Chef -> Through it we can configure linux machines (windows)

Ansible -> Through it we will only configure linux machine
gmail.com - web

In Interview they ask to write these

1) Writing recipe (chef)
2) Writing Dockerfile
3) playbook

Need to practise

Jira tool
to write the tickets

LINUX

Windows → win DOS, win 98, win xp, win vista, win 7, win 8, win 10

UNIX → Linux, HP-UX, AIX, MAC OS, Solaris

Linux → Fedora, RHEL, centos, Amazon linux, Ubuntu (Teamwork meaning) — 20 flavours of UNIX

Linux

1) open source

2) Security → hacker (less than window, because the too many commands line)
→ virus (whenever the virus is being downloaded with that file then at that particular file or folder are affected)

3) Customize → To modify in repective with our needs

Scanned with CamScanner

over all 3.8 years of experience
Knife rules from like mobile/network

4) Less resources

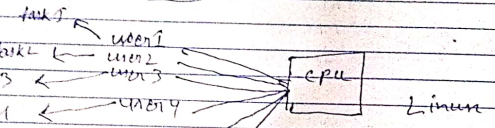
Song → MP3
→ MP4

Linux → CLI

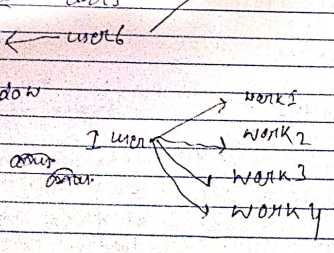
Window → CLI + GUI

1 GB of Linux = 50 GB of window
approx

5) Multiuser - multitask

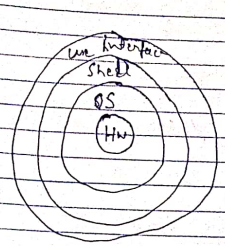


Window

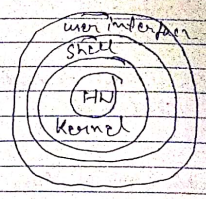


Architecture

Window

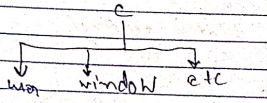


Linux

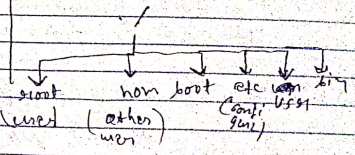


heriarichy

Window



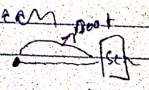
Linux



→ window → software

Linux → package

Boot → It is process taken from power on the button to on the screen



→ All package are stored under usr

Scanned with CamScanner

over all 3.8 years of experience
Knife rolls from file manipulation

There are four command to create files

-> cat -> cat > file1
-> touch ->
-> nano ->
-> vi/vim

-> To come out from the file
ctrl + d

cat > Ram Hanten
This is gud boy
ctrl + d

-> To add some more content in the existing file
cat >> filename (that is existing)

-> To open the file
cat filenam

Touch

-> It is used to create an empty file nam

touch file2

if we have to add content in this empty file which is created by touch. we have to use cat
cat > filenam

Nano

-> To create and inside the file
Nano filenam

-> To add some more content again
Nano filenam
-> Have to save the file
-> To exist out from the content
ctrl X

vi commands

-> To creat the file
vi filenam
-> To write the content in the file i (becaus to go insert mode from the command mode)
-> To com out from content file insert to command esc (button)

-> :wq! is used to exist from the content to command prompt

Cat → cant edit

touch → to creat empty files

Nano
vi } → Editors

Directory

→ To Create a directory

mkdir dirname (# mkdir dir1 dir2

→ To Create multiple directory

mkdir name1 name2 ....

Hidden file & Directory

put . in front of the command

touch .file5

mkdir .dir2

To check the hidden file & Directory

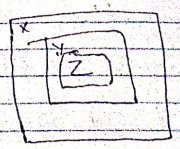
→ # ls -a

To check the full details of file and directories and also hidden file and directory

→ # ls -la
ls -al

To Create directory inside the another one

→ # mkdir -p dir1/dir2/dir3



To change the directory

cd directoryname

To go back in previous directory

→ # cd ..
cd ..dir2

To go directly from the existing directory

cd dir1/dir2/dir3
from X to from Z

To go multiple back directory

dir2] # cd ../../..

ec2-user] # c

#history → it shows all the Command which we have used

To copy one into another

cp source filename Destination file

cp file1 dir1

To cut and paste

mv source destination
mv file2 dir1

To rename

mv old name New name

This will not be present

means this file is not present and that dir1 because if it is present it will copy

To delete

rm -rf file or dir name

To show complete structure

tree
First we need to update
yum install

path are created the file without going directly to that file or dir

touch ~~dir1~~ filename1/filename2/filename3/ name
path
same as directory
mkdir dir1/dir2/dir3

→ Through path we can go directly to that file we can edit in that file from the existing or current file or directory

→ To pick the particular one word

~~# pick~~ # grep root /etc/passwd
word path

To open the file But we can't add the data

less

Exit from this quit (q)

Head

To see the lines from top

Ex

head -3 /etc/passwd

→ root : x : 0 : 0

bin : x : 1 : 1

daemon : x : 1 :

tail

To see the lines from the bottom

tail

Sort (# sort)

To arrange in the alphabetically

Ex

zn Jldn

x h fma

c fm

b neagn

n lnqdn

mK sng

sort test/file

a Jindin

bnean

cf

It only arrange in alphabatically

Not modified it

To see if

ifconfig (it will show more details)

hostname -i (only ip will be show)

To see the which operating system is being used

cat /os-release

it help to find the file with have pattern match

cat path filename

Scanned with CamScanner

install the package

yum install packageNam

update the packag

yum update packageNam

Remove the packag

yum remove packageNam

Listed of package which are installed in the machine

yum list installed

To start the service

service httpd start

To stop the service

service httpd stop

To restart an upgrade the service

service httpd restart

Sudo su -> what ever a super user do switch to that

if we restart the machine the service will be start automaticly

chkconfig httd on

chkconfig httpd off (to stop service permanently)

Echo

It will create the file and put the content on it

Ex-> # echo "sai" > mynewfile
↓ with file note exitted

After enter

mynewfile
↓
(sai) content

To find out the package

which packagenam

How to add users

#useradd hari

~~List of users~~ To check users

#cat /etc/~~passwd~~ passwd

To create a group

#groupadd name

To verify

#cat /etc/group

To add users in group

#gpasswd -a hari devops

user name group name

To verify

cat/etc

Tar -> it is used to pack

zip -> it is used to compressed

To make a softlink

ln -s filename softlinkfilename

Eg:- ln -s myfile softmyfile

-> if in ~~main~~ Main file will be edited or save content than in softlink also be copy

-> if we delete mainfile, softlink data will removed

Hard link

Eg # ln testfile hardtestfile

-> # ln filename hardlinknamefile

In hardlink if in mainfile we add the data or removed than it will also do same in harddink

-> if we delete the mainfile than there is no ~~defect~~ effect in the Hardlink

Scanned with CamScanner

Location of file	Location
dir	dir1
↓	↓
dir.tar	dir.tar
↓	↓
dir.tar.gz	dir.tar.gz

tar -cvf dir1.tar dir1

Tar -> it is used to packed
Zip -> it is used to compress
To convert the file into tar and zip

tar -cvf [illegible] [illegible]
file name file name

Extract the tar and zip

gunzip dir1.tar.gz
file name

tar -xvf dir1.tar
file name

To download the package from yum

yum install

Install the package

yum install

To verify the package installed or not

which chef

To delete entire file and directory

rm -rf *

Notation

drwxr-xr-x 2 root root 6 Sep 17 04:22 dir1
-rw-r--r--. 1 root root 6 Sep 17 04:22 file
↓ linked file

owner	group	other
rwx –	r-x	r-x
4+2+1	4+1	4
7	5	5
rw-	r-	r-
4+2	4	4
6	4	4

r – read – 4
w – write – 2
~~read~~ –
x – execute – 1

To give the permission of file

chmod 777 filename
chmod 700 filen
chmod 666 dirnc
chmod 764 dirni

To change the owner

chown ownernamefile filenameton to change ownerto normal user

To change the group

chgrp

DevOps Tools

GIT

SCM (Source code managment)

-> Storage (Any code)
  A) Developers (code)
  B) Testing team (test)
  C) DevOps (scripts)

-> Different people from different teams can store simultaneously (Save all changes separately)

-> pipeline b/w off shore & onshore teams

-> Helps in achieving team work

-> Track changes (Minute level)

Scanned with CamScanner

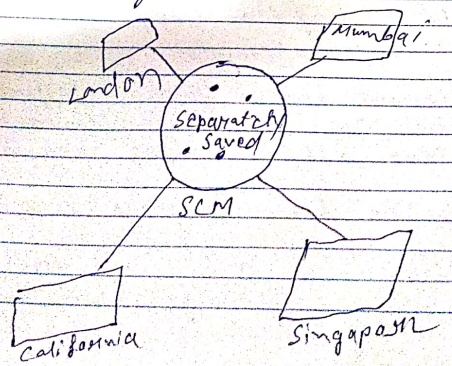
18/9/19

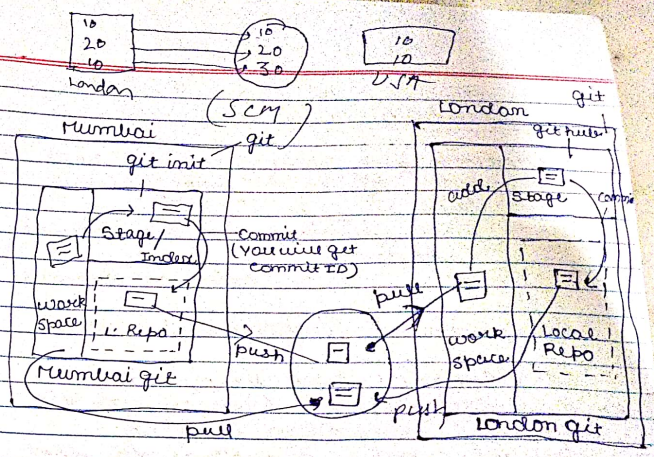
SCM Tools

- → Git (Most advanced tool)
- → SVN
- → perforce
- → Clearcase

Important Terminology

- → Repository / Depot
- → Server
- → Work space / work dir / Work tree
- → Branch / Trunk / code line
- → ~~Co~~ Commit.
- → Commit / check-in
- → version / version-ID / commit-ID
- → Tag





Important Terminology

- → Repository
  - A) Storage (Folder) related to one product
  - B) changes are personal to that particular repo
- → Server
  - A) Stores all repos
  - B) Metadata (products, people...)

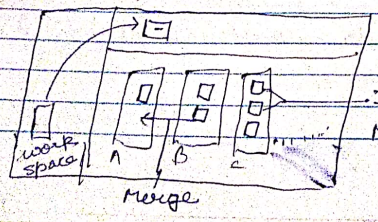
Scanned with CamScanner

Branch

→ product is same, so one repo but different feature features.
tasks
→ Each ~~task~~ task has one seperate branch
→ Finally merge (code) all branches

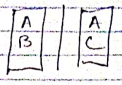
→ For parallel development

→ can create any no of branches
→ can create one branch on the basis of another branch
→ Changes are personal to that particular branch
→ can put files only in branches (not in repo directly)
→ Default branch is "Master"
→ Files created in workspace will be visible in any of the branch workspace untill you commit. once you commit, then that file belongs to that particular branch.



→ If working fine Merge with A

Note → For first time the Data is copy after that it will be personal



(A is copy but when we change B and C it not be copied)
B and C are kept seperately in respective branch

Important Terminology

→ Work space / working Directory

A) Where you work
B) Where you see files physically & do modification
C) Same & common workspace
d) At a time points to particular branch

→ Version / version-ID / commit-ID

A) Reference to identify each change
B) To identify each change and who did that change.

Tag
A) Meaningful name (we cannot remember commit-ID)

Scanned with CamScanner

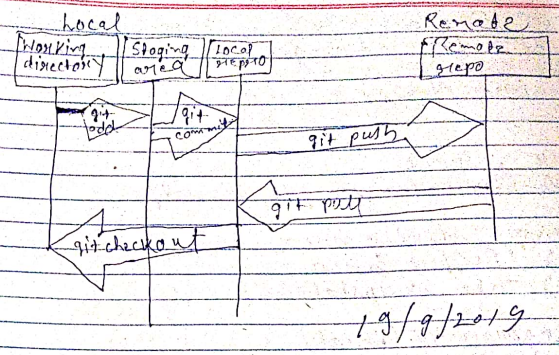
Why only Git?

-> Speed
A) Snapshots concept

-> parallel branching
A) Multiple branches at a time unlike other SCM tools

-> Fully Distributed

A) Backup copy is available in multiple location.
B) No Need internet connections so No network latency.
C) No need central server separately.
D) Each work space will have its own repo internally.
E) Can create any No of branches (No need to take prior permissions)
F) Can share code without using Central repo
G) That's why we call as DVCS (Distributed version control system)



19/9/2019

Terminology

-> Snapshots
A) Get any previous version (Backup)
B) Represents some data of particular time
C) Stores the changes (appended data) only. Not whole copy.

-> Commit.
A) Store changes in repo (Will get commit ID)
B) 40 Alpha - numeric characters
C) Concept - Checksum (It's a tool in Linux generates binary value equal to data present in file)
D) Even if you change one dot, commit ID will be changed

V.V.I NOTE :- ANS -> In this snapshot contain whole backup
DEVOPS -> In this snapshot contain incremental Data

- Tracks the changes.

Git stages

-> Work space
A) physically see file & Modify

-> Staging / Indexing area
A) Buffer area
B) Take snapshot

-> Respository (Local)
A) Store changes Locally

-> Respository (central)
A) Store changes Centrally

Scanned with CamScanner

Git Commands (in mumbai EC2 machine)

-> Create directory & go inside that
A) mkdir mumbaigit, cd mumbaigit
B) git init (to initialize git)

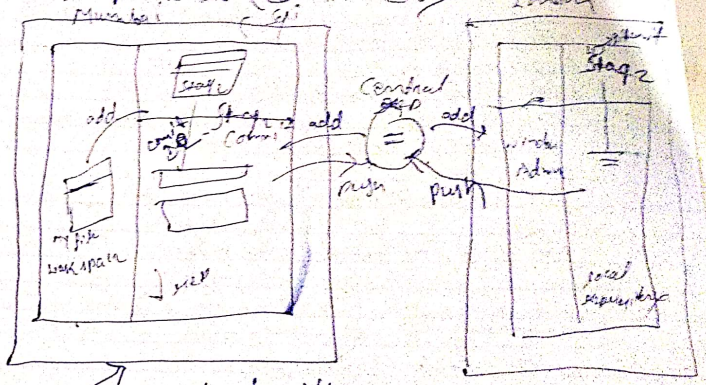
-> Create new file, see status, put in staging area & commit into local repo
A) touch my file (put some content)
B) git status
git add
C) git commit -m "1st commit from mumbai"

D) git log
E) git show <commitid>
F) git remote add origin <centralgit repo url>
G) git push -u origin master (give github credentials)
• verify in github. Can see the file.

H) (add some content to the file and repeat the same process. But need not to add again to central repo)

I) (observe the difference b/w untracked file & modified file)

J) git log
K) git show <commit-ID>



20/9/2019

Git Commands (In London EC machine)

-> Create directory & go inside that
A) mkdir londongit
B) git init (to initialize git)
C) git remote add origin <centralgit repo> repo url>
D) git pull origin master
E) git log
F) git show <commit-ID>
(ii) cat >> myfile (append with some content)
1) git status
2) git add.
3) git commit -m "1st commit from londongit"
4) git push -u origin master

Git Commands (in Mumbai EC machine)

-> To pull latest changes
A) git pull origin master

-> Again append with some content, add, commit & push.
A) cat >> myfile (append with some content)
B) git status
C) git add.
D) git commit -m "ANY commit msg"
E) git push origin master

Git Commands

-> To ignore the file while commiting (Retrospective effect is there)

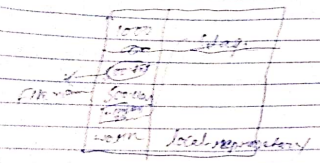
- .gitignore
- steps
  A) Create .gitignore file and give pattern matching (eg: *.class)
  B) Add and commit .gitignore file
  i) Create some class files and Java files for testing and add them by running "git add."

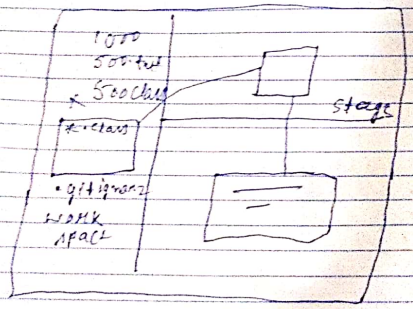
-> Git log options.
- git log
- git log -2
- git log --oneline

-> To pic commit based on commit msg
- gitlog --grep "any word of commit msg"

-> To see the content of particular commit
- git show <commit-ID>

Scanned with CamScanner





if ~~a have to show log 1~~

git log_1 (shows 1 latest

git log_2 (shows 2 latest)

23/3/2019

Branching

-> To see list of available branches (must have some thing)

A) git branch

-> Create a new branch

A) git branch <branchname>

-> To switch branch

A) git checkout <in which branch you want to go>

-> verify the content in both branches

-> Add content in both branches

-> Files are not personal to branch until you commit.

(Branching merge)

-> you can not merge branches of different repositories.

-> To merge branches (pulling mechanism)

A) git merge <branch name>

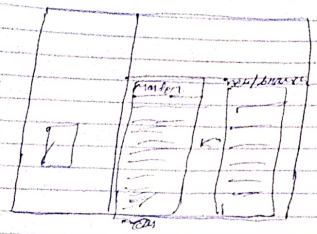
-> verify the merge

A) git log

-> push to central ~~repo~~ repo

A) git push origin master

Scanned with CamScanner



Git conflict only

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

→ Conflict occurs when
A) merging branches.

Git stashing

→ It removes content inside file from working directory and puts in stashing stage and gives clean working directory so that we can start new work freshly.

→ Later on you can bring back that stashed items to working directory and can resume your work on that file.

→ To stash an item (only applies to modified files, not new files)
A) git stash

Git Stash

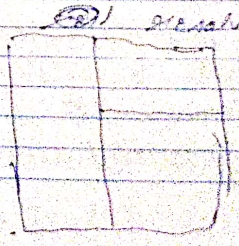
→ To see stashed items list
A) git stash list

→ To apply stashed items
A) ~~go~~ git stash apply stash @{number}
B) Then you can add & commit

→ To clear the stash items
A) git stash clear

Steps to be followed

A) Create any blank file add and commit
B) put some content, then run "git stash" (repeat twice)
C) Go to stash repo for verification.
D) Get from stash repo and then add and commit (if conflict occurs, resolve conflicts)



90

git stash apply stash

24/9/2019

Resetting changes (Before commit)

-> To reset from staging area

A) git reset <file name>

~~B) git reset~~

B) git reset.

-> To reset the changes from both staging area and working directory at a time (not individually)

~~A) git reset hard~~

A) git reset --hard

Reverting changes (After commit)

-> To revert the changes

A) git revert <commit-ID>

-> Revert will create a new commit-ID (all changes will be tracked)

-> You need not to add and commit again. Automatically new commit-ID will be generated.

Removing Files

-> To remove files

A) git rm <file name>

(Then commit (no need to add))

(It gets deleted from that particular branch)

Delete all untracked files

-> git clean -n (dry run)

-> git clean -f

Tag

-> To apply tag

A) git tag -a <tagname> -m <tag message> <commit-id>

-> To see the list of tags

A) git tag

-> To see particular commit content by using tag

A) git show <tag name>

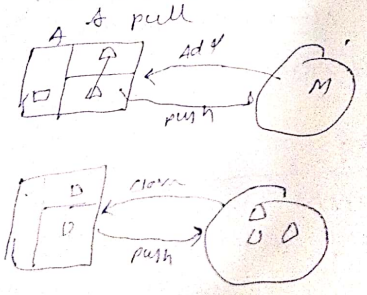
-> To delete a tag

A) git tag -d <tag name>

Scanned with CamScanner

Git Hub (clone)

-> Go to www.github.com
-> Choose an existing repository.
-> Run below command in local machine's current directory
A) git clone <URL of github repo>
-> Go inside that repo.
-> Then onwards, we can push & pull



Git Hub (branches & merging)

-> Create new branch from master (new branch)
-> Switch to new branch & create an new file & commit in new branch.
-> observe the differenced b/w master branch and new branch
-> To merge branches in git hub itself, Click on
A) New pull request
B) Create pull request
C) Merge pull request
D) Confirm merge
-> Go to master branch. You can find new file that u created in new branch

Scanned with CamScanner

Configuration Managament

→ Configuration Management

A) it's a method through which we automate admin tasks.
B) Configuration management ~~tool~~ tool turns your code into infrastructure
C) So your code would be Testable, Repeatable & Versionable.

→ Infrastructure refers to the composite of:

A) Software
B) Network
C) people
D) process

→ pain points:

A) managing users & group accounts
B) Dealing with packages
C) Taking backup
D) Deploying all kinds of applications.
E) Configure services.

Why Configuration Managment Tool?

→ Why configuration Managment Tool?

A) Complete Automation
B) Increase uptime
C) Improve performance
D) Ensure ~~Complain~~ Compliance
E) prevent Errors
F) Reduce Costs.

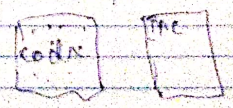
Configuration example

System admin Config

- files
- folders
- users
- group
- permission
- package
- services
- backup

→ configuration → manual by System Admin / automated by DevOps

IAC (Infrastructure as a code) through it in DevOps we can do automatically many code in small tap of time



Configuration Management

→ Configuration Management

A) it's a method through which we automate admin tasks.
B) Configuration management ~~tool~~ tool turns your code into infrastructure
C) So your code would be Testable, Repeatable & Versionable.

→ It infrastructure refers to the composite of:

A) Software
B) Network
C) people
D) process

→ pain points:

A) managing users & group accounts
B) Dealing with packages
C) Taking backup
D) Deploying all kinds of applications.
E) Configure services.

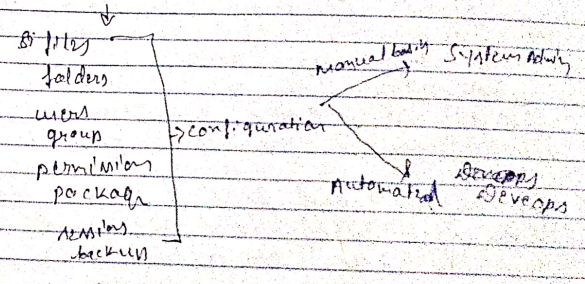
Why Configuration Management Tool?

→ Why configuration Management Tool?

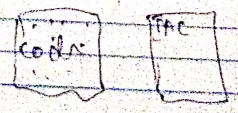
A) Complete Automation
B) Increase uptime
C) Improve performance
D) Ensure ~~Complain~~ Compliance
E) prevent Errors
F) Reduce Costs.

Configuration example

System admin Config



IAC (Infrastructure as a code)
Through it in Devops we can do automatically many code in small lap of time.



Scanned with CamScanner

Why Chef?

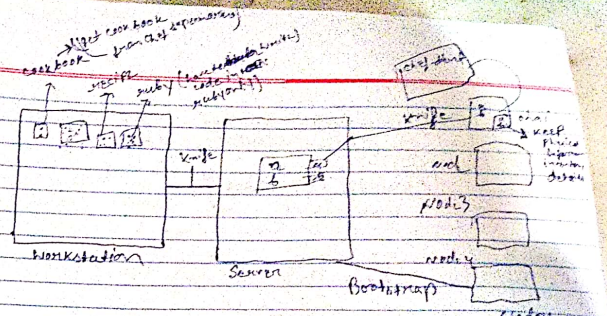
-> Chef is an administration tool. What ever system admins (Linux/windows) used to do manually, now we are automating all these tasks by using chef (Any CM Tools)
Eg: Waterpots

-> Can use this tool whether 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 applications.
A) Your code is versionable.
B) Your code is repeatable.
C) Your code is testable.

-> You only need to tell what the desired configuration should be; not how to achieve it

-> Through automation, get desired state of server..



Important Terminology

-> Chef workstation : Where you write code

-> Chef server : Where you upload code

-> Chef Node : Where you apply the code

-> Knife : Tool to establish communication among workstation, server & node.

-> Chef-client : Tools runs on every chef node to pull code from chef server

-> Ohai : Maintains current state information of chef node (System Discovery Tool)

-> Idempotency : Tracking the state of system resources to ensure that the changes should not re apply repeatedly.

Scanned with CamScanner

-> Chef supermarket : Where you get custom code

Install Chef

-> Go to chef.sh
-> Go to Downloads
-> ~~at~~ click on chef workstation
-> Go to Red Hat Enterprise Linux
-> copy unit of RHEL 7
a) wget unit
b) rpm install chef-workstation-x86_64.rpm

-> verify chef installation
A) chef -v (chef -v)
B) chef -version (chef -version)

Cookbook

-> Cookbook is a group of recipes and some other files & folders
-> Each cookbook defines a scenario.
e.g. web server cookbook

-> Create a new directory
A) mkdir cookbooks
-> Go inside to create a cookbook (what ever you do, do inside cookbooks folder)
A) Chef generate cookbook test-cookbook
B) time test-cookbook

Inside Cookbook :

A -> Chefignore : like .gitignore
B -> Kitchen.yml -> For testing cookbook
C) Metadata.rb -> name, author, version etc of cookbook
D) Readme.md -> info about usage of cookbook
E) Recipe -> Where you write code
F) Spec -> for unit test
G -> Test -> For integration test

(~~create~~ ->

To get root from directory -> cd:

Create your first Cookbook & Recipe

-> Already created cookbook (test-cookbook)

-> Go inside test-cookbook (But not inside recipes folder) & create a recipe

A) chef generate recipe test-recipe
name

-> vi test-cookbook/recipes/test-recipe.rb

```
file '/myfile' do
    content "Hello Dear Students!!"
    action :create
end
```

-> To verify the code (be inside cookbook folder)

a) chef exec ruby -c test-cookbook/recipes/test-recipe.rb

NO Apache installed!!

-> Apply that recipe locally (Be inside cookbooks folder)

A) Sudo chef-client -zm "recipe[test-cookbook::test-recipe]"

-> ~~ver~~ verify

A) ls

```
file '/myfile' do
    content "1st one stone !!
             2nd stone !!
             3rd stone !!
             4th stone !!
             5th stone !!"
    action :create
end
```

3-4 STR (If start have hat to ~~end~~ end
" " " " " " " " end

Create & write your second recipe

-> Go inside cookbooks (But not inside recipes folder) & create a recipe

A) chef generate recipe demo-recipe

-> vi test-cookbook/recipes/demo-recipe.rb

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

file '/myfile2' do
    content 'This is my second line'
    action :create
    owner 'root'
    group 'root'
end
```

-> Apply that recipe locally (be inside cookbooks folder)

-> A) Sudo chef-client -zm "recipe[test-cookbook::demo-recipe]"

Verify

A) ls /

B) which tree

Scanned with CamScanner

Deploying an Apache web server

-> Go inside cookbooks folder & create a cookbook

A) Chef generate cookbook apache-cookbook

-> Go inside cookbook (but not inside recipes folder) & create a recipe

A) Chef generate recipe apache-recipe

-> Go inside apache-recipe

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

file '/var/www/html/index.html' do
    content 'Hello Dear Students!!'
    action :create
end

service 'httpd' do
    action [:enable, :start]
end
```

This situation because that machine to we have to enable service & we give this command which automatically enable & start

-> Apply that recipe locally (be inside cookbooks folder)

A) sudo chef-client -zr "recipe[apache-cookbook::apache-recipe]"

verify -> paste the public IP in browser

Ohai

-> It is a system discovery tool.

-> It gathers system information

A) ohai

-> Now run

A) ohai ipaddress
B) ohai hostname
C) ohai memory/total
D) ohai cpu/0/mhz
zero

ATTRIBUTES

-> We have a web application to be deployed into 1000 nodes & we need to know some details of each server.

-> Because we need to mention that in configuration file of each server. This information is vary from system to system.

-> These details we call 'Attributes'

-> Chef-Client tool gathers these Attributes from ohai store and

file "/root/file" do
content "This is to get Attributes
~~HOSTNAME~~ HOSTNAME : #{node['hostname']}
IP ADDRESS : #{node['ipaddress']}
CPU : #{node['cpu'][0]['real']}
~~MEMORY : #{node['memory']['total']}~~
owner "root"
group "root"
action : create
end

puts in configuration files

-> Instead of hard coding thus attributes, we mention as variables

1/10/2019 Execute

Note -> Chef only consider root so if we get linux we have to use it

-> To execute Linux commands

```
execute "run a script" do
    command <<-EOH
    mkdir /saidir
    touch /saifile
    EOH
end
```

imp

User & Group

-> We can create users & groups

For users

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

verify: cat /etc/passwd

For group

```
group "devops" do
    action : create
    members 'raj'
    append true
end
```

verify: cat /etc/group

Chef-Client

-> We run chef-client to apply recipe to bring node into desired state.

-> We call this process as "convergence"

RUN List

-> Run the recipes in a sequence order that we mention in Run list

-> To apply each recipe from each cookbook (not multiple recipes from same cookbook) at using chef client (be in cookbooks folder to run the command)

A) Sudo chef-client -z -r "recipe[cookbook name::recipe name]"

A) sudo chef-client -zn "recipe[cookbook name
::
"recipe name]recipe[cookbook name :: recipe name]"

Chef Include - recipe

-> To call recipe/recipes from another recipe with in same cookbook

-> To run multiple recipes from same cookbook

-> Here comes the default recipe into action (can use any recipe)
A) Add below line in default.rb
B) include_recipe "cookbook :: recipe"

-> Now run (Be inside cookbooks folder)

A) sudo chef-client -zn "recipe[Cookbook name :: recipe name]"

3/10/2019

Chef server

-> chef server is going to be a hub for the code (cookbooks)

-> Nodes pull code (cookbooks) from chef server
A) sign up for chef-server account
B) Attach your workstation to chef server
C) Upload cookbooks to server
D) Bootstrap nodes
E) Apply cookbooks to nodes.

- Getting started with chef server

-> Signup in https://manage.chef.io
A) Get verification email. So get verified.

-> Create a new organization
A) Give full name
B) Give short name (must be unique) (we get url on this short name)
C) Now you can see the dash board.

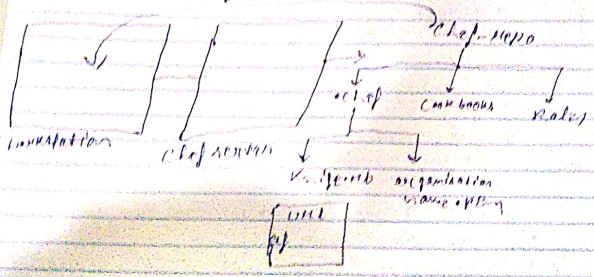
-> To establish communication b/w chefserver & workstation
-> Go to
A) Administration
B) organization

docs.chef.io → learning chef (coding)

1) Download starter kit
A) Unzip it, you can see chef-repo
2) move chef repo to chef workstation (under ec2-user directory) (by using winscp tools)

→ See the components of chef-repo

→ To verify whether communication established or not (be inside chef-repo)
A) knife ssl check



[Bootstrap a node]

→ Attaching a node to chef server is called Boot strapping (make sure, both workstation & node are in same AZ)

→ Now onwards, you have to be inside chef-repo directory to run any command.

→ The ~~command~~ actions will be done while bootstrapping
A) Adding node to chef server
B) Installing chef packages
(copy pem key of node into chef-repo folder in workstation)

Knife bootstrap <node-private-IP> -x <ssh user ec2-user> -i <.pem> -N <any-node-name (avoid hyphens name)>

→ Bootstrap 2 nodes
→ To see bootstrapped nodes
A) Knife node list.

This is only name of key that under this key pair and is present to my use of winscp. We have to move that pem key into chef-repo (current directory)

Cmd: Knife bootstrap IPL.

4/10/2019

Upload cookbook to server and apply to nodes

-> All cookbooks (apache & test..) must be inside cookbooks folder. Comes by default with chef-repo folder to upload to chef server.

-> Never ever delete the default cookbooks folder comes by default with chef-repo folder.

-> To upload cookbook to chef server
A) Knife cookbook upload <cookbook>

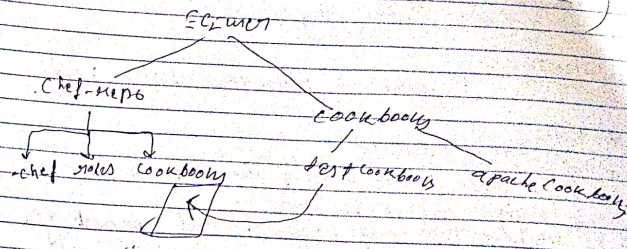
-> To verify cookbook uploaded or not (to see list of cookbooks which are present in chef server)
A) Knife cookbook list

-> To add cookbook's recipe to node (see runlist) (do it to both nodes)

A) Knife node run list set <node-name> "recipe[cookbook::recipe]" (~~to remove~~ then (to remove "remove")

-> To verify whether cookbook's recipe added to node or not (do it to both nodes)
A) Knife node show <node name>

-> Go inside node & run (do it to both nodes)
A) Chef-client -> ~~when only this command~~



Running chef-client in regular interval

-> Go to node, open "crontab" file inside /etc directory & give
* * * * * root chef-client
(in bootstrap script, can give below command to automate completely) ~~echo~~
echo "* * * * * root chef-client" >> /etc/crontab

-> Modify the recipe and upload that cookbook to chef server for testing

Scanned with CamScanner

Command to delete node
-> Knife node delete node
-> Knife cookbook delete apache-cookbook
-> Knife cookbook list

4) Knife cookbook upload <cookbook>

(Adding cookbook" recipe to node" run-list is one time effort. Need not to add again and again)

10/10/2019

Important Advanced topic (Roles)

* Roles:
  * custom run-list
  * We will create role & upload to chef server & assign them to node
* If we have 1000 nodes, need to add cookbook to run-list all 1000 nodes. It is very difficult instead, created role to 1000 nodes once. Next time onwards add cookbook to that role. Automatically all 1000 nodes will get the cookbook.
* So role is one time effort.
* instead of adding cookbooks to each & every node runlist always. Just create a role & attach that role to node.
* When we add cookbook to that role it will be automatically apply to all node that assingend with that role

Roles

Roles

* Terminate & delete node and apache-cookbook from chef server
* Be ready with fresh nodes, bootstrap them and automate chef-client
* Be inside chef-repo
* vi roles/web.rb
  name "web"
  description "Web server role"
  runlist "recipe[apache::cookbook::apache-recipe]", "recipe[test-cookbook::test recipe]", "recipe[cookbook-name]"
* upload role to chef server (every time you modify the role) (run inside chef-repo)
  Knife role from file roles/web.rb
* Attach role to all nodes (on one time effort) run inside chef-repo)
  Knife node run-list set <node-name> "role[web]"
  Knife node run-list set <node-name> "role[web]"
* To upload cookbook (everytime no modify) (run inside chef-repo)
  Knife cookbook upload apache-cookbook
* paste public IP of both nodes in web and verify

Test

* modify recipe
* upload cookbook

Scanned with CamScanner

4-10-19

Clean up the chef-server

* To see list of cookbook which are present in chef server
  - knife cookbook list
* To delete cookbooks from chef server
  - knife cookbook delete OK (cookbook name)
* To see list of nodes which are present in chef server
  - knife node list
* To delete nodes from chef server
  - knife node delete (node name)
* To see list of clients which are present in chef server
  - knife client list
* To delete clients from chef server
  - knife clients delete (clientname)
* To see list of roles which present in chef server
  - knife role list
* To delete roles from chef server
  knife role delete (node name)

Chef extra commands

* To upload all cookbook to chef server in a one go
  - knife cookbook -- all
* To deal with multiple resource at a time

```
['httpd', 'mariadb-server', 'unzip', 'git', 'vim']
each do |p|
package p do
action: install
end
end
```

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

10-10-2019

Roles

-> Terminate nodes & delete node and apache-cookbook from chef server.
-> Be ready with 2 fresh nodes, bootstrap them and automate chef-client.
-> Be inside chef-repo
-> vi roles/web.rb
```
name "web"
description "web server role"
run_list "recipe[apache-cookbook::apache-recipe]", "
```
No space
```
recipe[test-cookbook::test-recipe]", "recipe[
```
underbracket cookbookname]"
-> upload role to chef server (every time you modify the role) (run inside chef-repo)
Knife role from file roles/web.rb
-> Attach role to all nodes (one time effort) (run inside chef-repo)
* Knife node run_list set <nodename1> "role[web]"
Knife node run_list set <nodename2> "role[web]"
-> To upload cookbook (every time you modify) (run inside chef-repo)
Knife cookbook upload apache-cookbook
-> pass public IP of both nodes in web and verify

Test

* Modify recipe
* upload cookbook

11-10-2019

Chef extra commands

-> To upload all cookbooks to chef server in one go
Knife cookbook upload --all
-> To deal with multiple resources at a time

```
['httpd', 'mariadb-server', 'unzip', 'git', 'vim'].each do |p|
  package p do
    action :install
  end
end
%W(httpd mariadb-server unzip git vim).each do |p|
  package p do
    action :install
  end
end
```

PTO

Scanned with CamScanner

Clean up the chef-server

-> To see list of cookbook which are present in chef server

Knife cookbook list

-> To delete cookbooks from chef server

Knife cookbook delete <cookbook-name>

-> To see list of nodes which are present in chef server

Knife node list

-> To delete nodes from chefserver

Knife node delete <node-name>

-> To see list of clients which are present in chef server

Knife client list

-> To delete clients from chef server

Knife client delete <client-name>

-> 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>

Scanned with CamScanner

Important differences?

1) Container: A container like a virtual machine

2) Docker: Docker is a tool to create these virtual machines

What is Docker??

-) Docker is a tool that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and was developed by Docker Inc

-) Docker is a tool used to create virtual machines called "containers".

Docker

-> Took from shipping containers.

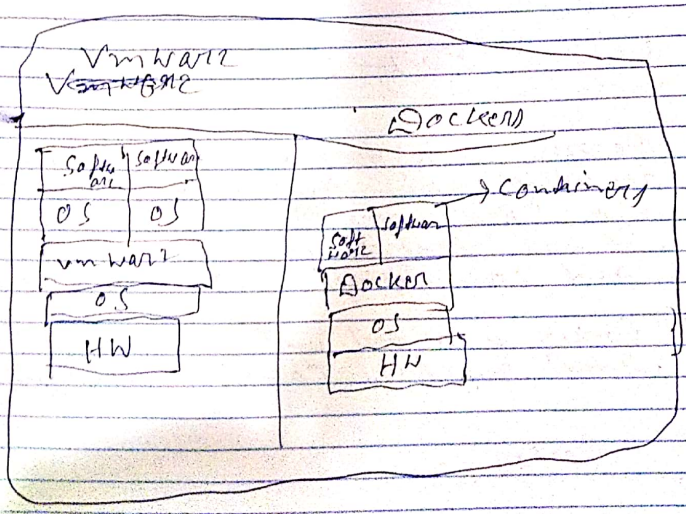
-> Docker is a tool designed to make containers in which we can deploy any type of

Scanned with CamScanner

applications easily.

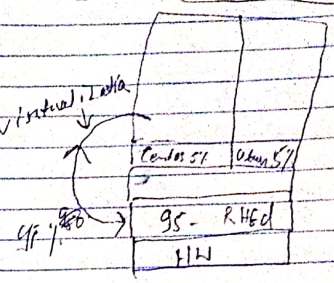
-> Docker uses union file system (layered)

-> Docker performance OS level virtualization



note:-) In Docker there is no pre-allocation of Ram

OS-level virtualization



Scanned with CamScanner

Docker benefits

-> Containerization (OS level virtualization (No need guest OS)

-> No pre-allocation of RAM (It take from base machine after finishing work return to base machine)

-> Can replicate same environment

-> Less cost

-> Less weight (MB's in size)

-> Fast to fire up

-> Can run on physical/virtual/cloud

-> Can re-use (same image)

-> Can create machines in less time.

Docker Components

-> Docker image: Contains OS (very small) (almost negligible) + soft-wares

-> Docker container: Container like a machine which is created from Docker image.

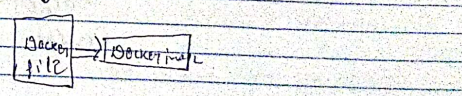
-> Docker file: Describes steps to create a docker image.

-> Docker hub/registry: Stores all docker images publicly.

-> Docker daemon: Docker service

Developer

↓



Scanned with CamScanner

Ways to Create Docker images

-> Take image from Docker hub
-> Create image from existing docker Containers
-> Create image from dockerfile.

Develop
↓
Dockerfile -> Dockering -> Dockerhub
↓
Dockercontainer

pullimage

Docker img
↓
Docker conflict
Dockering
↓
Docker Contly

Docker Download & install

-> wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
-> yum update -y
-> yum install docker -io -y
-> service docker start
-> chkconfig docker on
-> which docker

Docker installation

-> uname -r (OS)
-> yum update -y
-> yum install docker -y
-> docker (to verify)
-> docker --version (to verify)
-> docker info (running or not)
-> service docker start
-> ~~docker info~~
-> docker images (to see all images in our machine)

Scanned with CamScanner

-> docker ps (to see only running containers)

-> docker ps -a (To see all containers)

How to create container?

-> To create container

-> docker run -i -t ubuntu //bin/bash

-> i -> interactive mode

-> t -> Terminal

-> // -> s/by (some OS's s/will be neglected) so // will given path of one (unidentified)

-> hostname

-> cat /etc/os-release

-> exit

Docker Basic Commands (Images)

-> To see all images present in your local machine

A) docker images

-> To search image in online docker registry

A) docker search ubuntu

-> Download image from online docker registry to your machine

A) docker pull Jenkins

B) docker pull chef/chefdk (create container from this image)

Scanned with CamScanner

Docker Basic commands

-> Create Container by giving container name

A) docker ~~run~~ run --name saicontainer -it ubuntu //bin/bash

-> To start Container

A) Docker start saicontainer

-> To go inside Container

A) docker attach saicontainer
(First start than go inside the container)

-> To see all containers

A) docker ps -a

-> To see only running Containers

A) docker ps

-> To stop Container

A) docker stop saicontainer

-> To delete Container

A) docker rm saicontainer
(First stop than delete)

Note -> docker run --name saicontainer -it ubuntu //bin/bash
To keep the machine in run mode and exit back (only put d in place of i) — inplace of i put d

Create Container from own image

-> Create Container

A) docker run --name saicontainer -it ubuntu //bin/bash

B) touch /tmp/myfile

C) docker diff saicontainer (run outside of container)

-> To create image from Container (Be inside base machine)

A) docker Commit saicontainer

-> To give a name to your image

A) docker tag <image ID> saiimage

B) docker images

-> To give image name while creating image

A) docker Commit saicontainer (image name)

-> Create container from own image

A) docker run --name haricontainer -it saiimage ~~/bin/bash~~ //bin/bash

B) ls /tmp

17/10/2019

Dockerfile

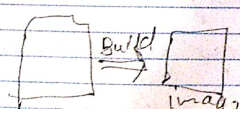
-> Dockerfile:
A) A text file with instructions to build image
B) Automation of Docker image creation
-> ~~Erase~~ FROM
-> RUN
-> CMD

-> Step1 -> create a file named Dokerfile.

-> Step2 -> Add instructions in Dockerfile.

-> Step3 -> Build dockerfile to create image

-> Step4 -> Run image to create container.



Dockerfile

-> vi Dockerfile
A) ~~FO~~ FROM ubuntu
B) RUN echo "Hi sai" > /tmp/testfile

-> To create Image out of Dockerfile

A) docker build -t test . (-t : tag to image)
↓ it will look to create where path is given in file
B) docker ps -a
C) docker images

-> Create container from the above image
A) docker run -it --name testcontainer test /bin/bash
B) cat /tmp/testfile

Scanned with CamScanner

tar and gz HAPPY DIWALI V TOO

Dockerfile

- vi Dockerfile → must be Dockerfile name

→ FROM

→ FROM ubuntu
→ WORKDIR /tmp
→ RUN echo "Hi Sai" > /tmp/testfile
↓ To create file
can give any path

→ ENV myname sai (To check echo $myname)

→ COPY testfile1 /tmp (testfile1 must be in current directory in EC2)

→ ADD test.tar.gz /tmp (test must be in current directory in EC2 in tar.gz format)

Volumes

→ Volumes is a directory inside your container

→ First ~~create~~ declare directory as a volume and then share volume.

→ Even if we stop container, ~~still~~ still we can access volume

→ Volume will be created in one container

→ you can declare a directory as volume only while creating container

→ you can't create volume from existing container.

→ you can share one volume across any no of containers

→ Volume will not be included when you a update an image.

→ Map volumes in two ways

A) Share host – container

B) Share container – container

Scanned with CamScanner

Volumes (declaration by using docker file)

-> Declare volume

A) FROM ubuntu

B) VOLUME ["/data"]

-> Docker build -t myimg.

myimg (image name we can give any name)

-> docker run -it --name myconti myimg //bin/bash

-> ls (you can see data folder)

-> touch /data/myfile (we can give any name)

-> exit

Volumes (container - container)

-> To share volume while creating another ~~cont~~ container

A) docker run -it --name myconti2 (new) --privileged=true --volumes-from myconti (old) ubuntu //bin/bash

B) ls /data/ (can see myfile)

-> What ever u do in one volume, can see from other volume

A) touch /data/test

-> docker start myconti

-> docker attach myconti

-> ls /data/ (can see the test file)

-> exit

Scanned with CamScanner

Volumes (declaration by using command)

-> docker run -it --name mycont3 -v/sai folder ubuntu//bin/bash

-> ls (can see sai folder. can mount like above. same)

Volumes (host - container)

-> verify file in /home/ec2-user

A) -> docker run -it --name nitcont -v/home/ec2-user:/raj --privileged=true ubuntu//bin/bash

B) -> cd raj

C) ls (can see all files created in your host)

D) touch rajfile (in container /raj)

E) exit

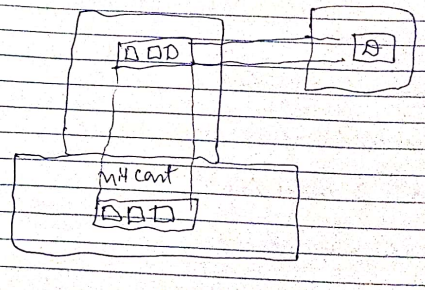
docker run -it --name nitcont -v/home/ec2-user:/raj --privileged=true ubuntu//bin/bash

F) ls (in host /home/ec2-user)

-> Can see rajfile

-> To see full details of container (even if it is in stopped state)

A) docker inspect nitcont



docker run -it --name name-volume

docker run -it

port expose

-> docker run -td --name webserver -p80(host):80(container) ubuntu (d-daemon)

-> docker port webserver

-> docker exec -it webserver /bin/bash (to go inside container)

-> apt-get update

-> apt-get install apache2

-> cd /var/www/html

-> echo "Hello" > index.html

-> service apache2 restart

-> In browser, public IP:80

(can change ~~host~~ host port. But not container port. Make sure, you are taking default port for container)

-> Try with Jenkins image (8080:8080) & open 8080 port in SG of EC2 machine (to unmap the port, stop the container or delete the container)

How to store Docker images in Docker hub & how to share to others

1) Be ready with Docker images (eg: saiimage)
2) Be ready with Docker hub account and to login in it.
3) docker login (run this command in base machine & give docker hub credentials)
4) docker tag saiimage sai3es/newsaiimage (sai3es is docker hub ID)
5) docker push sai3es/newsaiimage (to push image to docker hub)

- - - - -

6) Be in some other machine
7) docker pull sai3es/newsaiimage (to pull image to docker hub)

8) Can make docker image private also. So every time if some one wants to pull, they have to give your docker hub credentials. If it is public, no need to give credentials.

Scanned with CamScanner

Important Docker Commands

It is used to delete all ~~the~~

-> Stop all running containers : (docker stop $(docker ps -a -q))

-> Delete all stopped containers : (docker rm $(docker ps -a -q))

$(docker ps -a -q)

-> delete all images: ~~(docker rmi -f $(docker images)~~

docker rmi -f $(docker images -q)

If have to delete the particular one

-> Stop the running containers

(docker stop containernam)

-> Delete one stoped container

(docker rm Container name)

-> delete image

(docker rmi -f imagenam)

Ansible

-> Why Ansible ? Why Ansible

-> other tools in the market can be easily really complicated...

A) huge overhead of infrastructure setup
B) Complicated setups
C) pull mechanism
D) lot of learning required

-> pros of Ansible:

✓ Agentless: (No need to install Ansible)
✓ ~~Relies~~ Relies on SSH
✓ uses phython
✓ push mechanism

install & configure Ansible (server)

-> Launch Amazon linux (need not to install ansible in nodes)
-> yum install wget -y
-> wget http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
-> yum install epel-release-latest-7.noarch.rpm -y
-> sudo yum update -y
-> sudo yum install git python python-devel python-pip openssl ansible -y
-> ansible --version

-> Sudo vi /etc/ansible/ansible.cfg & enable the below lines
inventory = /etc/ansible/hosts
sudo_user = root

Ansible Inventory

Ansible recognize system listed in Ansible's inventory file (/etc/ansible/hosts)
Better to launch nodes as well in same AZ where server is there
The format for /etc/ansible/hosts looks like this: (vi /etc/ansible/hosts) (add under EN-I)
[groupname]
machinename(ar) machine IP

Ex:
[demo]
<host name / private-ip of node>

[local]
localhost

Test Environment setup (All machines)

-> adduser ansible (in all machines)
-> passwd ansible (in all machines)
-> visudo (add a line as below) (in all machine)
(adding ansible users into sudo users)
- ansible ALL=(ALL) NOPASSWD: ALL

To establish ssh configuration connection among all nodes (do it in all machines)

-> vi /etc/ssh/sshd_config (all machines)
permitrootlogin yes (uncomment line, set to "yes")
passwordAuthentication yes (")
permitRootlogin No (comment)
passwordAuthentication No (comment)
Service sshd restart

Test Environment setup (server)

Check sudo works without asking password (in server)

-> Su - ansible
-> whoami
-> sudo yum update
-> ssh <node private IP> (it prompts for password)
-> Run the following as ansible user
-> ssh-keygen (can see .ssh/ both keys in same directory)

Copy the ssh keys to all the nodes (be in master) (be in .ssh folder) (asks for password for the last time)

-> ssh-copy-id ansible@<node-private-ip>
(copy public key of server to node, so that private key will be authenticated)

Test ssh to test machine; it should not ask password

-> ssh <node-private-ip>

Scanned with CamScanner

Host patterns

vi /etc/ansible/hosts

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

ansible all --list-hosts

ansible <group_name> --list-hosts

Scanned with CamScanner

Ansible Ad-Hoc Commands

-> use ad-hoc tasks really quick & don't want to save for later

-> These are quick one-liner without writing a playbook

-> To run an arbitrary cmd use (-a)

A) ansible all --list-hosts
B) ansible demo -a "ls" (if all hosts, all)
C) ansible demo -a "ls -al"
D) ansible demo -a "touch myfile" (see run and verify indempotence)
E) ansible demo -a "yum install httpd -y"
(shows permissions denied)

-> To run anything with sudo, use -b

A) ansible demo -b -a "yum install httpd -y"
B) ansible local -b -a "yum remove httpd -y"

Ansible Modules

-> To run an arbitrary cmd use -a & use -m to run a module

A) ansible [group/host] -m <module> -a <cmd>
B) ansible demo -m ping

-> copy a file test.txt (first create) from server to node

A) ansible demo -b -m copy -a "src=test.txt dest=/temp/" (no need file name here)

-> Install/Remove a package

A) ansible demo -b -m yum -a "pkg=httpd state=present"
B) " " " " " "pkg=httpd state=latest"
C) " " " " "pkg=httpd state=absent"

State=present will install it
state = latest will update
State=absent will remove it

Scanned with CamScanner

Ansible Modules

-> Start/stop a service

started

A) ansible demo -b -m service -a "name=httpd state=started"
B) " " " " " " "name=httpd state=restarted"
C) " " " " " " "name=httpd state=stopped"

-> Create/Delete a user account

A) ansible demo -b -m user -a "name=raj"
B) ansible demo -b -m user -a "name=raj state=absent"

Gathering facts (Idempotence)

A) ansible demo -m setup

B) ansible demo -m setup -a 'filter=*ipv4*'

~~filter=*ipv4*~~

filter=*ipv4*

playbooks

-> Each playbook is composed of one or more 'modules' in a list

-> playbooks are divided into many sections like . . .

A) Target section -> Defines the hosts against which playbooks tasks has to be executed

B) variable section -> Defines variables

C) Tasks section -> List of all modules that we need to run, in an order

our first playbook 23/10/2019

-> All sections begin with "—"

YAML (yet Ain't markup language) Basics

-> For ansible, nearly every YAML file starts with a list

-> Each item in the list is a list of key/value pairs, commonly called a "dictionary"

-> All YAML files have to begin with "---" and end with "..."

-> All members of a list lines must begin with same identation level starting with "-"

```
--- # A list of tasty fruits
fruits:
  - Apple
  - orange
  - strawberry
...
```

A dictionary is represented in a simple key: value form (the colon must be followed by a space)

```
--- # An employee record
Employee:
    name: SAI
    Job: Devops Engineer
    Skill: Elite
...
```

Scanned with CamScanner

23/10/2019

Our First playbook

-> All sections begins with "-" & its attributes & parameters beneath it

-> indentation is imp, use only space & not tabs

-> create a folder (playbook) & go inside that (vi test.yml)

test.yml

```
--- # My first YAML playbook
- hosts: demo
  tasks:
    - name: install httpd on server
      action: yum pkg=httpd state=installed
```

Run ansible-playbook to call the playbook

ansible-playbook test.yml

Target section

create a file (vi first.yml)

Example:

```
--- # My first YAML playbook
- hosts: demo
  user: ansible
  become: yes          # yes or no
  connection: ssh      # ssh or paramiko
  gather_facts: yes    # yes or no
```

Run ansible-playbook to call the playbook

-> - ansible-playbook first.yml

Task section

Example:

```
--- # My first YAML playbook
- hosts: ~~ansible~~ demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: install HTTPD on centos 7
      action: yum name=httpd state=installed
    - name: install mysql on centos 7
      action: yum name=mysql state=installed
```

Run ansible-playbook to call

-> ansible-playbook first.yml

(TO ~~remove~~ ~~httpd~~ ->)

Scanned with CamScanner

Variables: inclusion types

→ Create a section called vars vars within a playbook.
→ put vars above tasks so that we define it first & use it later.

```
--- # My first YAML playbooks
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  vars:
    pkgname: httpd
  tasks:
    - name: installed HTTPD server
      action: yum name= "{{pkgname}}" state= installed
```

Handler section

→ Consists the ability to notify when something happens also called
→ Also call another set of tasks.
(Remove apache from node)

Example

```
--- # My first YAML playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: install httpd server on centos 7
      action: yum name=httpd state=installed
      notify: restart HTTPD (# this is called only if the action is run & successful #)

  handlers:
    - name: restart HTTPD (# this has to match the notify name. otherwise it shows error #)
      action: service name= httpd state= restarted
```

(install httpd in both nodes & don't start service. Then try running above script. it won't start service)

Run ansible-playbook to call the playbook

→ ansible-playbook first.yml

Dry Run

Check whether the playbook is formatted correctly
Test how the playbook is going to behave without running the tasks

→ ansible-playbook webserver.yml --Check

Loops

Often you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached

Example

```
--- # loop playbook
- hosts: demo
  user: ansible
  become: yes
  Connection: ssh
  tasks:
  - name: add a list of users
    user: name="{{ item }}" state=present
    with_items:
      - Bob
      - Sai
      - Hari
```

24/10/2019

Conditionals

Few tasks might be needed to execute only on specific scenario

When statement:
Sometimes you will want to skip a particular step on a particular host

Example 1

```
--- # When playbook example
- hosts: demo
  user: ansible
  become: yes
  Connection: ssh
  tasks:
    - name: install apache for debian
      Command: apt-get -y installed apache2
      When: ansible_os_family == 'Debian'
    - name: install apache for redhat
      Command: yum -y install httpd
      When: ansible_os_family == 'Redhat'
```

Cat /etc/os-rele*

Vault

Ansible allows keeping sensitive data such as passwords or key in encrypted files, rather than as plaintext in your playbooks

(A)
-> Creating a new Encrypted playbook (put ...) (verify file permissions) (open playbook now)

-> ansible-vault create playbook.yml

-> (B) Edit the Encrypted playbook
-> ansible-vault edit playbook.yml

-> (C) Change the password
-> ansible-vault rekey playbook.yml

-> (D) Decrypt the playbook & verify by opening playbook

ansible-vault decrypt playbook.yml

(E)
-> Encrypt an existing playbook (verify by opening playbook)

~~ansible-vau~~ ansible-vault encrypt playbook.yml

Roles (Theory)

Adding more & more functionality to the playbook will make it difficult to maintain in a single file
We can organize playbooks into a directory structure called roles

Creating Roles Framework (eg:

playbooks/roles/webserver)
/master.yml

/master.yml
playbook/roles/webserver/tasks/main.yml
/vars/main.yml
/handlers/main.yml

Scanned with CamScanner

(Roles) (Lab)

Master.yml (master.yml & roles folder must be in same directory level)

vi roles/webserver/tasks/main.yml
- name: install Apache on centos
  yum: pkg = httpd state = latest

playbook/roles/webserver/tasks/main.yml
/master.yml
/ansible-playbook master.yml (command run place)

This is how we mention role in play-books

Example

Create master.yml (inside playbook folder)

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

ansible-playbook master.yml
(run being present in playbook folder)

playbooks/roles/webserver/task/main.yml
var/main.v

~~By my info~~
By my inovation

playbook/roles/webserver/tasks/main.yml
playbook/roles/Dataserver/ " / "
playbook/roles/Roushanserver/ " / "
playbook/roles/ X / " / "
variablity

Change playbook name and give

playbook1/roles/Dataserver1/handler/main.yml
playbook1/roles/Rahulserver1/handler/main.yml
playbook1/roles/Kinanserver1/ " / "

Run as only

~~ansible playbook~~
ansible-playbook master.yml

not as

ansible-playbook1 master.yml

Scanned with CamScanner

Kubernetes

-> K8S is a container orchestration technology that creates, deploy and manages clusters (bunch of docker containers)

-> it schedules, runs and manages isolated containers which are running on virtual / physical cloud machines

-> Convert isolated containers running on different H/W into cluster

-> All 3 clouds support kubernetes

-> K8S originated at Google

Features of Kubernetes

-> Orchestration (clustering of any no of containers running on different H/W)

-> Autoscaling (new containers in place of crashed containers)

-> Load-Balancing (distributes client requests)

-> platform independent (cloud/virtualization/physical)

-> Fault tolerance (node/pod failures)

-> Rollback (bring back to previous versions)

Kubernetes

-> Working with kubernetes

A) we create manifest (.yml)

B) Apply this to cluster (to master) to bring into desired state

C) pod runs on node, which is controlled by master

-> Role of master node

A) Kubernetes cluster contains containers running on a Bare metal / VM instances / cloud instances / all min.

B) Kubernetes designates one or more of these as master and all others as workers

C) The master is now going to run a 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 so run cluster smoothly.

Constituents of control plane

-> Kube-apiserver:

A) This apiserver interacts directly with user (i.e we apply yaml or Tom manifest to kube-apiserver)

B) This kube-apiserver is meant for scale automatically as per load.

C) kube-api server is front end of control plane

-> etcd: (cluster store)

A) Stores metadata and status of cluster

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

C) source of truth for cluster state (info about state of cluster)

Constituents of control plane

-> kub-scheduler:

A) when users make request for the creation & managment of pods, kube-scheduler is going to take action on these requests.

B) Handlers pod creation and managment

C) kube-scheduler match/assign any node to create and run pods

-> controller-manager:

A) make sure actual state of cluster matches to desired state.

B) 2 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.

Scanned with CamScanner

→ What runs on each node of Cluster?

A) node/minion is going to run 3 imp pieces of software

B) Kubelet:
A) Agent running on the node
B) Listens to Kubernetes master (eg: pod creation request)
C) Port 10255
D) Sends success/fail reports to master

C) Container engine: (Docker)

A) Works with kubelet
B) pulling images
C) Start/stop containers
D) Exposing containers on ports specified in manifest

Nodes (kube-proxy)

→ Kube-proxy: (assigns IP to each Pod)

A) It is required to assign IP addresses to pods (dynamic)

B) Kube-proxy runs on each node & this make sure that each pod will get its own unique IP address.

→ Above 3 components collectively consists "node"

→ K8s for Hybrid & Multi-cloud

A) Hybrid: on premise datacenter + public cloud
B) Multi cloud: More than one public cloud provider

What is pod?

A) Atomic unit of deployment in Kubernetes
B) Consists of 1 or more tightly coupled containers
C) pod runs on node, which controlled by master
D) Kubernetes only knows about pods (doesn't know about individual containers)
E) Cannot start containers without a pod
F) A pod usually contains 1 container
G) Multi-container pods are possible too

-> Such Containers are tightly coupled.

PODS

-> Multi Container pods:
A) Share access to memory space
B) Connect to each other using localhost:(Container port)
C) Share access to the same volumes

-> Atomic unit of kubernetes
A) Containers within pod are deployed in an all-or-nothing manner.
B) Entire pod is hosted on the same node (a scheduler will decide about which node)

-> pod runs on which node - scheduler will decide

POD Limitations

-> No auto-healing or scaling
-> pod crashes? must be handled at higher level
A) Replica set
B) Deployment

Higher level kubernetes objects

-> Replication set:
A) Scaling & Healing

-> Deployment
A) Versioning and rollback

-> Service:
A) Static (non-~~ephem~~ ephemeral) IP and networking

-> volume:
- Non ephemeral storage

Important

-> kubectl - single cloud
-> kubeadm - on premise
-> kubefed - federated

2 ~~8~~) object management methods
A) imperative method
B) Declarative method

-> Imperative: we say actions that we want kubernetes to take
-> Declarative: we tell ~~ou~~ only required output, won't tell how to get

LINUX

Q. To deplay package application through remote

#

vi remoteapplication.sh
any file name

```
#!/bin/bash
echo "remote login client"
echo "enter ip address"
ssh 192.168.1.6 → client ip
"enter remote system passwd"
echo "below pkg details"
rpm -qa nfs*

yum remove nfs* -y

yum install nfs* -y

yum lists nfs*
```

now give permission

chmod +x remoteapplication.sh

now execute

#./remoteapplication.sh

How to check the modified file date

cat modifiedfiles.sh

```
#!/bin/bash
# To find files that have been modified within the last 2 hours.
#-mmin n: File was modified n minutes ago
#-mmin -n: File was modified less than n minutes ago
#-mmin +n:
```

```
#!/bin/bash
# To find files that have been modified within the last 2 hours
(uncommend to run #)
find . -mmin -120 → as per our requirement only a hour not back day
```

Now give permission

chmod +x --

now execute

#./

How to check two days back file

#!/bin/bash

Find files that have been modified within the last 2 days

These will change as per our requirement as days

(uncomment to run it)

find . -mtime -2

remember

chmod +

Execute

#!

How to check the modified file as a date wise

vi datewise.sh

#!/bin/bash

Find files modified in a range of dates. from 2019-06-07 to 2019-06-08

as per our requirement

(uncomment to execute)

find . -type f -newermt 2019-06-07 ! -newermt 2019-06-08

To check file which is greater than 15mb

~~#!~~ #!/bin/bash

find files larger than 15MB:

find -type f -size +15M

put - if have to find less than 15mb

How to check Bios information an memory information

#!/bin/bash

extract hardware information by reading data from the DMI tables

dmidecode -t memory

#print information about BIOS
echo "below information bios"
dmidecode -t bios

To shutdown at the schedule time

vi shutdown.sh

#!/bin/bash
echo "The following example will schedule a system shutdown at 05 AM"
Shutdown 05:00
#echo "The following example will schedule a system shutdown in 20 minutes from now"
Shutdown +20

echo "shutdown your system immediately"
#shutdown now

Crontab

The crontab is a list of commands that you want to run on a regular schedule; and also the name of the command used to manage that list. crontab stands for "cron table," because it uses the job scheduler cron to execute tasks; cron itself is named after "chronos".
crontab -l (To check previous schedules)
#cat /etc/crontab

Field	Description	Allowed value
MIN	Minute field	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of month	1-31
MON	Month field	1-12
DOW	Day of week	0-6
CMD	Command	Any command to be executed

Cron special keywords and its meaning

Keyword	Equivalent
yearly	0 0 1 1 *
daily	0 0 * * *
hourly	0 * * * *
reboot	Run at startup

To build crontab

crontab -e
1 * * * * /usr/bin/date.sh >/dev/pts/0

* * * * *
min hour day month day of week

How to do addition

vi addition.sh

#!/bin/bash
symbol is used to add single line comment in bash script
#Add two numeric value
((sum= 20+30))
#print the result
echo $sum

How to subtract

```
#!/bin/bash
:
The following script calculates
the square value of the number 5.
:
((area=15-5))
echo $area
```

While loop if is used

```
#!/bin/bash
#While loop will iterate for 5 times. The
                 var value of
#count variable will increment by I in each step.
#When the value of count variable will 5 then the
# While loop will terminate.
#((count++))

valid=true
count=1
while [ $valid ]
do
echo $count
if [ $count -eq 50 ];
then
break
fi
((count++))
done
```

arithmetic opperations

```
# vi arithmetic.sh
#!/bin/bash
#Arithmetic operations in shell script
a=5
echo -e "value of a is $a\n"
echo -e "assigning value of a to b\n"
b=$a
echo -e "value of b is $b\n"
echo -e "adding two numbers\n"
echo expr $a + $b
echo " "
echo -e "subtracting two numbers\n"
echo `expr $a - $b`
echo " "
echo -e "multiplying two numbers\n"
echo `expr $a \* $b`
echo " "
echo -e "dividing two numbers\n"
echo `expr $a / $b`
```

Scanned with CamScanner