Cicd

Linux ✓

3+ year’s experience in it indusry devops

Diff b/w git fetch and pull✓

Git flow, branching statergy and what are the branches u have in✓

Concepet of Pull request ✓

Diffent Goals Maven install deploy plublish ✓

Shell script example ✓

Basic example of Grep ps aux |grep java

Docker file example file

Ansible play book with example what u have witten✓

Cicd pipline in Jenkins wt are the steps steps

Tell I don’t have knowledge on kubernetis but I know

I’m currently working in YuppTV it’s a OTT platform i have 3+ years’ experience in IT industry from 3+ years im working on Devops tools with AWS cloud

**We are dealing 13 OTT projects** -

In all projects we are using devops tools and AWS

**Daaily activities**

Checking monitoring tool notification and sns (Amazon CloudWatch Alarm )

Checking email’s

Checking jir tickets

And working on emails and jira tickets

Ticks and emails like

Giving access to user in git repo of the project

And giving access to the test servers with sshkey

Monitoring tool notification’s like

disk utilization , cpu utilization, memory utilization

in disk removing unwanted logs , or converting to file or copying to s3 its depends on impotency of data

when cpu utilization – stopping unwanted services, with sudo run.sh , it will stop and start services

Checking Jenkins jobs and build’s if there are any fails then trouble shooting analysing and providing to fix issues

When any deployments comes Integrating with QA team. And dev team for collaboration and enhance a daily integration

Working on automation on a daily basis

Making documentation on daily bases like server information daily configuration or modify the infra

**DevOps Tools**

* **For source code management ---------** **git**
* **For deployments using ----------------Jenkins**
* **For configuration and for windows deployments using----- Ansible**
* **To Monitoring servers using ----------Site 24/7**
* **Ticketing Tool -------------------------JIRA**
* **Notifications using ------------------Amazon cloud watch alarm SNS**

**GIT Tool**

Step 4 : Add your github email and username to git

git config --global user.email "rahulkasarla@yahoo.com.com"

git config --global user.name "rahulsaikasarla"

Add file/folders to git – tracking

Commands

terminal - goto the location of the folder/projec

git status

ls -lrt

git add file\_name

to add all files

git add .

git commit -m “commit message”

git remote add orgin git@github.com:rahulsaikasarla/sample-maven-project.git

git push -u orgin master

to check the code from where we downloaded

git remote --verbose

# if Git error Fatal: remote origin already exists

git remote set-url origin https://github.com/rahulsaikasarla/Terrafrom.git

* **git init**

**Initialized** empty Git repository

# make directory a git repository

$ git init

Initialized empty Git repository in /Users/computer-name/Documents/website/.git/

* **git add**

Adds files in the to the staging area for Git

# To add all files not staged:

$ git add .

# To stage a specific file:

$ git add index.html

# To stage an entire directory:

$ git add css

* **git commit**

changes made to the files to a local repository

# Adding a commit with message

$ git commit -m "Commit message in quotes"

$ git commit -m "My first commit message"

[SecretTesting 0254c3d] My first commit message

1 file changed, 0 insertions(+), 0 deletions

* **git checkout**

use *git checkout* to switch branches

# Checkout an existing branch

$ git checkout <branch\_name>

# Switching to branch 'new\_feature'

$ git checkout new\_feature

Switched to branch 'new\_feature'

# Creating and switching to branch 'staging'

$ git checkout -b staging

Switched to a new branch 'staging'

**git merge**

Integrate branches together

* # Merge changes into current branch
* $ git merge <branch\_name>
* # Merge changes into current branch
* $ git merge new\_feature
* Updating 0254c3d..4c0f37c
* Fast-forward
* homepage/index.html | 297 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* 1 file changed, 297 insertions(+)
* create mode 100644 homepage/index.html

**git clone**

To copy or download the repository to a local computer

$ git clone <remote\_URL>

$ git clone git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

Cloning into 'repository\_name'...

remote: Counting objects: 5, done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 5 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (5/5), 3.08 KiB | 0 bytes/s, done.

Checking connectivity... done.

**git pull**

* This pulls the New changes files from the remote repository to the local computer.
* It performs two functions using single command
* It fetches all the changes that were made to the remote branch and then merges those changes into you’r local branch

**git pull = git fetch+gitmerge**

$ git pull <branch\_name> <remote\_URL/remote\_name>

# Pull from named remote

$ git pull origin staging

From account\_name.git.beanstalkapp.com:/account\_name/repository\_name

\* branch staging -> FETCH\_HEAD

\* [new branch] staging -> origin/staging

Already up-to-date.

# Pull from URL (not frequently used)

$ git pull git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git staging

From account\_name.git.beanstalkapp.com:/account\_name/repository\_name

\* branch staging -> FETCH\_HEAD

\* [new branch] staging -> origin/staging

Already up-to-date.

**git fetch**

git fetch does only half the work of git pull.it just brings the remote changes into your local repo but does not apply them onto your branches. You have to explicitly apply those changes

**git push**

Sends local com mits to the remote repository

$ git push <remote\_URL/remote\_name> <branch>

# Push all local branches to remote repository

$ git push —all

# Push a specific branch to a remote with named remote

$ git push origin staging

Counting objects: 5, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (5/5), 734 bytes | 0 bytes/s, done.

Total 5 (delta 2), reused 0 (delta 0)

To git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

ad189cb..0254c3d SecretTesting -> SecretTesting

# Push all local branches to remote repository

$ git push --all

Counting objects: 4, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (4/4), 373 bytes | 0 bytes/s, done.

Total 4 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

To git@account\_name.git.beanstalkapp.com:/acccount\_name/repository\_name.git

0d56917..948ac97 master -> master

ad189cb..0254c3d SecretTesting -> SecretTesting

**git stash (To revert back changes )**

# **Store current work with untracked files**

$ git stash -u

**To add user in git**

Git config –global user.name “testuser”

**git log**

To show the commit history for a repository

# **Show entire git log**

$ git log

# **Show git log with date pameters**

$ git log --<after/before/since/until>=<date>

# **Show git log based on commit author**

$ git log --<author>="Author Name"

In Practice:

# **Show entire git log**

$ **git log**

commit 4c0f37c711623d20fc60b9cbcf393d515945952f

Author: Brian Kerr <my@emailaddress.com>

Date: Tue Oct 25 17:46:11 2016 -0500

Updating the wording of the homepage footer

commit 0254c3da3add4ebe9d7e1f2e76f015a209e1ef67

Author: Ashley Harpp <my@emailaddress.com>

Date: Wed Oct 19 16:27:27 2016 -0500

My first commit message

# **Show git log with date pameters**

$ git log --before="Oct 20"

commit 0254c3da3add4ebe9d7e1f2e76f015a209e1ef67

Author: Ashley Harpp <my@emailaddress.com>

Date: Wed Oct 19 16:27:27 2016 -0500

My first commit message

# **Show git log based on commit author**

$ git log --author="Brian Kerr"

commit 4c0f37c711623d20fc60b9cbcf393d515945952f

Author: Brian Kerr <my@emailaddress.com>

Date: Tue Oct 25 17:46:11 2016 -0500

Updating the wording of the homepage footer

**Difference between git fetch and git pull.**

**Git fetches**

* only downloads new data from a remote repository
* It does not integrate any of these new data into your working files.
* Can be done any time to update the remote-tracking branches

**Command - git fetch origin**

**Git Pull-**

* Git pull updates the current HEAD branch with the latest changes from the remote server.
* Downloads new data and integrate it with the current working files.
* Tries to merge remote changes with your local ones.

**Command - git pull origin master**

**Difference between git clone and git pull.**

**git clone**

* To copy or download the repository to a local computer

**git pull**

* This pulls the New changes files from the remote repository to the local computer.

**Difference between git clone and git fetch**

**git clone**

To copy or download the repository to a local computer

**Git fetches**

* only downloads new data from a remote repository
* It does not integrate any of these new data into your working files.
* Can be done any time to update the remote-tracking branches

**merge conflict in Git?**

* when Git is unable to resolve differences in code between the two commits automatically.
* Git is able to automatically merge the changes only if the commits are on different lines or branches.

**resolve a merge conflict?**

* To resolve the conflicted file is to open it and make the required changes
* After editing the file, we can use the **git add a** command to stage the new merged content
* Next create a new commit with the help of the **git commit** command
* Git will create a new merge commit to finalize the merge

**Git Commands to Resolve Conflicts**

1. **git log --merge**

The git log --merge command helps to produce the list of commits that are causing the conflict

**2. git diff**

The git diff command helps to identify the differences between the states repositories or files

**3. git checkout**

The git checkout command is used to undo the changes made to the file, or for changing branches

**4. git reset --mixed**

The git reset --mixed command is used to undo changes to the working directory and staging area

**5. git merge --abort**

The git merge --abort command helps in exiting the merge process and returning back to the state before the merging began

**6. git reset**

The git reset command is used at the time of merge conflict to reset the conflicted files to their original state

**What is the purpose of branching in GIT?**

The purpose of branching in GIT is that you can create your own branch and jump between those branches. It will allow you to go to your previous work keeping your recent work intact.

**What is the common branching pattern in GIT?**

The common way of creating branch in GIT is to maintain one as “Main“

branch and create another branch to implement new features. This pattern is particularly useful when there are multiple developers working on a single project.

**how can you bring a new feature in the main branch?**

To bring a new feature in the main branch, you can use a command “git merge” or “git pull command”.

**What is another option for merging in git?**

“Rebasing” is an alternative to merging in git

**Rebasing**

Feature branch to master branch

* A process to reapply commits on top of another base trip. It is used to apply a sequence of commits from different branches into a final commit. It is an alternative of **git**merge **command**.

[**Gitflow**](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) **is:**

* The **Gitflow**  defines a strict branching model designed around the project release
* This provides for managing larger projects.

Branching strategy :-

1. A **develop** branch is created from **master**
2. A **release** branch is created from **develop**
3. **Feature** branches are created from **develop**
4. When a **feature** is complete it is merged into the **develop** branch
5. When the **release** branch is done it is merged into **develop** and **master**
6. If an issue in **master** is detected a **hotfix**branch is created from **master**
7. Once the **hotfix** is complete it is merged to both **develop** and **master**

**Concept of pull request**

1. A developer creates the feature in a dedicated branch in their local repo.
2. The developer pushes the branch to a public repository git.
3. The developer files a pull request via git.
4. The rest of the team reviews the code, discusses it, and alters it.
5. The project maintainer merges the feature into the official repository and closes the pull request.

**Maven Tool Build tool**

**Def of Maven:** Maven is a build automation tool used for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies

* This build tool is responsible of dependency and compiling multiple(more then 100 or thousand) files like bundle of codes compiling with dependency
* Artifact: package of or bundle of complied code available in ..war or jar format
* .jdk it is java application is used for compiling java code
* pom.xml file is called it as definition file
* compile can be triggered in the location where pom.xml file located
* pom (project object model)

**What Maven does ?**

* Compilation of Source Code.
* Running Tests (unit tests and functional tests)
* Packaging the results into JAR’s,WAR’s,RPM’s,etc..
* Upload the packages to remote repo’s (Artifactory like Jfrog)

Diffent Goals Maven install deploy plublish or maven build cycle

* mvn validate - to check/scan the pom.xml file if there are any errors
* mvn clean to clean all packages
* mvn compile - it will compile source code(which includes multiple java files at once)
* mvn test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

to generate artifact

* mvn package - take the compiled code and package it in its distributable format, such as a JAR.
* mvn verify - run any checks on results of integration tests to ensure quality criteria are met
* mvn install - install the package into the local repository, for use as a dependency in other projects locally
* mvn deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

**Linux os**

**What is dollar $1 and $0 ?**

**$1** is the first command-line argument passed to the shell script. Also, know as Positional parameters. For example, $0, $1, $3, $4 and so on. If you run ./script.sh filename1 dir1, then:

* $0 is the name of the script itself (script.sh)
* $1 is the first argument (filename1)
* $2 is the second argument (dir1)
* $9 is the ninth argument
* ${10} is the tenth argument and must be enclosed in brackets after $9.
* ${11} is the eleventh argument.

**$1 and positional parameters example**

**What is a swap space?**

Swap space is a certain amount of space used by Linux to temporarily hold some programs that are running concurrently. This happens when RAM does not have enough memory to hold all programs that are executing.

**how much memory Linux is using?**

* command: cat /proc/meminfo
* You can also use commands

free - m

[**How do you change file permissions in Linux?**](https://chmod-calculator.com/)

**using the chmod**

u (user), g (group), o (others), a (all), r (read), w (write) and x (execute)

example, the command chmod go+rw FILE1.TXT grants read and write access to the file FILE1.TXT, which is assigned to groups and others.

### **How to create 100 files in single command**

* **using the {} wildcard**
* touch file-{001..100}.txt

**to remove all files**

* rm -f file-\*

**lsb\_release**

The **lsb\_release** command prints certain LSB (Linux Standard Base) and Distribution information

**How do u check how many files are there in particular server with command**

dir \*. \* /s  ------ **To count all the files** and directories in **the** current directory and subdirectories

**check Linux Kernel version**

**$ uname -r**

4.4.0-143-generic

**Find Linux kernel version**.

**$ cat /proc/version**

Linux version 4.4.0-143-generic (buildd@lgw01-amd64-041) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.10) ) #169-Ubuntu SMP Thu Feb 7 07:56:38 UTC 2019

**To print /to view ssh public -- lock**

cat ~/.ssh/id\_rsa.pub

**.pub is public a lock**

**Id\_rsa is a Private key**

**To generate ssh key**

ssh-keygen

**To add key in remote server**

sudo vi .ssh/authorized\_keys

**To check running ports**

netstat -tulpn

**To check particular services port**

sudo netstat -tulpn | grep nginx

**nginx configurations**

/etc/nginx/sites-enabled

**Gitlab configurations**

Sudo nano /opt/gitlab/embedded/service/gitlab-rails/config/gitlab.yml

sudo gitlab-ctl restart

sudo nano /etc/gitlab/gitlab.rb

sudo gitlab-ctl reconfigure

**To find file in linux**

sudo find / -name Jenkins.log

**To stop any particular services**

* Systemctl - sudo systemctl stop httpd
* Service - sudo service httpd stop

**How will u searching particular services is running or not**

* ps -ef | grep java
* **ps aux | grep java**

**Linux servers logs – how can u find out issues in logs**

/var/log

**logs to** view is the syslog

**What command u would be using the logs or analysing logs**

Sed Command

Grep,

Awak

Head /var/log.auth.log

**Issue yesterday we have to find out issue in logs to day**

Goto system logs and select by using date folder or logs

**what all the processes are running on linux**

ps -eaf

**pstree**

systemd─┬─Site24x7Agent─┬─Site24x7Applog───2\*[{Site24x7Applog}]

│ └─19\*[{Site24x7Agent}]

├─Site24x7AgentWa───{Site24x7AgentWa}

├─2\*[agetty]

├─amazon-ssm-agen───13\*[{amazon-ssm-agen}]

├─atd

├─auditd───{auditd}

├─chronyd

├─crond

├─dbus-daemon

├─2\*[dhclient]

├─gssproxy───5\*[{gssproxy}]

├─irqbalance───{irqbalance}

├─java───145\*[{java}]

├─java───215\*[{java}]

├─java───46\*[{java}]

├─lsmd

├─lvmetad

├─master─┬─pickup

│ └─qmgr

├─nginx───4\*[nginx]

├─postgres───39\*[postgres]

├─3\*[redis-server───3\*[{redis-server}]]

├─rngd

├─rpcbind

├─rsyslogd───3\*[{rsyslogd}]

├─sshd───sshd───sshd───bash───pstree

├─systemd-journal

├─systemd-logind

└─systemd-udevd

## **How to see every process except those running as root**

ps -U root -u root -N

### **Print a process tree using ps**

# ps -ejH  
# ps axjf

**How do u kill process id**

sudo kill psid

**How to see how much usage on disk free**

$ **sudo df -h**

Filesystem Size Used Avail Use% Mounted on

devtmpfs 15G 0 15G 0% /dev

tmpfs 15G 28K 15G 1% /dev/shm

tmpfs 15G 648K 15G 1% /run

tmpfs 15G 0 15G 0% /sys/fs/cgroup

/dev/xvda1 200G 96G 105G 48% /

tmpfs 3.0G 0 3.0G 0% /run/user/1001

**$ sudo df -hT (**file system type.)

Filesystem Type Size Used Avail Use% Mounted on

devtmpfs devtmpfs 15G 0 15G 0% /dev

tmpfs tmpfs 15G 28K 15G 1% /dev/shm

tmpfs tmpfs 15G 648K 15G 1% /run

tmpfs tmpfs 15G 0 15G 0% /sys/fs/cgroup

/dev/xvda1 xfs 200G 96G 105G 48% /

tmpfs tmpfs 3.0G 0 3.0G 0% /run/user/1001

memory management , diskmangement, dis addig dizk removal , oom, out of memory

node avarge , pid?, ppid ?

vi admin.sh

#!/bin/bash

top | head -10

df -h – file system size

free -m it displays free memory and swap memory

uptime -it show system uptime and user login and load

iostat -how much data giving to the use

how to check old files +90(90 days files)

find /root/scripts/realtime-scripts/ -mtime +90 -exec ls -l {} \;

delete\_old\_files.sh

#!/bin/bash

echo “deleting old 90 days files”

find /root/scripts/realtime-scripts/ -mtime +90 -exec rm {} \;

remae\_files.sh

#!/bin/bash

echo ‘renaming old files’

find /root/scripts/realtime-scripts -mtime +90 -exec mv {} {}.old \;

backup\_files.sh

#!/bin/bash

tar -cvf /tmp/backup.tar /etc /var

gzip /temp/backup.tar

find /tem/backup.tar.gz -mtime -1 -type f -print &> /dev/null

If [ $? -eq 0 ]

then

eco “backup was created”

echo

echo “archieving backup”

#scp /tmp/backup.tar.gz root@192.168.1.8:~/path

else

echo “backup failed”

fi

**Disk commands in linux**

**To display disk**

lsblk

xvda and xvdb will show all

after adding storage to the server

lsblk

xvdf –(disk part) this will be available

**to format disk**

fdisk /dev/xvfd ---- press enter

give ----- n (add new partion) and press enter

give 4 times enter

lsblk

xvdf1

now create file system

**To create file system give below command**

mkfs.ext3 /dev/xvdf1

and create one mount volume

mkdir /ebsvol

Scripting

* #! ---------- shebang

#!/bin/sh

Example

#!/bin/bash

today=`date |awk '{print $1, $2, $3}'`

last | grep "$today" | awk '{print $1}

**Example script**

**copy\_to\_s3.sh**

aws s3 sync /home/user/App\_Server/logs/ s3://ott-serverlogs/FrndlyTV/apiserver1new/ --region us-east-1

sleep 1

aws s3 sync /home/user/Logs\_backup/ s3://ott-serverlogs/FrndlyTV/apiserver1new/ --region us-east-1

**api\_deploy\_from\_test.sh**

#!/bin/bash

export TIMESTAMP="$(date +'%Y%m%d\_%H\_%M')"

PROCESSID=$(ps aux |grep App\_Server-assembly-0.0.0.1-SNAPSHOT.jar | grep -v "grep"| awk '{print $2}')

echo Before Start ProcessID $PROCESSID

user@192.168.1.146:~/App\_Server/lib/App\_Server-assembly-0.0.0.1-SNAPSHOT.jar /home/yuppflix/

scp server@13.232.21.220:~/App\_Server/lib/App\_Server-assembly-0.0.0.1-SNAPSHOT.jar /home/yuppflix/

sleep 5

mv App\_Server/lib/App\_Server-assembly-0.0.0.1-SNAPSHOT.jar /home/yuppflix/Backup/App\_Server-assembly-0.0.0.1-SNAPSHOT.jar\_$TIMESTAMP

sleep 1

mv /home/yuppflix/App\_Server-assembly-0.0.0.1-SNAPSHOT.jar App\_Server/lib/

sleep 1

sh api\_restart.sh

#!/bin/bash

export TIMESTAMP="$(date +'%Y%m%d\_%H\_%M')"

PROCESSID=$(ps aux |grep Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar | grep -v "grep"| awk '{print $2, $19}')

echo Before Start ProcessID $PROCESSID

scp -r yupptv@119.81.201.166:/home/yupptv/Analytics\_Server/lib/Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar .

sleep 1

cd Analytics\_Server

mv lib/Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar /home/yuppindia/Backup/Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar\_$TIMESTAMP

sleep 1

mv /home/yuppindia/Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar lib

sleep 1

ps -ef | grep Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar | grep -v grep | awk '{print $2}' | xargs kill

sleep 1

rm -rf logs/\*

sleep 1

sh run.sh

sleep 5

tail -n 100 /tmp/analytics\_server.log

PROCESS=$(ps aux |grep Analytics\_Server-assembly-0.0.0.1-SNAPSHOT.jar | grep -v "grep"| awk '{print $2, $19}')

echo After Start ProcessID $PROCESS

Ansible Tool

Ansible is configuration and desire state:

Ansible – it works on ssh and Python.

It’s also support network automation.

Push architecture

To find ansible configuration file

ansible –version

To disable Host file key checking –

go to nano /etc/ansible/ansible.cfg

press control+f = search host

remove # from host\_key\_checking = False

should be like below

host\_key\_checking = False

To ping all host

ansible -i hosts -m ping all

**Ad-Hoc Commands-** To management purpose

Example – to check storage in all servers

ansible -i host app -a free

to check os

ansible -i hosts client1 -m setup | grep -i ansible\_os

[**Write a playbook to install ‘git’  on all hosts if the file ‘/opt/collab’ exists on the system.**](https://www.zeolearn.com/interview-questions/ansible#collapse-beginner-1012)

---

- hosts: all

  vars:

      mario\_file: /opt/collab

      package\_list:

          - 'git'

  tasks:

      - name: Check for collab file

        stat:

            path: "{{ collab\_file }}"

        register: collab\_f

      - name: Install git if collab file exists

        become: "yes"

        package:

            name: "{{ item }}"

            state: present

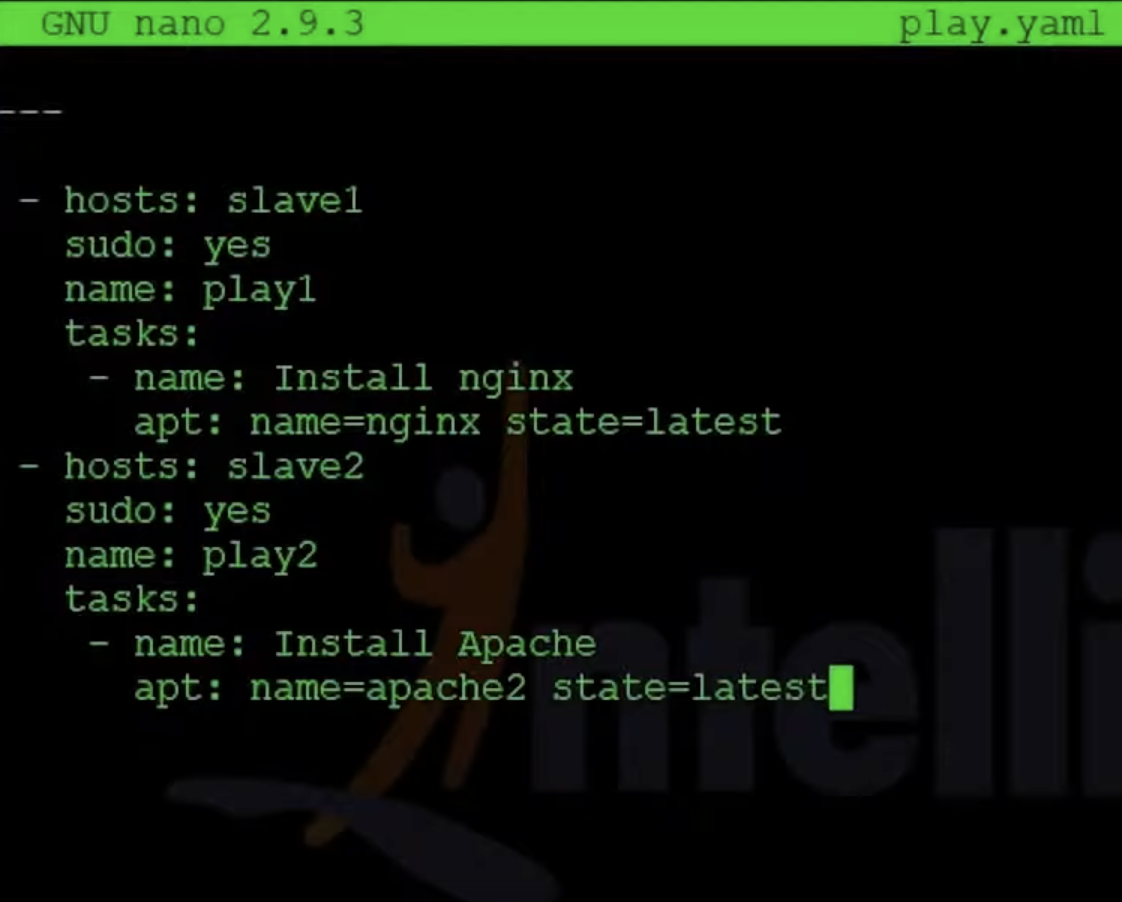
        with\_items: "{{ package\_list }}"

        when: collab\_f.stat.exists

**To install different software’s on different severs both servers are ubuntu**

Package management

vi install.yamal



**To run playbook**

**$ ansible-playbook install.yaml**

**Play book for Windows**

**Ansible installation on windows**

run .ps1 in windows

.\ConfigureRemotingForAnsible.ps1

(https://raw.githubusercontent.com/varmil/setup-winrm-for-windowsservercore/master/ConfigureRemotingForAnsible.ps1)

for winrm setup execute the below commands:

$url = "https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1"

$file = "$env:temp\ConfigureRemotingForAnsible.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)

powershell.exe -ExecutionPolicy ByPass -File $file

Get-ExecutionPolicy ( it most be in RemoteSigned )

if their is no value run the below command

Set-ExecutionPolicy "RemoteSigned"

to enable the winrm excute the below command

Enable-PSRemoting

After check once winrm https is allow or not

control pannel --> windows firewall --> Allow an apps through windows firewall

tick the box [Allow WinRM HTTPS]

pip install pywinrm ( linux machine )

vi /etc/ansible/hosts

[window]

192.168.\*\*\*.\*\*\*

[window:vars]

ansible\_user=@@@@@@@@@@@@@

ansible\_password=\*\*\*\*\*\*\*\*\*\*\*\*

ansible\_connection=winrm

ansible\_winrm\_server\_cert\_validation=ignore

$ ansible window -m win\_ping

output:

192.168.33.22 | SUCCESS => {

"changed": false,

"ping": "pong"

}

**copy play book**

**vi copy.yaml**

---

- hosts: windows

tasks:

- name: copy file local to windows machine

win\_copy:

src: /root/pra/cp.txt

dest: c:\Users\vagrant\Desktop\cp.txt

$ ansible-playbook copy.yaml

output:

PLAY [windows] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [192.168.33.22]

TASK [copy file local to windows machine] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [192.168.33.22]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

192.168.33.22 : ok=2 changed=1 unreachable=0 failed=0

**You can execute modules from the command line:**

ansible webservers -m service -a "name=httpd state=started"

ansible webservers -m ping

ansible webservers -m command -a "/sbin/reboot -t now"

**Ansible Modules-**

1. **Package management-** To install packages on host machines
2. **Service –** To start or restart or to check status of the service
3. **Copy –** To copy or deploy files in host machines
4. **Debug -** debugging variables or expressions without having to halt the playbook
5. **File -** manages the file and its properties
6. **Line in file -** manages lines in a text file
7. **Cli command -** This backs up a config to a different destination file.
8. **User -** Creates, deletes or alters user accounts on the configured server

## Module 1: Package management

**- name: Install a list of packages  
  yum:  
    name:  
     - nginx  
      - postgresql  
      - postgresql-server  
    state: present**

## Module 2: Service

After installing a package, you need a module to start it or restart or status

**- name: Start service foo, based on running process /usr/bin/foo  
  service:  
    name: foo  
    pattern: /usr/bin/foo  
    state: started**

**- name: Restart network service for interface eth0  
  service:  
    name: network  
    state: restarted  
    args: eth0**

## Module 3: Copy

copies a file from the local or remote machine to a location on the remote machine.

**- name: Copy a new "ntp.conf file into place, backing up the original if it differs from the copied version  
  copy:  
    src: /mine/ntp.conf  
    dest: /etc/ntp.conf  
    owner: root  
    group: root  
    mode: '0644'  
    backup: yes**

**- name: Copy file with owner and permission, using symbolic representation  
  copy:  
    src: /srv/myfiles/foo.conf  
    dest: /etc/foo.conf  
    owner: foo  
    group: foo  
    mode: u=rw,g=r,o=r**

## Module 4: Debug

prints statements during execution and can be useful for debugging variables or expressions without having to halt the playbook.

### Example 1:

**- name: Display all variables/facts known for a host  
  debug:  
    var: hostvars[inventory\_hostname]  
    verbosity: 4**

This displays all the variable information for a host that is defined in the inventory file.

### Example 2:

**- name: Write some content in a file /tmp/foo.txt  
  copy:  
    dest: /tmp/foo.txt  
    content: |  
     Good Morning!  
      Awesome sunshine today.  
    register: display\_file\_content  
- name: Debug display\_file\_content  
    debug:  
      var: display\_file\_content  
      verbosity: 2**

This registers the content of the copy module output and displays it only when you specify verbosity as

**ansible-playbook demo.yaml -vv**

## Module 5: File

manages the file and its properties.

* It sets attributes of files, symlinks, or directories.
* It also removes files, symlinks, or directories.

### Example 1:

**- name: Change file ownership, group and permissions  
  file:  
    path: /etc/foo.conf  
    owner: foo  
    group: foo  
    mode: '0644'**

## Module 6: Lineinfile

manages lines in a text file.

* It ensures a particular line is in a file or replaces an existing line using a back-referenced regular expression.
* It's primarily useful when you want to change just a single line in a file.

### Example 1:

**- name: Ensure SELinux is set to enforcing mode  
  lineinfile:  
    path: /etc/selinux/config  
    regexp: '^SELINUX='  
    line: SELINUX=enforcing**

This sets the value of **SELINUX=enforcing**

### Example 2:

**- name: Add a line to a file if the file does not exist, without passing regexp  
  lineinfile:  
    path: /etc/resolv.conf  
    line: 192.168.1.99 foo.lab.net foo  
    create: yes**

This adds an entry for the IP and hostname in the **resolv.conf** file.

## Module 7: Cli\_command

**- name: configurable backup path  
  cli\_config:  
    config: "{{ lookup('template', 'basic/config.j2') }}"  
    backup: yes  
    backup\_options:  
      filename: backup.cfg  
      dir\_path: /home/user**

This backs up a config to a different destination file.

**8.User add**

**-** name**:** Add the user 'Testuser' with a specific uid and a primary group of 'admin'

ansible.builtin.user**:**

name**:** johnd

comment**:** John Doe

uid**:** 1040

group**:** admin

**remove user**

**-** name**:** Remove the user 'johnd'

ansible.builtin.user**:**

name**:** johnd

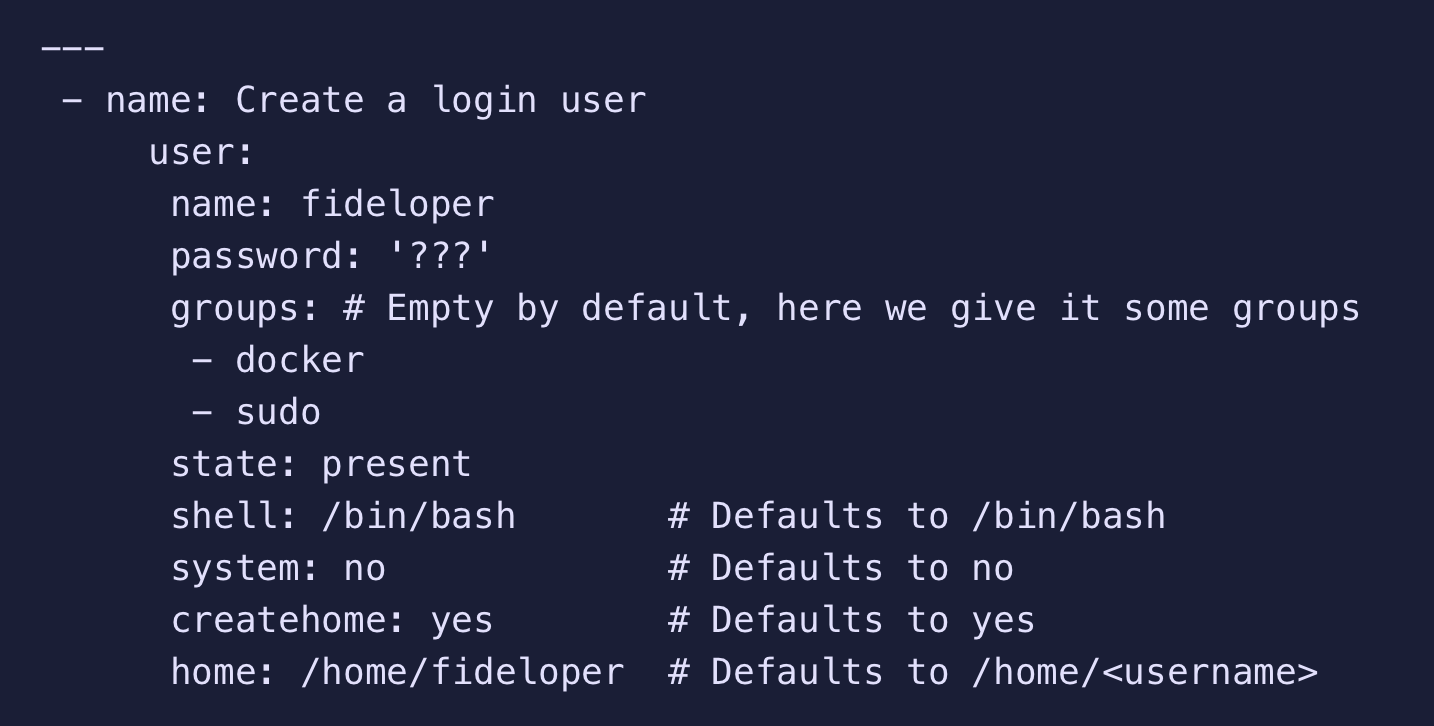
state**:** absent

remove**:** yes

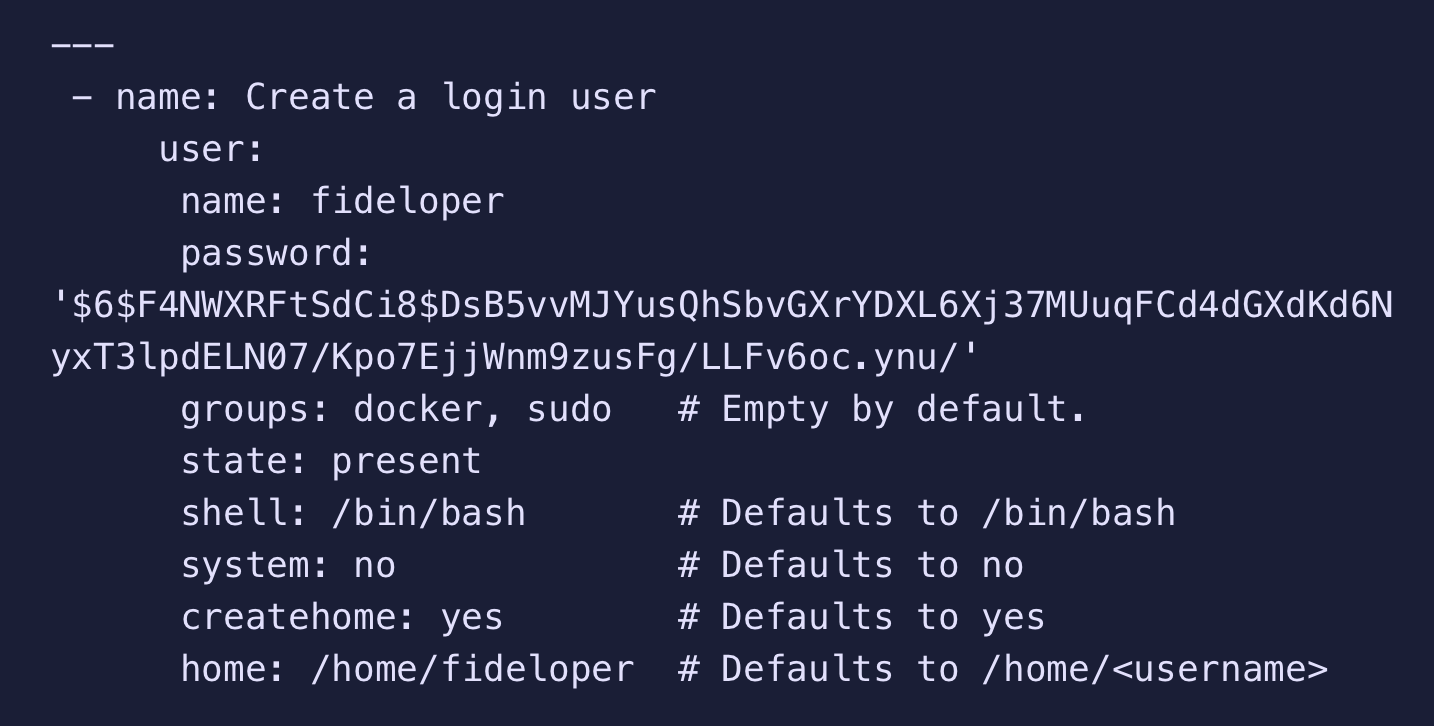
code

ansible all -m command -a "useradd testuser -G testgroup1,testgroup2,testgroup3"

[**To add in server**](https://serversforhackers.com/c/create-user-in-ansible)

****

mkpasswd --method=sha-512



**What is Ansible Galaxy?**

**Ansible Galaxy** is a repository for **Ansible** Roles that are available to drop directly into your Playbooks to streamline your automation projects

**Ansible** automates using the SSH protocol

**Ansible tower**

****

#### Deploy to Amazon ECS in Git.

eploy a container to an Amazon ECS service powered by AWS Fargate or Amazon EC2.

# This workflow will build and push a new container image to Amazon ECR,

# and then will deploy a new task definition to Amazon ECS, when a release is created

#

# To use this workflow, you will need to complete the following set-up steps:

#

# 1. Create an ECR repository to store your images.

# For example: `aws ecr create-repository --repository-name my-ecr-repo --region us-east-2`.

# Replace the value of `ECR\_REPOSITORY` in the workflow below with your repository's name.

# Replace the value of `aws-region` in the workflow below with your repository's region.

#

# 2. Create an ECS task definition, an ECS cluster, and an ECS service.

# For example, follow the Getting Started guide on the ECS console:

# https://us-east-2.console.aws.amazon.com/ecs/home?region=us-east-2#/firstRun

# Replace the values for `service` and `cluster` in the workflow below with your service and cluster names.

#

# 3. Store your ECS task definition as a JSON file in your repository.

# The format should follow the output of `aws ecs register-task-definition --generate-cli-skeleton`.

# Replace the value of `task-definition` in the workflow below with your JSON file's name.

# Replace the value of `container-name` in the workflow below with the name of the container

# in the `containerDefinitions` section of the task definition.

#

# 4. Store an IAM user access key in GitHub Actions secrets named `AWS\_ACCESS\_KEY\_ID` and `AWS\_SECRET\_ACCESS\_KEY`.

# See the documentation for each action used below for the recommended IAM policies for this IAM user,

# and best practices on handling the access key credentials.

on:

release:

types: [created]

name: Deploy to Amazon ECS

jobs:

deploy:

name: Deploy

runs-on: ubuntu-latest

environment: production

steps:

- name: Checkout

uses: actions/checkout@v2

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-east-2

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-login@v1

- name: Build, tag, and push image to Amazon ECR

id: build-image

env:

ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

ECR\_REPOSITORY: my-ecr-repo

IMAGE\_TAG: ${{ github.sha }}

run: |

# Build a docker container and

# push it to ECR so that it can

# be deployed to ECS.

docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG .

docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

echo "::set-output name=image::$ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG"

- name: Fill in the new image ID in the Amazon ECS task definition

id: task-def

uses: aws-actions/amazon-ecs-render-task-definition@v1

with:

task-definition: task-definition.json

container-name: sample-app

image: ${{ steps.build-image.outputs.image }}

- name: Deploy Amazon ECS task definition

uses: aws-actions/amazon-ecs-deploy-task-definition@v1

with:

task-definition: ${{ steps.task-def.outputs.task-definition }}

service: sample-app-service

cluster: default

wait-for-service-stability: true

**Any project starts with dev**

**Dev ------- Test --------Deploy**

**Ci= Continues integration**

* The ci process happens mostly into dev environment
* Now ci is the process where the team members integrate their code or their work in a shared repository this process is best achieved by using some scm or source code management tool like git so they integrate their work on daily basis it is mostly done daily once a day
* Everyone in the dev team will integrate and check in the shared repository and this check-in is then followed and validated by our automated build and automated unit tests and they can also be some automated integration test that we run to make sure that everything is fine

There are no errors and this process is mostly done daily that is once a day as per your need as per the requirement of the project it can be done as frequent as needed this is the process of ci

Dev-------- Test

**CD – continues delivery**

* After the ci process , Deploying on a prod like env and running automation tests to ensure that builds is ready for release
* Ensures the build is always in a Deployable state

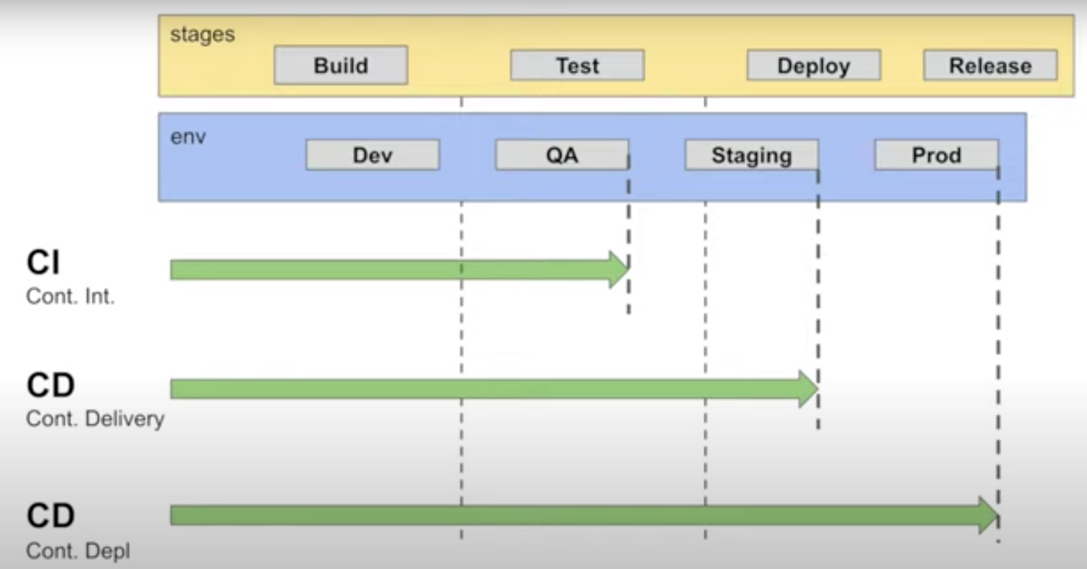
Dev--------Test--------Staging env

**CD- Continuous Deployment**

Automated Deployment to Production

Every change that passes through Automation Tests is deployed to Production

This may not be practical in some of the organizations that they do not have so frequent and so short release cycle so continues deployment is not done automatically.



Pipelines

**Terraform**

To build infrastructure it can be any provider like aws azure google ibm its supports all providers.

**Explain the uses of Terraform CLI and list some basic CLI commands?**

* **terraform init**: Prepare your working directory for other commands
* **terraform validate**: Check whether the configuration is valid
* **terraform plan**: Show changes required by the current configuration
* **terraform apply**: Create or update infrastructure
* **terraform destroy**: Destroy previously-created infrastructure

---- I build infrastructure with terraform and for manage this infrastructure used ansible

Packer – Image building -AMI creation

I done some small task

Git-cicd-trigger

* Stage 1- AMI build with packer
* Stage 2- Infra Build (Terraform)
* Stage3- Build war/jar file for java
* Stage 4 Copy this war/jar file to the server and Deploy

Copy the terraform in **/usr/local/bin/ (**to work terraform from any location**)**

We can create multiple files in terraform

Terraform file

Main.tf

References.tf

Outputs.tf

Test.tfvars

Variables.tf

**Main.tf ------------in real time only provider will be in main.tf file**

Provider “aws” {

Access\_key=

Secret\_key=

Region=

}

References.tf

Resource “aws\_vpc” “default”{

Cidr\_block= “${var.vpc-cidr}”

Enable\_dns\_hostname=true

Tags{

Name= “${var.vpc\_name}”

Internet gateway

Resource “aws\_internet\_gateway “default” {

Resource “aws\_subnet” “subnet1-public”

Resource “aws\_route\_table” “terraform-public”

Resource “aws\_security\_group” “allow\_all”

**test.tfvars**

aws\_access\_key=

aws\_secret\_key=

aws\_region= “us-east-1”

vpc\_cidr= “10.1.0.0/16

public\_subnet1\_cidr= “10.1.1.0/24”

public\_subnet2\_cidr= “10.2.1.0/24”

public\_subnet3\_cidr= “10.1.3.0/24”

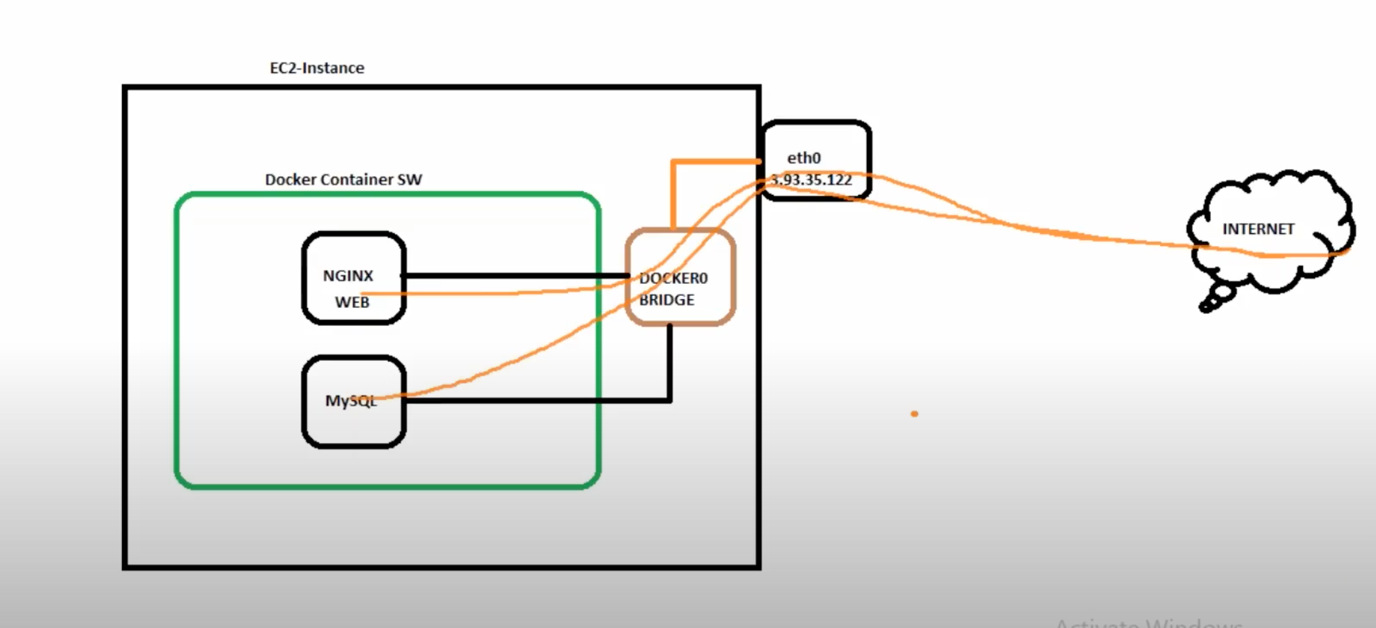
public\_subnet4\_cidr= “10.1.4.0/24”

vpc\_name = “terraform-aws-testing

IGWname= “terraform-aws-igw”

DOCKER Tool

Through host we will connect docker containers



Imutable

**SQL DB**

**Data Definition Language [DDL]**

* **CREATE**
* **DROP**
* **ALTER**

**Data manipulation language [DML]**

* **SELECT**
* **INSERT**
* **UPDATE**
* **DELETE**

**DDL**

* List of databases ---- \l
* List of roles -----------\du
* Change password for user-- ALTER USER postgres WITH PASSWORD ‘xx’;
* Create a new user --- CREATE USER name WITH PASSWORD ‘xxx’;
* Providing super user privileges --- ALTER USER name WITH SUPERUSER;
* Removing user --- DROP USER user\_2;

**DML**

* SELECT \* FROM *table\_name*;
* INSERT INTO first\_table\_name
* update yuppeducation.user\_preference set a5= null, a6=null, a7= null where user\_id in (92138);

25 -30 members