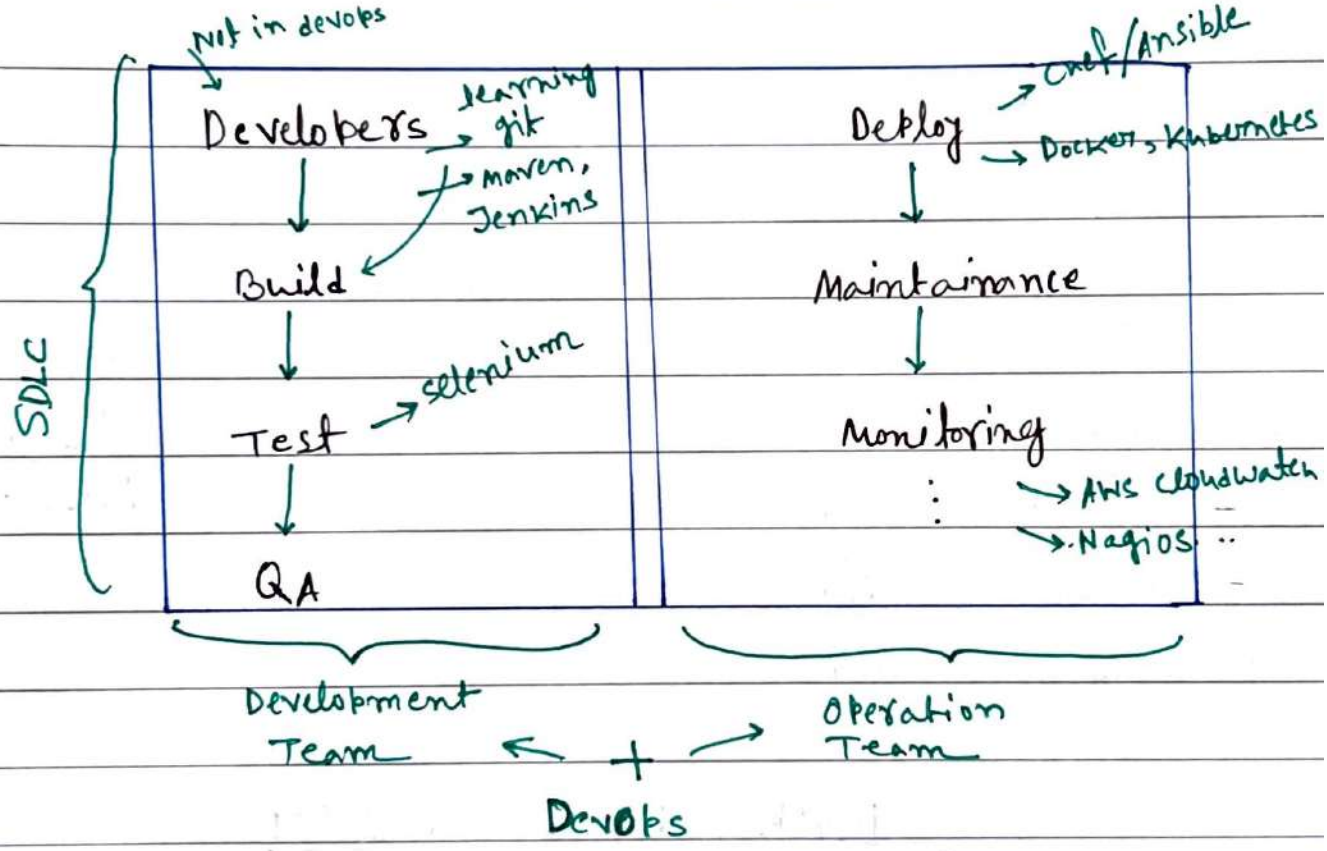
DevOps

Pre requisite for DevOps

- Where client sit that place called on-site.
- Where our team sits that called off-shore.



* Waterfall Model is also famous once (step-by-step)

What is DevOps, DevOps stages, Agile

- The term DevOps is a combination of two words Development and Operations.

- DevOps is a methodology that allows a single team to manage the entire application development

life cycle that is development, testing, deployment and Operations.

- The objective of DevOps is to shorten the system's development life cycle.

- DevOps is a software development approach through which superior quality software can be developed quickly and with more reliability.

*. Waterfall > Agile > DevOps
↳ silos issue (won't collaborate)

* meetings term
  - Sync Up Meeting
  - Scrum meeting
  - Stand up meeting

DevOps Stages CI/CD

Version Control	Continuous Integration	Continuous Delivery	Continuous Deployment
-Maintain different version of the Code Git	- Compile, validate, Code review, Unit testing, Integration testing Jenkins	-Deploy the build app to test servers Maven	- Deploying the test app on the production server for release

Basics of AWS Cloud required

* Cloud provides
  - IAAS ▸ Infrastructure as a service
  - Paas ▸ Platform as a service
  - Saas ▸ Software as a service

* Cloud - Cloud is a service it's gives us resources which can use remotely.

Creating AWS account

1. AWS free tier search
2. create free account
3. Enter details...
4. After all completion go to AWS Management console.
5. select root user.
6. Select your region

Making a Linux Machine

1. Services > EC2 (Compute) > Instances (running) > Amazon Linux 2 (AMI) > (Free tier eligible) > Next : Configure Instance details > (Name-Linux server) Next > Next > Next > Type

Download puttygen and putty

- from website, if forgot watch video

Working on server

- Creating instance and Copy the ID
- Now Open Puttygen
  - ↳ select load
  - ↳ Select our Linux123 pass file
  - ↳ Save private key
- Now open putty
  - ↳ Paste our server's public IP
  - ↳ Left side select SSH
    - ↳ select Auth
    - ↳ Browse select Linuxkey.ppk

On Terminal

↳ ec2-user

↳ sudo su → for getting all privelages.

$ → #

→ switch user

↳ super user do

After Completing LAB delete the server.

- Instance State
  - ↳ Terminate
    - ↳ It will delete the Instance.

Everything about Linux P-1

UNIX Project

1964 → Bell Laboratory (New Jersy)

1969 → withdraw

Dennis ritchie, Ken Thompson

They made

UNICs → Uniplexed Information & Computing services.

1975 → UNIX V6 → popular

- UNIX Versions

UNIX → IBM-AIX
→ Sun Solaris
→ Mac OS
→ HP-UX
→ Linux

* Linus Torvald → 1991 → study this OS then made Linux

A professor → Andrew Tanenbaum made → MINIX

GNU → Free software movement

→ Combination of software.

- Linux + GNU → OS

↓

Kernel

* Linux → RHEL
→ Fedora
→ Debian
→ others...

Community
Enterprise

→ Ubuntu, CentOS, AmazonLin, Kali

3rd most used OS

Created by an American

(Ubuntu → Teamwork)

Note

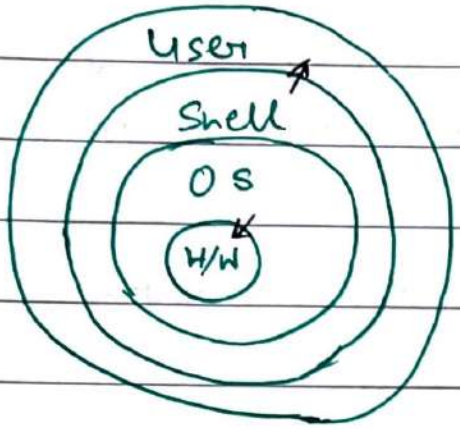
- Linux is a Kernel not OS
- Linux is not a UNIX derivative. It was written from scratch.
- A Linux distribution is the Linux kernel and a Collection of Software that together, create an O.S.

Features

1. Open source
2. Secure
3. Simplified updates for all installed software.
4. Light weight
5. Multi user - multi task.

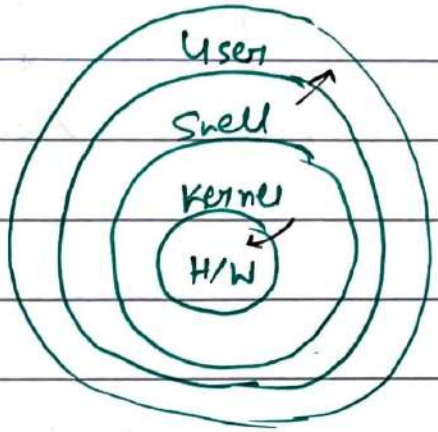
Linux part-2 Theory

Windows



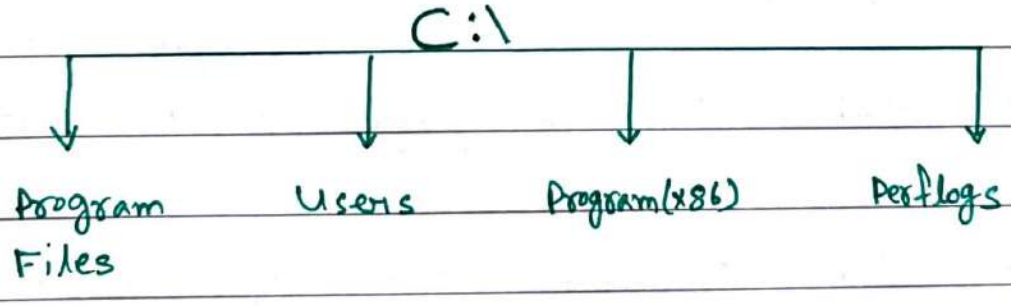
- Folder
- Adminstrator
- Software

Linux

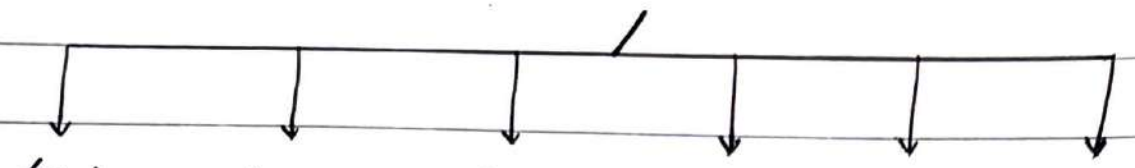


- Directory
- root user
- Package

File system for Windows



File system Hierarchy for Linux



/root - It is home directory for root user.

/home - home directory for other users.

/boot - It contains bootable files for Linux.

/etc - It contains all configuration files.

/usr - By default software are installed in this directory.

/bin - It contains commands used by all users.

/sbin - It contains commands used by only root user.

/opt - Optional application software packages.

/dev - Essential device files. This include terminal devices, USB or any device attached to the system.

Linux Commands

- How to create a file

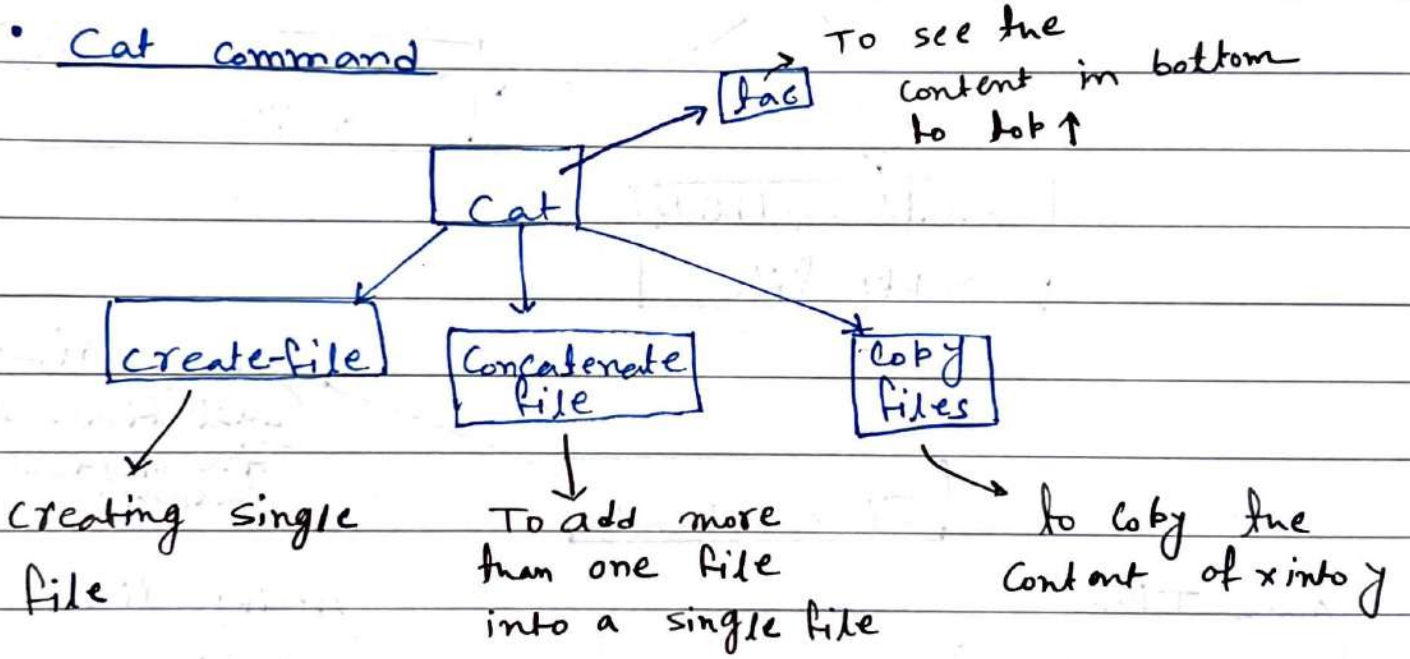
→ Cat

→ touch

→ vi/vim

→ nano

- **Cat command**



- **Creating and add content**

→ Cat > file ↵ (Redirect)

Ctrl+d } Save and quit

- Add some existing lines to the same file

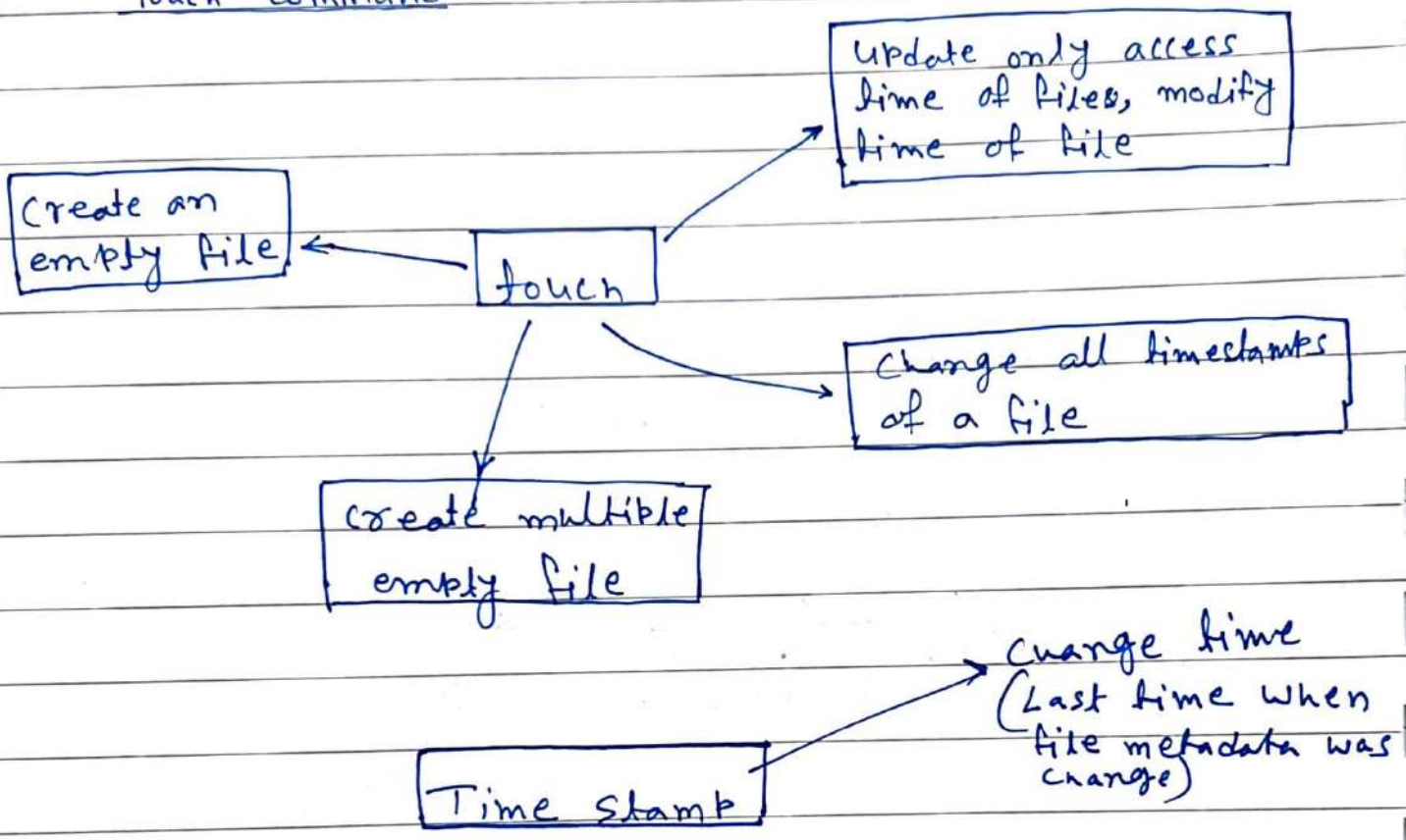
→ Cat >> file1

Ctrl+d

- Add two files content to another one

→ cat file1 file2 > all

- **Touch command**



Access Time
(Last time when a file was access)

touch -a

modify time
(Last time when a file was modified)

touch -m

- cmnd
  - create file

→ touch file1

- multiple files

→ touch file1 file2 file3

- change all time stamps
→ touch file1

- check timestamps
→ stat file1

- change specific time (access time)
→ touch -a file1

- change modify time
→ touch -m file1

- **Vi editor**

Note: :w → to save
:wq or, :x → to save and quit
:q → quit
:q! → force quit, no save

- **nano command**

- new file and save
→ nano file

Ctrl+x → Y

Linux part-4

- How to create directory

→ mkdir dir1

- Directory inside a directory

→ mkdir dir2/dir3/dir4

or,

mkdir -p dir2/dir3/dir4

- Multiple directory

→ mkdir dir5 dir6 dir7

- How to hide file/folder

→ touch .file1 → ls -a show all files.

or

→ mkdir .dira

- How to copy a file

→ cp file1 file2

↑ src. ↑ dest.

- How to cut & paste a file

→ mv file1 dir1

cut → moved → paste

- Rename file/dir

→ mv file1 myfile

↑ old name ↑ new name

- Remove directory

→ rmdir

*.	Remove even non empty file & dir.

rm -rf <file/dir>

- Remove both the parents and child dir.

→ rmdir -p

- Removes all the parent and subdirectories along with the verbose

→ rmdir -pv

- Remove non-empty dir. including parent & sub directories

→ rm -rp

- remove empty dir

→ rm -r

- How to go dir. inside a dir

→ cd dirc/dird/dire ← c → e

- How to go outside dir. out of dir.

→ cd ../../.. ⟶ 3 times

- To watch first 10 lines

→ head <file-name>

- To watch last 10 lines

→ tail file-name

Linux Command part-5

- Detail about Machine

→ hostname / hostname -i

→ To show IP

- IP show

→ ifconfig

- Which version of Linux

→ cat /etc/os-release

*. yum - Yellow dog updater modified

- Install apache using yum

→ yum install httpd

- Remove a package

→ yum remove httpd

- Update a package

→ yum update httpd

- Starting a service

→ service httpd start

- Check service status

→ service httpd status

- By default automatically start the service
→ chkconfig httpd on / chkconfig httpd off

- Checking package installed or not
→ which <Package_Name>

- Check all installed software
→ yum list installed

(***)
- Finding special names or just for finding purpose

→ grep .... ex)

Finding this
user details
grep root /etc/passwd
Finding place

- For sort alphabatically
→ sort

• echo

- Print a message
→ echo "Hello"
- Print a message in a file
→ echo "welcome" > file2
- Update or append with echo
→ echo "Message" >> file2
- Delete or clearing full file with echo
→ echo . > file2

Linux Command part - 6

- Create a user

→ useradd (check) / cat /etc/passwd

Ex) useradd <user-name>

* By creating user, group will create automatically.

- create a group

→ groupadd / check cat /etc/group

- To add user into group, to add multible user

→ gpasswd -a / -M → multiple user

-a: single user, <username>

-M: <user1, user2 ... n>, <group Name>

<group_Name>

- Hardlink (backup)

→ ln |Ex| ln <file> <backupfile>

- softlink (shortcut)

→ ln -s |Ex| ln -s <file> <softfile>

- Tar is an archiver used to combine multible files into one

→ tar |Ex| tar -cvf dirx.tar dirx → file

c: create, v: verbose, f: forcefully, dirx.tar: after tar

- gzip is a Combination tool used to reduce the size of a file.

→ gzip / unzip → gunzip dirx.tar.gz / Extract tar → tar -xvf dirx.tar

- wget (non interactive network downloader)

→ wget <url>

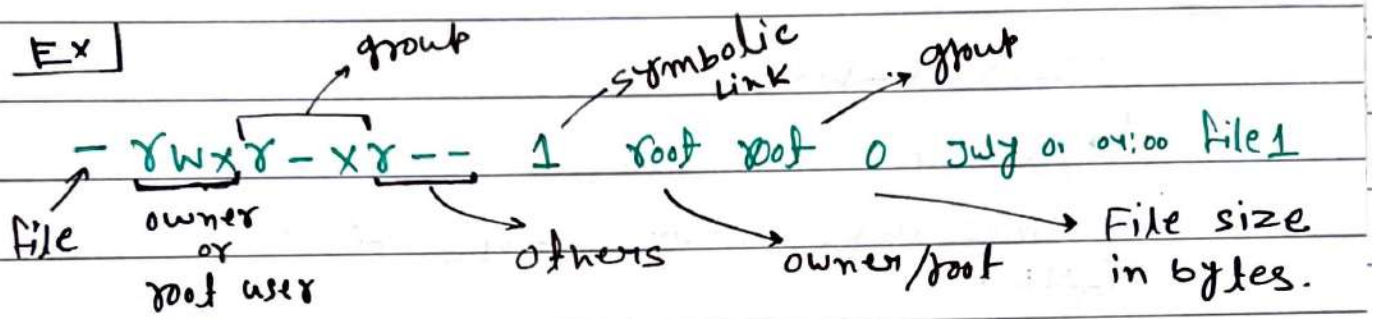
Linux Command Part-7 | File Permission

→ chmod → used to change the access mode of a file.

→ chown → Change the owner of file or dir.

→ chgrp → Change the group of file or dir.

Ex]



r = read

w = Write

x = execute

- Access Modes/permissions

Access-Modes		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

- Another method

u = user/owner
g = group
o = others

= + -

* If we won't user get full Permiss:
u = rwx

* If only on permission we want to provide
u+r

* If remove specific permission
u-r

Ex-1

r-- rwx-wx
↳ Command

chmod u=r, g=rwx, o=wx file1
After

Ex-2

drwxr-xr--

chmod u-wx, g+w, o=wx file1
After drw--rwxrwx

- If change the owner
→ chown swaib file1

- If change the group
→ chgrp devops file1

What is Git, Github, DVCS, CVCS

- Terms used in software company

→ software Configuration Management

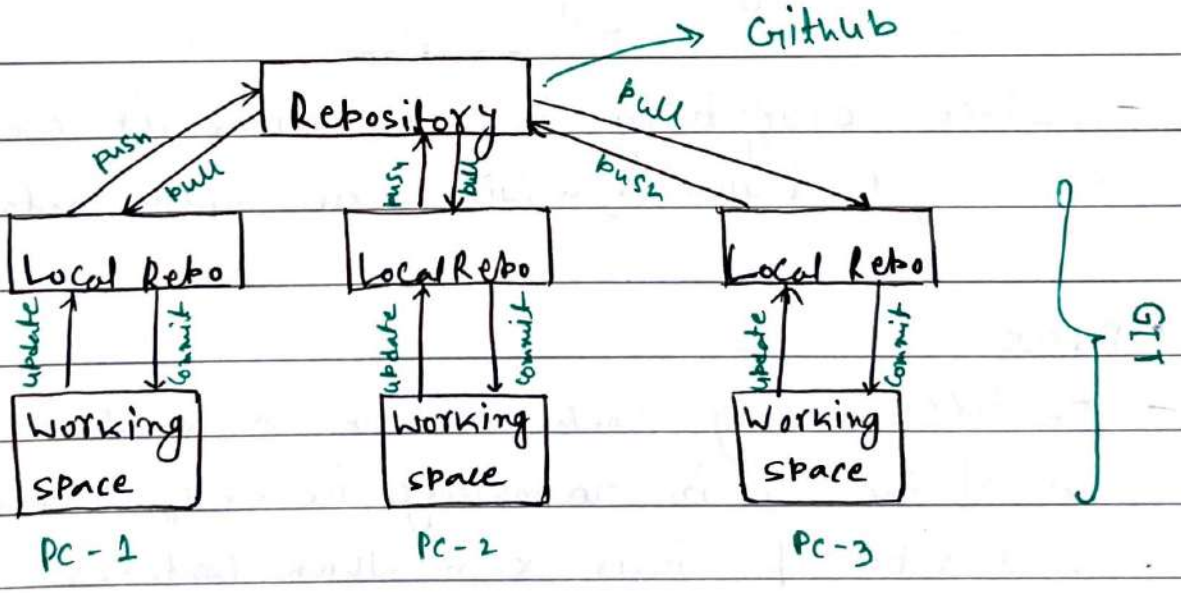
or,

→ Source Code Management

Centeralised Version Control System (CVCS)

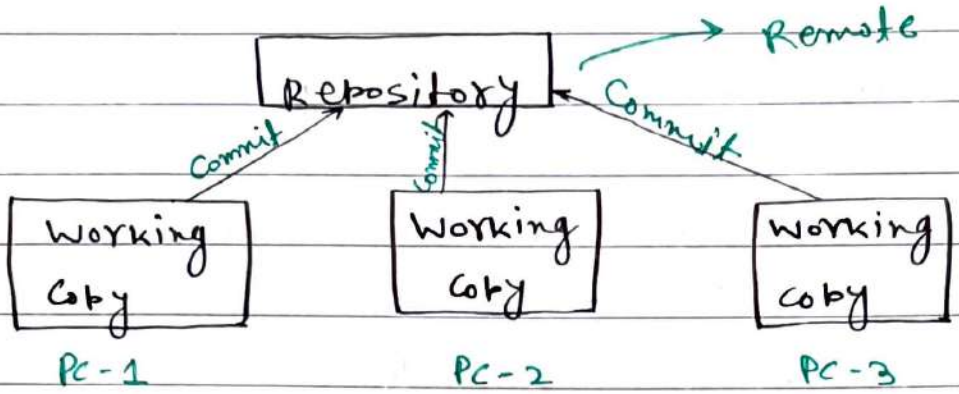
Distributed Version Control System (DVCS)

Distributed Version Control System (DVCS)



- Git created by Linus Torvald (2005)
- Because BitKeeper services stops git created.

Centralised version control system (CVCS)



- **CVCS Drawbacks**

- It is not locally available, meaning you always need to be connected to a network to perform any action.

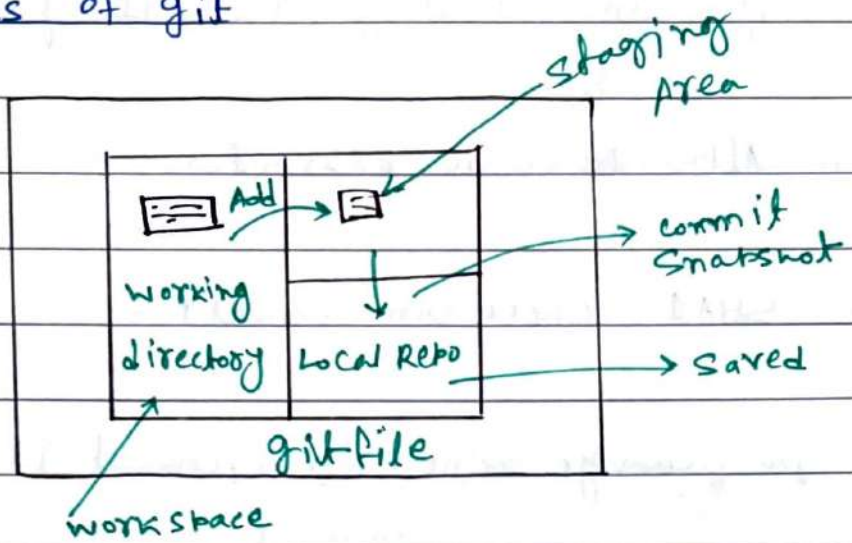
- Since everything is centralised, if central server gets failed, you will loose entire data. eg - SVN tool

- **DVCS**

- In DVCS, every Contributer has a local copy or 'clone' of the main repository i.e everyone maintains a local repo. of their own which contains all the files & metadata present in main repository.

Git three stages

- stages of git



Repository

- Repository is a place you have all your codes or kind of folder on server.

- It is a kind of folder related to one product.

- Change are personal to that particular Repo.

Server

- It stores all Repository

- It contains metadata also

Working directory

- Where you see files physically and do modification

- At a time, you can work on particular branch.

Commit

- Store changes in repository. You will get one Commit-ID
- It is 40 Alpha-Numeric characters
- It uses SHA1 checksum concept.
- Even if you change one dot, Commit-ID will change.
- 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 a specific version in the repo. Once a tag is created for a particular save, even if you create a new commit, it will not be updated.

Snapshots

- Represents some date of particular time.
- It is always incremental i.e It stores the change (updated data) only. Not entire copy.

Push

- Push operations copies changes from a local repository server to a remote or centrel repo. This is used to store the changes permanently into the git repo.

Pull

- Pull operation copies the changes from a remote repo. to a local machine. The Pull operation is used for synchronisation between the repo.

Branch

- Product is Same, So one repository, but different task.

- Each task has one Seperate branch.

- Finally merges (code) all Branches.

- Changes are present to that particular branch

- Default Branch is 'Master'.

- File created in workspace will be visible in any of the branch workspace, until you commit. once you comm then that file belongs to that particular branch.

Installing git on AWS

In Mumbai Server

→ sudo su
→ yum update -y
→ yum install git -y
→ git --version
→ git config --global username "<name>"
→ git config --global user.email "<email>"

How to add, Commit, Push/Pull git → github

- Login into Mumbai EC2 instance
- Create one directory and go inside it

→ git init
→ touch myfile → put something
→ git status
→ git add .
→ git commit -m "1st commit from Mumbai"
→ git status
→ git log
→ git show <commit-ID>
→ git remote add origin <centrel-git-URL>
→ git push -u origin master

[Enter username and pass]

In Singapore server

- Create one directory and go inside it

→ git init

→ git remote add origin <github-repo-url>

→ git pull origin master

→ git log

→ git show <Comment_ID>

- Now add some code in the file

→ git status

→ git add .

→ git commit -m "Singapore update 1"

→ git status

→ git log

→ git push origin master

To ignore some files while commenting

- Create one hidden file .gitignore and enter file format which we want to ignore.

for eg → vi .gitignore

*.css

*.java

→ git add .gitignore

→ git commit -m "Latest update exclude"

→ git status

- create some text, java & css files and add them by running "git add"

  • for eg → touch file1.txt file2.lvt
  file3.java file4.~~txt~~ css

→ ls
→ git status
→ git add .
→ git commit -m "My text files only."

Create Branch, git stash, git reset, Merge, conflict

- Each task has one seperate branch
- After done with code, Merge other branches with master
- This concept is useful for parallel development.
- You can create any no. of branches.
- Changes are personal to that particular branch.
- Default branch is "Master"

*** - Files will be created in workspace visible in any of the branch workspace until you commit, Once you commit, then that files belongs to that particular branch.

- When created new Branch, data of existing is copied to new Branch.

Commands for Branch

- For show all Branches
  → git branch

- create new Branch
  → git branch <branch-name>

- For going to a specific branch / change branch
  → git checkout <branch_name>

- Delete branch
  → git branch -d <branch_name>

Command for Merge

*. We can't Merge branches of different Repositories.

*. We use pulling mechanism to merge Branches.

CMND

→ git merge <branch-name>

Conflict

- When same file having different content in different branches, if you do merge, conflict occurs (Resolve conflict then add and commit)

- Conflict occurs when you merge branches

Git Stashing

- Suppose you're implementing a new feature for your product. Your choice is in progress and suddenly a customer esclation comes because of this, you have to keep aside your new feature work for few hours. You cannot commit your partial code and also cannot throw away your changes. So you need some temporary storage, where you can store your partial changes and later on commit it.

→ To stash an item (only applies to modified files not new file)

Command for stash

- To stash an item

→ git stash

- To see stashed items list

→ Git stash list {0} → New file
{1}
{2} → old file

- To apply stashed items

→ git stash apply stash@{...}

- To clear the stash items

→ git stash clear

Grit Reset

- Grit reset is a powerful command that is used to undo local changes to the state of a git rebo.

Cmnd

- To reset staging area

→ git reset <filenames>

* git reset ← forall reset

- To reset the changes from both staging area and working directory at a time.

→ git reset --hard

Git revert, Tag in git, Github clone, reset vs revert

Git revert

- The revert Command helps you undo an existing comment.

- It does not delete any data in this process, instead rather git creates a new commit with the included files reverted to their previous state so, your version control history moves forward while the state of your file moves backward.

reset → before Commit.

revert → After Commit.

Command

→ sudo su

→ cd mumbaigit

→ ls

→ git status

→ cat > newfile

Hi Final Code for abb

→ git add

→ git commit -m "Code"

→ git log --online

→ git revert <comment_id>

Other Commands

- Remove untracked files

→ git clean -n (dry run/warning)

→ git clean -f (forcefully)

Tags

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

- To apply Tag

→ git tag -a <tagname> -m <message> <comment_id>

- To see list of Tags

→ git log

- To see particular commit content by using tag

→ git show <tag_name>

- To delete a Tag

→ git tag -d <tagname>

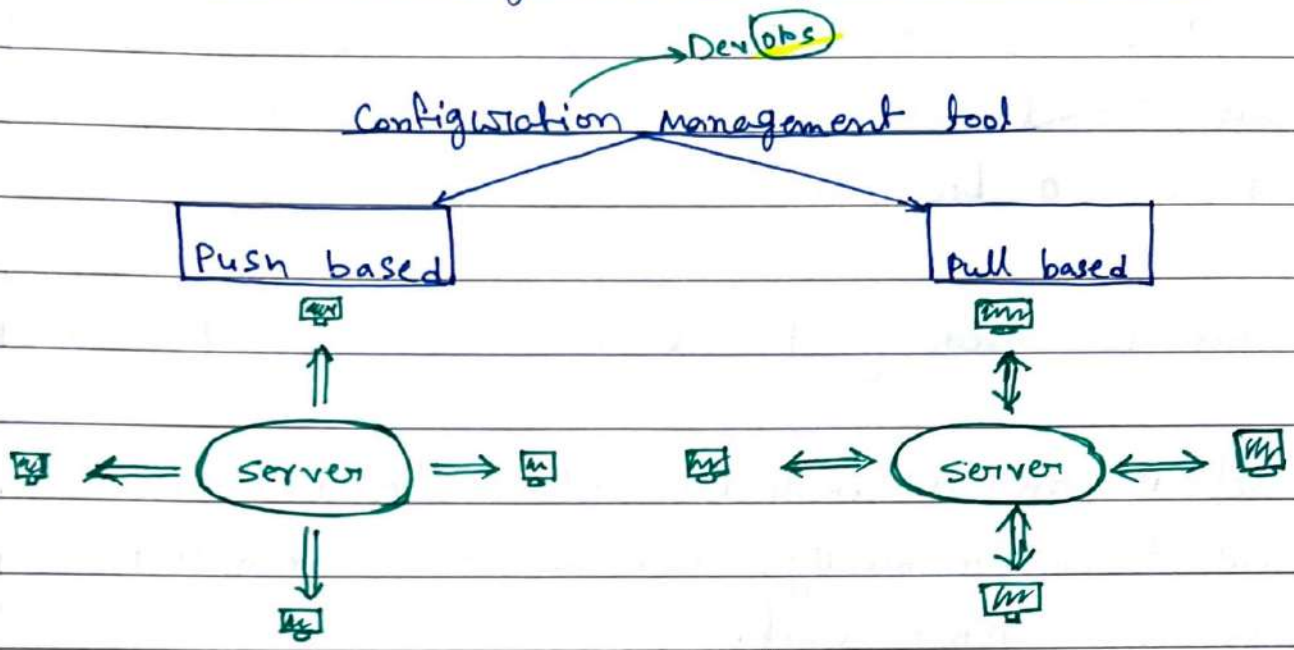
Github Clone

→ Open github website
→ Login and choose existing repo.
→ Now, go to your linux Machine and run cmnd.

→ git clone <url_of_github-repo>

→ It creates a local repo automatically in linux machine with the same name as in github acnt.

Chef | Configuration Management Tools | Ohai



Push Conf. server pushes Conf. to the nodes. (Ansible, Salt Stack)	Pull Conf. nodes check with the server periodically and fetches the conf. from it. (Chef, Puppet)

* IAC - Infrastructure as Code
  - By using code we Configure our Infra.

Note

- Chef is a Company and the name of Configuration management tool written in Ruby and Erlang.
- Founded by Adam Jacobs in year 2009.
- Actual name was "Marrionate" Later renamed to "Chef".

- On Abril 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 an adminstrator tool whatever system admins used to do manually, now we are automating all those task by using Chef.

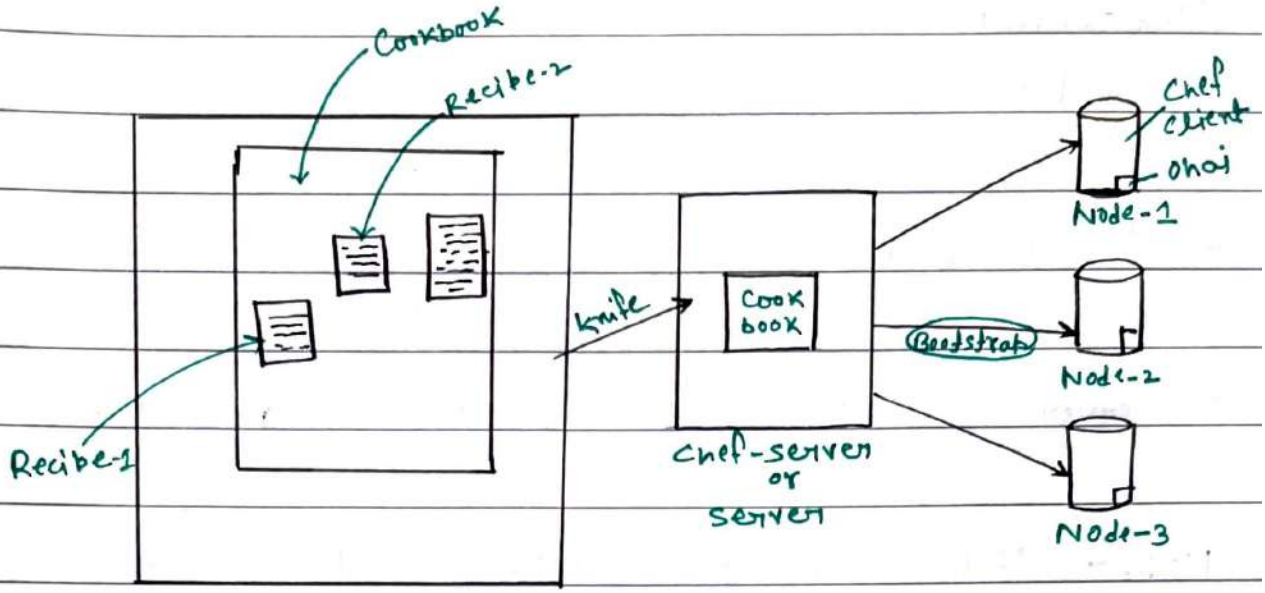
- Configuration Management → It is a method through which we automate admin tasks.

- Configuration Management tool turns your code into (IaC) so your code can be repeatable, testable and versionable.

■ Advantages

- Complete Automation
- Reduce Cost
- Increase Uptime
- Prevent errors
- Improve Performance
- Ensure Compliance (Rule and Regulation follow)

Chef Architecture or Process



*. Chef supermarket for predefined Cookbooks for various purpose.

Note

Workstation - Workstation are personal Computers or virtual servers where all Conf. code is tested, created or change.

- DevOps engineer actually sits here and write Codes. This code is called recipe. A collection of recipes are known as Cookbook.

- Workstation Communicate with the chef-server using knife.

- Knife is a CL(command Line) Tool that uploads the Cookbook to the server.

Chef-Server - 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 - Nodes are the systems that require the Conf.

- Ohai fetches the current state of the node it's located in.
- Node Communicate with the Chef-server using Chef-client
- Each node Can have a different Configuration Required.
- Chef-client is installed on every node.

SUMMARY

Chef-workstation - Where you write Code.

Chef-server - Where you Upload Code.

Chef-Node - Where you apply Code.

Knife - Tool to establish Communication among workstation, server and node. Knife is a CLI that runs on workstation.

Chef-client - Tool runs on every Chef node to pull Code from Chef-server.

Chef client will -

- Gather current system configuration.

- Download the desires system configuration from the Chef-server.

- Configuration the node such that it adhere to the policy.

Ohai - Maintain current state information of chef-Node

Idempotency - Tracking the state of system resources, to ensure that the changes should not re-apply Repeatedly.

Chef part-2 | How to create Cookbook and Recipe

- 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

- goto downloads, > chef-workstation

  > enter name, email, combany → Automatically starts downloading process > Go to downloads & Copy the URL.

→ Now to Linux Machine

→ wget <url>

→ Paste the url which you copied

→ ls

It shows Chef package.rpm

→ yum install <Chef_Workstance>

→ Paste the package name

→ which chef → To check installed or not

→ chef --version

Cookbook

Cookbook 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
↘ ruby

Readme.md → Information about usage of Cookbook.

Recipe → where you write code.

Spec → For unit test.

Test → For integration test.

LAB

→ which chef

→ mkdir Cookbooks

→ ls

→ cd Cookbooks

→ chef generate cookbook <test-Cookbook> → If we want to recipe then write recipe

→ yum install tree -y

→ tree

→ cd test-Cookbook → Inside generated Cookbook we've to generate recipe

→ chef generate recipe <name>

→ tree

→ cd .. → After changing directory to normal Cookbook then write code

→ vi test-Cookbook/recipes/test-recipe.rb

Code

```
file '/myfile' do
content 'Welcome Md Shoaib'
action :create
end
```

- Check Syntax

→ chef exec ruby -c test-Cookbook/recipes/test-recipe.rb

- Build & Run

→ chef-client -zr "recipe[test-Cookbook::test-recipe]"

Local Machine — Run list

■ Create and write 2nd Recipe

→ cd test-Cookbook

→ chef generate recipe recipe2

→ cd ..

→ vi test-Cookbook/recipes/recipe.rb

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

■ Deploying an Apache Server:-

→ chef generate Cookbook apache-Cookbook

→ ls

→ cd apache-Cookbook

→ chef generate recipe apache-recipe

→ tree

→ cd ..

→ ls

→ vi apache-Cookbook/recipes/apache-recipe.rb

Code

```
package "httpd" do
action :install
end
file '/var/www/html/index.html' do
content "welcome Md Shoaib"
action : 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]"

Note

Resource → It is the basic Component of a recipe used to manage the infrastructure with different kind of states. There can be multible resources in a recipe, which will help in configuring and managing the infrastructure- for eg → Infrastructure as Code

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 embeded Ruby template

Cookbook-file → Transfer 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.

cron → Edits an existing cron file on the node.

directory → manages the dirctory on the node.

Attributes in Chef

Attribute - Attribute is a key-value pair which represents a specific detail about a node.

Use of Attribute - To determine current state of the node
- To determine what the state of the node was at the end of the previous chef-client run.

- To determine the state of the node should be at the end of the current chef-client run.

Types of Attributes - i) Default ← least priority
ii) force default
iii) normal
iv) override
v) force override
vi) automatic ← highest priority

Who define attribute -

Node, Cookbooks, Roles, Environment, Recipe
↑ Collected by Ohai at the start of each chef-client run.

Note:

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

LAB

→ sudo su

→ ohai

→ ohai ipaddress

→ ohai memory/total

→ ohai CPU/0/mhz

→ ls

→ cd Cookbooks

→ ls

→ cd apache-Cookbooks

→ chef generate recipe recipe3.

→ cd..

→ vi apache-Cookbook/recipes/recipe3.rb

Code

```
file '/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
```

→ chef exec ruby -c apache-Cookbook/recipes/recipe3.rb

→ chef-client -zr "recipe[apache-Cookbook::recipe3]"

→ Paperkraft

→ cat /basicinfo

Chef Runlist, Run Multible Recipes & other

- How to execute Linux Commands if Ruby code is not possible to write

→ sudo su

→ cd Cookbooks

→ ls

→ vi test-Cookbook/recipes/test-recipe.rb

Code

```
execute "run a script" do
Command <<-EOH
mkdir /rajputdir
touch /rajput file
EOH
end
```

End of Here
or,
End of Hunk

→ chef-client -zr "recipe[test-Cookbook::test-recipe]"

*. Drawback for use direct Linux Command

→ Chef-client Can't Idembotency in this process.

Create a user/group

→ vi test-Cookbook/recipes/test-recipe.rb

Code

```
user "rajput" do
action : create
end
```

→ chef-client -zr "recipe [test-Cookbook :: test-recipe]"

→ vi test-Cookbook/recipes/test-recipe.rb

Code

```
group "technical" do
action :create
members 'rajput'
append true
end
```

← Add new group without overwrite

→ chef-client -zr "recipe [test-Cookbook :: test-recipe]"

→ Cat /etc/group

*. First we need to create user then group.

Note:

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

What is runlist

- To run the recipes in a sequence order that we mention in a runlist.

- With this process, we can run multiple recipes, but the condition is, there must be only one Recipe from one Cookbook.

cmnd for that

→ sudo chef-client -zr "recipe[test-Cookbook::test-recipe],recipe[apache-Cookbook::apache-recipe]"

How to include recipe

- To call recipe/recipes from another recipe with in same Cookbook.

- To run multible recipes from same Cookbook.

- Here comes the default recipe into action(you can use any)

- We can run any no. of recipes with this command, but all must be from Same Cookbook.

→ vi test-Cookbook/recipes/default.rb

```
include_recipe "test-Cookbook::test-recipe"
include_recipe "test-Cookbook::recipe2"
```

→ chef-client -zr "recipe[test-Cookbook::default]

- Now we are Combining previous two concepts, so that we can run multible recipes from multible Cookbooks simultaneously.

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

(before this include first)

or,

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

Bootstraping in Chef | Upload Cookbook in chef-server and apply to node | Automation

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

LAB

- Login into Amazon Linux Machine using Putty.

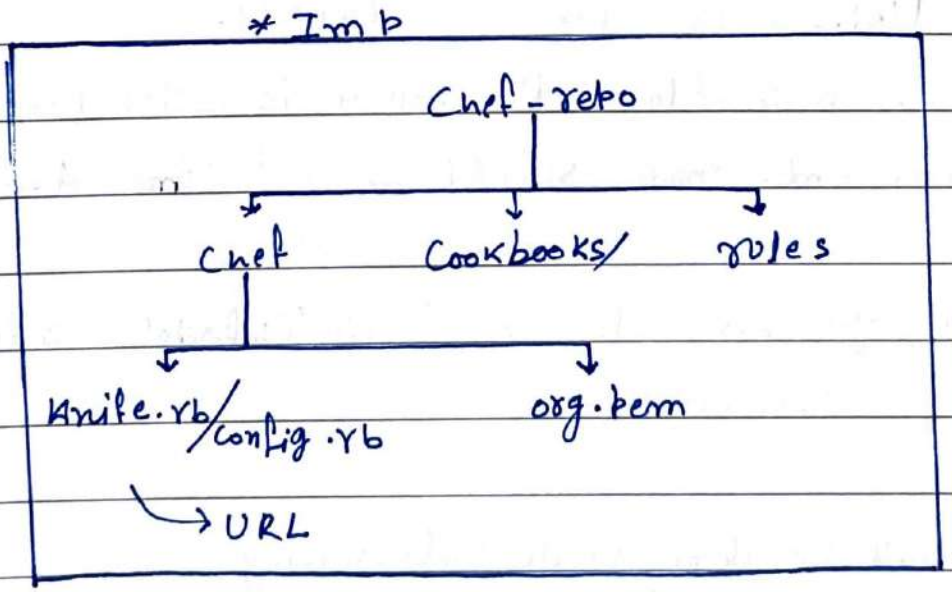
→ ls

→ cd Cookbooks/

→ ls

o/p → apache-Cookbook test-Cookbook

- Now open Chrome → search manage.chef.io
- create one account
- Go to Chef account → click on Organisation → starter kit → Download starter kit
→ open the download content → unzip → chef-repo
→ Now Download "winscp" → Login (use key in adv) with ec2 creds
→ Now Drag & Drop "chef-repo" folder from Win → Lin.
- Now open workstation in AWS again



→ [Cookbook] # ls

→ cd ..

→ [ec2-user] # ls

O/P → Chef-repo Cookbooks

→ [ec2-user] # cd chef-repo

→ ls -a

→ [chef-repo] # cd .chef/ → Hidden file

→ [.chef] # ls ← Everything from chef-repo

O/P → knife.rb shoaib.pem

→ [.chef] # cat config.rb

You will get URL of chef server

→ [.chef] # cd ..

→ [chef-repo] # knife ssl check ← To check that chef server is connected properly.

Bootstrap a Node

- Attaching a node to chef server is called Bootstrapping (Both workstation and node should be in same Az.)

- Now onwards, you 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.

*. Key pair Name → Node-key.pem

- Create one Linux Machine (Node1), launch in same availability zone

Advance Details

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

- Now go to chef-workstation

{ paste node-key.pem in chef-repo folder }

→ Using WinSCP

- Bootstrating to the node

→ [chef-repo]# knife bootstrap <Private_IP_Node> \
--ssh-user ec2-user --sudo -i node-key.pem \
-N <node-name>

i for our key

N — our choice

Node name

This key need to transfer in Linux via then bootstrap "WinSCP"

{ Paste node-key.pem in chef-repo folder }

- To see bootstrap node

→ [chef-repo]# knife node list

Create a server (Instance) again to use, as a node

- Instance should be in same AZ.
- Type

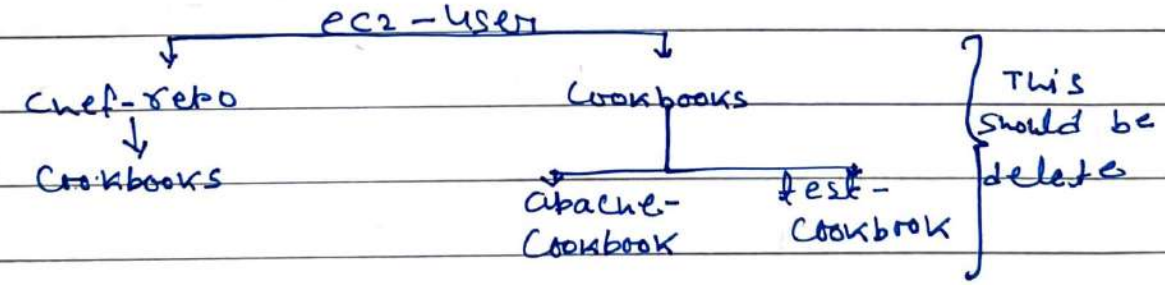
Advance details

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

- Name Chef Node-1
- New security group → node-sg
- 2 rules SSH, HTTP

Deleting Cookbooks for Shorten Confusion

- We have two Cookbooks so we need to Copy one Cookbook files into chef-repo/directory.



→ [chef-repo] # ls

O/p → Cookbooks rules

→ [chef-repo] # cd ..

→ [ec2-user] # ls

O/p → Cookbooks

→ [ec2-user] # mv Cookbooks/test-Cookbook chef-repo/Cookbooks

→ [ec2-user] # mv Cookbooks/apache-Cookbook chef-repo/Cookbooks

→[ec2-user]# rm -rf Cookbooks/

→[ec2-user]# cd chef-repo

→[chef-repo]# ls

O/P → Cookbooks

→[chef-repo]# ls Cookbooks/

→ O/P → apache-Cookbook test-Cookbook

■ Now we have to upload apache-Cookbook into-chef-server

→[chef-repo]# knife Cookbook upload apache-Cookbook

- Now to check, wheather Cookbook is uploaded or not.

→[chef-repo]# knife Cookbook list

- Now, we will attach the recipe, which we would like to run on node. (your node name)

→[chef-repo]# knife node run-list set node1 "recipe[apache-Cookbook::apache-recipe]"

- For checking which things connected into our runlist

→[chef-repo]# knife node show node1

■ Now, take access of Node1 with the help of putty [semi-Automode]

*. key name save node1

→[ec2-user]# sudo su

→[ec2-user]# chef-client

- Now all files would be updated, go to browser, paste public-IP of the node, you will get webpage.

Now in workstation

- Changing some content in workstation to check that node (semi-Automation is working or not)

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

- Upload the cookbook

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

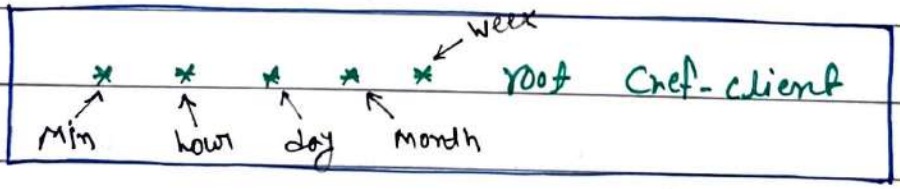
Now, again go to node & run chef-client command

Full Automation Using crontab

- Now we do not want to call chef-~~repo~~ client everytime, We want to automate this process -

→ go to Node-1

→ [ec2-user]# vi /etc/crontab



- Now go to workstation and change some content to check that this full automation working →

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

→ Upload the file to chef server

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

* check node ip to the browser.

■ Create one more Linux m/c (node2)

* Advance details

```
#!/bin/bash
Sudo su
yum update -y
echo "* * * * * root chef-client" >>/etc/crontab
```

- Now go to workstation and run bootstrap cmnd

→ knife bootstrap <ip_address> --ssh-user ec2-user
--sudo -i node-key.pem -N <node-name>

- Now attach Cookbook to node run-list

→ knife node run-list set node2 "recipe[apache-Cookbook
::apache-recipe]

- Now check in browser, node2 shows webpage.

Creation and use of ROLE and LAB again

Command to delete & clean chef-server

- To see list of Cookbooks which are present in chef-server
→ knife cookbook list

- To delete Cookbooks from chef-server
→ knife cookbook delete <cookbook_name> -y

- To see list of nodes which are presented in chef-server.
→ knife node list

- To delete node 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 client 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

Commands for role

→ [chef-repo]# ls -a

o/p → .chef roles

→ [chef-repo]# cd roles/

→ [roles]# ls

→ o/p → starter.rb

→ [roles]# vi devops.rb

Code

```
name "devops"
description "Web server role"
run_list 'recipe[apache-Cookbook::apache-recipe]"
```

- Now comeback to chef-repo

→ [roles]# cd..

→ [chef-repo]#

- Now upload the role on chef-server

→ [chef-repo]# knife role from file roles/devops.rb

o/p → updated role devops

If you want to see the role created

→ [chef-repo]# knife role list

o/p → devops

Now create instances as node1 & Node2 ... at same Az as of workstation.

* same key node-key.pem

* Advance details

```
#!/bin/bash
Sudo su
yum update -y
echo "* * * * * root chef-client" >>/etc/crontab
```

- Now bootstrap the node

→ [chef-repo]# Knife bootstrap <private_ip_node> --ssh-User ec2-user -- sudo -i nodekey.pem -N node1

Same for every node

Change Just

*. Now, Connect this nodes to role

Other Commands

→ [chef-repo]# knife node list

- Attaching nodes using roles

→ [chef-repo]# Knife node run-list set node1 "role[devops]"

→ [chef-repo]# knife node run-list set node2 "role[devops]"

- Show nodes are in roles or not

→ [chef-repo]# knife node show node1

O/P → Runlist - role[devops]

- Upload Cookbook

Now, check public-ip of any node in browser

Changing recipe of recipe1 | Just checking

→ [chef-repo]# Cat Cookbooks/apache-Cookbook/recipes/recipe1.rb

- Changing recipe1

→ [chef-repo]# vi roles/devops.rb

Code

```
name "devops"
description "Web Server role"
run-list "recipe[apache-Cookbook::recipe]"
```

↑ All recipe run

- Upload role

→ [chef-repo]# knife role from file roles/devops.rb

- Now, take access of any node via putty & check.
- Now again go to workstation

→ [chef-repo]# vi roles/devops.rb

Code

```
name "devops"
description "web server role"
run-list "recipe[apache-Cookbook]"
```

↳ All Cookbooks will run

- Upload role

→ [chef-repo]# knife role from file roles/devops.rb

→ [chef-repo]# knife Cookbook upload apache-Cookbook

- Now we are attaching two Cookbooks in a role.

→ [chef-repo]# vi roles/devops.rb

Code

```
name "devops"
description "web server role"
run-list "recipe[apache-Cookbook]", ↵
         recipe[test-Cookbook]"
```

- Now upload this role to server

→ [chef-repo]# knife role from file roles/devops.rb

o/p → updated role devops

- Now Upload test-Cookbook

→ [chef-repo]# knife Cookbook upload test-Cookbook

* For installing many packages at one script

→ [chef-repo]# vi Cookbooks/test-cookbook/recipes/test-recipe.rb

Code

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

* go any node and check install or not.

What is Docker | O.S Level virtualisation | Docker Hub

- Docker was first release in March 2013. It is developed by Solomon Hykes and Sebastin Paul.

- 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 O.S.

- 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 OS Level virtualization, also known as Containerization.

- Before Docker, many users faces the problem that a particular Code, is running in the developer's System but not in the User's system.

- Docker is a set of Platform as a service (PaaS) and provides O.S Level virtualization. VM ware Provides Hardware virtualisation.

Docker Architecture | Docker Advantages & Limitation

Advantages of Docker

- No pre-allocation of RAM.

- CI Efficiency → 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 / virtual H/W or on cloud.

- You can re-use the image

- It took very less time to create Container

Note

- When Image is running we can say Container;

- When we send a Container or not-runnable state we can say Image.

Disadvantages of Docker

- Docker is not good solution for application that requires rich GUI.

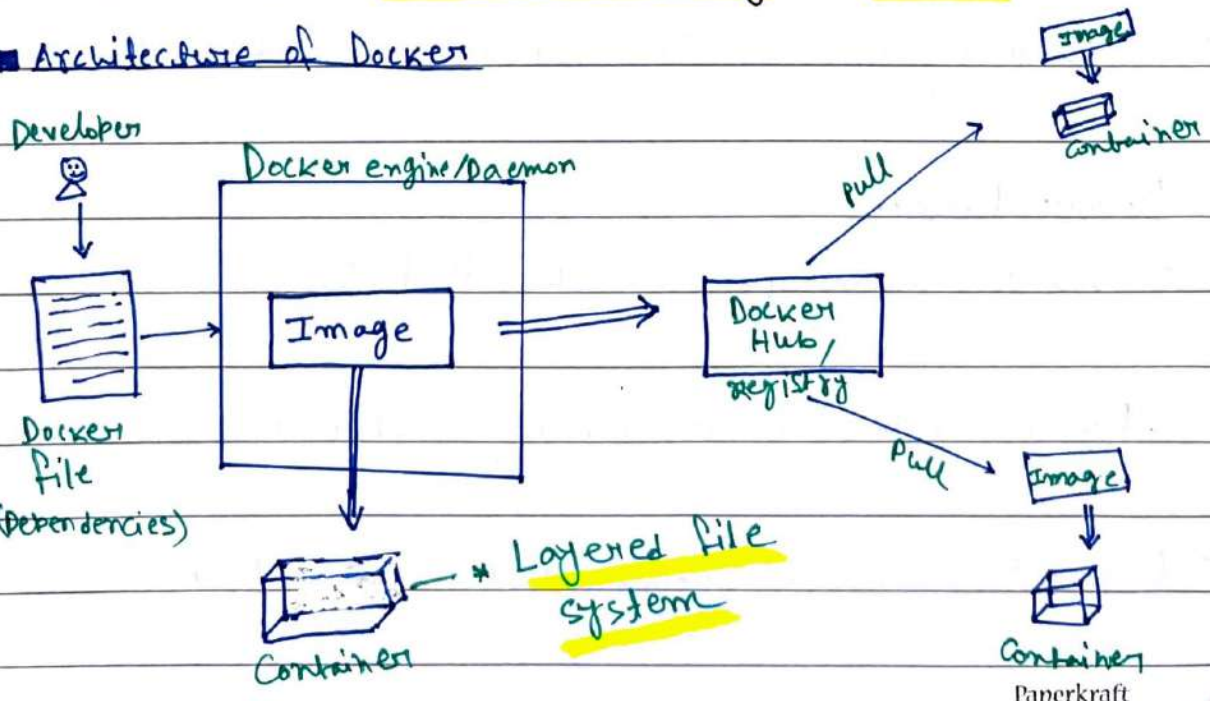
- Difficult to manage large amount of Containers.

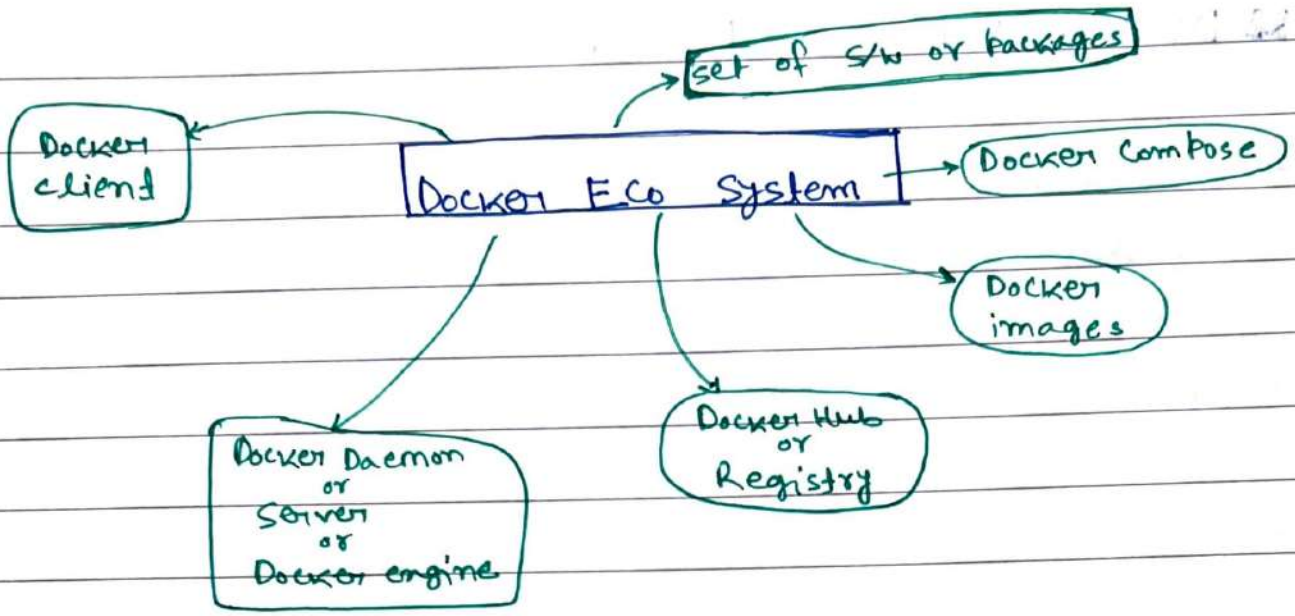
- Docker does not provide Cross-platform Combatibility means if an application is designed to run in a docker Container on windows, then it can't run on linux or vice-versa.

- Docker is suitable when the development O.S and testing os are same, if the O.S is different, we should use VM.

- No solution for Data Recovery & Backup.

Architecture of Docker





Components of Docker

Docker Daemon

→ Docker daemon runs on the Host O.S

- It is responsible for running containers to, manages docker services.

- Docker Daemon can Communicate with Other Daemons.

Docker Daemon

- Docker daemon runs on the Host OS

- It is responsible for running Containers to, manages docker ser

- Docker Daemon can communicate with other daemons.

Docker Client

→ Docker users can interact with docker daemon through a client

- Docker client uses Commands (CLI) and Rest API to communicate with the docker daemon.

- When a client runs any server command on the docker client terminal, the client terminal sends the 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 an environment to execute and run applications. It contains the docker daemon, images, containers, networks 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 also called docker Hub.

2) Private Registry → It is used to 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.

- Ways to create an images
  1) Take image from Docker Hub
  2) Create image from Docker File
  3) Create image from existing docker containers

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 a copy of that template.

- Container is like a virtualization when they run on docker engine.

- Images beomes Container when they run on docker engine.

Docker Install | start and Stop a Container

LAB

- create a machine on AWS with Docker installed AMI, and install Docker if not installed.

- To see all images present in your local

→ docker images

- To find out images in docker hub

→ docker search <image_name>

- To download image from docker-hub to local machine.

→ docker pull <image_name>

- To give name to Container and run

→ docker run -it --name <container_name> <image-name>/bin/bash

(or ubuntu / centos)

-it: create + start; Interactive mode; also -i -t; terminal

- To Check, Service is start or not

→ Service docker Status / docker info

- To start service

→ service docker start

- To start Container / stop

→ docker start <Container_name> / stop <Container_name>

- To go inside Container

→ docker attach <Container_name>
↳ Ctrl+Swoosh

- To see all Containers

→ docker ps -a ← all
← process status

- To see only running Containers

→ docker ps

- To stop Container

→ docker stop <container_name>

- To delete Container

→ docker rm <container_name>

- Exiting from docker Container

→ exit

- To delet Images

→ docker image rm <Image_name>

Dockerfile creation

Docker image creation using existing docker file

- Login into AWS account and start your EC2 instance access it from putty.

- Now we have to create Container from our own image, Therefore create one container first -

→ docker run -it --name <container_name> <image_name>
ubuntu
/bin/bash

→ cd tmp/ ← inside directory to create a file

- Now create one file inside this tmp directory

→ touch myfile

- Now if you want to see the difference between the base image & changes on it then -

→ docker diff <old_container_name>

O/P → C /root
A /root/.bash_history
C /tmp
A /tmp/myfile

cmnd outside container

D → Deletion
C → Change
A → Append / addition

→ comm outside [ec2-user]

- Now create image of this Container

→ docker commit <container_name> <container_image-name>

→ docker images

<container_name> → our container

<container_image-name> → Name of image which we want to give

- Now create Container from this image

→ docker run -it --name <container_name> <update_image_name> /bin/bash

our created image

assign name

→ ls

→ cd fmb/

→ ls

o/p → myfile { you will get all files back }

Docker file creation Using dockerfile

Dockerfile (Dockerfile) ← first capital

- Dockerfile is basically a test file it contains some set of instruction.

- Automation of Docker image creation

Docker Containerids

FROM (FROM) ← Capital All

- For base image. This Command must be on top of the dockerfile.

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 can't download file from internet and ~~only~~ 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 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 has higher priority over CMD. first commands will be execute by ENTRYPOINT only.

ENV

- Enviroment Variables

ARG

- To define the name of a parameter and its default value, difference between ENV and ARG is that after you set an env. using ARG, you will not be able to access that later on when you try to run the Docker container.

LAB

creation of Dockerfile

1) Create a file named Dockerfile
2) Add instructions in Dockerfile
3) Build dockerfile to create image
4) Run image to create container

→ vi Dockerfile

```
FROM ubuntu
RUN echo "Tech md ..." > /tomp/testfile
```

To create image out of dockerfile

→ docker build -t myimg . ← current directory

tagname

- check process state

→ docker ps -a

- see images

→ docker images

- To create container from the above image

→ docker run -it --name <container_name> <img_name> /bin/bash

↳ which we made

Ex. Command for next image creation

- No need to create new Dockerfile we will just update the Dockerfile.

→ vi Dockerfile

Code

```
FROM ubuntu
WORKDIR /tmp
RUN echo "Hello world" > /tmp/testfile
ENV myname Md Shoaib
COPY testfile /tmp
ADD test.tar.gz /tmp
```

- Open Dockerfile with vi remove all and inset this
- We have to create testfile, test, and a tar.gz (resp.) the file
- then build the image
- Then create the image

Docker Volume creation

Volume

- Volume is simply a directory inside our container.
- Firstly, we have to declare this directory as a Volume and then share Volume.

- Even if we 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 container.

- You can't create volume from existing container.

- You can share one volume across any number of containers.

- Volume will not be included when you update an image.

→ we can mapped volume in two ways -

1) Container ↔ container
2) Host ↔ Container

Benefits of Volume

- Decoupling Container from storage.
- Share Volume among different Containers.
- Attach Volume to Containers
- On deleting Container Volume does not delete.

Creating Volume from Dockerfile

- Create a Dockerfile and write

```
FROM ubuntu
VOLUME ["/myvolume1"]
```
code

- Then create image from this dockerfile

→ docker build -t myimage .
(. ← working directory)

- Now create a Container from this image & Run.

→ docker run -it --name container1 myimage /bin/bash
(container1 ← container_name; myimage ← container_Image)

- Now do ls, you can see myvolume1

- Now share Volume with another Container

Container ↔ Container

→ docker run -it --name <container_name> (New container) --Privileged=true
--volumes-from <container_name> ubuntu /bin/bash

(<container_name> ← old container; ubuntu ← OS image)

It will pull our default Volume created

- Now after creating Container2, myvolume1 is visible Whatever you do in one volume, Can see from other volume.

■ Now for checking ex:-

→ touch /myvolume1/Simplefile

→ docker start Container1

→ docker attach Container1

→ ls /myvolume1

- We can see Sample-file here

→ exit - Exit to the container

■ Now try to create Volume by using Command

→ docker run -it --name <container_name> -v

/volume2 <os_name> /bin/bash

vol. name, ex-name, Ex-ubuntu, ex. container3

- Do

→ ls

→ cd /volume2

■ Now create one more Container, and share Volume (2)

→ docker run -it --name <container_name> --privileged=true

--volumes-from <container_name> <os_name> /bin/bash

Ex-container4, Ex-container3, ubuntu

- Now you are inside Container; do ls, you can see volume2

- Now create one file inside this volume and then check in Containers, you can see that file.

Volumes (Host ↔ Container)

- Verify files in /home/ec2-user

- Create Volumes in host and Container and mapped them

→ docker run -it --name hostCont -v /home/ec2-user:/rajput --privileged=true ubuntu /bin/bash

(container name; host; mapped; volume name)

- Check

→ cd /rajput

- Do ls, now you can see all files of host M/c.

→ touch rajput (In container)

→ exit

- Now check in EC2, you can see them files.

Some other commands

- To see all created Volumes

→ docker volume ls

- To create docker Volume(normal)

→ docker volume create <vol_name>

- To delete Volume

→ docker volume rm <vol_name>

- To remove all unused docker Volumes

→ docker volume prune

- To get volume details

→ docker volume inspect <vol_name>

- To get container details

→ docker container inspect <cont

Docker port mapping, restart container, container and docker expose | Docker port expose

Docker port expose

- we have 0–65535 Logical ports.

Difference between docker attach and docker exec

- Docker `exec` creates a new process in the Container's enviroment while `docker attach` just connect the standard I/O of the main process inside the Container to corresponding standard I/O error of current terminal.

- `docker exec` is stecially for running new things in a already started container, be it a shell or some other process.

Difference between `expose` and `publish` a docker

- we have three options
  1) Neither specify `expose` nor `-p`
  2) Only specify `expose`
  3) Specify `expose` and `-p`

Explain

① If we specify neither `expose` nor `-p`, the service in the container will only be accessible from inside the container itself.

② If we `expose` a port, the service in the container is not accessible from outside docker, but from inside other docker container's, so this is good for inter-container communication.

③ If we `expose` and `-p` a port, the service in the container is accessible from anywhere, even outside docker.

Note

- If we do `-p` but ~~also~~ do not `expose`, docker does an implicit expose. This is because, if a port is open to the public, it is automatically also open to the other docker containers.

LAB

- Login into AWS account, create one linux instance.
- Now go to putty → Login as → ec2-user
  - → sudo su
  - → yum update -y
  - → yum install docker -y

→ service docker start

- Port mapping and run the container ex-techserver

→ docker run -td --name <container_name> -p 80:80 <os>

Host Container ex-ubuntu

-td: run and create but not will we go to the m/c (daemon)

-p: Port or Publish

80:80: Mapping

- Now to check processes (Running Containers)

→ docker ps

- To check open ports in container

→ docker port <container_name>

ex-techserver

- Execute the docker container

→ docker exec -it <container_name> /bin/bash

ex-techserver

To get insert container work like as attach container

- Update the ubuntu container

→ apt-get update

- Install apache for host as webserver

→ apt-get install apache2

- Create index.html for host

→ cd /var/www/html

→ echo "welcome MD" > index.html

When we exit or quit the container all mapped ports will be terminate/ unmapped.

- Start apach and surf web of the container ip

→ service apache2 start

Docker hub | Docker hub push and pull

Image creation and Docker Hub

- Go to AWS Account → select Amazon Linux
- Now go to putty → Login as → ec2-user

→ sudo su

→ yum update -y

→ yum install docker -y

→ service docker start

→ docker run -it ubuntu /bin/bash

- Now create some files inside Container
- Now create image of this Container

→ docker commit Container1 image1

(Container name, Image_name)

- Now create account in hub.docker.com
- Now go to EC2 Instance

→ docker login (Enter username password

- Now give tag to our image for identity

→ docker tag <image_name> dockerid/<Tag_name>

(ex- Image1, Ex- newImage)

- Now push to dockerhub

→ docker push dockerid/<Tag_name>

(Ex- NewImage)

- Now we can see this image in dockerhub account.
- Now create one instance in tokyo region and pull

image from hub (In instance install docker then the cmd)

→ docker pull dockerid/<Tag_name>
ex- newimage

- Now run

→ docker run -it --name <container_name> dockerid/<Tag_name> /bin/bash
ex-myCon ex-newImg
Image_name

■ Some Other Important Commands

- stop all running containers

→ docker stop $(docker ps -a -q)
bash script (delete in loop)

- Delete all stopped Containers

→ docker rm $(docker ps -a -q)

- Delete all images

→ docker rmi -f $(docker images -q)
forcefully

What is Ansible.

→ Ansible is an Open-Source IT-Configuration Management, Deployment and Orchestration Tool. It aims to provide Large Productivity gains to a wide Variety of Automation Challenges.

Ansible History

- Michael Dehan developed Ansible and the ansible project began in February 2012.

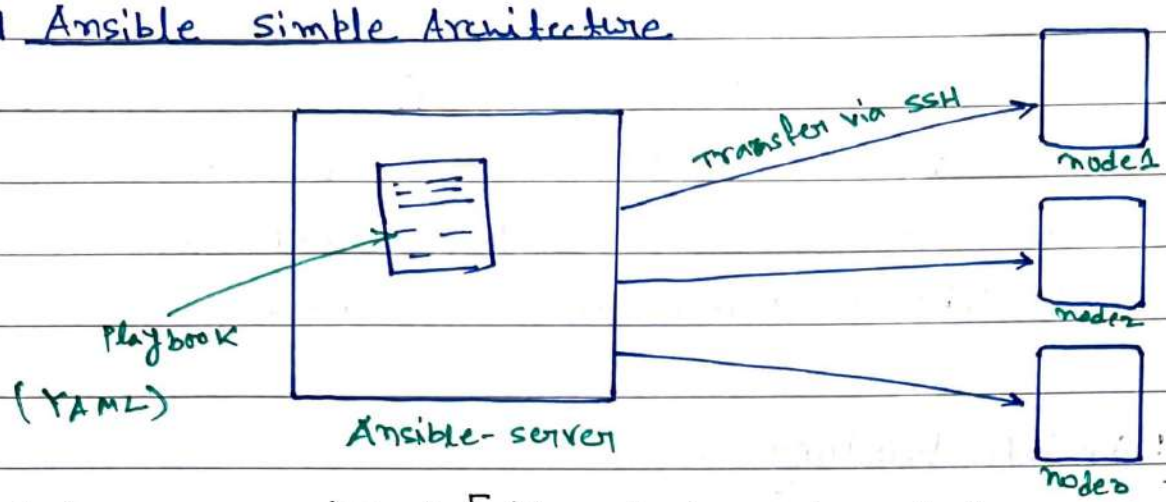
- RedHat acquired the Ansible tool in 2015.

- Ansible is available for RHEL, Debian, CentOS, Oracle.

- Ansible can use wheather your servers are in on-premises or in the cloud.

- It turns your Code into infrastructure i.e your computing environment has some of the same attributes as your application.

Ansible simple Architecture



* Ansible uses YAML [Yet Another Markup Language]

Ansible Advantages & Disadvantages

Advantages	Disadvantages
- Ansible is free to use by everyone	- Insufficient user Interface, though Ansible Tower is (GUI), but it is still in Development stage.
- Ansible is very Consistent and Lightweight and no Constraints regarding the O.S or Underlying h/w are present.	- Cannot achieve full automation
- Very secure due to its agentless Capabilities and Open SSH security feature	- New to the market, therfore limited support and document is not find much available.
- Does not need any special system admin. skills.	
- Push Mechanism	

Terms used in Ansible

Ansible Server - The machine where ansible is installed and from which all tasks and playbooks will be ran.

Module - Basically a module is a command or 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 organising 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 servers.

Play - Execution of a Playbook.

Handler - Task which is called only if a notifier is present.

Notifier - Section attributed to a task which calls a handler if the output is changed.

Playbook - It consist code in YAML format, which describes tasks to be executed.

Host - Nodes, which are automated by ansible.

Ansible inventory, Host pattern, establish SSH (Host↔N

LAB

Go to AWS account → Create 3 EC2 instances in same AZ and in Advance details

* Key Name → ansiblekey

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

- Take access of all machines via putty.

- Now go Ansible server (m/c) and download ansible package

→ wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rp

- Now do ls

→ [ec2-user@]# yum install epel-release-latest.7.noarch.rpm -y (Extra packages for enterprise)

→ [ ]# 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 hosts file inside ansible server and paste private-ip of Node1 & Node2.

→ [ ]# vi /etc/ansible/hosts ← create a group under ungrouped hosts then it

→ [demo]
grp_Name <private_ip_of_Node1>
" " " " 2>

- Now this hosts file is only working after updating ansible.cfg file.

→ [ ]# vi etc/ansible/ansible.cfg

- Uncomment this lines

remove Hash #inventory = /etc/ansible/hosts

remove Hash #sudo-user = root

- Now create one user, in all the three instances* Same

→ [ec2-user]# adduser ansible

- Now set password for this user

→ passwd ansible

→ xxxx....

* password - technical

↑ all same in other nodes

- Now switch as ansible user in all m/c

→ [ec2-user@ip]# su - ansible

→ [ansible@ip] $

- The ansible user don't have sudo privileges right now. if you want to give sudo priviledge to ansible user —

→ [ec2-user@ip]# visudo

outside ansible

↑ give priviteges in every machine

- Now go inside the file, under

Allow root to run any commands anywhere

root ALL=(ALL) ALL

ansible ALL=(ALL) NOPASSWD: ALL ← write this Line and save

- Now do this thing in all other nodes also.

Now go to Ansible server and try to install httpd package as a ansible user.

→ [ansible@ip]$ sudo yum install httpd -y

■ Now establish Connection between server and Node, Go to ansible-server.

→ [ansible@ip]$ ssh <private_ip_node1>

O/p → Permission Denied

- Now we have to do some changes in sshd_config file. go to ansible server

→ [root@ip...ec2-user]# vi /etc/ssh/sshd_config

- Now do some Changes and save the file

#Authentication:

:

remove hash → ~~#~~PermitRootLogin yes

:

To disable tunneled clear text passwords, change to no here!

remove hash → ~~#~~PasswordAuthentication yes

...

Put hash → # passwordAuthentication no

- we have to restart the sshd.config file then it will work

→ [root@ip...ec2-user]# service sshd restart

- Do this work in all other nodes also.

- Now verify in ansible server

→ [root@ip...ec2-user]# su - ansible

→ [ansible@ip]$ ssh <node1_private_ip>

- Now it ask for password, enter the password after that you will be inside Node 1

- Voila! we get the access ☺

■ Now we have to make Trust Relationship (using keys) and it only will work in same position

Root ⟷ Root
User ⟷ User

Root ⟷ User

- Now go to ansible server and create keys, Run this cmnds as ansible user.

→ [ansible@ip] $ ssh-keygen

→ [ansible@ip]$ ls -a

O/P → .ssh

→ [ansible@ip]$ cd .ssh/

→ [ansible@ip]# ls → public key that needs to be copied on the nodes

O/P → id_rsa (id_rsa.pub) known_hosts

- Now we need to copy public keys in both the nodes.

→ [ansible@ip].ssh]$ ssh-copy-id ansible@<node1 Private_ip>

ansible → username; Node Private-IP

Asking Last time Password ——

- same for other node

- Now verify, go to ansible server -

→ [ansible@ip .ssh]$ cd ..

→ [ansible@ip]$ ssh <node1_Private_ip>

↑ Now you will enter into node1 without password.

Host Patterns

→ "all" pattern refers to all the machines in an inventory.

→ ansible all --list-hosts → preview all node list

→ ansible <groupname> --list-hosts → Preview group node list

→ ansible <groupname>[ ] --list-hosts → specific node list

groupname[0] → picks first machine of group
groupname[1] → " second " " "
groupname[-1] → " last " " "
groupname[0:1] → " first-two machines in the group
groupname[2:5] → " 3, 4, & 5 & 6 machine in the group.

- Group seperated by a Colon can be used to use hosts from multible groups.
  → groupname1:groupname2
  ex) demo[1:5]:devops[2:10]

Ad-hoc Commands, Modules & Playbook

Note

1) Ad-hoc Commands (simple Linux) No Idempotency
2) Modules → single work
3) Playbooks → More than one Module

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, because these Commands are of one time usage.

- The ansible ad-hoc commands uses the /usr/bin/ansible Command Line tool to automate a single task

* Go to ansible-server

→ [ansible@ip] $ ansible demo -a "ls"

→ [ansible@ip] $ ansible demo[0] -a "touch file2"

→ [ansible@ip] $ ansible all -a "touch file4"

→ [ansible@ip] $ ansible demo -a "ls -al"

→ [ansible@ip] $ ansible demo -a "sudo yum install httpd -y"

or,

For sudo privilege

ansible demo -ba "yum install httpd -y"

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

Ansible Modules

module library

→ Ansible ships with a number of modules (called 'module lib') 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 for the inventory file is

/etc/ansible/hosts

Cmnds for Modules

group

[ansible@ib]$ 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"

* install = Present

Uninstall = absent

Update = latest

Ansible Modules setup

This module checks the Idempotency of nodes

→ [ansible@ib]$ ansible demo -m setup

→ [ansible@ib]$ ansible demo -m setup -a "filter=*ipv4*

filter partial line

Ansible Playbook, Handlers, Variables, Dry run and Looks

YAML

→ Playbook in ansible are written in YAML Format.

- It is human readable data serialization language, it is commonly used for configuration files.

- Playbook is like a file where you write codes. Consist of Vars, tasks, handlers, files, templates and roles.

- Each Playbook is composed of one or more 'modules' in a list. Modules is a Collection of Configuration files.

- Playbooks are divided into many sections like-

  Target section :- Defines the host against which Playbooks task has to be executed

  Variable section :- Define Variables

  Task section :- List of all 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 a dictionary.

- All ~~item~~ YAML files have to begin with "---" and end with "..." ← mandatory

- All members of a list lines must begin with same indentation level starting with "-" for eg →

```
--- # A list of fruits
Fruits:
  - Mango
  - Strawberry
  - Apple
  - Banana
...
```

- A dictionary is Represented in a Simple Key: Value form.

for eg:-

```
---# Detail of Customer
Customer:
  name: Shoaib
  Job: Trainer
  Skills: Ansible
  Exp: 5
```

→ Extension for Playbook files is .yml

Note:- There should be space between : _ value

LAB

- Go to Ansible-server
- Now create one Playbook -

→ [ansible@ip]$ vi target.yml

Code

```
---# Target Playbook
- hosts: demo          → group name
  User: ansible
  become: yes          → sudo privilege
  Connection: ssh
  gather_facts: yes    → setup (shows info)
```

esc → :wq!

- Now, to execute the Playbook

→ [ansible@ip] $ ansible-playbook target.yml

- Now create one more Playbook in ansible-server

→ [ansible@ip] $ vi task.yml

```
---# Target and Task Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: Install HTTPD on Centos 7
      action: yum name=httpd state=installed
```

esc → :wq!

- Now execute this Playbook

→ [ansible@ip] $ ansible-playbook task.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 & use it later.

- Now go to ansible server and create one Playbook,

→ [ansible @ ip] $ vi vars.yml

```
---  # My Variable Playbook
- hosts: demo
  user: ansible
  become: yes
  Connection: ssh
  vars:
      pkgname: httpd
  tasks:
      - name: install HTTPD server on ..7
        action: yum name='{{pkgname}}'
                state=present
```

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 wheather the playbook is formatted correctly.

→ ansible-Playbook handlers.yml --check

- Go to ansible server

→ [ansible@ip] vi handles.yml

```
---  # Handlers Playbook
- hosts: demo
  user: ansible
  become: yes
  Connection: ssh
  tasks:
    - name: install httpd server on centos
      action: yum name=httpd state=installed
      notify: restart httpd
  handlers:
    - name: restart httpd
      action: service name=httpd state=restarted
```

Loops

→ Sometimes you want to repeat a task multiple times. In computer programming, this is called 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 –

→ [ansible@ib]$ vi loops.yml

```
---
# My Loops Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: Add list of users in my nodes
      user: name='{{item}}' state=present
      with_items:
        - shoaib
        - sachin
        - simran
        - simmi
```

esc → :wq!

[ansible@ib]$ ansible-playbook loops.yml

*verify →

[ec2 ...] cat /etc/passwd

What is Conditions, Ansible Vault and Ansible Roles

Conditions

→ Whenever we have different different scenarios, we put conditions according to the scenario.

When statement

Sometimes you want to skip a particular command on a particular node.

```
---# Condition Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: Install apache server for debian family
      command: apt-get -y install apache2
      when: ansible_os_family == "Debian"
    - name: install apache server for redhat
      command: yum -y install httpd
      when: ansible_os_family == "RedHat"
```

Vault

→ Ansible allows keeping sensitive data such as Passwords or keys in encrypted files, rather than a plain-text 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 an existing Playbook
  → ansible-vault encrypt target.yml

- To Decrypt an Encrypted Playbook
  → ansible-vault decrypt target.yml

Roles

Default → It stores the data about role/application Default Variables eg → If you want to run to port 80 or 8080 then variables needs to define in this path.

Files → It contains files need to be transfered to the remote VM (static files)

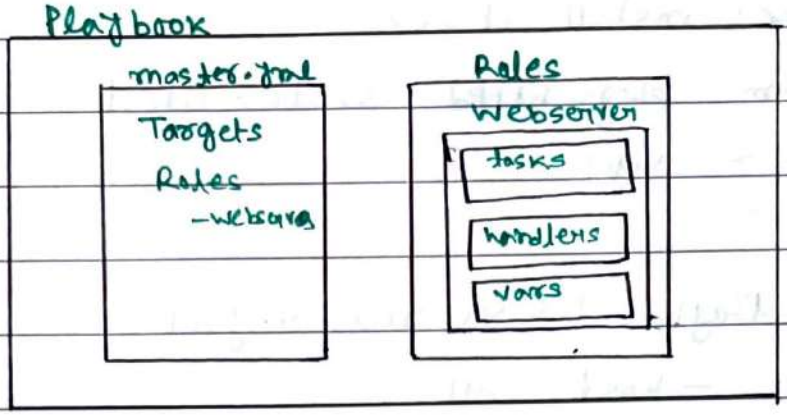
Handlers → They are triggers or task we can segregate all the handlers required in Playbook.

Meta → This directory contain files that establish roles Dependencies, eg → Author Name, Supported Platform, Dependencies if any.

Tasks → It contains all the tasks that is normally in the Playbook. eg → Installing Packages and copies files etc.

Vars → Variables for the role can be Specified in this directory and used in your configuration files Both Vars and default stores variables.

• Basic Architecture of Roles



LAB

→ [ansible@ip]$ mkdir -p Playbook/roles/webserver/tasks

→ [ansible@ip]$ tree

```
O/P → Playbook
        └→ roles
             └→ Webservers
                  └→ tasks
```

→ [ansible@ip]$ cd Playbook/

→ [ansible@ip]$ tree

```
O/P → .
      └→ roles
           └→ Webserver
                └→ tasks
```

→ [ansible@ip Playbook]$ touch roles/webserver/tasks/main.yml

→ [ansible@ip..Playbook]$ touch master.yml

→ [ansible@ip... Playbook]$ vi roles/webserver/tasks/main.yml

- Inside main.yml

```
- name: install apache
  yum: pkg=httpd state=latest
```

esc → :wq!

→ [ansible@ip...Playbook] $ vi master.yml

```
--- # master playbook for server
- hosts: all
  user: ansible
  become: yes
  connection: ssh
  roles:
    - webserver
```

→ Target for Nodes

→ [ansible@ip... Playbook] ansible-Playbook master.yml

■ Important points for roles

- We can use two techniques for reusing a set of tasks:- includes and roles

- Roles are good for organising tasks and encapsulating data needed to accomplish those task.

- We can Organize Playbooks into a directory structure called roles.

- Adding more and more functionality to the playbooks will make it difficult to maintain in a single file.

What is CI/CD Pipeline | What is Jenkins

Jenkins

→ Jenkins is an Open-Source project written in Java that runs on Windows, MacOs and other Unix-like Operating systems. It is free, community supported and might be your first choice tool for CI.

- Jenkins automate the entire software Development Life cycle.

- Jenkins was originally developed by Sun Microsystem in 2004 under the name hudson.

- The project was later named Jenkins when Oracle bought Microsystems.

- It Can run on any major platform without any Compatibility issues.

- Wherever developers write code, we integrate all that code of all developers at that point of time and we build, test and deliver/deploy to the client. This process is called CI/CD.

- Because of CI, bugs will be reported fast and get rectified fast. so the entire software development happens fast.

Workflow of Jenkins

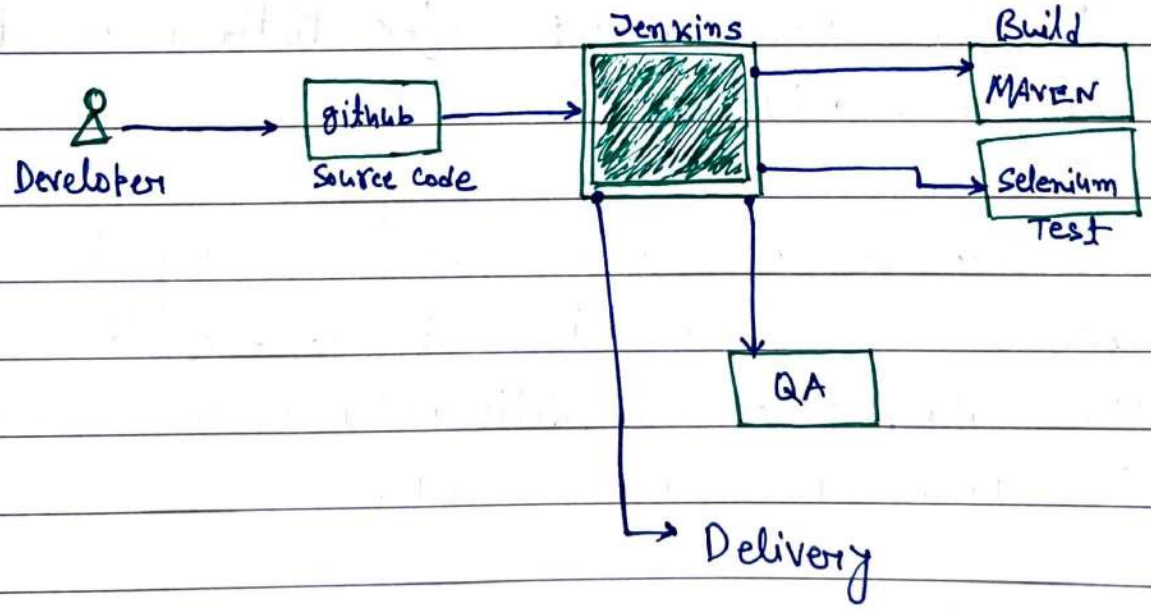
→ We can attach git, Maven, Selenium and Artifactory plugins to Jenkins.

- Once developers puts code in github, Jenkins pull that code & send to Maven for build.

- Once build is done, jenkins pull 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.

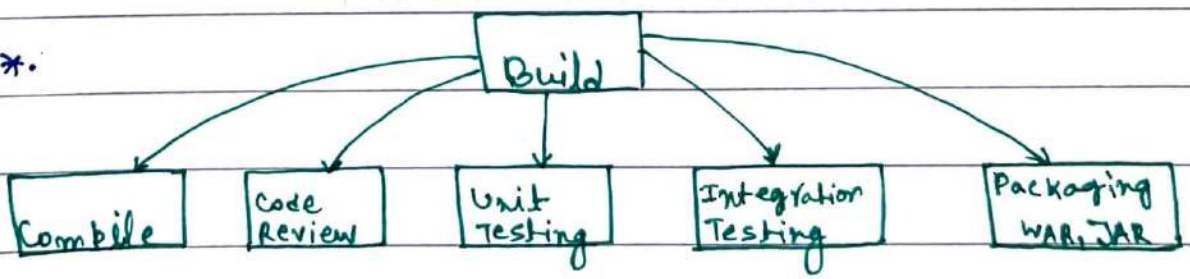
- We can also deploy with jenkins.



Advantages of Jenkins.

- It has lot of plug-ins available.
- You can write your own plug-in.
- You can use Community plug-in.
- Jenkins is not just a tool. It is a framework. i.e. You can do whatever you want. All you need is plug-ins.
- We can attach slaves (nodes) to Jenkins master. It instruct other (slaves) to do job. If slaves are not available, Jenkins itself does the job.
- Jenkins also behave as Cron-server Replacement. i.e- Can do Scheduled task.
- It Can Create Labels.

*.



CI/CD Pipeline Project | Jenkins

Git Installation | steps for inst. git, java, maven, and jenkins on Local M/c

→ Go to Google Chrome → search 'git'
download → Download 2.31.1 for win.

- Open the download file
Preamble → C:\Programfiles\git → select
Components → select start Menu folder
→ Choosing the default editor Vim
→ ⊙ Let git decide → git from the command line and also from the 3rd Party software → Use the OpenSSL Library.

→ Checkout as-is, Commit Unix-style line endings → Use MinTTY → Choose default behaviour → git Credentials Manager Core.
→ ☑ enable file system Caching → install

- Go to Command Prompt in Laptop →
→ git Config --global User.name "<name>"
→ git Config --global User.email "<mail_id>"
→ git Config --global --list

Download and Installation of JDK

→ Go to Chrome → search "Java Development kit Download" → Download Java SE Development kit 16 for win x64.

→ Run and follow the steps to install
Now, go to C drive → Program files → Java → JDK16.
Select path and Copy it.

→ Search 'Edit system Environment variables" in laptop
→ go to user variables → New
Variable Name → JAVA_HOME
Variable Value → Paste the path here

→ Now go to system variable → New
Variable Name → JAVA_HOME
Variable Value → Paste the path here

→ Now go to inside programfiles → bin → paste the path.

→ Again go to "Edit system Environment Variables"
→ system Variable → Path → New → Paste Path
Now verify in command Prompt.

```
C:\Users\home\ echo %JAVA_HOME%
```

o/p → C:\Programfiles\Java\JDK16

MAVEN Download and Configure

→ Go to Chrome → search Maven.apache.org

→ download → Binary zip archive

extract files → c:\DevTools

go to c:\ → DevTools → Apache-Maven → Copy the path.

→ Now search "Edit system Environment Variables"

→ system Variable → New

Variable Name → M2_HOME

Variable Name → Paste the path

→ Now go inside Apache Maven folder → bin → Copy path.

→ Now again go to "Environment variables"

→ system Variable → Path → New → Paste Path.

→ Now open CMD

c:\Users\home> mvn -version

c:\Users\home> echo %M2_HOME%

→ Restart Laptop

Jenkins Download and Installation

→ Go to google Chrome → Jenkins.io → Download
→ select LTS → windows → download.

→ open download file → Run and install
→ After installation it automatically open as localhost:8080

→ Unlock the page by wing password

→ Now install suggested Plugins
→ ask for username & password
→ Username - admin
→ Password — admin123
→ email — <Youremail>
→ Save & continue...

Plugins

- Plugins are small Libraries that add new abilities to jenkins and can provide integration points to other tools.

Maven Job, Schedule Project and Source Code Polling

→ Go to Google Chrome → localhost:8080 → log in

→ Go to Manage jenkins on left side of jenkins dashboard→
Manage Plugins → Available → select Maven Integration
& Green balls → Install without restarrt.

Go to New item → Maven Project

Now go to Manage jenkins → Global tool Configuration
go to add JDK.

☐ Uncheck this install Automatically Option

NAME → JAVA

JAVA_HOME → C:\Programfiles\Java\JDK

Now go to MAVEN

Name _ MAVEN

MAVEN_HOME → C:\Dev Tools\aPache-Maven

■ Maven Project (By Maven)

goto https://github.com/technicalguftgu/time-tracker

→ click on time-tracker repo.

→ "Fork" to Copy this repo

→ sign-in into your github account

→ Click on time-tracker repo

→ clone

→ go to c drive

→ git clone <url_of_time-tracker_repo>

→ cd time-tracker

→ c:\time-tracker > ~~mvn~~ mvn clean package

Maven Project (By Jenkins)

Now go to Jenkins → New item → Entername
→ MyMavenProject

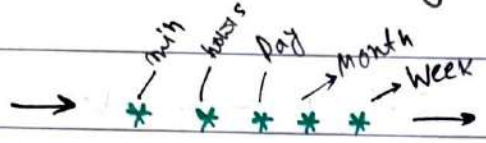
- Then select Maven Project → ok
- Source code Management → git ⊙
→ Repository URL

- Build Option → Root POM → Pom.xml
- Goals & options → clean package → save

- go to Jenkins home page → click on
MyMavenProject → Build Now

Scheduled Project

Click on any Project → Configure → build triggers
→ build Periodically.



→ Can see automatic. Builds after every 1 min.

→ You can manually trigger build as well.

Source Code Polling (Poll SCM)

→ Now go to Jenkins Homepage
→ go to MyMavenProject → Configure

Now go to build trigger

☑ Poll SCM
Schedule * * * * * → save

Now go to github account → do some changes in Readme.md →
Commit changes
You can see, after 1 min, it build automatically.

What is 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).

→ Maven can build any number of projects into desired output such as .jar, .war, metadata.

- Mostly used for Java based Projects.
- It was initially released on 13 July 2004. based on Ja
- Meaning of Maven 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

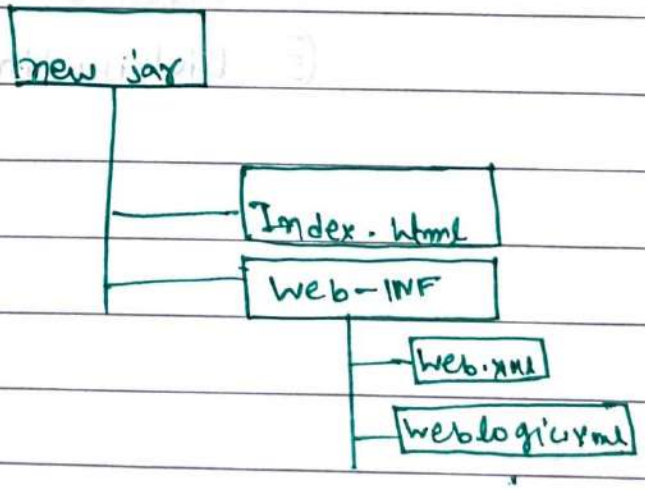
Java → Ant, Maven, gradle

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 jar also.

(2) Creating the right Project structure → we must create the right project structure in servlet, struts, etc. otherwise it will not be executed.

fog eg →



③ Building and deploying the project → We must have to build and deploy the project so that it may work.

What MAVEN does ?

- It makes a project easy to build.
- It provides Project Information. (for eg → log docs, cross reference sources, mailing list, dependency list, unit test)
- Easy to add new dependencies.

Therefore Apache Maven helps to Manage :-

① Build
② Dependencies
③ Reports
④ Releases
⑤ Distribution

What is Build tool ?

→ A build tool take care 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 Reko, server repo or Control Repository.

POM (Project Object Model)

→ POM refers to the XML files that have all the information regarding project and configuration details.

- Main Configuration file is POM.xml
- It has the description of the project, details regarding the versioning and configuration management of the project.
- The xml file is in the project home directory.

Pom.xml Contains

- Metadata
- Dependencies
- kind of project
- kind of output (.jar, .war)
- Description

One Project → One Workspace → one Pom.xml

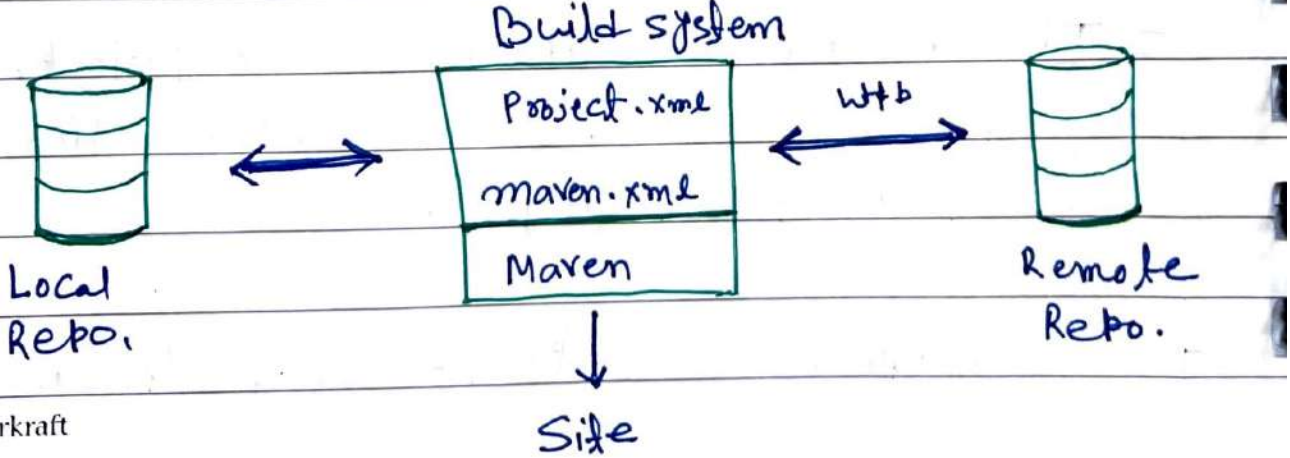
Requirements for Build

→ Source code (Present in Workspace)

→ Compiler (Remote repo → local repo → Workspace)

→ Dependencies (Remote repo → local repo → Workspace)

■ Architecture of Maven



■ Maven Build Life-cycle

Goals:-

1. Generate Resource (Dependencies)
2. Compile Code
3. Unit test
4. Package (Build)
5. Install (into local repo & artifactory)
6. Deploy (to serve)
7. Clean (delete all run time files)

eg- mvn install

mvn clean package

* 1 - 6 Default & sequence order.

7 → Not default & it won't allow sequence.

→ Build lifecycle consist 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 information of the project structure in build.xml file.

- Ant is procedural, you need to provide info about what to do and when to do through code.
- There is no lifecycle in Ant.
- It is a tool box.
- It is mainly a build tool.
- It is less prefered than Maven.

Vs.

Maven → Maven has a convention to place source code, Compiled code etc. So we don't need to provide information about the project structure in pom.xml

- Maven is declarative, everything you define in the pom.xml file.
- There is a lifecycle in Maven.
- It is mainly a project Management tool.
- It is a framework.

Nagios Continuous Monitoring tool

- **Nagios** → Nagios is an Open-source software for Continuous Monitoring of systems, networks and infrastructure. It runs plugins stored on a server which is connected with a host or another server on your 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 Glastad developed it as a part of NetSaint distribution.

- 2002, Ethan renames the project to "Nagios" because of trademark issues with the name "Netsaint."

- 2009, Nagios releases its first Commercial Version, Nagios XI.

- In 2012, Nagios again renamed as Nagios Core.

- It uses port number 5666, 5667, 5668 to monitor its client.

■ Why Nagios ?

→ Detect all types of Network, or server issues.

- Helps you to find the root cause of the problem which allows you to get the permanent solution to the problem.

- Reduce downtime

- Active Monitoring of entire infrastructure.

- Allow you to monitor and troubleshoot server performance issues.

- Automatically fix problem.

■ Features of Nagios

→ Oldest and Latest

- Good log and database system

- Informative and attractive web Interface.

- Automatically send alert if condition changes.
- Helps you to detect network errors or server crashes.
- You can monitor the entire business process and IT infrastructure with a single pass.
- Monitor network services like http, smtp, ftp, snmp, ssh, pop, DNS, LDAP, IPMI etc.

Phase of Continuous Monitoring

1. Define → develop a monitoring strategy.
2. Establish → How frequently you are going to monitor it.
3. Implement
4. Analyze data and Report finding
5. Respond
6. Review and Update

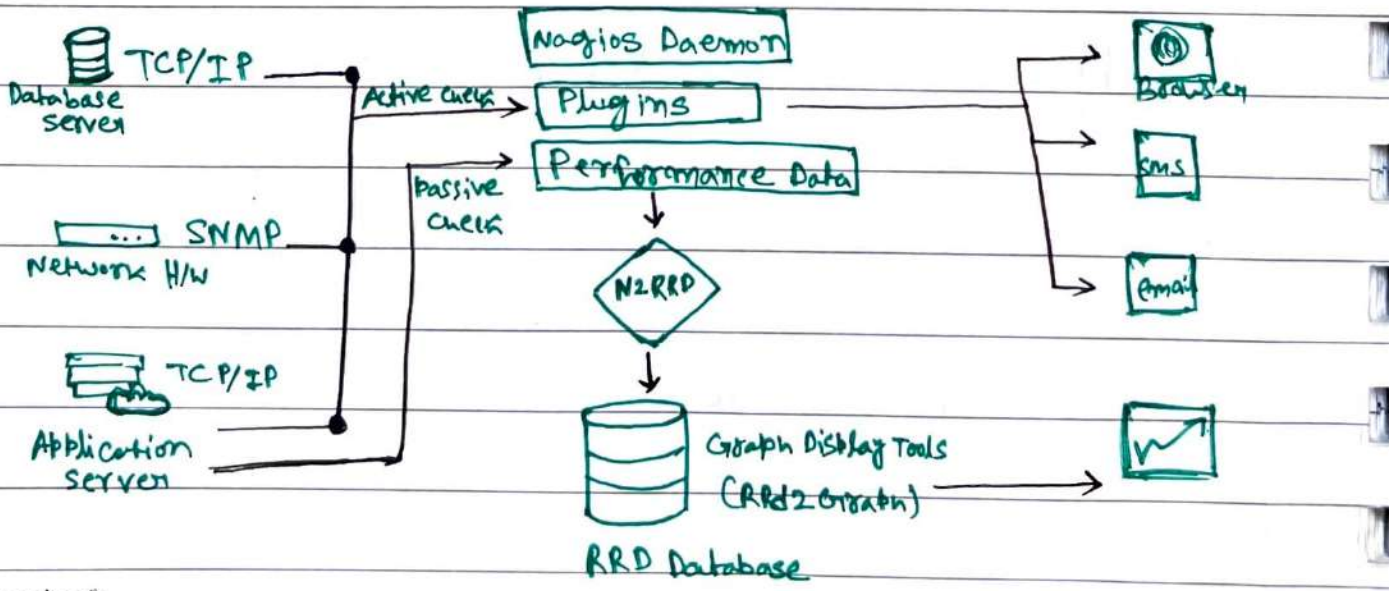
Nagios Architecture

→ Nagios 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 you monitor.

How does it works?

→ Mention all details in Configuration files.

- Daemon read those details what data to be collected.
- Daemon uses NRPE (Nagios Remote Plugin Executor) Plugin to Collect data from nodes and ~~their~~ store in its own database.
- Finally shows everything in dashboard.



■ Pre-requisite

→ httpd (Browser)
→ php (dashboard)
→ gcc & gd (Compile, to Convert raw code into binaries)
→ make file (to build)
→ Perl (script)

→ Main Configuration file —
/usr/local/nagios/etc/nagios.cfg

→ All monitoring things called as 'service'.

For eg → 5 servers → 4 checks each you have to monitor → 5 × 4 = 20 services.

■ Dashboard Overview

In dashboard, you can see →

Host → down
unreachable
up
recovery
none

Services → Warning, Unknown, critical, recovery, Pending.

Nagios installation on AWS Linux Machine

→ To start Nagios Core installation you must have your EC2 instance up and run and have already configured SSH access to the instance.

http → All traffic

Step-1 → Install Pre-requisite softwares on your EC2 machine. Prior to Nagios installation like Apache, PHP, gcc compiler and gd development libraries.

→ sudo su
→ yum install httpd php
→ yum install gcc glibc glibc-common
→ yum install gd gd-devel

Step-2 → Create account information you need to setup a nagios user. Run the following cmds.

→ adduser -m nagios
→ passwd nagios

- Now, it ask to enter new password give '12345' as password.

→ groupadd nagioscmd
→ usermod -a -G nagioscmd nagios
→ usermod -a -G nagioscmd apache

step-3 → Download nagios core and the Plugins. Create a directory for storing the downloads.

→ mkdir ~/downloads

→ cd ~/downloads

- Download the source code tarballs of both nagios and the nagios plugins -

→ wget http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-4.0.8.tar.gz

→ wget http://nagios-plugins.org/download/~~nagios-plugins.org/download~~/nagios-plugins-2.0.3.tar.gz

step-4 → Compile and install Nagios extract the nagios source code tarball.

→ tar zxvf nagios-4.0.8.tar.gz

→ cd nagios-4.0.8

- Run the configuration script with the name of the group which you have created in above step.

→ ./Configure --with-command-group=nagios cmd

- Compile the Nagios source code

→ make all

- Install Binaries, init script, sample config files and set permissions on the external command directly.

→ make install
→ make install-init (To compile init script)
→ make install-config
→ make install-commandmode

Step-5 → Configure the web interface

→ make install-webconf

Step-6 → Create a 'nagiosadmin' account for login into the nagios web interface. Set password as well.

→ htpasswd -c /usr/local/nagios/etc/htpasswd.users ␣ nagiosadmin

- Asking for password, set a new password -
- Restart httpd service

→ service httpd restart

Step-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 the plugins.

→ ./configure --with-nagios-user=nagios \
--with-nagios-group=nagios

→ make ← to compile
→ make install

Step-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

Step-9 → Verify the sample Nagios Configuration files

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

- if there are no errors, start nagios.

→ service nagios start
→ service httpd restart

Step-10 → Copy Public IP of ec2 instance and paste in google chrome, in given way

for eg → 20.1*1.1/nagios/ → instance public ip

ask for username → nagiosadmin
password → 12345

What is Kubernetes (Kubernetes Introduction)

→ Kubernetes is an Open-source Container Management tool which automates Container deployment, Container scaling & Load balancing.

- It schedules, runs and manages isolated Containers which are running on Virtual/Physical/Cloud Machines.

- All fob cloud providers support Kubernets.

- one fopular name for Kubernetes is **K8s**.

History

→ Google developed an internal system Called 'borg' (later named as Omega) to deploy and manage thousands google application and services on their cluster.

- In 2014, google introduced Kubernetes on Open Source Platform written in 'Golang' and later donated to CNCF.

Online Platform for K8s

→ Kubernetes Playground
→ Play with k8s
→ Play with Kubernetes Classroom

■ Cloud based K8s Services

→ GKE - [Grogle Kubernets Engine]
→ AKS - [Azure Kubernetes service]
→ Amazon EKS - [Amazon Elastic Kubernetes service]

■ Kubernetes Installation tool

→ Minikube
→ kubeadm

■ Problems with Scaling up the Containers

→ Containers Cannot Communicate with each other.

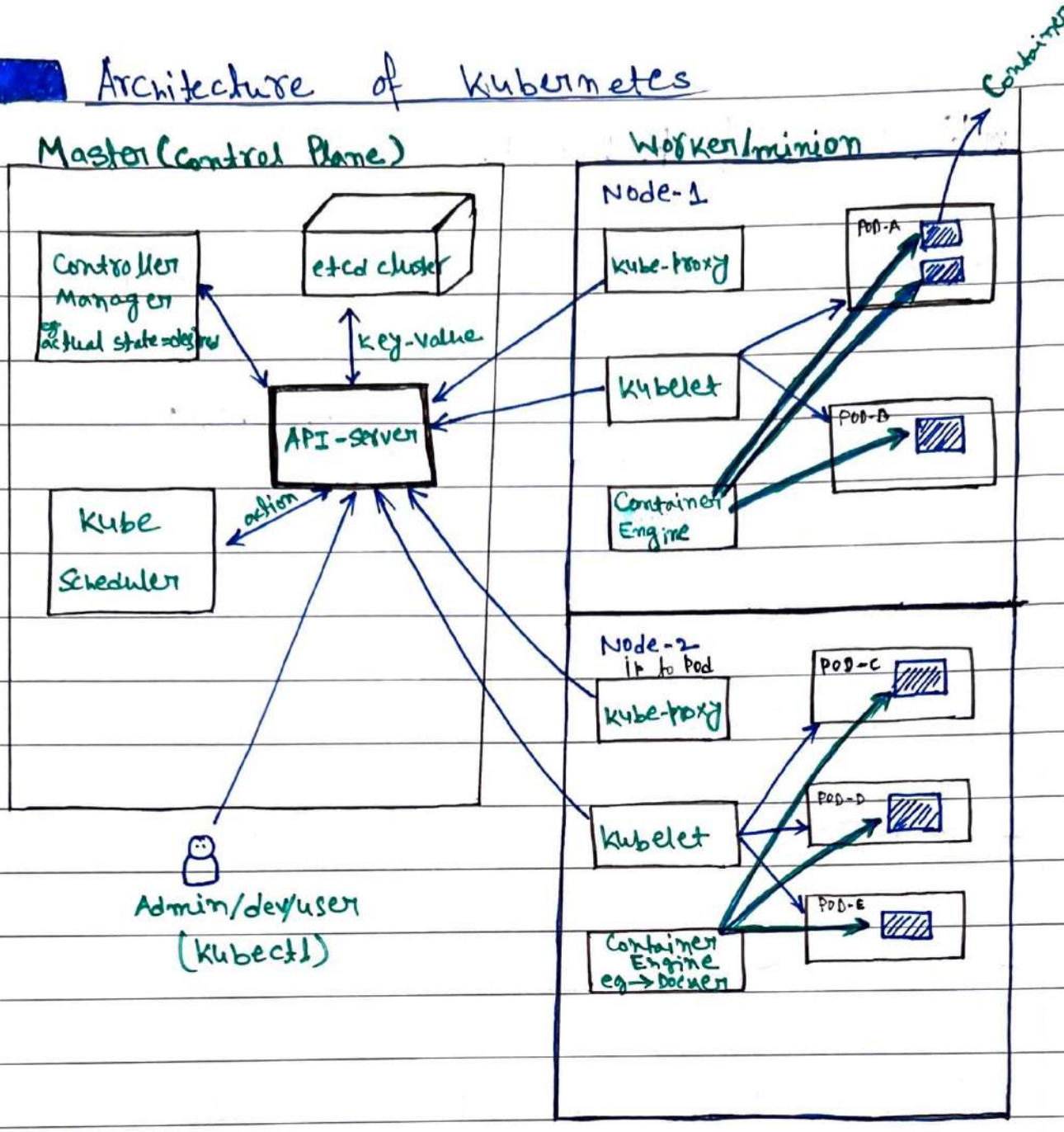
- Autoscaling and Load Balancing was not possible.
- Containers had to be managed Carefully.

Features of Kubernetes

→ Orchestration (clustering of any no. of Containers running on different n/w.)

- Autoscaling (Vertical & Horizontal) → more prefer
- Auto-Healing
- Load Balancing
- Platform Independent (cloud / virtual / physical)
- Fault Tolerance (Node/POD failure)
- Rollback (going back to previous Version)
- Health Monitoring of Containers
- Batch Execution (One time, sequential, parallel)

Architecture of Kubernetes



Working with Kubernetes

- We create Manifest (.yml)
- Apply this to cluster (to master) to bring into desired state.
- Pod runs on node, which is controlled by master.

■ Role of Master Node

→ Kubernetes cluster contains Containers running or Bare Metal/VM instances/cloud instances/all mix.

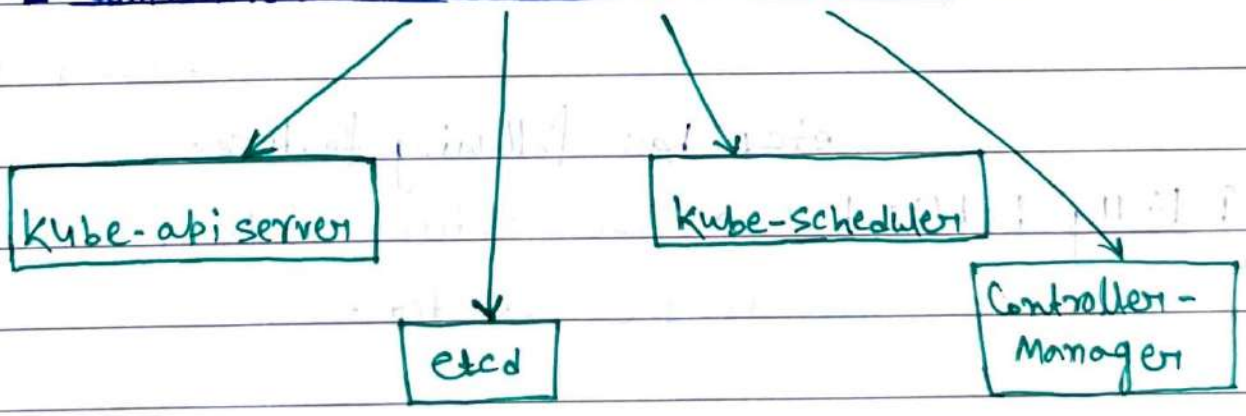
- Kubernetes designates one or more of these as master and all others as Workers.

- The master is now going to run set of K8s processes. These processes will ensure smooth functioning of cluster. These processes are called "Control Plane".

- Can be Multi-master for high availability.

- Master runs Control Plane to run cluster smoothly.

■ Components of Control Plane (Master)



Kube-api-server → (For all communications)

→ This api-server interacts directly with user (i.e we apply .yml or .json manifest to kube-api-server)

- This kube-api-server is meant to scale automatically as per load.

- Kube api-server is front-end of control-plane.

etcd →

→ stores metadata and status of cluster.

- etcd is consistent and high-available store (key-value store)

- Source of truth for cluster state (info about state of cluster)

etcd has following features

① Fully Replicated → The entire state is available on every node in the cluster.

② Secure → Implements automatic TLS with optional client-certificate authentication.

③ Fast → Benchmarked at 10,000 writes per second

Kube-schedular (action)

→ When users make request for the creation & Management of Pods, kube-schedular is going to take action on these requests.

- Handles Pod creation and Management
- kube-schedular match/assign any node to create and run pods.
- A schedular watches for newly created Pods that have no node assigned. For every Pod that the schedular discovers, the schedular becomes responsible for finding best node for that Pod to run on
- Schedular gets the information for hardware configuration from configuration files and schedules the Pods on nodes accordingly.

Controller-manager →

→ Make sure actual state of cluster matches to desired state.

Two possible choices for Controller Manager —

① If K8s on cloud, then it will be cloud-controller-manager

② If K8s on non-cloud, then it will be Kube-controller-Manager

■ Components on master that runs Controller

Node Controller → For checking the cloud provider to determine if a node has been detected in the cloud after it stops responding.

Route-Controller → Responsible for setting up network, routes on your cloud.

Service-Controller → Responsible for load Balancers on your cloud againest services of type load Balancer.

Volume-Controller → For creating, attaching and mounting Volumes and interacting with the Cloud Provider to Orchestrate Volume.

Nodes (Kubelet and Container Engine)

→ Node is going to run 3 important piece of software / process.

Kubelet

→ Agent running on the node

- Listens to kubernetes master (eg- Pod creation request)

- Use Port 10255.

- Send success/Fail reports to master.

Container Engine (Docker)

→ Works with Kubelet

- Pulling images

- start/stop Containers

- Exposing Containers on ports specified in manifest.

Kube-Proxy

→ Assign IP to each Pod.

- It is required to assign IP addresses to Pods (dynamic)
- Kube-Proxy runs on each node & this make sure that each Pod will get its own unique IP address.
- This 3 Components Collectively Consist "node".

POD

→ smallest unit in Kubernetes.

- POD is a group of one or more Containers that are deployed together on the same host.
- A cluster is a group of nodes.
- A cluster has atleast one worker node and master node.

- In Kubernetes, the Control unit is the POD, not Containers.

- Consist of one or more tightly coupled Containers.

- POD runs on node, which is Control by Master.

- Kubernetes only knows about PODs (does not know about individual Container)

- Cannot start Containers without a POD.

- One POD usually contains One Container.

Multi Container Pods →

→ share access to memory space.

- Connect to each other using Localhost:<Container-Port>

- Share access to the Same Volume.

- Containers within POD are deployed in an all-or-nothing manner.

- Entire POD is hosted on the same node (scheduler will decide about which node)

POD Limitations

→ No auto-healing or scaling by default.

→ POD crashes

Higher level Kubernetes Objects

Replication set → auto Scaling and auto healing

Deployment → Versioning and Rollback

Service → Static (Non-ephemeral) IP and Networking

Volume → Non-ephemeral Storage.
→ storage outside the node.

Important

kubectl → Single Cloud

kubeadm → On Premise

kubefed → Federated

Setup Kubernetes Master and Worker node on AWS

→ Login into AWS account → Launch 3 Instance → Ubuntu 16.04 (t2.medium).
Master must have 2 vCPU and 4 GB RAM

→ Now using puttygen, create Private key and save.

→ Access all the 3 instances (1 Master, 2 Nodes) using putty.

Commands Common for Master & Node

→ sudo su
→ apt-get update
- Now install https package
→ apt-get install apt-transport-https
- This https is needed for intra cluster communication. (Particularly from Control Plane to industrial Pods).

- Now Install docker on all 3 instances
→ sudo apt install docker.io -y

- To check, wheather docker is installed or not.
→ docker --version
→ systemctl start docker
→ Systemctl enable docker

- Setup Open GPG-key. This is required for intra cluster Communication. It will be added to source key on this node. i.e when K8s sends signed info to our host, it is going to accept those information because this Open GPG key is present in the source key.

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

- Edit Sources list file (apt-get install nano)

→ nano /etc/apt/sources.list.d/kubernetes.list

→ deb http://apt.kubernetes.io/ kubernetes-xenial main
(← space)

exit from nano and Save.

→ apt-get update

→ apt-get install -y kubelet kubeadm kubectl kubernetes-cni

Bootstrapping the Master Node (In Master)

- To initialize K8s cluster

→ kubeadm init

- You will get one long command started from "kubeadm join 172.31.6.165:6443 ....

Copy this Command and Save on notepad

- Create both .kube and its parent directories (-p)

→ mkdir -p $HOME/.kube

- Copy configuration to kube directory (in config file)

→ sudo cp -i /etc/kubernetes/admin.conf
$HOME/.kube/config
(No space)

- Provide user permissions to Config file

→ chown $(id -u):$(id -g) $HOME/.kube/config
(space) (space)

- Deploy flannel node network for its repository path. Flannel is going to place a binary in each node.

→ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

→ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifest/kube-flannel-rbac.yml

- Configure worker node

→ Paste long command in both the nodes.

[go to master]

→ kubectl get nodes

Installation of Minikube and Detailed LAB

Kubernetes Objects

→ Kubernetes uses objects to represent the state of your cluster.

- What Containerized applications are running (and which node)
- The policies around how those applications behave, such as restart policies, upgrades and fault tolerance

- Once you create the object, the Kubernetes system will constantly work to ensure that object exist and maintain's cluster's desired state.

- Every kubernetes object includes two nested fields that govern the object Config. The object spec and Object Status.
- The spec, which we provide, describes your desired state for the Object - the characteristics that you want the object to have.

- The Status describes the actual state of the object and is supplied and Updated by the Kubernetes system.

- All objects are identified by a Unique name and a UID.

→ The Basic Kubernetes Objects include.

1. Pod
2. Service
3. Volume
4. Namespace
5. Replicasets
6. secrets
7. Configmaps
8. Deployments
9. Jobs
10. Daemonsets

■ Relationship b/w these Objects

- Pod manages Containers.

- Replicaset manage Pods.

- services expose Pod processes to the outside world.

- Configmaps and secrets helps you Configure Pods.

Kubernetes Object

→ It represents as JSON or YAML files.

- You create these and then push them to the Kubernetes API with Kubectl.

State of the Object

→ Replicas (2/2)
→ Image (Tomcat/ubuntu)
→ Name
→ Port
→ Volume
→ Startup
→ Detached (default)

■ Kubernetes Objects Management

→ The kubectl command Line tool supports several different ways to create and manage Kubernetes Object.

Management Technique	Operates on	Recommended Environment
Imperative cmds	Live objects	Development projects
Declarative object Configuration	Individual files (yml/json)	Production

- Declarative, is about describing what you are trying to achieve, without instructing how to do it.

- Imperative, explicitly tells "how to accomplish it".

■ Fundamental of Pods

→ When a Pod gets created, it is scheduled to run on a node in your cluster.

→ The Pod remains on that node until the process is terminated, the Pod object is deleted, the Pod is evicted for lack of resources, or the node fails.

→ If a Pod is scheduled to a node that fails, or if the scheduling operation itself fails, the Pod is deleted.

→ If a node dies, the Pods scheduled to that node are scheduled for deletion after a timeout period.

→ A given Pod (UID) is not "rescheduled" to a new node, instead it will be replaced by an identical Pod. With even the same name if desired, but with a new UID.

→ Volume in a POD will exists as long as that POD (with that UID) exist. If that POD is deleted for any reason, Volume is also destroyed and Createad as new on new Pod.

→ A Controller can Create and manage multiple Pods, handling replication, rollout and providing self-healing Capabilities.

Kubernetes Configuration

→ All in one single node installation	With all-in-One, all the master and worker Components are installed on a single node. This is very useful for learning, development and testing this type should not be used in Production. Minikube is one such example, and we are going to explore it soon.
→ Single-node etcd, Single-master and Multi-Worker installation	In this step, we have a single master node, which also runs a single-node etcd instance. Multiple worker nodes are Connected to the master node.

→ Single-node etcd, Multi-master and Multi-worker installation :	In this step, we have multiple master nodes, which works in an HA mode, but we have a single-node etcd instance. Multiple worker nodes are Connected to the Master node.

LAB

- Go to AWS account → Launch instance → Ubuntu 18.04
  → t2.medium (2 VCPU)

- Now access EC2 via Putty → Login as "Ubuntu".
  → sudo su
  → sudo apt update && apt -y install docker.io

- Now install Kubectl
  → curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl && chmod +x ./kubectl && sudo mv ./kubectl /usr/local/bin/kubectl

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

→ apt install Conntrack

→ minikube start --vm-driver=none

→ minikube status

→ kubectl version

⟶ Now onwards, we will use kubectl Commands

→ kubectl get nodes

O/P → Name Status Roles Age Version

ip.... Ready Master 2m v1.20.7

→ kubectl describe node ip-<ip_of_your_node> (» private)

→ vi Pod1.yml

Code

```
kind: Pod                    (Capital)
apiVersion: v1
metadata:
    name: testPod
spec:
  containers:
    - name: c00
      image: ubuntu
      command: ["/bin/bash", "-c", "while true; do echo Hello-Master; sleep 5; done"]
                                  (No space)
  restartPolicy: Never    # Default to Always
```

- create Pod and execute

→ Kubectl apply -f pod1.yml

o/p → pod/testpod created

- If you want to see, where exactly Pod is running.

→ Kubectl get pods -o wide

o/p → testpod 1/1 running .... and other details

- Details of the Pod

→ Kubectl describe pod testpod

(object name → testpod; object → pod)

- What are runnings on our node

→ kubectl logs -f testpod

- To see running services on a specific Pod

→ kubectl logs -f testpod -c coo

- To delete a Pod

→ kubectl delete pod testpod

■ Example of Annotations

→ vi pod1.yml

(Existing file)

```
metadata:
    name: testpod
    annotations:
        description: our first pod creation
```

Esc → :wq | - Now you can see Annotations somewhere.

→ Kubectl apply -f pod1.yml

→ Kubectl delete pod pod1.yml

- To delete the Pod using cmnd or deleting YAML file
→ Kubectl delete pod testPod or, kubectl delete -f Pod2.yml

■ Ex. MultiContainer Pod

→ vi pod2.yaml

```
Kind: Pod
apiVersion: v1
metadata:
   name: testPod3
spec:
   Containers:
      - name: c00
        image: Ubuntu
        Command: ["/bin/bash", "-c", "while true; do echo
                                    Md-Shoaib; sleep 5s;u done"]
      - name: c01
        image: Ubuntu
        Command: ["/bin/bash", "-c", "while true; do echo
                                    Hello-my-Master; sleep 5s;u done"]
```

→ Kubectl apply -f pod2.yml

→ Kubectl get pods

- Some minor changes

→ vi pod1.yml

Code under previous code for adding one more cont.

```
- name: co2
  image: ubuntu
  Command: ["/bin/bash", "-c", "while true; do echo
            Welcome-Guru; sleep 5 ; done"]
```

→ Kubectl apply -f pod2.yml

→ Kubectl delete Pod testpod3

→ Kubectl apply -f pod2.yml

→ Kubectl get pods

→ Kubectl describe pod testpod3

→ Kubectl logs -f testpod3

→ Kubectl logs -f testpod3 -c co2 ← Mention specific container

→ Kubectl exec testpod3 -c co2 --hostname -i

O/p → IP_Address ← To get IP of the Pod

→ root@ip... /home/ubuntu# Kubectl exec testpod3 -c co1 -- /bin/bash ← go inside the container

→ root@testpod3:/# ps

→ root@testpod3:/# ps -ef

→ root@testpod3:/# exit

→ Kubectl exec testpod3 -it -c co1 -- /bin/bash

→ root@testpod3:/# ps -ef

→ root@testpod3:/# exit

- Delete the Pod by deleting YAML file

root@.....Ubuntu# Kubectl delete -f pod2.yml

Ex - Environment Variables in Pod

→ Vi Pod3.yml

Code

```
Kind: Pod
abiVersion: v1
metadata:
    name: environments
spec:
  Containers:
    - name: c00
      image: Ubuntu
      Command: ["/bin/bash", "-c", "while true; do echo
          Hell-Brother; sleep 5 ; done"]
      env:    #list of env variables to be used inside the pod
      - name: MYNAME
        value: SHOAIB
```

→ Kubectl apply -f pod3.yml

O/P → Pod/environments created

→ Kubectl get Pods

→ Kubectl exec environments -it -- /bin/bash

→ root@environments:/# env

→ root@environments:/# echo $MYNAME

exit then

O/P → SHOAIB

→ Kubectl delete -f pod3.yml

■ Example Pod with ports.

→ vi pod4.yml

Code

```
kind: Pod
apiVersion: v1
metadata:
    name: testpod4
spec:
  Containers:
    - name: c00
      image: httpd
      ports:
        - containerPort: 80
```

→ Kubectl apply -f pod4.yml

→ Kubectl get pods

→ Kubectl get pods -o wide

→ curl <private-ip>:80

Minikube Lab-2 | Labels, Selectors, Replication Controller and Replicaset

Labels and Selectors

→ Labels are the mechanism you use to organise Kubernetes Objects.

- A Label is a key-value pair without any predefined meaning that can be attached to the objects.

- Labels are similar to tags in AWS or git where you use a name to quick reference.

- So you are free to choose labels as you need it to refer an environment which is used for dev or testing or Production, refer a product group like Department A, Department B.

- Multiple labels can be added to a single object.

→ vi pod5.yml

```
kind: Pod
apiVersion: v1
metadata:
    name: delipod
    labels:
```

```
    env: development
    class: pods
spec:
    containers:
      - name: c00
        image: ubuntu
        Command: ["/bin/bash", "-c", "while true; do echo
                  Hello-Master; sleep 5 ; done"]
```

ESC ⟶ :wq

→ Kubectl apply -f pod5.yml

→ Kubectl get pods --show-labels

- Now, if you want to add a label to an existing pod

→ kubectl label pods delhipod myname=master

→ Kubectl get pods --show-labels
↖ object (if node then type node)

- Now, list pods matching a label.

→ Kubectl get pods -l env=development

- Now, give a list, where 'development' label is not present.

→ Kubectl get pods -l env!=development

- Now, if you want to delete Pod using labels
→ Kubectl delete Pod -l env!=development
→ Kubectl get Pods

Labels - Selectors

→ Unlike name/UIDs, labels do not provide Uniqueness, as in general, we can expect many objects to carry the same label.

- Once labels are attached to an object, we would need filters to narrow down and these are called as label selectors.

- The api currently supports two types of selectors-
  (i) Equality based and
  (ii) Set based.

- A label selector can be made of multiple requirements which are comma-separated.

- Equality Based:- (=, !=)
  name: master
  class: nodes
  project: development

- Set based: (in, notin and exists)

  env in (production, dev)

  env notin (team1, team2) → No space

- Kubernetes also supports set-based selectors i.e match multiple values.

→ Kubectl get pods -l 'env in (development, testing)'

→ Kubectl get pods -l 'env notin(development, testing)'

→ Kubectl get pods -l class=pods, myname=master

To check multiple labels

■ Node Selector

→ One use case for selecting labels is to Constrain the set of nodes onto which a pod can schedule i.e you can tell a pod to only be able to run on particular nodes.

- Generally such constraints are unnecessary, as the scheduler will automatically do a reasonable placement, but on certain circumstances we might need it.

- We can use labels to tag nodes.

- You the nodes are tagged, you can use the label selectors to specify the pods run only of specific nodes.

- First we give label to the node.

- Then use node selector to the pod Configuration.

■ Scaling and Replication

→ Kubernetes was designed to orchestrate multiple Containers and replication.

→ Need for multiple Containers/replication helps us with these.

Reliability → By having multiple versions of an application, you prevent problems if one or more fails.

Load Balancing → Having a multiple versions of a Containers enables you to easily send traffic to different instances to prevent overloading of a single instances or node.

Scaling → When load does become too much for the number of existing instances, Kubernetes enables you to easily scale up your application, adding additional instances as needed.

Rolling updates → Updates to a service by replacing pods one by one.

Replication Controller

→ A replication Controller is a object that enables you to easily create multiple pods, then make sure that number of pods always exist.

→ If a pod created using RC will be automatically replaced if they does crash, failed, or terminated.

→ RC is recommended if you just want to make sure 1 pod is always running, even after system reboots.

→ You can run the RC with 1 replica & the RC will make sure the pod is always running.

Replication Controller

→ vi myrc.yml → object

Kind: ReplicationController → this defines to create the object of Replication type

apiVersion: v1

metadata:

name: myreplica

spec:

replicas: 5 → this element defines the desired no. of pods

Selector: → tells the controller which pods to watch/belong to this RC

myname: Master → this must match the labels:

template: → template element defines a template to launch a new pod.

metadata:

name: testpod6

labels: → selectors values need to match the labels values specified in the pod template.

myname: Master

spec:

Containers:

- name: c00

image: Ubuntu

Command: ["/bin/bash", "-c" "while true; do echo Hello-Master; sleep 5; done"]

→ Kubectl apply -f myrc.yml

→ Kubectl get rc

→ Kubectl describe rc myreplica

→ Kubectl get pods --show-labels

- To Scale up your Pods

→ Kubectl scale --replicas=8 rc -l myname=Master

→ Kubectl get rc

→ scaling down rc

→ Kubectl scale --replicas=1 rc -l myname=Master

→ Kubectl delete -f myrc.yml

→ Kubectl get rc

→ Kubectl get pods

Replica set

→ Replica set is a next generation Replication Controller

- The replication Controller only supports equality-based selector whereas the replica set supports set-based selector i.e. filtering according to set of values.

- Replicaset rather than the Replication Controller is used by other objects like deployment.

→ vi myrs.yml

```
Kind: ReplicaSet
apiVersion: apps/v1
metadata:
    name: myrs
spec:
  replicas: 2
  selector:
    matchExpressions:                    # these must match the labels
      - {key: myname, operator: In, values: [Bhupinder, Bubinder, Bhokendra]}
      - {key: env, operator: NotIn, values: [production]}
  template:
    metadata:
      name: testpod7
      labels:
        myname: Bhupinder
    spec:
      Containers:
        - name: Coo
          image: Ubuntu
          command: ["/bin/bash", "-c", "while true; do echo Master-MD; sleep 5 ; done"]
```

→ Kubectl apply -f myrs.yaml

→ Kubectl get rs

→ Kubectl get pods

- Scale up our Replica set Pods

→ Kubectl scale --replicas=1 rs/myrs

→ Kubectl get pods

→ Kubectl get rs

→ Kubectl delete pod myrs-77nmt
(← Pod-Name)

→ Kubectl get rs

→ Kubectl get pods

→ Kubectl delete rs/myrs

→ Kubectl get pods

→ Kubectl get rs

Minikube Lab-3| Deployment object in K8s

Deployment & Rollback

→ Replication Controller & Replica Set is not able to do updates & Rollback apps in the cluster.

- A deployment object act as a supervisor for pods, giving you fine-grained control over how and when a new pod is Rolled out, updated or Rolled Back to a previous state.

- When using deployment object, we first define the state of the app, then K8s cluster schedules mentioned app instance onto specific individual nodes.

- K8s then monitors if the node hosting an instance goes down or Pod is deleted the deployment Controller replaces it.

- This provides a self-healing mechanism to address machine failure or maintenance.

- A deployment provides declarative updates for pods & ReplicaSet.

• The following are typical use cases of Deployments -

① Create a deployment to rollout a Replicaset → The Replicaset creates Pods in the background. Check the status of the Rollout to see if it succeeds or not.

② Declare the new state of the Pods → By updating the PodTemplateSpec of the deployment. A new Replicaset is created and the Deployment manages moving the Pods from the old Replicaset to the newone at a controlled rate. Each new Replicaset updates the revision of the Deployment.

③ Rollback to an earlier Deployment Revision → If the current state of the Deployment is not stable. Each rollback updates the revision of the Deployment.

~~④ Rollback to an earlier Deployment Revision → If th~~

④ Scale up the Deployment to facilitates more load.

⑤ Pause the Deployment to apply multiple fixes to its PodTemplateSpec and then resume it to start a new rollout.

⑥ Cleanup older Replicasets that you don't need anymore.

→ If there are problems in the deployment, kubernetes will automatically roll back to the previous version, however you can also explicitly rollback to a specific revision, as in our case to Revision 1 (the original Pod version)

- You can rollback to a specific version by specifying it with --to-revision

for eg → kubectl rollout undo deploy/mydeployments --to-revision=2

deploy → type; mydeployments → object type Name

■ Note: That the name of the Replicaset is always formatted as [Deployment-name]-[Random string]

Cmnd → kubectl get deploy

- When you inspect the deployments in your cluster, the following fields are display.

NAME → List the names of the deployments in the namespace.

READY → Display how many replicas of the application are available to your users. It follows the pattern ready/Desired.

UP-TO-DATE → Display the number of replicas that have been updated to achieve the desired state.

AVAILABLE → Displays how many replicas of the application are available to your users.

AGE → Display the amount of time that the application has been running.

LAB

→ Login to AWS Management console, create one Ubuntu (t2.medium) instance. Take access using putty.

→ sudo su

→ sudo apt update && apt -y install docker.io

→ install Kubectl from given link

→ install minikube from link

→ apt install conntrack

→ vi mydeploy.yml

Code for kubectl get nodes

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: mydeployments
spec:
  replicas: 2
  selector:
    matchLabels:
      name: deployment
  template:
    metadata:
      name: testpod
      labels:
        name: deployment
    spec:
      containers:
        - name: c00
          image: ubuntu
          command: ["/bin/bash", "-c", "while true;
            do echo Technical-Master; sleep 5 ; done"]
```

ESC ⟶ : wq

→ kubectl apply -f mydeploy.yml

→ kubectl get deploy

- To check deployment was created or not
→ Kubectl get deploy

- To check, how deployment creates RS & Pods.
→ Kubectl describe deploy mydeployments
→ Kubectl get rs
(deploy: obj type; mydeployments: obj. Name)

- To scale up/down
→ Kubectl scale --replicas=1 deploy mydeployments
(deploy: object type; mydeployments: object Name)

- To check, what is running inside container
→ Kubectl logs -f <Podname>

- To rollout / Rollback Status
→ Kubectl rollout status deployment mydeployments

- To see history of your versions or about deployments
→ Kubectl rollout history deployment mydeployments

- To go previous step
→ Kubectl rollout undo deploy/mydeployments

■ Failed Deployment

→ Your deployment may get stuck trying to deploy its newest Replicaset without ever completing. This can occur due to some of the following factors.

1. Insufficient Quota
2. Readiness Probe failures
3. Image Pull errors
4. Insufficient Permission
5. Limit Ranges
6. Application runtime misconfiguration

Kubernetes Networking, Services, Nodeport & Volumes

→ Kubernetes Networking addresses four concerns —

① Containers within a Pod use networking to communicate via loopback.

② Cluster Networking provides communication between different Pods.

③ The service resources lets you expose an application running in Pods to be reachable from outside your cluster.

④ You can also use services to publish services only for consumption inside your cluster

→ Container to Container Communication on same Pod happens through localhost within the containers.

LAB

Install Minikube on your AWS EC2 Instance ...

→ sudo apt update && apt -y install docker.io

→ Install Kubectl from the link

→ Install minikube from the link

- start Minikube

→ apt install Conntrack

→ minikube start --vm-driver=none

→ minikube status

Code for Kubernetes Networking

→ vi pod1.yml

```
kind: Pod
apiVersion: v1
metadata:
    name: testpod
spec:
  containers:
    - name: c00
      image: Ubuntu
      command: ["/bin/bash", "-c", "while true; do echo
          Hello-Master; sleep 5 ; done"]
    - name: c01
      image: httpd
      ports:
        - containerPort: 80
```

ESC ⟶ :wq

→ kubectl apply -f pod1.yml

→ kubectl get pods

→ kubectl exec testpod -it -c c00 -- /bin/bash

→ root@testpod:/# apt update && apt install curl

→ root@testpod:/# curl localhost:80

. . . . . → then exit.

→ kubectl delete -f pod1.yml

- Now try to established Communication between two different Pods within same machine(node) -

→ Pod to Pod Communication on same worker node happens through Pod IP.

→ By default Pod's IP will not be accessible outside the node.

LAB

→ vi Pod2.yml

```
kind: Pod
apiVersion: v1
metadata:
  name: testpod1
spec:
  Containers:
```

- name: Co1
  image: nginx
  Command: ["bin/bash", "-c", "while true; do echo \
    Hello-Master; sleep 5 ; done"]
  - ContainerPort: 80
  Esc ⟶ :wq

⟶ vi pod3.yml

kind: Pod
apiVersion: v1
metadata:
  name: testpod4
spec:
  Containers:
    - name: CO3
      image: httpd
      ports:
        - containerPort: 80

⟶ Kubectl apply -f pod2.yml

⟶ Kubectl apply -f pod3.yml

⟶ Kubectl get pods

→ kubectl get pods -o wide

→ curl <ip....> :80

■ Object - Services

→ Each Pod gets its own IP address, however in a deployment, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later.

→ This leads to be a problem: If some set of Pods (call them 'backends') provides functionality to other Pods ('Call them frontend') inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload

→ When using RC, Pods are terminated and created during Scaling or replication Operations.

→ When using deployments, while updating the image version the Pods are terminated and new pods take the place of other Pods.

→ Pods are very dynamic i.e they come & go on the K8s cluster and on any of the available nodes & it would be difficult to access the pods as the pods IP changes once its recreated

→ Service object is an logical bridge between pods and end users, which provides Virtual IP(VIP).

→ Service allows clients to reliably connect to the containers running in the pod using the VIP.

→ The VIP is not an actual IP connected to a network interface, but its purpose is purely to forward traffic to one or more pods.

→ Kube proxy is the one which keeps the mapping between the VIP and the pods upto date, which queries the API server to learn about new services in the cluster.

→ Although each pod has a unique IP address, those IP's are not exposed outside the cluster.

→ Services helps to expose the VIP mapped to the pods & allows application to receive traffic.

→ Labels are used to select which are the pods to be put under a service.

→ Creating a service will create an endpoint to access the pods/application in it.

→ Service can be exposed in different ways by specifying a type in the service spec.

① → cluster IP

② → Nodeport

③ → LoadBalancer created by cloud Providers that will route external traffic to every node on the Nodeport. (eg- ELB on AWS)

④ → Headless → Creates several endpoints that are used to produce DNS Records. Each DNS Record is bound to a ~~pod~~ Pod.

→ By default Service can run only between ports 30,000 - 32767.

→ The set of pods targeted by a service is usually determind by a selector.

■ Cluster IP

→ Expose VIP only reachable from within the cluster.

→ Mainly used to communicate between components of microservices.

LAB

→ Vi deploy_httpd.yml

```
kind: Deployment
apiVersion: apps/v1
metadata:
    name: mydeployments
spec:
    replicas: 1
    selector: # tells the Controller which pods
      matchLabels:                to watch/belong to
        name: deployment
    template:
      metadata:
        name: testpod1
        labels:
          name: deployment
      spec:
        Containers:
```

```
- name: c00
  image: httpd
  ports:
    - containerPort: 80
```

→ kubectl apply -f deployhttpd.yml

→ kubectl get pods

→ kubectl get pods -o wide

→ curl <ip...>:80

→ vi service.yml

```
kind: Service          # Define to create service type obj.
apiVersion: v1
metadata:
  name: demoservice
spec:
  ports:
    - port: 80          # Containers port exposed
      targetPort: 80    # Pods port
  selector:
    name: deployment    # Apply this service to any
                        pods which has the specific label
```

type: clusterIP # specifies the service type ie clusterIP or NodePort

ESC ⟶ :wq

→ kubectl apply -f service.yml

→ Kubectl get svc

root@ip... ubuntu# curl <ip...>:80

root@ip... ubuntu# Kubectl get Pods

.... # Kubectl delete Pod mydeployments-----...

→ kubectl get Pods

→ curl <ip...>:80

→ kubectl delete -f service.yml

→ Kubectl delete -f Pod2.yml

→ Kubectl delete -f Pod3.yml

→ kubectl delete -f deployhttpd.yml

NodePort

→ Makes a service accessible from outside the cluster.

→ Exposes the service on the same port of each selected node in the cluster using NAT.

LAB

→ vi deployhttpd.yml ← same script nothing changes

→ kubectl apply -f deployhttpd.yml

→ vi svc.yml

```
kind: Service
apiVersion: v1
metadata:
    name: demoservice
spec:
  ports:
    - port: 80
      targetPort: 80
  selector:
      name: deployment
  type: NodePort
```

ESC ⟶ :wq

→ kubectl apply -f svc.yml

→ kubectl get svc

→ kubectl describe svc demoservice

→ kubectl delete -f svc.yml

- Now copy your public DNS of Node and attach the port no.

Volumes

→ Containers are short lived in Nature.

- All data stored inside a Container is deleted if the Container crashes. However the kubelet will restart it with a clean state, which means that it will not have any of the old data.

- To overcome this problem, kubernetes uses Volumes. A Volume is essentially a directory backed by a storage medium. The storage medium and its content are determined by the Volume type

- In kubernetes, a Volume is attached to a Pod and shared among the Containers of that Pod.

- The Volume has the same life span as the Pod, and it outlives the Containers of the Pod this allows data to be preserved across Container Restarts.

Volume Types

→ A Volume type decides the properties of the directory, like size, content etc. Some eg. of Volume types are-

- node-local type such as emptydir and hostpath.
- File sharing types such as nfs
- Cloud provider-specific types like awselasticblockstore, azuredisk
- Distributed file system types, for example glusterfs or cephfs
- Special purpose types like secret, gitrepo.

EmptyDir

→ Use this when we want to share contents between multiple Containers on the same Pod & not to the host machine.

- An emptydir Volume is first created when a Pod is assigned to a node, and exist as long as that Pod is running on that node.

- As the name says, it is initially empty.

- Containers in the Pod can all read and write the same files in the emptydir volume, though that volume can be mounted at the same or different paths in each containers.

- When a Pod is removed from a node for any reason, the data in the emptydir is deleted forever.

- A Container crashing does not remove a Pod from a ~~Pod~~ so the data in an emptyDir volume is safe across Container crashes.

LAB

→ vi emptydir.yml

```
apiVersion: V1
kind: Pod
metadata:
    name: myvolemptydir
spec:
  containers:
  - name: C1
    image: centos
```

```
Command: ["/bin/bash", "-c", "sleep 15000"]
volumeMounts:
  - name: xchange
    mountPath: "/tmp/xChange"
- name: c2
image: centos
Command: ["/bin/bash", "-c", "sleep 10000"]
VolumeMounts:                    # Mount definition inside the cont.
  - name: xChange
    mountPath: "/tmp/data"
Volumes:
  - name: xchange
    emptyDir: {}
```

ESC ⟶ :wq

→ Kubectl apply -f emptydir.yml
→ Kubectl get pods
→ Kubectl exec myvolemptydir -c c1 -it -- /bin/bash
→ [root@myvolemptydir /]# cd /tmp
. . . . . # ls
- - - - - - xchange
. . . . . # cd xchange
. . . . . # vi abc.yml ⟶ write something
. . . xchange]# ls
. . . # exit

→ kubectl exec myvolemptydir -c c2 -it -- /bin/bash
(-c → Container, c2 → Container_Name)

→ [root@myvolemptydir /]# cd /tmp

→ [root@myvolemptydir tmp]# ls

O/P → data . . . .

→ [root@myvolemptydir tmp]# cd data

→ [root@myvolemptydir data]# ls

O/P → abc.yml technical

← cat abc.yml

again vi abc.yml and wrote some extra Content then exit.

→ kubectl exec myvolemptydir -c c1 -it -- /bin/bash

→ [root@myvolemptydir /]# cat /tmp/xchange/abc.yml

. . . } Your Content then do exit.

→ kubectl delete -f emptydir.yml

HostPath

→ Use this when we want to access the Content of a Pod/Container from hostmachine.

— A hostPath Volume mounts a file or directory from the host node's filesystem into your Pod.

LAB

→ vi hostPath.yml

```
apiVersion: V1
kind: Pod
metadata:
```

```
    name: myvolhostpath
  spec:
    containers:
    - image: centos
      name: testc
      Command : ["/bin/bash", "-c", "sleep 15000"]
      VolumeMounts:
        - mountPath: /tmp/hostpath
          name: testvolume
    volumes:
    - name: testvolume
      hostPath:
        path: /tmp/data
```

→ kubectl apply -f hostpath.yml

→ kubectl get pods

→ ls /tmp

→ cd /tmp/data

→ ls

→ kubectl exec myvolhostpath -- ls /tmp/hostpath

cd to direct this directory

→ kubectl exec myvolhostpath -- ls /tmp

→ echo "Hello My Master" >myfile

→ kubectl exec myvolhostpath -- ls /tmp/hostpath

→ cat myfile

→ kubectl exec myvolhostpath -- cat /tmp/hostpath/myfile

O/P → Hello My Master

→ cd /tmp/hostpath/

→ kubectl exec myvolhostpath -it -- /bin/bash

→ [root@myvolhostpath /]# cd /tmp/hostpath

ls

O/P → myfile

echo "this is my second file" >file

ls

exit

Persistent Volume & LivenessProbe in Kubernetes

→ In a typical IT environment, Storage is managed by the storage/system adminstrator. The end user will just get instructions to use the storage, but does not have to worry about the underlying storage management.

- In the containerized world, we would like to follow similar rules, but it becomes challenging, given the many volume types we have seen earlier Kubernetes resolves this problem with the Persistent Volume (PV), subsystem.

- A Persistent Volume (PV) is a cluster-wide resource that you can use to store data in a way that it persists beyond the lifetime of a Pod.

- The PV is not backed by locally attached storage on a worked node but by networked storage system such as EBS or NFS or a distributed filesystem like ceph.

- K8s provides APIs for users and adminstrator to manage and consume storage. To manage the Volume, it uses the Persistent Volume API resource

type and to Consume it, uses the PersistentVolume-claim API resource type.

Persistent Volume Claim

→ In order to use a PV you need to claim it first, using a Persistent Volume claim (PVC)

- The PVC requests a PV with your desired specification (size, accessmodes, speed etc) from Kubernetes and once a Suitable Persistent Volume is Found, it is bound to a PersistentVolumeClaim.

- After a successful bound to a Pod, you can mount it as a Volume.

- Once a user finishes its works, the attached PersistentVolume can be released. The underlying PV can there be reclaimed and Recycled for future usage.

AWS EBS (Elastic Block Storage)

→ An AWS EBS Volume mounts an AWS EBS Volume into your Pod. Unlike emptyDir, which is erased when a Pod is removed, the content of an EBS Volume are Preserved and the Volume is merely Unmounted.

There are some Restrictions:

① The nodes on which Pods are running must be AWS EC2 instances.

② Those instances need to be in the same region and availability zone as the EBS Volume.

③ EBS only supports a single EC2 instance mounting a volume.

LAB

- Install Docker

→ Sudo apt update && apt -y install docker.io

- Install Kubectl from the link
- Install Minikube from the link

- Start Minikube

→ apt install conntrack

→ minikube start --vm-driver=none

→ minikube status

→ Now select Volumes under AWS Elastic Block store title > then add size of 20 GiB and copy Vol-ID.

→ vi mypv.yml

```
apiVersion: v1
kind: PersistentVolume
metadata:
    name: myebsvol
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  PersistentVolumeReclaimPolicy: Recycle
  awsElasticBlockStore:
      VolumeID: <your_vol-ID>
      fsType: ext4
```

ESC → :wq

→ kubectl apply -f mypv.yml

→ kubectl get pv

→ vi mypvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myebsvolclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

ESC ⟶ :wq

→ kubectl apply -f mypvc.yml

→ kubectl get pvc

→ vi deployPvc.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pvdeploy
spec:
  replicas: 1
  selector:          # tells the controller which pods to watch/belong to
    matchLabels:
      app: mypv
  template:
    metadata:
      labels:
        app: mypv
    spec:
      Containers:
        - name: shell
          image: centos
          command: ["/bin/bash", "-c", "sleep 10000"]
          volumeMounts:
            - name: mypd
              mountPath: "/tmp/persistent"
      Volumes:
        - name: mypd
          PersistentVolumeClaim:
            claimName: my ebsvolclaim
```

→ Kubectl apply -f deployprc.yml

→ Kubectl get deploy

→ Kubectl get rs

→ Kubectl get pods

→ Kubectl exec prdeploy..... -it -- /bin/bash

→ [root@... /]# cd /tmp/persistent/

ls

vi testfile >.... Hello my tmp file..

exit

→ Kubectl delete pod pv deploy ---

→ Kubectl get pods

→ Kubectl exec prdeploy... -it -- /bin/bash

→ [root@... /]# cd /tmp/persistent/

ls

O/P → test file

cat testfile

exit

→ Kubectl delete -f deployprc.yml

→ Kubectl delete -f mypvc.yml

→ Kubectl delete -f mypv.yml

→ then delete your EBS from AWS

HealthCheck/LivenessProbe

→ A Pod is Considered ready When all of its Containers are ready.

- In order to verify if a container in a Pod is Healthy and ready to serve traffic, Kubernetes provides for a range of healthy checking mechanism

- Health checks or Probes are Carried Out by the kubelet to determine when to restart a Container (For liveness Probe) and used by services and deployments to determine if a Pod should receive traffic.

for eg → Liveness Probes Could catch a deadlock, When an application is running, but unable to make progress. Restoring a container in such a state can help to make the application more available despite bugs.

- One use of readiness probes is to control which Pods are used as backends for services. When a Pod is not ready, it is removed from service load Balancers.

- For running healthchecks, we would use cmnds specific to the application.

- If the cmnd succeeds, it returs 0, and the kubelet considers the container to be alive and healthy. If the command returns a non-zero value, the kubelet kills the pod and recreate it.

LAB

→ Vi liveness.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: myLivenessProbe
spec:
  containers:
    - name: liveness
      image: ubuntu
      args:
        - /bin/bash
        - -c
        - touch /tmp/healthy; sleep 1000
      livenessProbe:  # define the health check
        exec:
```

Command: # Command to run periodically

- cat
- /tmp/healthy

initialDelaySeconds: 5 # wait for the specified time before it runs the first probe

periodSeconds: 5 # Run the above command every 5 sec

timeoutSeconds: 30

ESC ⟶ :wq

→ kubectl apply -f liveness.yml

→ kubectl get pods

→ kubectl describe pod mylivenessprobe

→ kubectl get Pods

→ kubectl exec mylivenessprobe -it -- /bin/bash

→ root@mylivenessprobe:/# cat /tmp/healthy

→ # echo $?

O/P → 0

→ # cat /tmp/shraib

O/P → No such file or directory

echo $?

O/P → 1

→ # ls tmp/healthy

Configmap and Secrets in Kubernetes

→ While performing application deployments on K8s cluster, Sometimes we need to change the application Configuration file depending on environments lik dev, QA, stage or Prod.

- Changing this application Configuration file means we need to change source code, Commit the Change, creating a new image and then go through the Complete deployment process.

- This is where Kubernetes Configmap comes handy, it allows us to handle Configuration files much more efficiently.

- Configmaps are useful for storing and sharing non-sensitive, Unencrypted Configuration information. Use secrets otherwise.

- Configmap can be used to store fine-grained information like individual properties or entire configfiles.

- Configmap are not intended to act as a replacement for a properties file.

→ Configmap can be accessed in following ways:-

① As environment Variables
② As Volumes in the Pod

→ kubectl create configmap <mapname> --from-file=<file_to_read>

■ Secrets

→ You don't want sensitive information such as a database password or an API key kept around in clear text.

- Secrets provide you with a mechanism to use such information in a safe and Reliable way with the following properties:-

- Secrets are namespaced objects, that is exist in the Context of a namespace.

- You can access them via a Volume on an environment Variable from a Container running in a Pod.

- The Secret data on nodes is stored in tmpfs volumes (tmpfs is a file system which keeps all files in virtual Memory). Everything in tmpfs is temporary in the sense that no files will be created on your hard drive.

- A per-Secret size limit of 1 MB exist.

- The API server stores secrets as plaintext in etcd. Secrets can be created -
  1. from a text file
  2. from a yaml file

LAB

Start your Linux machine and install Docker, Kubectl and minikube from link-

→ Vi sample.conf

... this is my configuration....

→ Kubectl create Configmap mymap --from-file=sample.conf
(↑ obj. Name)

→ Kubectl get Configmap

→ Kubectl describe Configmap mymap

→ Vi deployConfigmap.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myvolConfig
spec:
  containers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "while true; do echo
             Technical-Master; sleep 5 ; done"]
    VolumeMounts:
      - name: testconfigmap
        mountPath: "/tmp/config" # the config
files will be mounted as ReadOnly by default here
    Volumes:
      - name: testconfigmap
        ConfigMap:
          name: myfmap # this should match
the config map name created in the first step.
          items
            - key: sample.conf
              path: sample.conf
```

Esc ⟶ :wq

→ kubectl apply -f deploy configmap.yml

→ kubectl get pods

→ kubectl exec myvolConfig -it -- /bin/bash

→ [root@myvolConfig /]# cd /tmp

ls

O/P → Config .....

cd Config

ls

O/P → Sample.conf → you can cat this file / then exit

→ kubectl delete -f deploy configmap.yml

→ vi deployenv.yml

apiVersion: v1

kind: Pod

metadata

name: myenvConfig

spec:

Containers:

- name: c1

image: centos

Command: ["/bin/bash", "-c", "while true; do echo Technical-Master; sleep 5 ; done"]

env:

- name: MYENV # env name in which value of the key is stored.

```
        valueFrom:
          configMapKeyRef:
            name: mymap #name of the config created
            key: sample.conf
```

ESC → :wq

→ kubectl apply -f deployenv.yml

→ kubectl get pods

→ kubectl exec myenvConfig -it -- /bin/bash

→ [root@myenvConfig /]# env

↳ exit

→ kubectl delete -f deployenv.yml

→ echo "root" > username.txt; echo "mypassword123" > password.txt

→ Cat username.txt

→ Cat password.txt

→ kubectl create secret generic mysecret --from-file=username.txt --from-file=password.txt

→ kubectl get secret

→ kubectl describe secret mysecret

→

→ vi deploysecret.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myredsecret
```

spec:

containers:

- name: c1

image: centos

command: ["/bin/bash", "-c", "while true; do echo Technical-Master; sleep 5 ; done"]

volumeMounts:

- name: testsecret

mountPath: "/tmp/mysecrets" # the secret files will be mounted as ReadOnly by default here.

volumes:

- name: testSecret

secret:

secretName: mysecret

Esc ⟶ :wq

→ kubectl apply -f deploysecret.yml

→ kubectl exec myvolsecret -it -- /bin/bash

→ [root@myvolsecret /]# cd /tmp

ls

cd mysecrets

ls

o/p → password.txt username.txt → You can cat the files

Namespaces, Limits & Request in Kubernetes

Namespaces

→ You can name your object, but if many are using the cluster then it would be difficult for Managing.

- A namespace is a group of related elements that each have a Unique name or identifier. Namespace is used to uniquely identify one or more names from other similar names of different objects, groups or the namespace in general.

- Kubernetes namespaces help different project teams or customers to share a Kubernetes cluster & Provides-

  - A scope for every names.

  - A mechanism to attach authorization and policy to a subsection of the cluster.

Namespace

→ By default, a kubernetes cluster will instantiate a default namespace when provisioning the cluster to hold the default set of pods, services and Deployments used by the cluster.

- We can use resource quota on specifying how many resources each namespace can use.

- Most Kubernetes resources (eg. Pods, services, replicatio controllers and others) are in some namespaces and low-level resources such as nodes and Persistent Volumes, are not in any namespace.

- Namespaces are intended for use in environments with many ~~years~~ users spread across multible teams, or projects. For cluster with a few to tens of users, you should not need to create or think about namespaces and all.

Create new namespaces

→ Let us assume we have shared K8s cluster for Dev and production use cases.

- The dev team would like to maintain a space in the cluster where they can get a view on the list of pods, services and Deployments they use to build and run their application In this, no restrictions are put on who can or cannot modify resources to enable agile development.

- For Production team we can enforce strict procedure on who can or cannot manipulate the set of Pods, Services and deployments.

cmd → kubectl get namespaces

LAB

→ kubectl get pods

→ No resources found in default namespace

→ kubectl get pods -n dev

→same

→ kubectl get namespace

:

:

→ vi devns.yml

apiVersion: V1

kind: Namespace

metadata:

name: dev

labels:

name: dev

ESC → :wq

→ kubectl apply -f devns.yml

→ kubectl get namespace

→ vi pod.yml

```
kind: Pod
apiVersion: v1
metadata:
    name: testpod
spec:
    Containers:
        - name: c00
          image: ubuntu
          command: ["/bin/bash", "-c" "while true; do
echo Technical-Master; sleep 5 ; done"]
    restartPolicy: Never
```

Esc ⟶ :wq

→ kubectl apply -f pod.yml -n dev

→ kubectl get pods

→ kubectl get pods -n dev

→ kubectl delete -f pod.yml

→ kubectl delete -f pod.yml -n dev

→ kubectl apply -f pod.yml -n dev

→ kubectl get pods

→ kubectl config set-context $(kubectl config current-context) \
--namespace=dev

→ kubectl get pods
- Searching for specific using grep
→ kubectl config view | grep namespace:
o/p → namespace: dev

→ kubectl get pods
→ kubectl get pods -n default
→ kubectl delete -f pod.yml

Managing Compute Resources for Containers

→ A Pod in Kubernetes will run with no limits on CPU and memory.

- You can optionally specify how much CPU and memory (RAM) each container needs.

- Scheduler decides about which nodes to place pods only if the Node has enough CPU resources available to satisfy the Pod CPU request.

- CPU is specified in Units of Cores and memory is specified in Units of Bytes.

- Two types of Constraints can be set for each resource type - Request and Limits.

→ A **request** is the amount of that resources that the system will guarantee for the Container and Kubernetes will use this value to decide on which node to place the Pod.

- A **limit** is the max amount of resources that Kubernetes will allow the container to use. In the case that request is not set for a Container, it default to limits. If limit is not set, then it default to 0.

- CPU Values are specified in 'MilliCPU' and memory in MiB.

Resource Quota

→ A Kubernetes Cluster can be divided into namespaces. If a Container is created in a namespace that has a default CPU limit, and the container does not specify its own CPU limit, then the Container

is assigned the default CPU limit.

- Namespaces can be assigned Resource Quota objects, this will limit the amount of usage allowed to the objects in that namespace. You can limit —
  1. Compute
  2. Memory
  3. Storage

- Here are two restrictions that a resource quota imposes on a namespace:
  - Every container that runs in the namespace must have its own CPU Limit.
  - The total amount of CPU used by all containers in the namespace must not exceed a specified limit.

LAB

→ vi podresources.yml

```
apiVersion: V1
kind: Pod
metadata:
    name: resources
```

```
spec:
  containers:
    - name: resource
      image: centos
      command: ["/bin/bash", "-c", "while true;
    do echo Technical-master; sleep 5 ; done"]
      resources:
        requests:
          memory: "64Mi"
          CPU : "100m"
        limits:
          memory: "128Mi"
          CPU: "200m"
```

Esc → :wq

→ kubectl apply -f podresources.yml

→ kubectl config view | grep namespace:

→ kubectl get pods

→ kubectl describe pod resources

→ kubectl delete -f podresources.yml

→ vi resourcequota.yml

```
apiVersion: v1
kind: ResourceQuota
metadata:
```

```
    name: myquota
spec:
  hard:
    limits.cpu: "400m"
    limits.memory: "400Mi"
    requests.cpu: "200m"
    requests.memory: "200Mi"
```

(Esc → :wq.

→ kubectl apply -f resourcequota.yml

→ vi testpod.yml

```
kind: Deployment
apiVersion: apps/v1
metadata:
    name: deployments
spec:
    replicas: 3
    selector:
      matchLabels:
        objtype: deployment
    template:
      metadata:
        name: testpods
```

```
labels:
  objtype: deployment
spec:
  containers:
    - name: coo
      image: ubuntu
      command: ["/bin/bash", "-c", "while true; do echo
"Technical-Master; sleep 5 ; done"]
      resources:
        requests:
          CPU: "200m"
```

Esc ⟶ : wq

→ kubectl apply -f testpod.yaml
→ kubectl get deploy
→ kubectl get pods
→ kubectl get rs

→ kubectl delete -f resourcequota.yaml
→ kubectl delete -f testpod.yaml

→ vi cpudefault.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-limit-range
spec:
  limits
    - default:
        cpu: 1
      defaultRequest:
        CPU: 0.5
      type: container
```

→ kubectl apply -f cpudefault.yml

→ kubectl apply -f pod.yml

→ kubectl get pods

→ kubectl delete -f pod.yml

→ vi cpu2.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo-2
spec:
  containers:
  - name: default-cpu-demo-2-ctr
    image: nginx
    resources:
      limits:
        cpu: "1"
```

ESC → :wq

→ kubectl apply -f cpu2.yml

→ kubectl get pods

→ kubectl describe Pod default-cpu-demo-2

→ kubectl delete -f cpu2.yml

→ vi CPU3.yml

```
apiVersion: V1
kind: Pod
metadata:
  name: mypod
spec:
  Containers:
    -name: mycontainer
     image: nginx
     resources:
       requests:
         cpu: "0.75"
```

Esc → :wq

→ Kubectl apply -f CPU3.yml

→ Kubectl describe pod mypod

→ Kubectl delete -f CPU3.yml

Kubernetes Horizontal Pod Autoscaler

→ Kubernetes has the possibility to automatically scale pods based on observed CPU Utilization, Which is horizontal Pod Autoscaling.

- Scaling can be done only for scalable objects like Controller, deployment or Replica set.

- HPA is implemented as a Kubernetes API resource and a Controller.

- The Controller periodically adjust the number of replicas in a replication Controller or deployment to match the observed average CPU Utilization to the target specified by user.

- The HPA is implemented as a Controlled loop with a Period Controlled by the Controller manager's -horizontal-pod-autoscaler-sync-period flag (Default Value of 30 sec).

- During each period, the Controller manager Queries the resource Utilization againest the metrices specified in each horizontalPod-Autoscaler definition.

- For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each pod targeted by the HorizontalPodAutoscaler.

- Then if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod.

- If a target raw-value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value across all targeted pods, and produces a ratio used to scale the number of desired replicas.

- Cool down period to wait before another downscale operation can be performed is controlled by --horizontal-pod-autoscaler-downtime-stabalization flag (Default value of 5 min)
  → changeable

- Metric server needs to be deployed in the cluster to provide metrics via the resource metrics API.

- Install metricserver

→ wget -O metricserver.yml https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

Now open the metricserver.yml file and insert
- --kubelet-insecure-tls under the args of cert.

→ kubectl apply -f metricserver.yml
→ kubectl get pods -n kube-system
→ kubectl logs -f metric-server ... -n kube-system

→ vi deployhpa.yml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: mydeploy
spec:
  replicas: 1
  selector:
    matchLabels:
      name: deployment
  template:
    metadata:
      name: testpods
```

```
labels:
  name: deployment
spec:
  containers:
    - name: c00
      image: httpd
      ports:
        - containerPort: 80
      resources:
        limits:
          cpu: 500m
        requests:
          cpu: 200m
```

ESC ⟶ :wq

→ kubectl apply -f deploywa.yaml

→ kubectl get all

→ kubectl autoscale deployment mydeploy --cpu-percent=20 --min=1 --max=10

— Now open one another window again(same instance)

→ kubectl exec mydeploy... -it -- /bin/bash

→ watch kubectl get all

In 2nd window of our instance we increase the CPU Utilization and check the autoscaling pods on our first window- Sudo apt-get update.

Kubernetes Jobs, Init Containers & Pod Lifecycle

Jobs

→ We have ReplicaSets, DeamonSets, StatefulSets and deployments they all share one common property: they ensure that their pods are always running. If a pod fails, the controller restarts it or reschedules it to another node to make sure the application the pods is hosting keeps running.

Use Cases

1. Take Backup of a DB.
2. Helm charts uses jobs.
3. Running Batch Processes.
4. Run a task at an schedule interval.
5. Log rotation.

LAB

Start your Instance and install minikube ... after go his

→ vi job.yml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    metadata:
      name: testjob
    spec:
      containers:
      - name: counter
        image: centos:7
        command: ["bin/bash", "-c", "echo Technical-Master; sleep 5"]
      restartPolicy: Never
```

ESC → :wq

→ kubectl apply -f job.yml

→ kubectl get pods

→ kubectl delete -f job.yml

Job 2

→ vi job2.yml

```
apiVersion: batch/v1
kind: Job
```

```
metadata:
  name: testjob
spec:
  parallelism: 5                  # Runs for pods in parallel
  activeDeadlineSeconds: 10       # Timesout after 30 sec
  template:
    metadata:
      name: testjob
    spec:
      Containers:
      - name: Counter
        image: centos:7
        command: ["/bin/bash", "-c", "echo TechniCal-Master; sleep 20"]
      restartPolicy: Never
```

ESC → :wq

→ kubectl apply -f job2.yaml

→ kubectl delete -f job2.yml

→ watch kubectl get pods

- The cron Job Pattern

→ If we have multiple nodes hosting the application for high availability, which nodes handles cron?

→ What happens if multiple identical cron jobs run simultaneously?

→ vi Cronjob.yml

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: master

Spec

Schedule: "* * * * *" (/

jobTemplate:

spec:

template:

spec:

Containers:

- image: ubuntu

name: master

command: ["/bin/bash", "-c", " echo Technical-master;

sleep 5"]

restartPolicy: Never

ESC → :wq

→ kubectl apply -f cronjob.yml

→ watch kubectl get pods

■ Init Containers

→ Init Containers are Specialised Containers that run before app Containers in a Pod.

→ Init Containers always run to completion.

- If a Pod's init Container fails, Kubernetes repeatedly restarts the Pod until the init container succeeds.

- Init Containers do not support Readiness Probe.

Use cases

→ Seeding a database

- Delaying the application launch until the dependencies are ready.
- clone a git repository into a volume.
- Generate Configuration file dynamically.

→ Vi init.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: init Container
spec:
  initContainers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "echo Master > /tmp/xchange/testfile; sleep 30"]
```

```
volumeMounts:
  - name: xchange
    mountPath: "/tmp/xchange"
containers:
- name: c2
  image: centos
  command: ["/bin/bash", "-c", "while true; do echo `cat /tmp
    /data/testfile`; sleep 5; done"]
  volumeMounts:
    - name: xchange
      mountPath: "/tmp/data"
  volumes:
  - name: xchange
    emptyDir: {}
```

Esc ⟶ :wq

→ kubectl apply -f init.yml

→ watch kubectl get pods

■ Pod Lifecycle

→ The phase of a Pod is a simple, high-level summary of where the Pod is in its lifecycle.

Pending

→ The Pod has been accepted by the k8s system, but its not running.

- One or more of the Container images is still downloading.

- If the Pod Cannot be scheduled because of resource constraints.

Running

→ The Pod has been bound to a node.

- All of the Containers have been created.

- Atleast one Container is still running or is in the process of starting or restarting.

Succeeded

→ All Containers in the Pod have terminated in success, and will not be restarted.

Failed

→ All Containers in the Pod have terminated and atleast one Container has terminated in failure.

- The Container either exited with non-zero status or was terminated by the system.

Unknown

→ State of the Pod could not be obtained.

- Typically due to an error in network or communicating with the host of the Pod.

Completed

→ The Pod has run to Completion as there's nothing to keep it running.

eg:- Completed jobs

Pod Conditions

→ A Pod has a PodStatus, which has an array of PodConditions through which the Pod has or has not Passed.

→ Using 'Kubectl describe Pod PODNAME' you can get Condition of a Pod.

These are the possible types →

PodScheduled

The Pod has been Schedule to a node.

Ready

The Pod is able to serve request and will be added to the load Balancing Pods of all matching services.

Initialized

→ All init Containers have started successfully.

UnScheduled

→ The Schedular Cannot Schedule the Pod right now, for eg- due to lacking of Resources ~~are~~ or other Constraints.

ContainerReady

→ All Containers in the Pod are ready.