Operating System

We have two types of Operating Systems:

1. Textual User Interface Operating System
    a. DOS (Disk Operating System)
    b. UNICS (Uniplexed Information and Computing Services) – Open Source Software
2. Graphical User Interface Operating System

MIMX-OS – before Unics

Operating Systems developed based on UNICS:

- IBM Z/OS
- IBM AIX
- Sun Solaris
- Mac OS
- HP UX

Linus Torvald: Invented Linux Operating System

- Linux is Kernel, not OS.
- Linux is not a Unix derivative, it is written from the scratch.
- A Linux distribution is the Linux Kernel and a collection of software that together can create a OS.

Types of Linux Operating Systems:

- RHEL (Red Hat Enterprise Linux)
- Fedora
- Debian
- Ubuntu
- CentOS
- Amazon Linux
- Oracle Linux
- Kali Linux (Hacking Operating System)
- Mandriva

File Level Permissions:

- File Level permissions are managed under 10 chars.
- The First char resembles whether it is a file or a directory (Directory(d), File(-))
- Next 3 chars resembles, owner permissions for read(r), write(w), execute(x) (4,2,1)
- Next 3 chars resembles, owner group permissions.
- Last 3 chars resembles other Group permissions.

- Example: drwxr-xr-- 2 <Owner> <group-name> <size in bytes> <date & time> <file/directory name>

Links:

We have two types of links:

1. Soft link/Symbolic link: This is equivalent to the shortcut in windows.
    a. Ex: ln -s <original-file> <link-file-name>
    Note: once original file is deleted soft link doesn't work anymore
2. Hard Link: This is equivalent to back of a folder/file which is in sync with the original file.
    a. Ex: ln <original-file> <link-file-name>
    Note: Even after original file is deleted, link file exists with the full backup

File Creation Methodologies:

1. cat
2. touch
3. vi editor
4. nano editor

Cat command:

- The cat command is one of the universal tools, yet all it does is copy standard input to standard output.
- **To create a file**:
    - cat > <file-name>
    - write the file content & CNTRL+D
- **To View a File**: Display's the content of the file
    - cat <file-name>
- To **append** content to an existing file:
    - cat >> <file-name>
    - Write the content of the file & CNTRL+D
- To **concatenate** two files:
    - cat <file1> <file2> **>** file3
- To **copy file** into new file:
    - cat <file1> > <file2>
- To **print/view** the content of the file in **reverse order**:
    - tac <file-name>

01-10-2021 08:30 PM

touch command:

- Touch can be used to create an empty file
- Touch can create multiple empty files at a time.
    - touch <file-name1> <file-name2> <file-name3>…

- Touch can also be used to change timestamp of a file.
- Touch can also be used to update access time of the file, modify timestamp of the file.
- **Timestamp**:
    - Access time: Last time when a file was accessed
        - touch -a <file-name>
    - Modified time: Last time when a file was modified
        - touch -m <file-name>
    - Change time (meta data change): Last time when a file metadata was changed

vi editor(command):

- A programmer/Administrator text editor.
- It can be used to edit all kinds of plain text or program files or configuration files. It is specially useful for editing UNIX programs.
  Note: Commands inside the vi editor
    - i - for insert mode
    - :w – to save the file
    - :wq – to save and quit from the editor
    - :q – to quit from the editor
    - :q! – force quit without saving file from the editor

nano editor:

- "vi" is a standard whereas nano has to be available depending on Linux that we use.
- If not available bydefault, we can install using "yum" command.

stat command:

- stat command is used to view the status of all the timestamps of the file.
- It shows timestamps like access time stamp, modified timestamp and change timestamp

Ex: stat <file-name>

mkdir command:

- This is used to make a directory or create a directory (folder).
- By default every directory created using this command will contain two hidden folders inside the directory.
    - . – resembles root
    - .. – resembles home
- Ex: mkdir <directory-name>

cd command:

- This command is used to change the directory path from current path to another directory path.
- Ex: cd /home/<user>
- cd .. : to come out of directory at one level.

pwd (present working directory):

- This command shows you the path where you are currently located.
- Ex: pwd

cp command (copy):

- Used to copy only files.
- We can use this command to copy a file from current path(absolute path) to another absolute directory path.
- Ex: cp <file1> <file-name/dir-path>
- Note: There is no change the stats of access timestamp & modified time stamp.

mv command (move):

- Used to move file content from one file to another (similar to rename file).
- Used to move a file from one path to another path.
- Ex: mv <file1> <file-name/dir-path>
- Note: There is no change the stats of access timestamp & modified time stamp.

rmdir command (remove directory):

- Remove directory command works only on empty directories.
- Remove directory can delete a single directory or it can also delete a recursive way of deletion only if all the directories are empty.
- Ex: rmdir <dir-name>
- Ex: rmdir -p <dir-name> (will delete parent and child directories)
- Ex: rmdir -pv <dir-name> (will delete parent and child directories in verbose mode (information about the process))

rm command (remove):

- Remove command works for files as well as directories.
- Remove command also works for non-empty directories.
- Ex: rm <file/dir>

- Ex: rm -rf <file/dir> (removes all the files/directories in the parent directory forcefully, even the non-empty)
- Ex: rm -r <file/dir> (removes all the files/directories only if they are empty in a recursive manner)

hostname command:

- Hostname command gives the host name of the computer (similar to computer name in windows).
- Ex: hostname

ifconfig command:

- It shows both IPV4 & IPV6 addresses.
- It also shows subnet masks of our network.
- It shows information about our public and private network details.

cat /etc/os-release command:

- This shows the OS version and patches details.

yum command:

- It is a powerful command in Linux which can install/remove/update any package(software).
- yum should be executed only in the root/sudo users.
    - Ex: yum install <package-name>
    - Ex: yum update /<package-name>
    - Ex: yum remove <package-name>
    - Ex: yum list installed – shows all the installed packages

which command:

- This is one of the command used in day-to-day life to know if the package is installed (where) or not.
- Ex: which <package-name>

whoami command:

- To find the user which you are working on.
- Ex: whoami

hostname -i command:

- This command displays only the ip address.

04-10-2021 08:30PM

Creation of New USER:

- useradd command: To create a new user in linux environment
  - syntax:]# useradd <user-name>
- To attach password to the newly user created:
  - syntax:]# gpasswd <user-name>
  - enter password & re-type new password again

Creation of a New Group:

- groupadd command: TO create a new group in linux environment
  - syntax:]# groupadd <group-name>
- to attach a user to the group:
  - syntax:]# gpasswd -a <user-name> <group-name>
- to attach multiple users at a time to a group:
  - syntax:]# gpasswd -m <user1,user2…userX> <group-name>

Viewing the Users & Group details:

- cat /etc/passwd → to view the list of users created
- cat /etc/groups → to view the list of groups created with users assignments

File permissions change commands:

- chmod command
- chown command

1. **chmod command**: Change Moderator command
   - This command can change the read, write and execute permissions for owner, owner group & other groups.
   - Read resembles 4, write resembles 2 & execute resembles 1. So total 7
   - If read and write are provided as a permission, you must enable mandatorily execute permission.
   - chmod 700 resembles, full permission to the owner i.e. r,w,x
   - chmod 500 resembles, read and execute permissions for the owner i.e. r,-,x
   - chmod 400 resembles, read only permission for the owner i.e. r,-,-
   - chmod 750 resembles, full permission to the owner. Read and execute permission to the owner group
   - chmod 740 resembles, full permission to the owner. Read only permission to the owner group.

- chmod 774 resembles -> full permission to the owner, Full permission to the owner group & Read only permission to other groups.
- chmod 775 resembles -> full permission to the owner, Full permission to the owner group and Read & Execute permission to other groups.

2. **chown command**: This command helps to change the ownership of the file/directory. Only either root or the file owner can execute this command.
   - Syntax: chown <new-owner-name> <file-name>

chgrp command: This command can change the user from existing group to new group.

Syntax:]# chgrp <new-group-name> <user-name>

Downloading a file from internet using command prompt:

- wget command: This command is used to download any packages or files or software using url
  - syntax:]# wget <url>

Compression/Zipping in Linux:

1. tar command
2. gzip command

1. tar command: (No compression)
   - tar is an archival command used to combine multiple files into one single file.
     - Syntax:]# tar -cvf <directory-name> <tar-file-name>.tar.gz
2. gzip command: (compression)
   - gzip is a compression tool used to reduce the file size.
     - Syntax:]# gzip -cvf <directory-name> <tar-file-name>.tar.gz

-**cvf options**:

- c resembles create
- v resembles verbose mode
- f resembles forcefully

Installing a web server on a Linux Environment:

- First we need to install WebServer package.
  - Syntax:]# yum install httpd
- We need to verify whether the webserver started or not!
  - Syntax:]# service httpd status
- If the server is not started, we need to start the server.
  - Syntax:]# service httpd start

- To verify the server from the browser, type the url in the browser as
    - <host-name>:80
    - It should display the Apache webserver home page.
- To host a website, place the website content into **/var/www/html/**

Sort command:

- This is used to sort the content of the file by alphabetical order in a line-by-line manner.
    - Syntax: sort <file-name>

Tree command:

- This shows the current directory structure in the form of a tree.
    - Syntax: tree <directory-name>
- Note: tree is not available by default in Linux environment, to install “tree” tool/package use yum command.

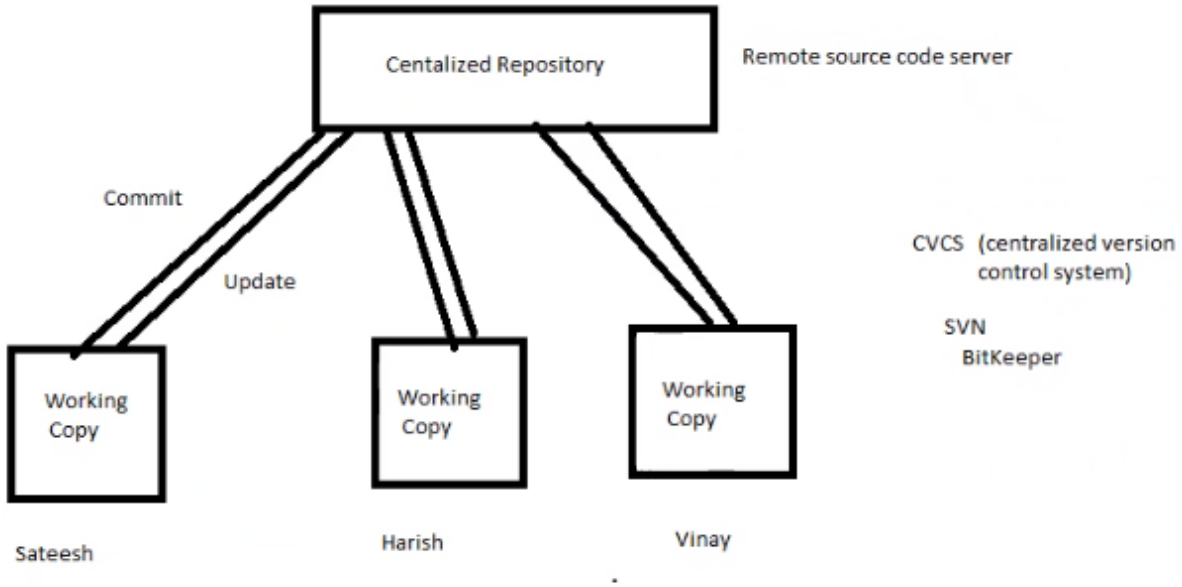
GIT

Source Code Management:

We have two kinds of Source Code Management Systems

1. Centralized Version Control System (CVCS)
2. Distributed Version Control System (DVCS)

1. Centralized Version Control System (CVCS):

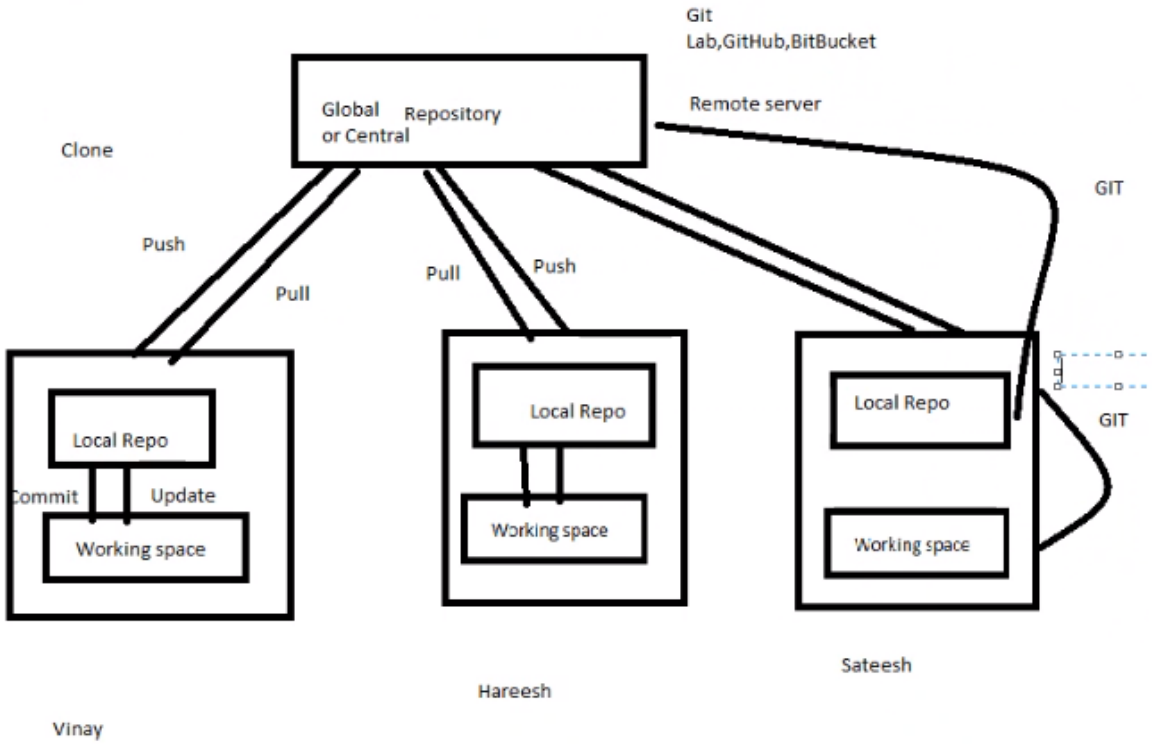


Drawbacks:

- It is not locally available, meaning you always need to be connected to the network to perform any action.
- Since everything is centralized, if central server gets failed, we will loose entire data.

2. Distributed Version Control System (DVCS):

- This concept was introduced by LINUS TORVALD in 2005.

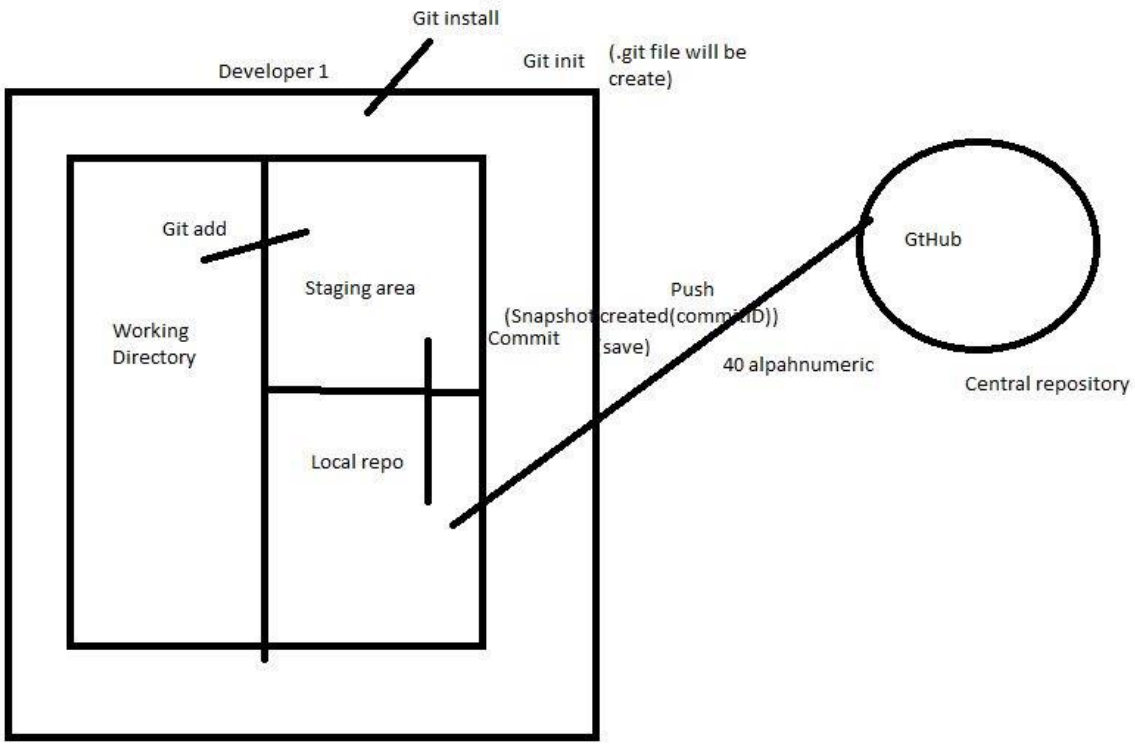


- In Distributed Version Control System every Contributor/Developer has a local copy or clone of Main/Global repository.
- Everyone maintains a local repository of their own which contain all the files and metadata present in main repository along with their own “commits” of local.

Difference between CVCS & DVCS:

	CVCS	DVCS
1	In CVCS, a client need to get a local copy of source from the server, make changes and commit those changes to central source on the server.	In DVCS, each client can have a local repository and have a complete history on it. Client need to push the changes to the server repository.
2	CVCS systems are easy to learn and setup.	DVCS are difficult for beginners as multiple commands need to be remembered.
3	Working on branches is difficult, Developer often faces merge conflicts.	Working on branches is easier, Developer faces less conflicts.
4	Don’t provide offline access.	Provides offline access.
5	Slower, as every command need to be communicated with the server.	Faster, as mostly the user deals with local copy without hitting server every time.
6	If server is down, developers cannot work.	Even if server is down, developers can work on their local copy.

Stages of GIT:



Repository:

- Repository is the place where we have all the code or kind of folder on the server.
- It is a kind of folder that is related to one Project or Product.
- Changes are personal to that particular repository.

Server:

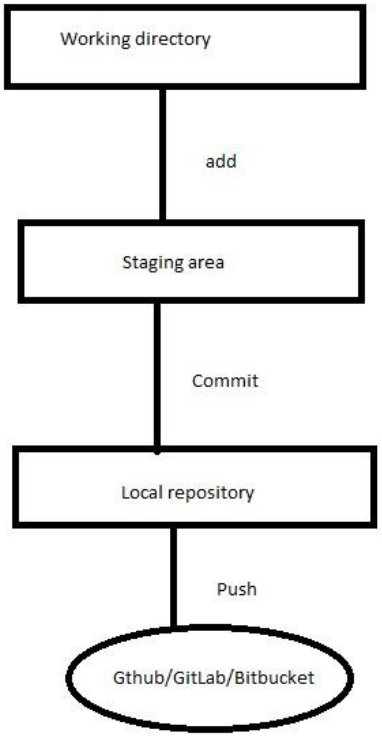
- It stores all the repositories.
- It contains metadata also.

Working Directory:

- Where you see files physically and do modifications on it.
- At a time, we can work on only a particular branch.

Note:

- Never work on old code.
- In CVCS, developers generally make modifications and commit changes to the central repository, but GIT uses a different strategy. GIT doesn't track each and every file modified, whenever you do the "commit", GIT will look for the files present in the staging area and further considered for the commit into local repository and then maintains a **commitID** (snapshot) for the modifications.



Commit:

- Store changes in the local repository. We will get one COMMITID for each commit command.
- COMMITID is 40 alphanumeric chars autogenerated ID.
- It uses SHA-1 checksum concept.
- If you change even one dot in the code, the checksum on the COMMITID will change. So the changes get rejected.
- It actually helps you to track the changes in the code.
- Commit is also named as SHA1# (Secure Hashing Algorithm).

Commit-ID/Version-ID/Version:

- Reference to identify each change.
- To identify who made the changes to the file.

TAGS:

- TAGS assign a meaningful name with the specific version in the repository.

Snapshot:

- Represents some data (code) of particular time.
- It is always incremental, i.e. it stores the changes (appended data but not the entire copy).

.git file (log):

- This fille is located in the master/branch root folder and it stores all the log for the changes with commit-IDs in the repository.
- It also has the users IDs who made the changes.
- It is a hidden file.

Push:

- Push operation copies changes from local repository to the remote/Central repository.
- This is used to store changes permanently into the GIT repository.

PULL:

- Pull operation copies the changes from remote repository to the local machine.
- The pull operation is used for synchronization between two repositories.

Branch:

- Each Task/Feature has one separate branch.
- Finally merge all the branches to the master branch.
- Useful when we want to work parallelly.
- Can create one branch basis of another branch.
- Default branch is **"master branch"**.
- File created in workspace will not be visible in any of the branch workspace until you commit. Once you 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 **"Secured Hash Algorithm – SHA1"**.
- No need of a powerful hardware to maintain GIT.
- Easier branching, If we create a new branch it will copy all the code to the new branch.

Sprint-Branch(multiple)

||

Develop Branch

||

QA Branch

||

UAT Branch

||

Master/Main Branch

26-10-2021 08:05 PM

How to use GIT and Create GIT Hub/Lab/Bitbucket Account?

1. Create an EC2 Instance and login to the Instance.
2. Run command “SUDO SU”
3. Run command “yum update -y”
4. Run command “yum install git -y”
5. After installation “git --version”
6. Configure git account with commands “git config --global username <user-name>”; “git config --global user.email <email>”
7. Create a directory for our working directory and get into it
8. Run command “git init”
9. Add some files to our project
10. Run command “git add .”
11. Run command “git commit”
12. Run command “git status”
13. Run command “git log” – to verify the number of commits with commitID
14. Run command “git show <commit-ID>” – to view the details of the commitID
15. Run command “git push -u origin <master/main>” – to push/update the code to main repository.

GIT Ignore:

To ignore some files while committing the code to the central repository, create one hidden file named “.gitignore” by running command “vi .gitignore” and add some extensions which we don’t want to commit to central repository. We have to commit “.gitignore” file to the central repository to ignore extension files added in the .gitignore file.

Ex:

vi .gitignore

*.jpeg

*.ideaj

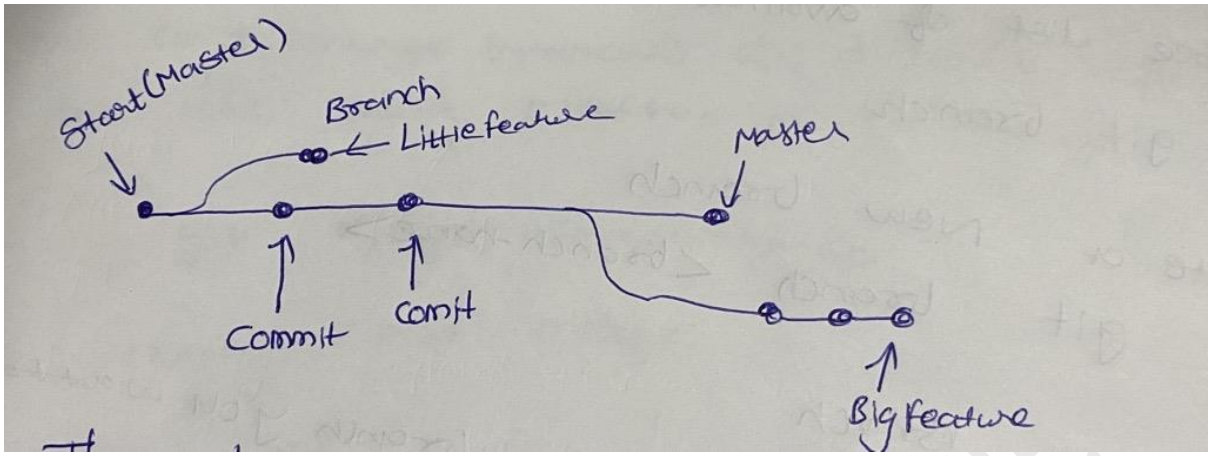
git add .gitignore

git commit -m “committing git ignore file”

git status

git push

How to create a branch, merge and stash repositories?



The diagram above visualises a repository with two isolated lines for development, one for a "little feature" and another "big feature" as a long running feature. Its not only possible to work on both in parallel, but it also keeps the main master branch free from errors.

- Each task has one separate feature branch
- After making the code changes, merge other branches with the master branch once the code is verified.
- This concept is useful for parallel development
- We can create any number of branches within
- Changes are personal to that particular branch, until we merge it to the parent branch
- Default branch is called Main/Master branch.
- Files created in workspace will be visible in common workspace until you commit. Once we commit then that file belongs to that particular branch.
- When we create a new branch from an existing branch, data/code of existing branch will be copied to the new branch.

Git commands with uses:

"git branch" - To see a list of available branches

"git branch <branch-name>" – create a new empty branch

"git checkout <branch-name>" – to switch between branches

"git log --online" – to see all branches in the remote/global/central repository

"git branch -d <branch-name>" – to delete the branch

27-10-2021 08:11 PM

MERGE:

- We can merge branches of different repositories.
- Merging can be done either by automatic or manually.

- Automatic merge can happen only if both branches don't have any conflicts. We can do this by using *PULL* (fetch + checkout) mechanism to merge branches.
- Command to merge branches manually:
    - "git merge <**source-branch**-name>"
    - Before running the above command, we must checkout the code to the **target branch** to which we are merging.
- To verify whether the merge is performed properly or not, run command "git log".
- But the merge happens only on the local repository for the above command.
- To make that merge on the remote repository, run the command "git push origin <branch-name>".

GIT Conflict:

- When same file having different content in different branches or same branch, if we do merge then conflict occurs (resolve conflicts first then add & commit).
- Conflicts occurs even when you merge branches.

GIT Stashing:

- Suppose we are implementing a new feature for your product, your code is in progress and suddenly a customer escalation or a requirement comes at priority. Because of this, we have to keep aside our new feature work for few hours. You cannot commit your partial code and also cannot throwaway your changes. So, we need some temporary storage, where we can store your partial changes and later commit it to the local repository.
- We can stash only modified items/files, we cannot stash new files.
- Command to stash an item:
    - "git stash"
- Command to list items in a stash:
    - "git stash list"
- Command to apply back the stash items to current repository/branch:
    - "git stash apply stash@{0} stash@{1} … so on"
- Once you apply the stash items we need to add and commit the code.
- Command to delete or clear the stash items:
    - "git stash clear" or "git stash clear <items1>"

GIT reset:

- GIT reset is a powerful command that is used to undo local commit changes to the state of GIT local repository.
- Commands to reset staging area:
    - "git reset <file-names>"
    - "git reset ."
- To reset the changes from staging and working directory at a time, the command is:
    - "git reset --hard"

GIT Revert:

- The revert command helps you undo an existing commit in the local repository (before push).
- In this process, it doesn't delete any data instead git creates a new commit id with the included files reverted to the previous state. So version control history moves forward when the state of your files backward.
- Command to revert particular commit ID is: "git revert <commit-id>"
- Reset works before commit, where as revert works even after commit.

Removing untracked files:

- Command to remove untracked files by dry run: "git clean -n"
- Command to remove untracked files forcefully & permanently: "git clean -f"

Tags:

- Tags allows you to give a meaningful name for a specific version or changes in the repository.
- Command to apply/attach a tag to a particular commit id:
  - "git tag -a <tag-name> -m <message> <commit-id>"
- Command to list tags:
  - "git tag"
- Command to see/view contents of particular commit id or tag:
  - "git show <tag-name/commit-id>"
- Command to delete a tag:
  - "git tag -d <tag-name>"

Configuration Management Tool – CHEF

Configuration Management Tool:

- Push based tools
- Pull based tools

Push based tools:

- Pushes configuration by the server (tool server)Pushes configuration to the nodes (servers)
- Example of push-based tools are: ANSIBLE & SALT stack.

Pull based tools:

- Pull confirm nodes to check with the server (tool server).
- Nodes periodically fetches the configuration from the tool server.
- Examples of pull based tools are CHEF & PUPPET

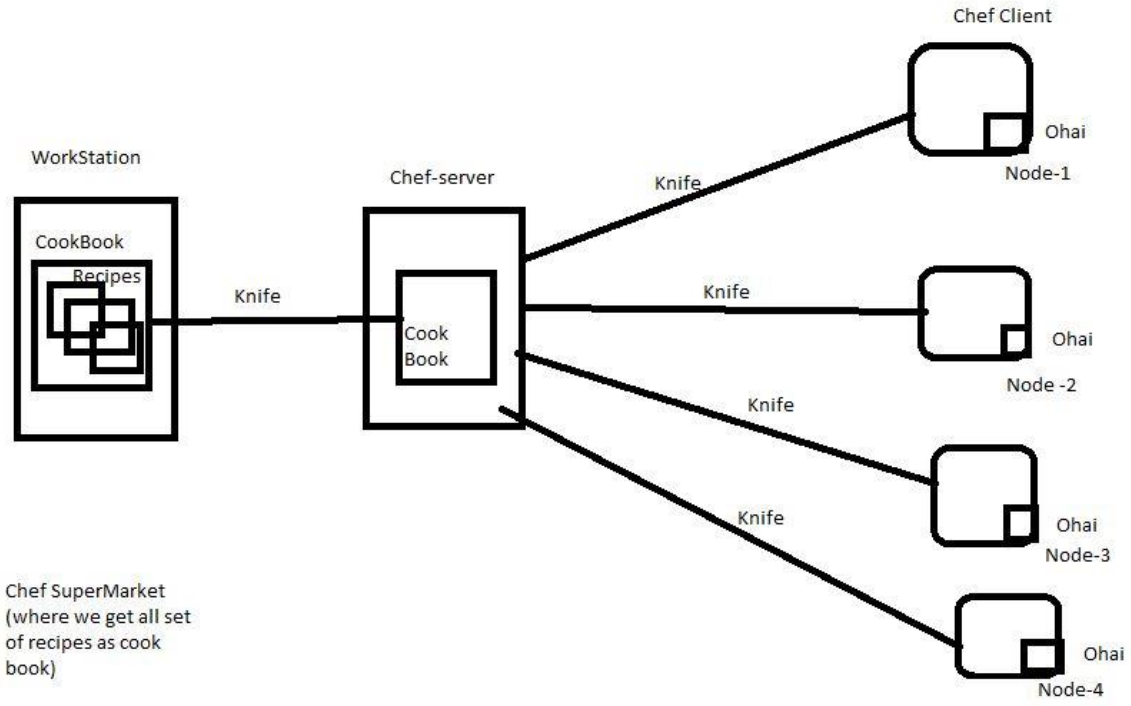
CHEF:

- Chef is a company and the name of the configuration management tool written in Ruby and Erlang.
- Founded by Adam Jacobs in the year 2009
- Actual name was "**Marinate**" later renamed to "**CHEF**"
- On April 2, 2019 the company announced that all their products are open source under Apache 2.0 license.
- **CHEF** is used by Facebook, AWS OpsWorks, HP Public cloud, etc.
- **Chef** is an administration tool, whatever system administrator used to do manually now we are automating all those tasks by CHEF.
- Configuration Management – it is a method through which we automate admin tasks.
- Configuration Management tool turns your code into infrastructure.
- So that your code will be repeatable, testable and versional.

Advantages of Configuration Management Tool:

- Complete automation

- Increase uptime
- Improve performance
- Ensure compliance
- Prevent Errors
- Reduce Cost



28-10-2021 09:22 PM

<u>Components of CHEF</u>:

- **Workstation**: Workstations are personal computers or Virtual servers where all configuration code is created, tested or changed.
- DevOps Engineers actually sit at workstations and write code. This code is called "**recipe**". A collection of recipes are called "**Cookbook**".
- Workstation communicate with the **CHEF server** using **KNIFE.**
- **KNIFE** is a command line tool that upload the cookbook to the server.

CHEF Server:

- The CHEF Server is a middleman between workstation and the Nodes.
- All cookbooks are stored in CHEF Server.
- The CHEF Servers may be hosted locally or remotely.

Node:

- Nodes are the systems that require configuration.
- **Ohai** fetches the current state of the Node of its location.
- Node communicate with the **CHEF Server** using the **CHEF Client**.
- Each Node can have a different configuration requirement.
- **CHEF Client** is installed on every Node (Client).

Key Words:

- **CHEF Workstation:** Where we write our code
- **CHEF Server:** Where we upload our code
- **CHEF Node:** Where we apply our code
- **Knife:** Tool to establish communication among workstation, server and Node. Knife is a command Line tool that runs on workstation.
- **CHEF Client:** Tool runs on every CHEF Node to pull code from the CHEF Server.
    - Chef Client will gather current system configuration, downloads the desired system configuration from the CHEF Server.
    - Configures the Node such that it adhere to the policy.
- **Ohai**: Maintains current state information of CHEF Code.
- **Idempotency**: Tracking the state of system resources to ensure that the changes should not reapply repeatedly.
- **CHEF SuperMarket**: Where we get custom code.

How to create a cookbook and recipe?

- Create AWS Linux Instance.
- Login to EC2 Instance as "ec2-user".
- Access Root permissions using command "*sudo su*".
- Acquire download URL of ChefWorkstation from www.chef.io
- Fill the form and download the chef software into Linux Machine.
- On Linux Machine using command "wget <url>" download the chef workstation package as **rpm**.
- Run command "*yum install <downloaded chef file> -y*"
- Run command "*which chef*" to verify where chef is installed.
- Run command "*chef --version*" to know the version of the chef software installed.
- **Cookbook**: cookbook is a collection of recipes and some other files and folders.
- Inside cookbook we can see the following:
    - Chefignore – just like gitignore
    - kitchen.yml – for testing cookbook
    - metadata.rb – name, version, author, etc of the cookbook
    - Readme.md – information about usage cookbook to be filled by users.
    - Recipes folder – where you store all your recipes, where each recipe hold the code of that recipe.
    - Spec – for unit testing
    - Test – for integration testing
- Steps for creating cookbook:

- mkdir cookbooks
- cd cookbooks
- chef generate cookbook <cookbook-name>
- yum install tree -y
- cd <cookbook-name>
- chef generate recipe <recipe-name>
- cd ..
- vi <cookbook-name>/recipes/<recipe-name>.rb
- write below **<sample-code>** and save the file

<sample-code>:

```
file '/home/ec2-user/<file-path>/<file.sh>' do
content "add content here"
action :create
end
```

- *run command "chef exec ruby -c <cookbook-name>/recipes/<recipe-name>.rb" to verify the syntax*
- *run command 'chef-client -zr "recipe[<cookbook-name>::<recipe-name>]" ' to execute the recipe*

<sample-code2>

<sample-code2>:

```
package 'git' do
action :install
end
```

<sample-code3>:

```
package "httpd" do
action :install
end
file '/var/www/html/index.html' do
content 'welcome to Static Web Site'
action :create
end
service 'httpd' do
action [:enable,:start]
```

end

Resource:

- It is a basic component of the recipe used to manage the infrastructure with different kind of states.
- There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure.
- For example:
    - **package**: manages the package on the Node.
    - **service**: manages the service on a Node.
    - **user**: Manages the users on the Node.
    - **group:** manages group
    - **template**: manages the files with embedded Ruby template
    - **cookbook-file**: Transfers the files from the files sub-directory in the cookbook to a location on the node.
    - **file**: manages the content of a file on Node.
    - **execute**: execute a command on the Node.
    - **cron**: edits an existing cron file on the Node.
    - **directory**: manages the directory on the Node.

04-11-2021 03:30 PM

CHEF Attributes:

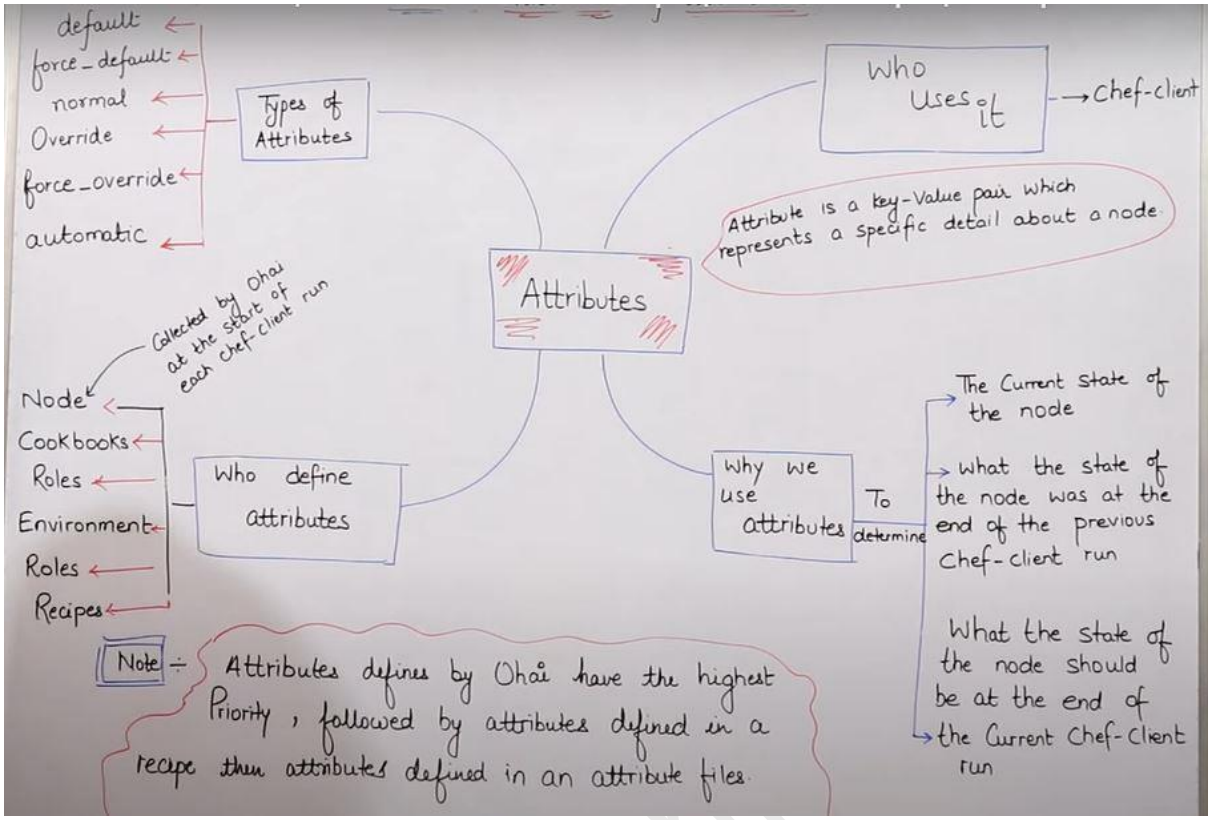
What is an Attribute? Attribute is a Key-Value pair which represents specific detail about a Node.

Types of Attributes:

1. Default
2. Force-Default
3. Normal
4. Override
5. Force Override
6. Automate

Note: Default has least priority whereas automate has the highest priority.

- CHEF Client uses the Attributes.
- Node, recipes/cookbooks, roles, environment can define attributes.
- We use attributes to determine the current state of Node, to know the state of the Node of previous CHEF Client run and can know the state of Node for next CHEF Client run.



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

Commands used in CHEF:

- Here is the list of commands:
  - #Ohai – it returns all attributes details stored in Ohai
  - #Ohai ipaddress – returns ip address of the Node
  - #Ohai memory/total – returns internal memory details of a Node
  - #Ohai CPU/0/mhz – returns the CPU process details of a Node

Sample Program:

cd <cookbook-name>

<cookbook-name>#chef generate recipe GettingAttributes

<cookbook-name>#cd ..

//for example recipe name is "GettingAttributes"

cookbooks# vi <cookbook-name>/recipes/GettingAttributes.rb

file '/home/BasicInfo' do

content "this is to get attributes

HOSTNAME : #{node['hostname']}

IPADDRESS : #{node['ipaddress']}

CPU : #{node['cpu']['0']['mhz']}

MEMORY : #{node['memory']['total']}

"

owner 'root'

group 'root'

action :create

end

run command "chef exec ruby -c <cookbook-name>/recipes/ GettingAttributes.rb" to verify the syntax

run command 'chef-client -zr "recipe[<cookbook-name>:: GettingAttributes]" ' to execute the recipe

---

08-11-2021 08:30 PM

How to execute Linux Commands from a recipe?

Here is a sample code for running the Linux commands through recipe:

#vi <cookbook-name>/recipes/<recipe-name>.rb

execute "run a script" do

command << -EOH

mkdir /home/ec2-user/veera

touch /home/ec2-user/veera/shekar.txt

EOH

end

How to run this recipe:

"chef exec ruby -c <cookbook-name>/recipes/ <recipe-name>.rb" to verify the syntax

chef-client -zr "recipe[<cookbook-name>:: <recipe-name>]"

Sample code to create a User using a recipe:

user "veera" do

action :create

end

Sample code to create a new group and add user to the new group:

group "<group-name>" do

action :create

members <user-name>

append true

end

Note: We run CHEF-Client to apply recipe to bring Node into desired state. This process is known as "***Convergence***".

RUNLIST

What is RUNLIST?

1. To run the recipes in the sequence order that we mention is known as "**RUNLIST"**.
2. With this process we can run multiple recipes, but the condition is there must be only one recipe from one cookbook.

   Command to run multiple recipes from DIFFERENT cookbooks:

   #chef-client -zr "recipe[test-cookbook::test-recipe],recipe[apache-cookbook::apache-recipe]"

How to run multiple recipes from a single or multiple cookbook?

- For this we can use "include_recipe"
- To call recipes from other recipes within the same cookbook we use "include_recipe" key word, this will help to run multiple recipes from the same cookbook.
- For every cookbook by default, we will have a "default.rb" recipe
- We can run any number of recipes with this command, but all must be from same cookbook.
- Sample:

  #vi test-cookbook/recipes/default.rb
  include_recipe "test-cookbook::test-recipe1"
  include_recipe "test-cookbook::test-recipe2"

Command to run multiple recipes from *SAME* cookbooks:

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

Note: Now we can combine previous two concepts, so that we can run multiple recipes from multiple cookbooks in a sequence order.

#chef-client -zr "recipe[test-cookbook::default],recipe[apache-cookbook::default]"

OR

#chef-client -zr "recipe[test-cookbook],recipe[apache-cookbook]"

BOOTSTRAPing

CHEF server is going to be a mediator between the **Node** and cookbooks(in workstation).

- Firstly, create an account in CHEF Server.
- Attach workstation to the CHEF Server.
- Now upload cookbooks from Workstation to CHEF Server.
- Attach Nodes to the CHEF Server via BOOTSTRAP process.
- Apply recipes of cookbooks from CHEF Server to Nodes.

ec2-user]# ls

ec2-user]# cd cookbooks

cookbooks]# ls

09-11-2021 11:10 PM

How to Create a CHEF Server:

- Open a web page "Manage.chef.io"
- Register a chef account and create an organization, then it allows you to download starter kit as a zip file.
- Unzip the downloaded content, it appears as a folder "chef-repo".
- Download "winscp" and login with ec2 credentials, then copy the "chef-repo" folder to the Linux Machine which is a "**chef-workstation**".
- Commands to create a **chef-server**.
    - Open **chef-workstation** and run the below commands:
        - $sudo su
        - #cd chef-repo
        - #ls -a → to View all list of files including Hidden files
          .chef cookbooks roles
        - # cd .chef
        - # ls -a
          Config.rb <server.pem>
        - To view whether workstation is connected to server or not run below commands
          #cat config.rb
          #cd ..
          #knife ssl check

Bootstrapping a Node:

Attaching a Node to Chef-server is called "**Bootstrapping**" (both workstation and Node should be in same availability zone)

- Now onwards we have to run all commands inside "**chef-repo**" directory.
- There are two actions that will be performed during bootstrapping
  1. Adding Node to Chef-Server
  2. Installing Chef Package

Node Creation:

Create a EC2 Linux instance (Node1), launch this instance in the same availability zone as in the Chef-Workstation. We need to add some bootstrap script while creation of instance in advance details.

Script:

```
#!/bin/bash

sudo su

yum update -y
```

Now go to Chef-Workstation, then add Node to the Chef-workstation & Chef-server

[chef-repo]#knife bootstrap <private-ip-address-of-node> --ssh-user ec2-user --sudo -i <private-key-of-Node> -N Node1

Note: Before running the above command we have to place the key of the Node in chef-repo directory

To verify whether the Node is added to the Chef-Server or not, run below command:

#knife node list

Uploading cookbooks to the Chef-Server:

Now we have to upload already created cookbooks into Chef-Server, to do that we have below command:

[chef-repo]*#knife cookbook upload <cookbook-name>*

Now to check if the cookbook is uploaded or not run the below command:

[chef-repo]#knife cookbook list

Creating a RUNLIST for a Node:

We will attach the recipe required for that specific Node.

Command: *[chef-repo]#knife node run_list set Node1 "recipe[test-cookbook::recipe1], recipe[apache-cookbook::apache-recipe]"*

To verify whether the runlist added to the Node or not:

Command: *[chef-repo]#knife node show Node1*

Note: As of now we are manually updating the Node, further we will try automation

Now access the Node1 with the help of putty/MobaXterm.

#sudo su

#chef-client

Now try modifying existing recipe:

[chef-repo]#vi cookbooks/apache-cookbook/recipes/apache-recipe.rb

Change some content of this file, save and reupload to the Chef-Server

[chef-repo]#knife cookbook upload apache-cookbook

Note: but in this case we have to rerun *chef-client* command on the Node

Automating the entire process of running recipe *on the Node*:

We don't want to call *chef-client* everytime manually. So, we have to automate this process.

Now connect to the Node instance:

[ec2-user]#vi /etc/crontab

** * * * * root chef-client*

-- this command is to run *chef-client* command every minute

First * is minutes (0-59)

Second * is hours (0-23)

Third * is day (1-31)

Fourth * is month (1-12)

Fifth * is week day(1-7)

Now go to the chef-workstation and update the recipe and reupload the cookbook to the chef-server using below command:

[chef-repo]#knife cookbook upload apache-cookbook

After this Node will automatically run the *chef-client* command every minute to apply the recipe to the Node.

Create a Bootstrap script so that it creates a crontab on Instance/Node creation.

Advanced details:

```
#!/bin/bash

sudo su

yum update -y

echo "* * * * * root chef-client" >> /etc/crontab
```

From now if we update any **cookbook** and reupload to **chef-server**, **Node** automatically picks the changes in the **recipes** which are in its **RUNLIST**.

10-11-2021 12:20 PM

Commands to delete and clean CHEF-Server:

- To see list of cookbooks which are present in CHEF-Server
  #knife cookbook list
- To delete cookbook from CHEF-Server
  #knife cookbook delete <cookbook-name> -y
- To see the list of Nodes which are present in the CHEF-Server
  #knife node list
- To delete nodes from CHEF-Server
  #knife node delete <node-name> -y
- To see list of clients which are present in CHEF-Server
  #knife client-list
- To delete clients from CHEF-Server
  #knife client delete <client-name> -y
- To see list of Roles which are present in CHEF-Server
  #knife role list
- To delete roles from CHEF-Server
  #knife role delete <role-name> -y

Roles

- Role is a concatenated list of RUNLISTs of cookbooks/recipes.
- "roles" folder is located in CHEF-REPO folder.

- By using ROLE we don't need to run RUNLIST for every change/create of the cookbooks/recipes for individual Nodes.
- Bydefault will have one role created by CHEF is "***starter.rb***"

Commands:

[chef-repo]#cd roles
[roles]#vi devops.rb

Name "devops"
Description "web server role"
run_list "recipe[apache-cookbook::apache-recipe],recipe[test-cookbook::test-recipe]"

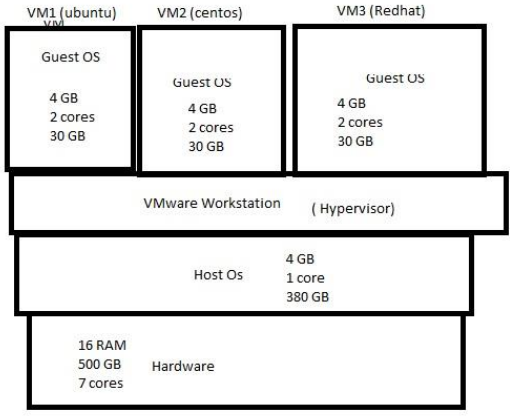
[roles]#cd ..

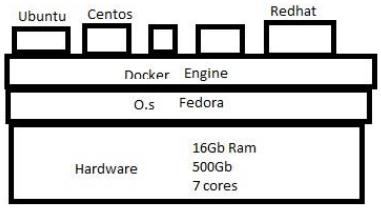
- Now upload the role to the CHEF-Server using below command
  [chef-repo]#knife role from file roles/devops.rb
- To list roles on the server use below command
  [chef-repo]#knife role list
- To list the Nodes in the server use below command:
  [chef-repo]#knife node list
- To execute RUNLIST of a role use below command:
  [chef-repo]#knife node run_list set node1 "role[devops]"
  [chef-repo]#knife node run_list set node2 "role[devops]"
- If we make any change/create a cookbook/recipe, then make those changes in the roles and run below command again to upload/update the role:
  [chef-repo]#knife role from file roles/devops.rb

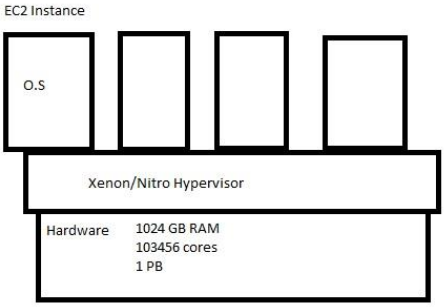
Sample Recipe:

```
%w(httpd mariadb-server unzip git vim)
    .each do |p|
        package p do
        action :install
    end
end
```

DOCKER







- Docker is an open-source centralized platform designed to create, deploy, and run applications.
- Docker uses "**container**" on the host OS 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 Operating System but Docker engine runs natively on Linux distribution.
- Docker written in **'GO'** language.
- Docker is a tool that performs OS level virtualization, also known as '**Containerization**'.
- Before Docker many users faced the problem that the particular code is running in a developer system but not in the user's system.
- Docker was first released in March 2013, is developed by Soloman Hykes & Sebastean pahl.
- Docker is a set of "Platform as a Service" (PaaS) that uses Operating System virtualization whereas VMWare uses hardware level virtualization.
- Docker uses or downloads 5% of OS file remaining will be taken from host OS.

Advantages of Docker:

- No pre-allocation of resources like RAM, Processor, Storage.
- Containerization Image (CI) efficiency → Docker enables you to build a container image and use that same image across every step of the deployment process.
- Less cost

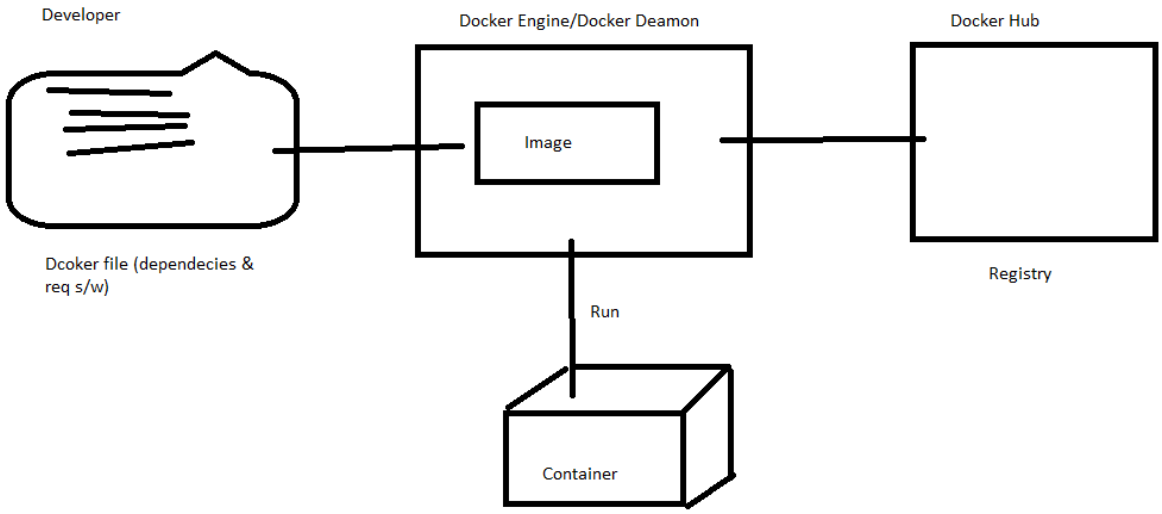
- Docker is lite in weight.
- Docker can run on physical hardware or virtual hardware or on Cloud environment.
- We can reuse the Docker Image.
- It will take very less time to create a **container**.
- Image → Container (running)

Run

Image ----> Container

Disadvantages of Docker:

- Docker is not a good solution for applications that requires rich GUI.
- It is 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 cannot run on the Linux environment or vice-versa.
- Docker is suitable when the development OS and testing OS are same, if the OS is different, we should use VMware/Virtualization.
- NO solution for data recovery and backup.



Docker Eco System:

1. Docker Client
2. Docker Daemon/Engine
3. Docker Hub
4. Docker Images
5. Docker Compose

11-11-2021 18:18 PM

Docker Daemon/Engine:

- Docker Daemon runs on the host OS.
- It is responsible for running containers to manage docker services.
- Docker Daemon can communicate with other Docker services (like Docker Client, Docker Hub, etc).

Docker Client:

- Docker users can interact with docker Daemon through a Docker Client (CLI).
- Docker Client uses commands and Rest-API to communicate with the Docker Daemon.
- When a client runs any server command on a Docker Client terminal, the client terminal sends 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 an environment to execute and run applications.
- It contains the docker daemons, images, containers, and storages.

Docker Hub/Registry:

- Docker Registry manages and stores the docker images.
- There are two types of registries in the docker.
    1. Public Registry: Public Registry is also called as "***Docker Hub***"
    2. Private Registry: It is a local repository to enterprise where we can share images within the enterprise.

Docker Images:

- Docker Images are the read only binary templates used to create docker containers
  (OR)
- Single file with all dependencies and configuration required to run a program as container.
- *Ways to create an Image*:
    1. Take image from docker hub.
    2. Create image from docker file.
    3. Create image from existing docker container.

Docker Container:

- Container hold the entire packages that is needed to run the application
  (OR)

- In other words we can say that the image is a template and the container is the copy of that template.
- Container is like a virtual machine (but not Virtual Machine).
- Image becomes container when they run on Docker Engine.

Commands helpful for running Docker (must run on Docker Client):

- To see all the Images present in your local machine use below command:
  ]#docker images
- To find out images in docker hub(public registry).
  ]#docker search Jenkins/<search-parameter>
- Run below command to download image from docker hub to local machine:
  ]#docker pull <image-name>
- Command to pull image, Start container, attach container and give name to the container:
  ]#docker run -it --name <container-name> <Image-name> /bin/bash
- Run below command to check Docker service is started or not:
  ]#service docker status
- Run below command to start a container:
  ]#docker start veera
- Command to get into the container:
  ]#docker attach veera
- Command to see all the containers:
  ]#docker ps -a
- Command to see only running containers:
  ]#docker ps
- Command to stop a container:
  ]#docker stop veera
- Command to delete a container:
  ]#docker rm veera

14-11-2021 11:00 AM

Docker diff commands:

Task: We have to create container from our own image

- First we have to create a container, so run below command:
  ]#docker run -it --name <container-name> ubuntu /bin/bash
- Once we log into the container make necessary setup required for environment and server setup. But for test purpose now we are making minor changes:
  ]#cd tmp/
  ]#touch Veera Vamsi Hareesh Vinay Sateesh
- To identify the changes in the container with the base image run below command:
  ]#docker diff <container-name>
- Command to make a new image out of a container: Container must be up and running

]#docker commit <container-name> <target-image-name>

- Command to create a new container using the custom image we created:
  ]#docker run -it --name <container-name> <target/custom-image> /bin/bash

Docker File:

- Docker file is basically a text file which contains some set of instructions.
- Automation of docker image creation:
- **Docker Components**:
  - **FROM**: for downloading the base image, this must be added as first command in the docker file.
  - **RUN**: to execute commands, it will create a layer in the image.
  - **MAINTAINER**: author/owner/description of that file.
  - **COPY**: Copy files from ***local*** system, we need to provide source and destination for this command. (this command cannot download file from internet/remote machine)
  - **ADD**: similar to copy command but it provides a feature to download files from internet also. This extracts a file at docker site.
  - **EXPOSE**: to expose ports such as port 8080 for TOMCAT, port 80 for NGINX, etc.
  - **WORKDIR:** to set working directory for a container.
  - **CMD:** execute commands but during container creation.
  - **ENTRYPOINT:** Similar to CMD but have higher priority over CMD. First commands will be executed by ENTRYPOINT only.
  - **ENV:** to set Environment variables.
- **How to create a Docker File**:
  - Create a file named "Dockerfile".
  - Add instructions in "Dockerfile".
  - Build the "Dockerfile" to create an image.
  - Run image to create container.
  - Sample Dockerfile:

    FROM ubuntu

    RUN echo "Technical expert" > /tmp/testfile
- How to build Dockerfile as image: We must be in the dockerfile location

  #docker build -t <image-name> .
- To verify the image created run below command:
  #docker images
- Command to start the container from the created image:
  #docker run -it --name <container-name> <image-name> /bin/bash

Docker Volume:

- Volume is simply a directory inside a container.
- Firstly, we have declare this directory as volume and then share volume.
- Even if you stop container still, we can access volume.
- Volume will be created in one container.

- You can declare a directory as a volume only while creating a container.
- You cannot create volume from existing container.
- Volume will not be included when you update an image.
- You can map volume in two ways:
    - Container to container
    - Host to Container

Benefits of Volume:

- Decoupling container from storage.
- Share volume among different containers.
- Attach volume to containers.
- On deleting container, volume doesn't get deleted.

Creating Volume from Docker File:

- Create a file with name "Dockerfile" and write below code in it:
    - *FROM ubuntu*
    - *VOLUME ["/mydrive1"]*
- Then create image from this Dockerfile
    - *docker build -t MyNewImage .*
- Now create a container from the image we created
    - *docker run -it --name Container1 MyNewImage /bin/bash*
- Now if we login to the container, we can see mydirve1 volume in it.
- Share volume between containers:
    - *docker run -it –name container2 --privileged=true --volumes-from container1 ubuntu /bin/bash*
- Now after creating container2 the mydrive1 is visible. Whatever the changes you made in mydrive1 volume, you can see in other container volume as they both are sharing the mydrive1 volume.

22-11-2021 20:20

Creating a volume while creating container using Commands:

#docker run -it --name <container-name> -v /mydrive1 ubuntu /bin/bash

- Volume can be created using only while creating a container.

Now, if we want to share this volume from between containers

- Share volume between containers:
    - *docker run -it --name container2 --privileged=true --volumes-from Container1 ubuntu /bin/bash*
- Now check container1's shared volume can be accessed from container2.

Sharing volume between Host OS and Containers:

- Let's share /home/ec2-user directory of host OS as a volume to containers:
- Command to do the above task:

  #docker run -it --name hostCont -v /home/ec2-user:/myVolume1 --privileged=true ubuntu /bin/bash
- Now once you are in the container do
  - cd /myVolume1
  - ls
- Now we can see all the files of host system which are located in /home/ec2-user.

Some useful commands of Docker Container for Volumes:

- *docker volume ls* → list of docker volumes
- *docker volume create <volume-name>* → to create a docker volume
- *docker volume rm <volume-name>* → to remove/delete a docker volume
- *docker volume prune* → it will remove all the unused docker volumes
- *docker volume inspect <volume-name>*→ provides all the details of the volume
- *docker container inspect <container-name>* → provides all the details of the container

Docker port expose:

- For deploying a website in its webserver inside a container, we need to expose ports with host. Let's see how to make it in the below process:
- Commands:

  $sudo su
  #yum install docker -y
  #service docker start
  #docker run -td --name webserver -p 80:80 ubuntu
  #docker ps
  #docker port webserver
  #docker exec -it webserver /bin/bash
  Now run the commands inside container:
  #apt-get update
  #apt-get install apache2 -y
  #cd /var/www/html
  #echo "Learn technical things">index.html
  #service apache2 start

Example for Jenkins container launch:

#docker run -td --name myJenkins -p 8080:8080 jenkins

Difference between *docker attach* and *docker exec*:

Docker exec *creates a new process* in the container environment while *docker attach just connect the* standard input/output of the main process inside the container to corresponding standard input/output error of current terminal.

Difference between expose and publish:

Basically, we have three options:

1. Neither specify expose nor -p (publish)
2. Only specify expose
3. Specify expose and -p

Note:

1. If you specify neither expose not -p, the service in the container only be accessible from inside the container itself.
2. Whereas if you expose a port, the service in the container is not accessible from outside the docker but accessible from inside other docker containers, so this is good for inter-container communication. We need to opt publish to share with all the containers (outside world).

Process to push docker image to docker hub:

$sudo su
#yum update -y
#yum install docker -y
#service docker start
#docker run -it --name MyFirstContainer ubuntu /bin/bash
#docker commit MyFirstContainer MyFirstContainerImage

Now create an account in hub.docker.com
On the source machine

#docker login

Provide username and password

#docker tag <source-image> <docker-hub-id>/<target-image-name>
#docker push <docker-hub-id>/MyFirstContainerImage

Now we can access this image in docker hub account.
Now create another ec2 instance in the required region where we want this image to be run as container.

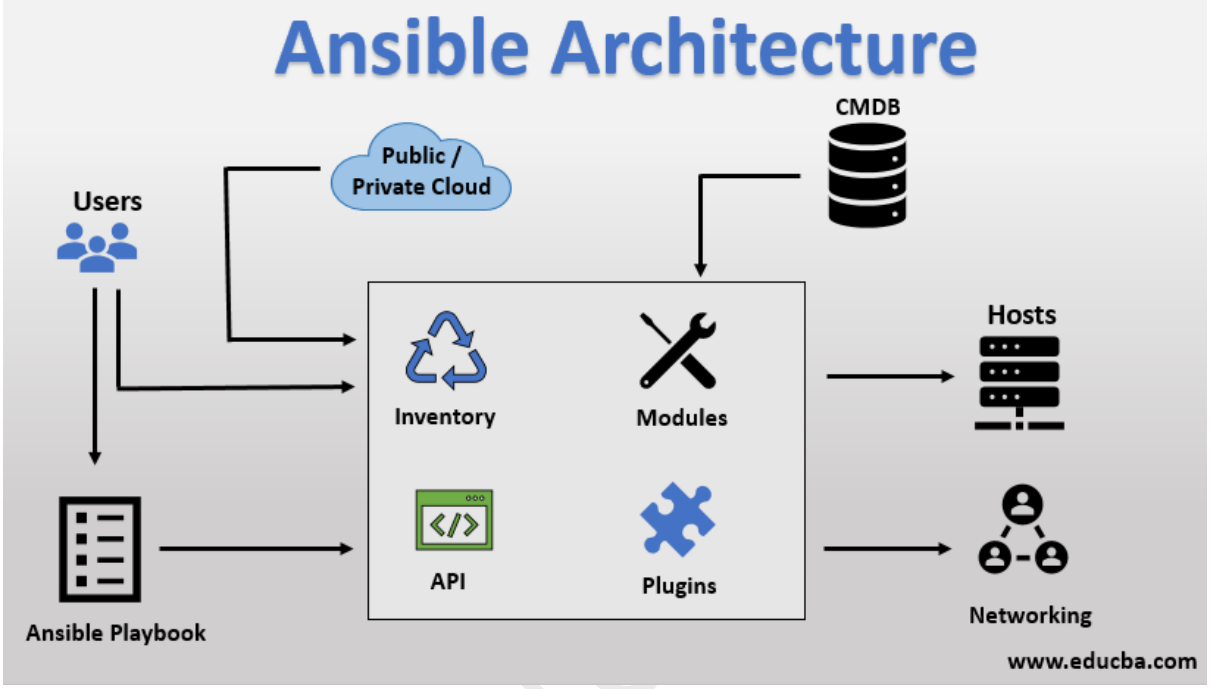
#docker pull <docker-hub-id>/MyFirstContainerImage
#docker run -it --name NewContainer <docker-hub-id>/MyFirstContainerImage /bin/bash

Important commands:

- Stop all running containers
  #docker stop $(docker ps -q)
- Delete all stopped containers:
  #docker rm $(docker ps -a -q)
- Delete all images from the host:
  #docker rmi -f $(docker images -q)

ANSIBLE

- Ansible is an open-source IT configuration management, deployment and orchestration tool. It aims to minimize productivity gaps to a wide variety of automation challenges.
- Ansible is a Push based configuration management tool.



Ansible History:

- Micheal Dehaan developed Ansible and the ansible project begin in Feb 2012.
- RedHat acquired the ansible tool in 2015.
- Ansible is available only for RHEL (RedHat Enterprise Linux), DEBIAN, CentOS (Community Enterprise Operating System) & Oracle Linux.
- We can use this tool either our servers are on-premise or in-cloud.
- Ansible turns your code into infrastructure i.e. our computing environment has some of the same attributes as our application.
- YAML (Yet Another Markup Language) is the language used in Ansible.
- SSH configuration is mandatory for Ansible communication.

Advantages of Ansible:

- Ansible is free to use.
- It is very consistent and lite weight.
- It has no constraints regarding the OS or underlying hardware (within supported OS).
- Ansible is very secure due to its agentless capabilities and open SSH security features.

- Ansible doesn't need any special system administrator skills to install & use it (YAML knowledge is enough).
- Ansible is push based tool (which means one point of control i.e. server).

Disadvantages of Ansible:

- Insufficient user interface. Though ANSIBLE TOWER has GUI, but it is still in development stage.
- We cannot achieve full automation using Ansible.
- Its new to the market therefore, limited support and documentation is available.

Terms used in Ansible:

- ***Ansible server***: The machine where Ansible is installed and from which all the tasks and playbooks are ran.
- ***Module***: Basically, module is a command or a set of similar commands meant to be executed on the client-side.
- ***Task***: A task is a section that consists of a single procedure to be completed.
- ***Role***: A way of organizing tasks and related files to be later called in a playbook.
- ***Fact***: Information fetched from the client system from the global variables with the gather facts operation.
- ***Inventory***: File containing data about the Ansible client hosts.
- ***Play***: Execution of a playbook.
- ***Handler***: Task which is called only if notifier is present.
- ***Notifier***: A Section attributed to a task which calls the handler if the output is changed or an event occurred.
- ***Playbooks:*** It consists code in YAML format which describes task to be executed.
- ***Hosts:*** Nodes, which are automated by Ansible.

Basic steps for Ansible setup:

- Create ansible server and host machines
- Create a user ansible in server and host machines.
- Update user ansible as sudo user in server and host machines using below commands:

  ```
  # visduo
          ansible ALL=(ALL)          NOPASSWD : ALL
  ```

- Install ansible using below commands:

  ```
  #sudo yum install epel-release
  #sudo amazon-linux-extras install epel
  #yum install git python python-level python-pip openssl ansible -y
  ```

- Update host inventory file by adding host private ips and group them using below commands:

  ```
  #vi /etc/ansible/hosts
          ## [webservers] – remove comments for the Group and add all hosts private
  ```

  IP addresses under group

- Enable Inventory and sudo user in the configuration file:
  # vi /etc/ansible/ansible.cfg
    #inventory = /etc/ansible/hosts – remove comments for the inventory
    #sudo_user = root – set root as sudo user
- Generate key using below command in the server:
  #ssh-keygen
- Go to the location /home/ansible/.ssh
- Update permit and authentication on Server and all the hosts using below given commands:
  #vi /etc/ssh/sshd_config
    PermitRootLogin yes
    PasswordAuthentication yes
    #PasswordAuthentication no
- Restart the sudo services after updating on server and all the host machines using below command:
  sudo service sshd restart
- Connect to host from server using ssh with below commands:
  #ssh <host-private-ip-address> - prompts for password everytime we connect
  #ssh-copy-id ansible@<host-private-ip-address> - prompts for password only for the first time login (do this for all the hosts)

Some commands to be used on Ansible Server:

#ansible all --hosts-list → to display all the hosts attached to ansible server
#ansible <group-name> --hosts-list → to display all the hosts attached to the specific group
#ansible <group-name>[0] --hosts-list → to display first host under a group
#ansible <group-name>[1] --hosts-list → to display second host under a group
#ansible <group-name>[-1] --hosts-list → to display last host under a group
#ansible <group-name>[0:1] --hosts-list → to display first two hosts under a group

25-11-2021 10:20AM

Adhoc Commands:

- Adhoc commands are commands which can be run individually to perform quick functions. These Adhoc commands are not used for configuration management and deployment, because these commands are of one time usage.
- The ansible adhoc commands use the /usr/bin/ansible command line tool to automate a single task.

Commands:

[ansible@ip]$ansible webservers -a "ls"
[ansible@ip]$ansible webservers[0] -a "touch file1"
[ansible@ip]$ansible all -a "touch file2"
[ansible@ip]$ansible webservers -a "ls -a"
[ansible@ip]$ansible webservers -a "sudo yum install httpd -y"
OR
[ansible@ip]$ansible webservers -ba "yum install httpd -y"

[ansible@ip]$ansible webservers -ba "yum remove httpd -y"

Modules:

- Ansible shifts with a number of modules (called module library) that can be executed directly on remote host or through playbooks.
- Your library of modules can reside on any machine(any host or server), and there are no servers, daemons or databases required.
- **Where ansible modules are stored??** They are stored in the default location for the inventory file i.e. /etc/ansible/hosts.
- **Modules using Command Line**:

  [ansible@ip]$ansible webservers -b -m yum -a "pkg=httpd state=present"
  [ansible@ip]$ansible webservers -b -m yum -a "pkg=httpd state=latest"
  [ansible@ip]$ansible webservers -b -m yum -a "pkg=httpd state=absent"
  [ansible@ip]$ansible webservers -b -m service -a 'name=httpd state=started' -become -u ansible
  [ansible@ip]$ansible webservers -b -m user -a 'name=veera state=present'
  [ansible@ip]$ansible webservers -b -m copy -a 'src=file4 dest=/tmp/'

Ansible Playbook:

- Playbooks in Ansible are written in YAML format. YAML → Yet Another Markup Language.
- It is human readable data serialization language, it is commonly used for configuration files.
- Playbook is like a file where we write code consists of var, tasks, handlers, templates, and roles.
- Each playbook is composed of one or more modules in a list.
- Playbooks are divided into many sections like:
  - **Target section**: defines the host against which playbook tasks has to be executed
  - **Variable section**: defines variables
  - **Task section:** list of all modules that we need to run in an order.
- All YAML files have to begin with '---' and end with '…'
- All members of list lines must begin with same indentation level starting with '-'.
- Example:

```
---# list of fruits
fruits :
   -mango
    Apple
    Banana
```

- A dictionary is represented in a simple key value form.
- Example for list of key-value pairs:

```
---# list of customer relationship
Customer :
   -name : Vinay
     Job : Administrator
     Skills : ANSIBLE
     Experience : 0.00005 years
```

- Extension for playbook is .yml

Note: There should be a space after every colon(:)

- Example of Ansible playbook:

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

- To execute the above playbook run below commands:

  $ansible-playbook target.yml

```
---
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  gather_facts: yes
~
```

- Sample Playbook to install HTTPD on CentOS7:

```
---
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  tasks:
      - name: install HTTPD on centOs7
        yum:
           name: httpd
           state: present
           update_cache: true
```

- **Variables**:
  - Sample playbook by Passing variables:

```
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  vars:
          pkgname: httpd
  tasks:

      - name: Remove HTTPD on centOs7
        yum:
           name: "{{ pkgname }}"
           state: absent
           update_cache: true

      - name: install HTTPD on centOs7
        yum:
           name: "{{ pkgname }}"
           state: present
           update_cache: true

      - name: start HTTPD on centOS
        service:
           name: "{{ pkgname }}"
           state: started
```

Handlers:

- A handler is exactly 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 will run only if the task contains a notify directive and also indicates that it "**changed**" something.

Note: https://docs.ansible.com/ansible/latest/user_guide/playbooks_handlers.html#handler-example

- Execution of playbook with handlers:

```yaml
---
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  vars:
          pkgname: httpd
  tasks:

      - name: Remove HTTPD
        yum:
           name: "{{ pkgname }}"
           state: absent
           update_cache: true

      - name: install HTTPD
        yum:
           name: "{{ pkgname }}"
           state: present
           update_cache: true
        notify:
            - start HTTPD


  handlers:
       - name: start HTTPD
         service:
             name: "{{ pkgname }}"
             state: started
```

Dry Run:

- Check whether playbook is formatted correctly.
- Command to check the formatting is right or not of the playbook:

  $ansible-playbook handlers.yml --check

Loops:

- Some times we want to repeat a task multiple times. In computer programming this is called loops. Common ansible loops include changing ownership on several files or directories with the **'file'** module, creating multiple users with the '**user'** module, and repeating a polling step until certain result is reached.

```
---
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  tasks:
      - name: ADD A LIST OF USERS
        user:
            name: '{{item}}'
            state: present
        loop:
                - veera
                - sateesh
                - vinay
                - vamsi
                - harish
~
~
```

Note: https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

Conditional Statements:

- Whenever we have a different scenarios we put conditions according to the scenario

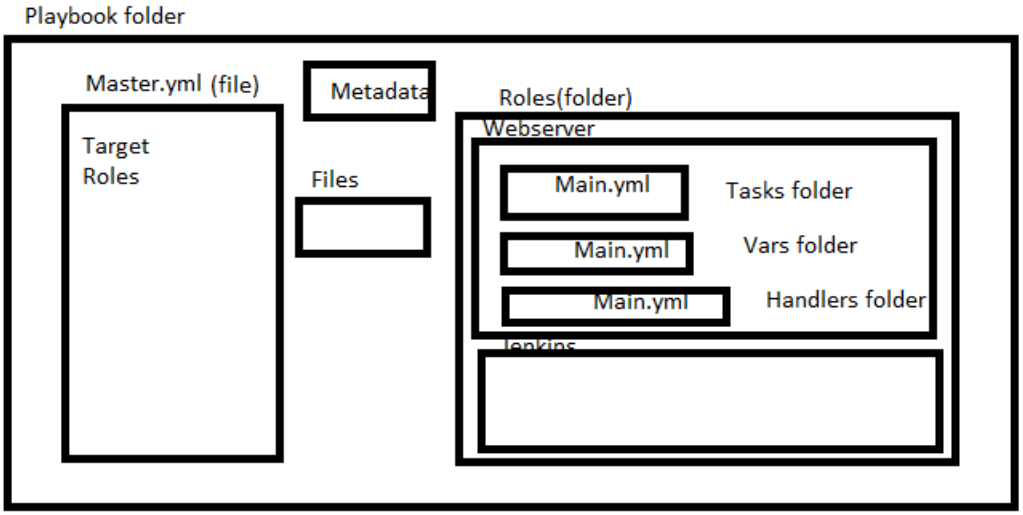
```
---
# target playbook named target.yml
- hosts: webservers
  user: ansible
  become: yes
  connection: ssh
  tasks:
      - name: UNINSTALL HTTPS
        command: sudo yum remove httpd -y
      - name: INSTALL APACHE ON DEBIAN
        command: apt-get -y install apache2
        when: ansible_facts['os_family'] == "Debian"
      - name: INSTALL APACHE ON CentOS or Redhat
        command: sudo  yum install httpd -y
        when: (ansible_facts['os_family'] == "CentOS") or (ansible_facts['os_family'] == "RedHat")
```

Link: https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html

Vault:

- Ansible allows keeping sensitive data such as passwords/keys in encrypted files rather than a plain text in playbooks.
- This vault feature present in /usr/bin/ansible-vault.
- Commands with ansible vault:
    - Creating a new encrypted playbook:
      $ansible-vault create passwords_encrypt.yml
    - Edit the encrypted playbook:
      $ansible-vault edit passwords_encrypt.yml
    - To change the password:
      $ansible-vault rekey passwords_encrypt.yml
    - To encrypt an existing playbook:
      $ansible-vault encrypt playbook1.yml
    - To decrypt an encrypted playbook:
      ansible-vault decrypt playbook1.yml

Roles:



Role directory structure:

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
    webservers/
        tasks/
        defaults/
        meta/
```

Link: https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

- We can use two techniques for reusing a set of tasks:
    - Include – we have to explore
    - Roles
- Roles are good for organizing tasks and encapsulating data needed to accomplish those tasks.
- **Ansible roles**:
    - Default
    - Files
    - Handlers

    - Meta
    - Templates
    - Tasks
    - Vars
- We can organize playbooks into a directory structure called roles.
- Adding more and more functionalities to a playbook will make it difficult to maintain in a single file, so we use roles.
- **Default**: It stores the data about role or application default variables.
    - Example: if we want to run to port 80 or 8080, then variables need to be defined in this path.
- **Files**: It contains files(static files) need to be transferred to the remote hosts.
- **Handlers**: They are triggers or task by notifiers. We can segregate all the handlers in this folder.
- **Meta:** This directory contains files that establish roles dependencies.
    - Example: Author name, supplied platform, etc.
- **Tasks**: It contains all the tasks that is normally executed through a playbook.
    - Example: installing packages, copying files, etc.
- **Vars**: Variable for the roles can be specified in this directory.

Commands:

```
$mkdir -p playbook/roles/webserver/tasks
$tree
$cd playbook
$touch roles/webserver/tasks/main.yml
$touch master.yml
$vi roles/webserver/tasks/main.yml
        -name: install webserver
         yum:
            name: httpd
            state: present
            update_cache: true
$vi master.yml
---
   -hosts: webservers
    user: ansible
    become: yes
    connection: ssh
    roles:
        -webserver
```

```yaml
---
  - hosts: webservers
    user: ansible
    become: yes
    connection: ssh
    roles:
        - webserver
~
```

```yaml
- name: install webserver
  yum:
    name: httpd
    state: present
    update_cache: true
~
```

```
├── playbook
│   ├── master.yml
│   └── roles
│       └── webserver
│           └── tasks
│               └── main.yml
```