconfig --add nagios
☐ chkconfig nagios on

9.Verify the sample nagios config files

☐ /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg

• If there no errors,start nagios and restart httpd
☐ service nagios start
☐ service httpd restart

10.Copy public ip o ec2 instance and paste in browser :ip/nagios/
☐ username : nagiosadmin.password:<given password>

9.Kubernetes

Understanding Containers:

• Images are like a linux DVD and contains all that is needed to start an application
• Containers are running instances of the application that is defined in the image
• Registries can be used to share conatiner images like docker hub

Container Solution :

• LXC offers linux native containers and has been around for long time
• systemd-nspawn offers systemd integrated containers
• Docker started in 2013 and has made containers the success its currently is
• Podman is offered in recent versions of RHEL and offers a more secure and lightweight solution to run containers

Corportae Container Requirements:

• Automated rollouts and rollbacks
• Container orchestration
• Storage Orchestration
• Service discovery and load balancing
• Self-healing
• High-availability
• Scalability
• Batch execution

Success of Kubernetes:

• Google released the large-scale cluster management at google with borg document in 2015: https://research.google/pubs/pub43438/
• This document was the blueprint for kubernetes
• As borg was not new,but already existed for 15 years,kubernetes was mature right from the moment it started
• The specifications were donated to cloud native computing foundation(CNCF) a foundation within linux foundation
• Donating it to CNCF guaranteed the public availability of kubernrtes code
• With the release of Kubernetes,all companies that were working on a solution to orchestrated containers have changed in order to work with kunernetes
• Kubernetes embodies the success of open source

Kubernetes Ecosystem:

• Kubernetes is core project managed by CNCF
• Kubernetes conference Kubecon is a major event about the current and future status of kubernetes projects
• A new release of kubernetes is published every 3 months

CNCF Projects:

• Diff CNCF projects exist and make kubernetes an easy to extend Solution
☐ CRI-O : a kubernetes Container Runtime
☐ CNI: common network interface
☐ Jaeger : an operator that eases packaging,deploying and managing applications
☐ Rook : a storage orchestrator for kubernetes
☐ and many more
• Notice that kubernetes itself is also “just” a CNCF project

Projects Status:

• Project status is determined by CNCF
☐ Sandbox : project is new and used by innovators only
☐ Incubating : adoption of project is becoming more wide spread
☐ Graduated : project is part of core kubernetes environment
• For current project status ,see https://www.cncf.io/projects/

Kubernetes Solutions:

• CNCF is owner of kubernetes code
• Diff commercial and open source distributions are available to work with kubernetes
☐ Rancher
☐ Red hat Openshift Container Platform
☐ SUSE containers as a service(Caas)
• Kubernetes can be used on premise but also as a managed solution in public cloud:
☐ AKS - Azure Kubernetes Service
☐ EKS : Amazon K Platform
☐ Google GKE is offered by google

Kubernrtes Certification

• CKAD(Certified Kubernetes Application Developer),CKA(Certified Kubernrtes Admin)

Understanding Containers:

• Applications need a runtime environment
• Traditional runtime env is a physical/virtua host on which an app is installed with all of its dependencies
• Dependencies can be a problem when updating apps
• As a complete runtime environment that is isolated from everything else on the systemc a container is offering a solution to that problem
• Containers are slef contained and never offer any dependency problems
• Containers are linux and are running on top of a container engine and as such dont offer any platform requirements either

Containers and VM's

• many containers can run on top of sam host kernel
• A container is not a VM

- Main operational diff is that a container has a default app that it must start
- Without specifying default app,the container will start and immediately stop again as it doesnt know what to do