Linux Commands

cd /home/ec2-user 🡺 change directory cd /

ls

you have learnt commands

clear

sudo -s

sudo -i

sudo su

pwd

cd

cat

ls

history

File & Directory commands:

Ls : List files

-l: long format listing

-a: include hidden files hidden ones

-h: Human – readable file sizes

Ls -al: display files and directories with detailed information

Ls -a: shows all files and directories including hidden

Ls -lh: displays files sizes in human readable format.

Ls -lt: displays files

Ls -ltr: reverse sorting

Ls new\* => to list files starting with new

-rw-r—r--. 1 root

drwxr-xr-xr-x. 2 root

* Represent file

d represent directory

cd: change directory

touch filename: to create a empty file

pwd: print current working directory

mkdir: create a new directory

cd .. : move back to the one directory

mkdir hello

rmdir hello 🡺 delete directory

rm

-r: Remove directories recursively

-f: Force removal without confirmation

rm : deletes the file

rm -r: deletes the directory and its contents

rm -f : forcefully deletes the file without confirmation

rmdir -rf directory

rm -rf \* 🡺 its removes all from directory path

cp : copy files and directories

cp file1.txt file2.txt

cp file.txt /test/file.txt

cat file 🡺 shows the content of the file data

cat > filename 🡺 its create file

cat >> filename

mv 🡺 moving file from to another/rename file

vi file.txt 🡺 editing the file

head -n 5 file.txt 🡺 displays first 5 lines of file

tail -n 5 file.txt 🡺 displays last 5 lines of file

sort file.txt : sorting content

sort -r file.txt : reverse sort

cmp file.txt file1.txt 🡺 to compare two files deferred by 3 lines

diff file file2 🡺 shows the difference

diff -u file file2 🡺 shows the line by line comparison

find /home/ec2-user/ -name java1 🡺 it finds the file name as java1 in the ec2-user folder

find \* -name java1 🡺 it finds all filenames with java1

man cp/man ls 🡺

bc 🡺 its calculator in cmd

cal 🡺 calender in cmd

cal AUG 2022 🡺 displays august 2022

who 🡺 user details

whoami 🡺 its shows the user

hostname -i 🡺 it shows ip address

hostnamectl set-hostname PERSONAL 🡺 it sets hostname as PERSONAL

hostname 🡺 it shows hostname

sudo -i 🡺 it will set our hostname as PERSONAL

vi file.txt 🡺

i 🡺 insert mode

I => start of line

A: end of line

O: new line above existing

o: below existing line

COMMAND MODE:

gg: first line of file

shift g: last line of file

10gg: 10th line of file

:10: 10th line of file

yy: copies a single line

4yy: copies four lines

p: to paste for one time

4p: to paste for four times

dd: to delete a single line

4dd: to delete four lines

U: to undo

/word: to search for a word

GREP:

grep reyaz file 🡺 it will search reyaz in the file and displays the content that having reyaz in the file(grep word filename)

frequently using commands:

ls -ltr & ls -al 🡪 display all hidden files

script 🡺 this cmd useful in realtime 🡺 used to record activities

Alias:

It gives shortcut to commands like below

Alias h=”ls -al” 🡺 so now if u use h then it will work as ls -al

gzip -k file1 🡺 its zip file1

gunzip file2.gz 🡺 its unzip file

tar -czf hello.tar.gz hello/ 🡺 to uncompress files

rm -rf hello

tar -xzf hello.tar.gz 🡺 to compress multiple files

tar -czf

SED: STREAM EDITOR (to replace words in a file)

sed -n ‘7,14p’ file1: to print from line 7 to 14 lines

sed -n ‘5, 15p’ file1: to print from line 5 to 15 lines

sed -i ‘s/word1/word2/’ file1 : to replace word1 with word2

sed -i ‘s/word1/word2/g; s/word3/word4/g’ file1: to replace multiple words, g globally

sed -i ‘4c goodbye’ file1: to replace a line

sed -i ‘4i goodbye’ file1: to add a line

sed -i ‘4d’ file1: to delete 4th line

wget pathoffile 🡺 its download file from internet

yum install -y wget 🡺install wget package

yum install -y nginx🡺install nginx

systemctl start nginx 🡺 to start the nginx service

systemctl status nginx

systemctl stop nginx

systemctl restart nginx

systemctl list-units --type=service --all

systemctl list-units --type=service –all|grep nginx 🡺 it will search nginx service in the all services

Process

ps

ps -aux 🡺 all ur processes

ps -ef|grep nginx 🡺 its show nginx process

kill -9 processids 🡺 it will kill processes forcefully

systemctl status nginx

pgrep nginx

System related commands

======================

uname -a

df -h 🡺 disk file system

fdisk -l 🡺 file system information

du -sh test 🡺 test directory size

free -h 🡺 how much memory for system

uptime 🡺 its for details like user login time

date 🡺 date information

rpm 🡺 redhat package manager

rpm -qa | grep sql

rpm -qa | grep nginx

rpm -qa | grep httpd

rpm -qa | grep java

rpm -qa | grep python

dnf list installed

yum list available

yum list available|grep nginx

ifconfig

ping google.com 🡺 uses icmp protocol to get the connectivity.

Netstat 🡺 to see TCP/UDP connections

curl 🡺 cmd retrieves content of the webpage

curl <http://numbersapi.com/random> 🡺 we can call API by using curl commands

check port is open or not

telnet localhost 80 🡺 its checking 80 port is opened or not

netstat -putan|grep 80 – see if port 80 is open or not

printenv 🡺 to show the environment variables

export course=linux 🡺 to export the path

unset course 🡺 it will remove the value

java –version

printenv – it shows JAVA\_HOME

export PATH=$JAVA\_HOME/bin:$PATH

=====================================================================

how to add env variable permanently on linux:

vi .bashrc

export JAVA\_HOME=”/usr/lib/jvm/java-11-openjdk-11.0.25.0.9-3.el9.x86\_64”

source .bashrc

=====================================================================

User Management Commands

=========================

USERS & GROUPS

------------------------

root: will have all permissions

ec2-user

useradd Bharath 🡺 to add Bharath user

passwd Bharath 🡺 to add password

cat /etc/passwd 🡺 we can see all users here

su – Bharath 🡺 it will set user as Bharath

visudo

groupadd devops

cat /etc/group 🡺 it shows all the groups

r w x

rwx rwx rwx

users groups others

r – read = 4

w – write = 2

x -- excute = 1

-rw-r—r-- 🡺 644

If u want to give full permissions then below command 🡺 chmod 777 file1

If u want to give read & write permissions then 🡺 chmod 764 file1

chmod 755 file1

yum install -y httpd

yum remove httpd

LOGGING AND AUDITING

=====================

#Display logs stored in the system journal

journalctl -u httpd (Journalctl -u servicename)

lscpu –CPU Info

lsmem - - mem info

fdisk -l – disk information

lsblk

ifconfig – to see the ip address

ip addr

hostname -I

WinSCP Setup | Transferring Files from Windows to Linux (Redhat)

----------------------------------------------------------------------------------------

Install WinSCP 🡺 click on newtab

Open WinSCP 🡪 SFTP 🡪 Hostname=IP, Port = 22, username = ec2-user Advanced 🡪 Authentication --> select ppk transfer files

Transfer files b/w Linux Servers using SCP Command

---------------------------------------------------------------------

SCP – Secure Copy Protocol, is a command line tool used for securely transferring files b/w local & remote host or between 2 remote hosts using SSH for Encryption and authentication

Syntax

---------

scp /local/file username@remotehost:/path/

scp -i key-pair-name.pem /path/my-file.txt ec2-user@privateip:/path/

Launch another Redhat Linux EC2

Create a pem file vi MyKey.pem

chmod 400 pemfile

scp -i MyKey.pem file1 [ec2-user@172.31.19.242:/tmp/](mailto:ec2-user@172.31.19.242:/tmp/)

or

scp -i MyKey.pem test.zip [ec2-user@172.31.6.82:/home/ec2-user/](mailto:ec2-user@172.31.6.82:/home/ec2-user/)

Login to another EC2 and check the files.

GIT: Global Information Tracker

==========================

Its used for source code management tool & used to maintain multiple versions of the same file.

Its also named as version control system.

DVCS: Distributed version control system, stores code on multiple repos. Ex: GIT

CVCS: Centralized version control system stores code on single repo. Ex: SVN(subversion)

GIT Stages:

---------------

Working Directory

Staging Area

Repository

Working Directory

Staging Area

GIT ADD

COMMIT

Repository

Working Directory: Its just aware of files which having in the project. It will not track these files until we commit these files.(Where we write the source code)

Staging Area: It’s a rough draft space i.e. where you can save the files by using git add version in next commit. (where we track the code)

Its like if u give “git add .” then all files saved into once space that’s called staging area.

Repository: its like folder having all the collection of data files and history changes made to these files which related to our project. (where we store the tracked code)

Local Repo: its local git repository folder .git directory, its area that saves everything.

Remote Repo: it’s a git repository stored in some remote system. (where we store the code in remote location)

Central Repo: this is will be present in GITHUB.

Git init 🡺 git add . 🡺 git commit 🡺 git push 🡺 (GitHub) Remote Repo

| |

-----------------------------------------------------------------------

git pull

To install GIT: YUM INSTALL GIT -Y

$git - -version 🡺 check the git version installed

$git init 🡪 to get empty repo

To track/add the files:

$git add file\_name 🡪 single file

$git add file1 file2 🡪 multiple files

$git add \* 🡺 all regular files

$git add . 🡺all hidden files

$git status 🡺 to check the status of the git files

Green color files 🡺 tracking files 🡺 its having record which we made the changes on file.it will track each and every change in the file.

Red colour files 🡺 untracking files

$git commit -m “first file commit” filename 🡺 it will commit the single file

$git commit -m “files commit” 🡺 it will commit the all the tracking files

$git rm – cachedfile 🡪 it will remove/untrack the files

$git log 🡪 its to check the history

$git log-2 🡪 to check the last commit ids

$git log –oneline -> used to see only commit id's and commit messages

$git config 🡪 its used to configure the user name & user email id for global user configuration

$git restore 🡪 its used to restore deleted files (git restore filename)

$git show 🡪 its used to show file names in log

$git commit –amend -m “new commit message” 🡺 its used to edit commit message

$git commit --amend --no-edit : used to add some files in previous commit. (-

-no-edit means that the commit message does not change.)

$git update-ref -d HEAD : used to delete all the commits in git

$git reset commit: used to delete all the commits (upto the commit id)

$git reset --hard HEAD~1 : used to delete latest commit along with the changes

$git reset --hard HEAD~N : upto nth commit

$git revert commit-id : used to undo a middle of the change (file also deleted)

$git rebase: if u want to change multiple commit histories

(if u want to use this command, you need to have minimum 3 commits in ur repo)

$git rebase -i HEAD~3 🡺 it means it will refer last 3 commits from the current branch head

And now we can replace pick with reword and keep changing the commit messages for 3 times and wq!

GIT BRANCHING

=================

Git branch command used to create, delete, list, rename the branches.

The default branch for the git branch is master.

$git branch 🡺 to see the current branch

$git branch branch\_name 🡺 to create a branch

$git checkout branch 🡺 to go to a branch

$git branch -d branch\_name 🡺 to delete a branch

$git branch -m old new 🡺 to rename a branch

$git checkout -b branch\_name 🡺 to create and switch at a time

GIT MERGE:

==========

It allows us to get the code from one branch to another. This is useful when developers work on the same code and want to integrate their changes before pushing them up in a branch.

command: git merge branch\_name

Merge conflicts happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge.

GIT makes merging super easy!

GIT MERGE CONFLICTS

CONFLICTS generally araise when two people two people have changed the same lines in a file (or) if one developer deleted a file while another developer is working on the same file!

In this situation git cannot determine what is correct!

Lets understand in a simple way!

cat>file1 : hai all

add & commit

git checkout -b branch1

cat>file1 : 1234

add & commit

git checkout master

cat>>file1 : abcd

add & commit

git merge branch1 : remove it

identify merge conflicts:

see the file in master branch then you will see both the data in a single file including branch names

resolve:

open file in VIM EDITOR and delete all the conflict dividers and save it! add git to that file and commit it with the command

(git commit -m "merged and resolved the conflict issue in abc.txt")

merge:

if you have 5 commits in master branch and only 1 commit in devops branch, to get all the commits from master branch to devops branch

we can use merge in git. (command: git merge branch\_name)

cherry-pick:

if you have 5 commits in master branch and only 1 commit in devops branch, to get specific commit from master branch to devops branch

we can use cherry pick in git. (git cherry-pick commit\_id).

git branch -d new ==> deleting new branch

git log ==> it will give the git log history

git stash ==> it will take the backup file and revert back to original position

git rm filename ==> removing the filename from the git

git reset –hard HEAD^ ---- no data loss

git reset –soft HEAD^ ---- data loss

git stash 🡺 keep the working files for aside and revert back to original position of working directory

git stash apply 🡺 it will add backup files/kept working files previously and showed them with all current files.

Git init

Git add .

Git commit -m “committing files”

Git remote add origin <https://github.com/bharath/testrepo.git> ==> it needs to done only first time

Git push -u origin master 🡺 this will be used to commit the changes from local to central repo

Git remove -v

Git push -u origin master

Git pull

Maven

Maven is used to build the code and once we build we get file like format EAR/JAR/WAR.

Maven used to add dependencies to application and based on POM.XML file.

POM file contains like metadata, kind of project, kind of output, description, dependencies).

Maven Build Life Cycle:

1.Generate Resource

2.Compile code

3.Unit Test

4.Package(build)

5.install (into Local repo or Artifactory)

6.Deploy(to servers)

7.Clean (to delete all the runtime files)

mvn archetype:generate

mvn compile: compiles the source code of the project

mvn test: runs the tests using a testing framework like junit

mvn package: packages the compiled code into a jar/war file.

mvn install: installs the project war/jar file into local repo

mvn deploy: deploys the built project to remote repo for sharing with other developers

mvn clean: deletes the target directory (where compiled code and other build artifacts are stored)

In realtime, we won’t use above steps one by one, just use the below command:

mvn clean package

**Docker**

Yum install docker -y

#docker -v 🡺 version of docker

#systemctl start docker 🡺 to start docker service

#docker images

#docker status 🡺 list the images

#docker ps -a 🡺 to show the docker all containers

#docker ps 🡺 to show the docker all containers running

#docker pull ubuntu 🡺 it will download it from ubuntu

#docker pull amazonlinux

#docker images 🡺 it will show images now (ubuntu)

#docker run -it –name amazonlinuxcont amazonlinux 🡺 to run the amazonlinux container

#docker run -it –name uibuntucont ubuntu 🡺 to run the ubuntu container

#docker run -it -d –name ubuntucont ubuntu 🡺 to run container with dettached mode

(by using above cmd we run container but we can enter inside into docker by default)

#docker attach ubuntucont 🡺 here u can enter the docker container

#apt update

#docker exec -it 33dcd2340a2a /bin/bash 🡺 we can use login/enter to container

(above cmd we use mostly in realtime)

Difference b/w docker attach & docker exec:

Docker attach interacts with the container’s main process(PID 1)

Docker exec starts a new, separate process inside the container. (i.e. already PID1 is running it will starts new separate process like PID2)

EXEC is preferred.

#docker stop ubuntucont 🡺 it will stop ubuntucont container from container images

#docker ps 🡺 now it will show only running containers

#docker inspect ubuntucont 🡺 description of docker container

#docker kill ubuntucont

#docker rm ubuntucont

#docker rmi ubuntu

#docker rmi amazonlinux 🡺 delete docker images

#docker commit ubucontainer mynewimage 🡺 it will create new container image from the existing imagecontainer

#docker images

#docker ps -aq 🡺 it will show the container ids

#docker kill $(docker ps -aq) 🡺 it will kill the container ids

#docker rm $(docker ps -aq) 🡺 it will remove the container ids

#docker images -q 🡺 it will show docker images ids

#docker rmi -f $(docker images -q) 🡺 it will remove the image ids

#docker restart ubucont 🡺 it will restart ubuntu container

If we want to automate all these by using Dockerfile:

Dockerfile

Its used to automate image creation, inside Dockerfile we use components to do our works.

Components will be on capital letter, in Dockerfile D will be capital.

We can create image directly without container help to create image from file we need to build it.

Components:

FROM: used to get base image

RUN: used to run linux commands (during image creation)

CMD: used to run linux commands (after container creation)

ENTRYPOINT: high priority than cmd

COPY: to copy local files to container

ADD: to copy internet files to container

WORKDIR: to open req directory

LABEL: to add labels for docker images

ENV: to set env variables (inside container)

ARGS: to pass env variables (outside container)

EXPOSE: to give port number

Dockerfile 🡪 Docker Build 🡪 Image 🡪 Container

#vi Dockerfile

FROM amazonlinux

RUN yum install -y git

CMD yum install mysql

:CTRL+pq; 🡺 to exit

#docker logs awscont 🡺 it shows the logs of the container

#docker top awscont 🡺 it shows the process of the container

#docker cp hello.txt awscont1:/tmp 🡺 copy files to folder in container (we use in realtime)

#docker save amazonlinux > myownimg.tar 🡺 it copies/save ur image as tar file

Dockerfile 🡺 to create a image

vi Dockerfile

FROM ubuntu

RUN apt update -y

RUN apt install git maven tree apache2 -y

RUN touch file1

:wq!

#docker build -t bharath:v1 .

#docker images

#docker run -it –name cont1 bharath:v1 🡺 to enter into the container

vi Dockerfile

FROM ubuntu

RUN apt update -y

RUN apt install git python3 -y

CMD apt install mysql-server -y

:wq!

#docker build -t bharath:v2 .

#dockerimages

#docker run -it –name cont1 bharath:v2 🡺 to enter into the container

Difference b/w CMD & RUN:

RUN will execute when docker build time

CMD will execute when docker run time(it will execute when container got created)

Vi Dockerfile

FROM ubuntu

COPY hello.txt /tmp

ADD <https://dldcn.apache.org/tomcat/tomcat-10/v10.1.35/bin/apache-tomcat-10.1.35.tar.gz> /tmp

#docker build -t bharath:v3 .

#docker images

#docker run -it –name cont3 bharath:v3

#cd /tmp

#ls 🡺 u can check apache tomcat 10 server zip file into temp folder

#mv Dockerfile to dockerfile4

Vi Dockerfile

FROM ubuntu

COPY hello.txt /tmp

ADD <https://dldcn.apache.org/tomcat/tomcat-10/v10.1.35/bin/apache-tomcat-10.1.35.tar.gz> /tmp

WORKDIR /tmp

LABEL author Bharath

:wq!

#docker build -t bharath:v4 .

#docker run -it –name cont4 bharath:v4

Vi Dockerfile

FROM ubuntu

ENV course devops

ENV trainer bharath

EXPOSE 8080

:wq!

#docker build -t bharath:v5

#docker run -it –name cont5 bharath:v5

#echo $course

#echo $trainer

#docker ps -a

#mv Dockerfile to dockerfile5

Vi Dockerfile

FROM ubuntu

RUN apt update -y

RUN apt install apache2 -y

RUN apt install apache2-utils -y

RUN apt clean

COPY website/ /var/www/html/

RUN service apache2 restart

EXPOSE 80

CMD [“/usr/sbin/apachectl”, “-D”, “FOREGROUND”]

:wq!

#docker build -t firstproject:v1

#docker run -itd –name newwebcont1 -p 80:80 firstproject:v1

#docker ps -a

#docker run -itd –name newwebcont2 -p 8080:80 firstproject:v1

Vi Dockerfile

FROM ubuntu

VOLUME [“volume1”]

:wq!

#docker build -t bharath:v1 .

#docker run -it –name cont1 bharath:v1

#ls

#cd volume1/

#touch file{1..5}

#ls

#cd /var/lib/docker/volumes

#ls

#cd \_data/

#docker ps -a

#docker attach cont1

#ls

#docker run -it –name cont2 --volumes-from cont1 ubuntu

#cd volume2/

#docker run -it –name cont5 –volumes-from cont4 ubuntu

#cd volume2/

#docker attach cont4

#cd /var/lib/docker/volumes/

#ls

#cd \_data/

#cd .

#docker volume ls

#docker volume create volume3

#docker volume ls

#cd /var/lib/docker/volumes/

#ls

#cd volume3/

#ls

#cd \_data/

#ls

#touch html{1..5}

#docker run -it –name cont6 –mount source=volume3, destination=/volume3 ubuntu

#docker run -it –name my-container1 -v /mnt/ebs-volume/:/mynewebsvolume ubuntu

#cd mynewebsvolume/

#touch hello.txt

#ls

#docker run -it –name mynewcont2 –volumes-from my-container1 ubuntu

#ls

#cd mynewebsvolume/

#ls

#docker system df

#docker ps -a

#docker stats mynewebsvolume

#docker ps -a

#docker run -it –name cont7 –cpus=”0.1” –memory=”300mb” ubuntu

#docker stat cont7

#docker update cont7 –cpus=”0.5” –memory=”200mb”

#

Terraform

Terraform

=========

In AWS till now, we have created infrastructure manually, created EC2, ELB, ASG, S3, RDS, VPC etc

If we create infra manually

1.time consume

2.mistakes

3.tracking

if you want to repeat these creating infra multiple times, you need to automate creating infrastructure:

AWS introduced a service called CloudFormation in 2011. Its very cool service to automate infrastructure using

JSON & YAML.

AWS has CloudFormation(CF)

Azure has AzureResourceManager(ARM)

Google has Deployment manager

But common tool to automate the infrastructure for all cloud providers i.e. Terraform.

Terraform

---------

Terraform is an open-source Infrastructure as Code(Iac) tool developed by HashiCorp.

It allows users to define and provision infrastucture resources in a consistent, repeatable manner using a high level

configuration language know as HashiCorp Configuration Language(HCL).

It was a free and opensource tool but now its not opensource (check in google, opentofu is a forked version of terraform)

Alternatives

------------

PULUMI

ANSIBLE --- is mainly used for configuration management, TF will create infra

CHEF

PUPPET

OpenTofu

TF VS Ansible

=============

TF will create infra and these servers will be orchestrated/configured by Ansible.

Terraform Structure

-------------------

Mainly most of the people use 3 files in Terraform

main.tf: contains all providers, resources and data sources

variables.tf: contains all defined variables

output.tf: contains all output resources

The issue with this structure is that most logic is stored in the single main.tf file which therefore becomes pretty complex and long.

Terraform, however doesnot mandate this structure, it only requires a directory of Terraform files.

Filenames don't matter to Terraform, to make it simple we prefer the following structure:

Projectname/

|

|-- provider.tf - plugin to connect to cloud, search in google, TF providers --> AWS --> use provider

|-- version.tf - sets required terraform and provider versions.

|-- backend.tf - this file is used to configure the backend, which determines how and where terraform stores its

state data.

|-- main.tf - contains the terraform block and provider block(AWS,Azure etc)

|-- variables.tf - declares input variables

|-- terraform.tfvars - Assigns values to variables

|-- outputs.tf

provider.tf: contains the terraform block and provider block (AWS, Azure etc)

data.tf: contains all data sources

variables.tf: contains all defined variables

local.tf: contains all local variables

output.tf: contains all output resources

vi main.tf

----------

provider "aws" {

region = "ap-south-1"

}

resource "aws\_instance" "myinstance" {

count = 3

ami = "ami-0f2ce9ce760bd7133"

instance\_type = "t2.micro"

}

if u want to make terraform file u can use command like ==> #terraform fmt

once we created main.tf then first cmd which needs to perform:

#terraform init ==> it will download all plugins and dependencies after using this command(it will create lock file which has all dependencies of the provider information and .terraform directory)

#terraform validate ==> it will validate the syntax

#terraform plan ==> it will show just suppose to plan(after giving this cmd nothing will create) - dry run - it will shows what are the things it will create resources

#terraform apply ==> it will create all the resources(it will approve all the plan)

#terraform apply --auto-approve ==> it will auto approve terraform plan

#terraform destory ==> it will destory the resources created by terraform

#terraform state list ==> it will list out all the instances created by terraform

aws\_instance.myinstance[0]

aws\_instance.myinstance[1]

aws\_instance.myinstance[2]

#terraform destory --auto-approve -target=aws\_instance.myinstance[2]

COUNT argument ==> terraform will create the instances based on this argument

##Initialize Infrastructure###

terraform init : Initialize a working directory, it will download the providers plugins

terraform plan: creates an exection plan (dry run)

terraform apply: Executes changes to the actual environment

terraform apply --autoapprove: Apply changes without being prompted to enter "yes"

terraform destroy --auto-approve: Destroy/cleanup without being prompted to enter "yes"

All tf code should be written in tf files, extension is .tf, also called as configuration files.

Main things in TF file

----------------------

Blocks

Labels

argument

A tf file has blocks, for ex: provider is a block, "aws" is a label, for single block we can have multiple labels 0, or

1 or 2 etc, optional whatever we write in {} is called arguments, arguments can called as input for the block, argument need to have key & value.

For provider we have one label aws, but for resource block has 2 labels, it can be multiple labels \_ is called identifier, if u want to combine 2 words use \_

we can't use aws instance (space between) so use aws\_instance, instance\_type

First TF Example:

-----------------

vi main.tf

provider "aws" {

region = "ap-south-1"

}

provider is block

aws is label {} inside is arguments

--terraform fmt [used to format ur configuration files into a canonical format and style]

terraform fmt -recursive -- for all files

--terraform init

whenever you have a new or existing terraform directory (containing ur terraform configuration files) you need to

run terraform init to prepare that directory for other terraform commands.

Provider Plugins: Terraform uses plugins to interface with cloud providers (like AWS, Azure, Google cloud, etc).

The init command checks the configuration files to see which providers you are using and fetches the required provider plugins.

Provider Versions: If u are specified a particular version of a provider in ur configuration, terraform init will download that version. If not, it will

get the latest compatible version.

-- ls -al

--cd .terraform --> Navigate & see aws plugin with version

--cd

vi main.tf

provider "aws" {

region = "ap-south-1"

}

resource "aws\_instance" "myinstance" {

ami = "ami-02ddd77f8f93ca4ca"

instance\_type = "t2.micro"

}

No need to do again terraform init, because provider already downloaded and we didn't changed in that block

--terraform validate

--terraform plan ====> attach a IAM role to EC2

--terraform apply

--terraform apply --auto-approve

+ : Creating

- : Deleting

~ : Update

I - Init

P - Plan

A - Apply

D - Destroy

.terraform contains lots of information (providers plugins will be stored in this directory)

-- cd .terraform

STATE FILE

==========

Terraform must store state about ur managed infrastructure and configuration. This state is used by Terraform to map real

world resources to ur configuration, keep track of metadata, and to improve performance for large infrastructures.

This state file is extremely important, it maps various resource metadata to actual resource IDs so that Terraform knows what it is managing.

Local State & Remote State

--------------------------

"By default, Terraform stores state locally in a file named terraform.tfstate. When working with terraform in a team, use of a local file makes

terraform usage complicated because each user must make sure they always have the latest state data before running terraform and make sure

that nobody else runs terraform at the same time.

"With remote state, Terraform writes the state data to a remote data store, which can then be shared b/w all members of a team." Ex: S3

--cat terraform.tfstate

--terraform state list ==> list all the resources that are currently being tracked in the terraform state file

-- terraform destory --auto-approve

.terraform.lock.hcl

===================

When u run terraform init, it downloads the required providers & dependencies and generates the .terraform.lock.hcl file if it doesn't already exist.

If the file does exist, Terraform checks the versions specified in the lock file and installs those versions.

COUNT Argument

TARGET = is used to delete a specific resource

==============================================

single target:

--------------

-- terraform destroy --auto-approve -target=aws\_instance.myinstance[0]

Multi Target:

-------------

--terraform destroy --auto-approve -target=aws\_instance.myinstance[1] -target=aws\_instance.myinstance[2]

terraform state list

terraform destroy --auto-approve