APPLICATION: COLLECTION OF SERVICES
EX: WHATSAPP
Calls, Media, Messages, Payment, Location, Meta AI

PAYTM:
Movies, Flight, Dth, Payment, Recharge, Train ---

WHERE THE APPLICATIONS WILL BE HOSTED ?
ANS: SERVER

WHAT IS SERVER ?
PROVIDES SERVICES TO USER (CLIENT)

DATA CENTERS FOR AWS
TOTAL AZ : 117
REGIONS : 37
MIN : 3
MAX : 6
BIGGEST : N-VIRGINIA

PHYSICAL SERVER = ON-PREM

PHYSICAL SERVER	CLOUD SERVER
INFRA	NO NEED FOR INFRA
MAINTAINACE	NO MAINTAINACE
HIGH COST	LOW COST
SKILLED PEOPLE	NO SKILLED PEOPLE

WHAT IS EC2 ?
IN AWS SERVER CAN BE CALLED AS EC2
EC2: ELASTIC COMPUTE CLOUD

EC2 = SERVER = INSTANCE

STEP = 7

STEP-1: NAME & TAGS
TO IDENTIFY SERVERS

STEP-2: AMAZON MACHINE IMAGE

PROVIDES OS, SOFTWARES ----------

STEP-3: INSTANCE TYPE
PROVIDES HARDWARE (CPU & RAM)
1 CPU & 1 RAM

STEP-4: KEY PAIR
TO SAFE AND SECURE LOGIN TO SERVER

STEP-5: NETWORK & SECURITY
VPC: VIRTUAL PRIVATE CLOUD
TO CREATE PRIVATE NETWORK

FIREWALL:
TO ALLOW/BLOCK USERS TO CONNECT TO SERVER

SSH : 22 : TO CONNECT WITH SERVER
HTTP/S : 80,443 : TO CONNECT WITH APPLICATION

STEP-6: EBS
PROVIDE STORAGE TO SERVER
MIN: 8GB MAX: 16TB

STEP-7: ADVANCE

```
#! /bin/bash
sudo -i
apt update
apt install nginx  -y
cd /var/www/html/
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* .
```

TIP-1: IF APPLICATION IS LOADING IN BROWSER ?
CEHCK PORT 80 AND 443

TIP-1: IF SERVER IS NOT CONNECTING TO BROWSER ?
CHECK PORT 22

TITLE: MULTI-CLOUD DEVSECOPS WITH AI
DURATION: 4.5 MONTHS +

TIMINGS: 4:00 TO 5:30
CLASSES: MON - SAT
FEE: 16K (2 INSTALLMENTS)
REC: 5K
PROJECTS: 10
TOOLS: 25
AWS: 25
AZURE: 25
GCP: 25
EXP: 4

1. apt update
2. apt install apache2 -y
3. git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
4. mv online-clone-amazon/* /var/www/html

git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* /var/www/html

sudo -i
apt update
apt install apache2 -y
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* /var/www/html

AWS	AZURE	GCP
EC2	VM	

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

HOW TO STORE DATA IN AWS ?
USING S3

WAHT IS S3 ?
SIMPLE STORAGE SERVICE

IN S3 HOW WE CAN STORE DATA ?
USING BUCKETS

WHAT KIND OF DATA CAN WE STORE ON S3 ?
IMAGES, AUDIO, VIDEO, FILES -----

S3:
CRR: CROSS REGION REPLICATION : COPY DATA FROM ONE REGION TO ANOTHER REGION

WHY TO CRR ?
FOR BACKUP
LATENCY ISSUES

SRR: SAME REGION REPLICATION : COPY DATA FROM TO SAME REGION

PRACTICAL PATH:
1. CREATE 2 BUCKETS (1=SOURCE (mumbai), 1=DESTINATION (usa))
2. UPLOAD SOME DATA TO SOURCE BUCKET (Mumbai)
3. CONFIGURATION NEED TO DONE ON SOURCE BUCKET

Amazon S3
Buckets
mynetflixbucket007-hyd (bucket-one-spotifyaudiofile)
management
Replication rules
Create replication rule
Replication rule name: Rule-01
Apply to all objects in the bucket

Destination:
Choose a bucket in this account
rahamclientbuxket (BUCKET-2)
IAM role: Create New Rule
save
no

S3 PRE SIGNED URL:
TO SHARE THE OBJECTS FOR SPECIFIC TIME (EX: 1MIN)
MIN: 1-MIN MAX: 720-MIN
IT COMES WITH ADDITIONAL SECURITY TOKEN IN URL.
select object -- > actions -- > share with pre signed url -- > time -- > save

LIFE-CYCLE:
TO TRANSFER OBJECTS BLW STORAGE CLASSES
https://docs.aws.amazon.com/AmazonS3/latest/userguide/storage-class-intro.html

GO TO BUCKET
MANAGEMENT
lifecycle rule
Create lifecycle rule
Lifecycle rule actions
Transition current versions of objects between storage classes

BATCH OPERATIONS: TO PERFORM SAME OPERATIONS FOR MULTIPLE FILES AT SAME TIME

S3 TRANSFER ACCELERATION: TO TRANSFER FILES VERY FAST.
IT IS PAID FEATURE.

Q1. HOW TO STORE DATA IN AWS CLOUD?
A. USING AWS S3 SERVIE

Q2. WHAT IS S3 ?
A. SIMPLE STORAGE SERVICE
TO STORE OBJECT

Q3. WHAT KIND OF OBJECTS CAN S3 STORE ?
A. IMAGES, VIDEOS, AUDIOS, FILES, CODE ----------

Q4. WHAT IS BUCKET ?
A. ITS A FOLDER WHICH HAS COLLECTION OF FILES

Q5. WHICH COMPANIES ARE USING S3 IN REAL TIME ?
A. NETFLIX, BBC, UNION BANK, SAMSUNG, NASA, SPOTIFY, SNAPCHAT -------

Q6. AS A CLOUD ENGINNER HOW CAN YOU SAVE COST IN AWS STORAGE SERVER ?
A. USING S3 LIFE CYCLE RULES

Q7. HOW CAN YOU SHARE S3 OBJECTS TO USERS VERY SAFE AND SECURLY ?
A. USING S3 PRE-SINGED URL

Q8. HOW TO SHARE DATA BLW TWO AWS ACCOUNTS ?
A: CRR

Q9. HOW DO YOU REDUCE LATENCY FOR USERS IN DIFFERNTE LOCATIONS ?
A : CRR

Q10. HOW DO YOU TRANSFER FILES VERY FASTLY FROM ONE BUCKET TO ANOTHER ?
A. TRANSFER ACCLERATION

Q11. HOW DO YOU WORK WITH MULTIPLE FILES IN S3 ?
A. BATCH OPERATIONS

Q12. WHAT IS THE DEFAULT STORAGE CLASS IN AWS S3 ?
A. STANDARD

Q13. WHAT IS THE LAST STORAGE CLASS IN AWS S3 ?
A. GLACIER DEEP ARCHIVE

Q14. WHAT IS THE AVAILABILITY OF AWS S3 ?
A. 99.999999999

Q15. HOW MUCH OF DATA IS FREE TO STORE IN S3 ?
A. 5 GB

Q16. HOW MANY BUCKETS WE CAN CREATE IN S3 ?
A. 10,000

==========
LINUX COMMANDS:

ROOT USER : ALL PERMISSIONS
NON-ROOT USER : LIMITED PERMISSIONS

sudo -i : to switch to root user
mkdir raham : to create directory (folder)
cd raham : change directory to raham
ls/ll : to list files/folders
touch file1 : to create a file
clear/ctrl l : to clear the screen

vim/vi : its an editor used to write data

i : to insert data
esc :wq : to save and quit the file
cat file1 : to print the content of a file

cp file1 file2 : copy the content from file1 to file3
mv file1 raham : to rename file1 to raham
rm file1 : to remove file1
rm * : to remove all files (* = all)
touch file{1..100}: to create file1 to file100
rm file{1..24} : to remove file1 to file24

SCENARIO BASED QUESTIONS:

1. HOW TO CHECK LOAD OF A SERVER ?
A. htop/top

2. HOW TO CHECK CPU INFROMATION OF A SERVER ?
A. lscpu or cat /proc/cpuinfo

3. HOW TO CHECK RAM INFROMATION OF A SERVER ?
A. lsmem or cat /proc/meminfo

4. HOW TO CHECK RAM INFROMATION OF A SERVER ?
A. lsblk or cat /proc/meminfo

5. HOW TO CHECK UPTIME/RUNTIME OF A SERVER ?
A. htop or uptime

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

AMI: AMAZON MACHINE IMAGE
PURPOSE: TO PROVIDE THE OPERATING SYSTEM, SOFTWARES TO SERVER
WHO: AWS CLOUD WILL GIVE AMI FOR ALL USERS
TYPES: FREE & PAID

CATEGORIES:

QUICK START : AWS
MY AMI : YOU
MARKET PLACE : TO SELL OUR AMIS
COMMUNITY : WE CAN USE OTHER PEOPLE AMI FOR FREE

CREATE A SERVER AND LOGIN TO SERVER
sudo -i

1. apt update
2. apt install apache2 -y
3. git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
4. mv online-clone-amazon/* /var/www/html

CREATING AMI
SELECT SERVER-1
CLICK ON ACTIONS
IMAGE AND TEMPLATES
CREATE IMAGE

IMAGE NAME: SWIGGY-IMAGE
DESCRIPTION: OPTINAL
SAVE

IT WILL TAKE FEW MINS TO MAKE AMI AVAILABLE

HOW TO MIGRATE SERVER :

SELECT AMI
ACTIONS
COPY AMI
DEST: MUMBAIL
COPY

CREATE SERVER IN MUMBAI:

SELECT AMI
LAUNCH INSTANCE FROM AMI
CREATE INSTANCE
START SERVICE

HOW TO RECOVER DELETED AMI ?
Recycle Bin
Retention rules
Create retention rule
Retention rule name - rule01
Resource type: ami
Retention period: 7
create retention rule

AMI -- > SELECT YOUR AMI -- > ACTIONS -- > DEREGISTER AMI
RECYCLE BIN -- > RESOURCES -- > SELECT AMI -- > RECOVER

EARNING-1: WHEN WE USE MORE SERVERS
AWS WILL BLOCK THE CPUS
WE NEED TO INCREASE QUOTAS

LEARNING-2: IF HTTPD SERVICE STOPPED APP WILL NOT RUN
systemctl start httpd
chkconfig httpd on --- > this command make sure our service will stop

QUESTIONS: EXP
1. HOW DO YOU MIGRATE SERVER/APP FROM ONE REGION TO ANOTHER ?
A: CREATE AMI -- > COPY AMI TO OTHER REGION -- > CREATE SERVER FROM AMI

2. HOW DO YOU CREATE BACKUP FOR SERVER ?
A: CREATE AMI/SNAPSHOT

3. HOW TO RECOVER DELETED AMI ?
A: RECOVER FROM RECYCLE BIN (BY CREATING RETENTION RULE)

4. WHAT IS GOLDEN AMI ?
A: It contains the latest security patches, software, configuration for logging and security performance.
==========

STEP-1: CREATE A SERVER-1 AND LAUCH SWIGGY APP
STEP-2: CREATE A SERVER-2 AND LAUCH SWIGGY APP

WHAT IS HA ?
HIGH AVAILABILITY: MAINTANING MORE THAN ONE SERVER.

13.200.212.65

NOTE: RUN THE BELOW SCRIPT ON USER DATA.

```
#! /bin/bash
sudo -i
apt update
apt install nginx  -y
git clone https://github.com/karishma1521success/swiggy-clone.git
mv swiggy-clone/* /var/www/html/
```

52.200.154.169

NOTE: WITHOUT ELB USERS CAN GET APP FROM ONLY ONE SERVER
SO LOAD WILL NOT BE DISTRIBUTED
ULTIMATELTY THE SERVER-1 WILL BE CRASH AND USERS WONT GET ANY APP.

WHEN USERS ARE INCREASING LOAD IS INCREASING.
IF I DONT DISTRIBUTE THE LOAD THEN SERVER WILL BE CRASHED.
TO DISTRIBUTE THE LOAD WE NEED TO CREATE LOAD BALANCER.

WHY TO USE LOAD BALANCER IN REAL TIME:
TO DISTRIBUTE THE LOAD/TRAFFIC

TRAFFIC: INCOMMING REQUEST AND OUTGOING RESPONSE.

TYPES:
1. APPLICATION LOAD BALANCER (REAL TIME)
2. NETWORK LOAD BALANCER
3. GATEWAY LOAD BALANCER
4. CLASSIC LOAD BALANCER

HTTP: HYPERTEXT TRANSFER PROTOCOL : 80
HTTPS: HYPERTEXT TRANSFER PROTOCOL SECURITY: 443

TARGET: SINGLE SERVER
TARGET GROUP: MULTIPLE SERVERS SERVING SAME APPLICATION

PRACTICAL PART:

STEP-1: CREATE 2 SERVERS AND LAUCH SWIGGY APP
STEP-2: SELECT THE LOAD BALANCER
STEP-3: CREATE LOAD BALANCER -- > APPLICATION LOAD BALANCER
STEP-4:
Load balancer name: swiggy
Scheme: Internet-facing
Load balancer IP address type: ipv4
Network mapping: select all Availability Zones
Security groups: select the sg with 80 & 443

STEP-5:
Crete target group
Target group name: amazon-target-group
next
select 2 servers
include as pending below
create target group

Go back to previous tab and click on refresh
and finally click on create load balancer.

HOW TO AD NEW SERVER TO TARGET GROUP:
CREATE SERVER-3 AND DEPLOY SWIGGY APP
TARGET GROUP
SELECT TARGET GROUP
ACTION
REGISTER TARGET
include as pending below
Register pending target

HOW TO ENABLE NEW AZ TO LB:
SELECT LB
NETWORK MAPPING
EDIT
SELECT 1-C SUBNET
SAVE

PROVISIONING MEANS CREATING.

TYPES OF ALOGRTHMS:
1. ROUND ROBIN (DEFAULT)
2. STICKY ROUND ROBIN

3. WEIGHTED
4. IP/URL
5. LEAST CONNECTIONS
6. LEAST TIME
7. RANDOM
8. LEAST BAND WIDTH

HOW TO CHANGE ALOGORITHMS:
SELECET TAGRGET GROUP -- > ATTRIBITES -- > Load balancing algorithm

IMP POINTS:
1. LB WORKS ON OSI MODEL.
2. LB DISTRIBUTE TRAFFIC ON ALOGRITHEMS.
3. DEAFULT ALOGORITHM FOR LB IS ROUND ROBIN.
4. TO CREATE A LB ATLEAST WE NEED TO HAVE 2 SERVERS.
5. IN REAL TIME SERVERS WILL AUTOMATIACLLY ADD TO LB.
6. TO CREATE LB WE NEED TO SELECT ATLEAST 2 AZ.

TYPES OF ROUTING:
1. HOST BASED ROUTING: www.train.paytm.com
2. PATH BASED ROUTING: www.paytm.com/movies, www.paytm.com/recharge
==========

ASG = AUTO SCALING GROUP
WHY: TO ADD/REMOVE SERVERS AUTOMATICALLY.
IF WE DELETE A SERVER ASG WILL RECREATE SAME SERVER.

LOAD IS HIGH -- > ADD THE NEW SERVERS
LOAD IS LOW -- > TO DELETE EXISTING SERVERS

WHEN WE USE AUTO SCALING GROUPS:
IF THE LOAD IS CHANGING FREQUENTLY WE CAN USE ASG.

TYPES OF SCALING:
1. HORIZONTAL SCALING : WE CREATE NEW SERVERS
2. VERTICAL SCALING   : FOR EXISTING SERVERS WE CAN INCREASE CPU & RAM

WEB & APP : HORIZONTAL SCALING
DB   : VERTICAL SCALING

VERTICAL SCALING
1. STOP SERVER
2. ACTIONS -- > INSTANCE SETTINGS -- > CHANGE INSTANCE TYPE -- > t2.medium

3. START SERVER

TRACKING POLICY: USED TO SCALE THE SERVERS
1. CPU
2. NETWORK IN
3. NETWORK OUT
4. COUNT PER TARGET

TEMPLATE: IT CONSIST OF CONFIGURATION OF A SERVER WHICH IS CREATED BY ASG.
WITHOUT TEMPLATE WE CANT CREATE ALL SERVERS WITH SAME CONFIG.

STEPS TO CREATE ASG:
AUTO SCALING GROUP -- > CREATE
NAME: SWIGGY
CREATE A LAUNCH TEMPLATE
NAME: SWIGGY-TEMPLATE
NOTE: GIVE THE CONFIGURATION JUST LIKE WE GIVE FOR EC2.
USE SCRIPT ON USERDATA

SCRIPT:
```
#! /bin/bash
sudo -i
apt update
apt install nginx  -y
cd /var/www/html/
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* .
```

CREATE TEMPLATE

AZ AND SUBNETS: US-EAST-1A, US-EAST-1B, US-EAST-1C -- NEXT
LOAD BALANCING: ATTACH TO A NEW LOAD BALANCER
INTERNET FACING
SELECT CREATE A NEW TARGET GROUP
NEXT

DESIRED : 2 -- > HOW MANY SERVERS YOU WANT NOW
MIN: 2 -- > ALWAYS ATLEAST I WANT 2 SERVERS
MAX: 10 -- > IF LOAD IS INCREASE I WANT 10 SERVERS

AUTOMATIC SCALING POLICY:
TARGET TRACKING SCALING POLICY:
CPU -- > VALUE: 50 -- > NEXT

http://internal-swiggy-1-1906495425.ap-south-1.elb.amazonaws.com/

NOTIFICATION -- > CREATE A TOPIC -- > NAME: SWIGGY SERVERS -- > EMAIL: give your email -- > NEXT -- > NEXT -- > CREATE AUTO SCALING GROUP

HOW TO INCREASE LOAD:

LOGIN TO SERVER:
apt update
apt install stress -y
stress
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 500s

NOTE: TO DELETE SERVERS WE NEED TO DELETE ASG.
if you remove servers directly because of auto scaling group servers are going to recreate.

Q1. WHY WE NEED ASG ?
A: IF THE LOAD IS CHANGING FREQUENTLY WE CAN USE ASG.

Q2. TYPES OF ASG ?
A. 1. HORIZONTAL SCALING : WE CREATE NEW SERVERS
2. VERTICAL SCALING : FOR EXISTING SERVERS WE CAN INCREASE CPU & RAM

Q3. HOW ASG IS GOING TO SCALE SERVERS ?
A. TRACKING POLICY: USED TO SCALE THE SERVERS
1. CPU
2. NETWORK IN
3. NETWORK OUT
4. COUNT PER TARGET

Q4. HOW WILL ALL THE SERVERS IN ASG WILL GET SAME CONFIG ?
A. TEMPLATE

Q5. WHAT IS COOLING PERIOD ?
A. AMOUNT OF TIME TAKEN BY SERVER BEFORE DELETE WHEN LOAD IS DECREASED

Q6. IF YOU DELETE A SERVER WILL ASG CREATE NEW SERVER ?

A. YES

Q7. WHAT IS WARMUP PERIOD ?
A. AMOUNT OF TIME TAKEN BY SERVER BEFORE LANUCHING IN ASG.

Q8. HOW WILL ASG KNOWS ABOUT CPU PERCENTAGE OF SERVER ?
A. USING CLOUD WATCH

==========

MONITORING:

CLOUD INFRA: COMBINATION OF HARDWARE & SOFTWARE
RESOURCES USED TO RUN APP ON CLOUD.
EX: EC2, ELB, ASG, S3, VPC ----------

CLOUD WATCH: TO MONITOR THE CLOUD RESOURCE
NOTE:BY DEFAULT MONITORING FOR EC2 IS ENABLED

SERVER : APP
AMI : COPY OF SERVER
ELB : DISTRIBUTE LOAD BLW SERVERS
ASG : CREATE SERVERS AUTOMATICALLY
CLOUD WATCH : TO GET METRICS OF CLOUD RESOURCES

METRICS MEANS INFORMATION ABOUT SERVER.
METRIC : CPU, RAM, DISK -------

STEP-1: CREATE EC2 AND DEPLOY THE APPLICATION

CREATE A SERVER AND LOGIN TO SERVER
sudo -i

#this command will download webserver(nginx) git(to get code)
apt update
apt install nginx -y

#This is the path where we store frontend code in server

cd /var/www/html/
rm index.html (opt)

#This command will download the code from GitHub
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git

mv online-clone-amazon/* .

STEP-2: CREAT A DASHBOARD
CLOUD WATCH:
CREATE DASHBOARD
NAME: DASHBOARD1
WIDGET: LINE
EC2
EC2: PER INSTANCE METRICS
GIVE ID OF YOUR SERVER
SELECT CPUUTILIZATION
CREATE WIDGET

TO GET MORE LOAD:
sudo -i
apt install stress -y
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 100s

EPEL: EXTRA PACKAGES FOR ENTERPRISE LINUX

GO TO DASHBOARD AND CLICK ON SHARE THE DASHBOARD
GENERATE A LINK AND COPY PASTE

CREATE ALARM
CLOUD WATCH
ALL ALARM
CREATE ALARM
SELECT METRIC
EC2
EC2: PER INSTANCE METRICS
CPUUtilization
60 -- > NEXT

CREATE A SNS TOPIC -- > new topic -- > email
EC2 ACTION:STOP

==========

EFS: ELASTIC FILE SYSTEM
PURPOSE: TO SHARE DATA BLW TWO SERVERS
SIZE: GROW UP TO PETABYTES
TYPE: SERVER LESS
PROTOCOLS: NFSV4.0 & NFSV4.1

INTEGRATIONS: EC2, ECS, EKS, Lambda, Fargate.
MODES: 1. GENERAL PURPOSE 2. ELASTIC
PRICING: 5 GB/YEAR FREE
BACKUP: WE CAN GET BACKUPS

EFS -- > CUSTOMIZE -- > NAME: ONE -- > REGIONAL -- > ELASTIC -- > GIVE SG WITH NFS ENABLE -- > NEXT -- > CREATE.

NOTE: IF NFS IS NOT ENABLE ON SG YOU CANT GET THE DATA.

PRACTICAL PART:

GO TO SERVER AND RUN BELOW COMMANDS:

```
yum install httpd git -y
systemctl start httpd
ls /var/www/html/
```

GO TO EFS -- > SELECT EFS -- > ATTACH -- > COPY PASTE COMMANDS ON BOTH SERVERS

```
sudo mount -t nfs4 -o
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport
fs-02ea223ceb1c70423.efs.us-east-1.amazonaws.com:/
/var/www/html
```

NOW LETS CHECK WITH CODE

```
cd /var/www/html
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* .
```

NOTE:
WE CANT INTEGRATE EFS DIRECTLY TO ASG
CEATE A SERVER -- > CREATE AMI -- > ASG

==========

MODIFIYING VOLUME:
SELECT SERVER -- > CLICK ON STORAGE -- > SELECT VOLUME -- > ACTIONS -- > MODIFY

ADDING ANOTHER VOLUME
CREATE A VOLUME
VOLUME -- > CREATE -- > SIZE: ABC -- > AZ: SELECT AS YOUR SERVER AZ -- > TAGS: Name = New server -- > CREATE

ATTACHING VOLUME TO SERVER:
SELECT VOLUME -- > ACTIONS -- > ATTACH -- > SELECT YOUR SERVER -- > /dev/xvdb -- > save

BOOT/ROOT VOL : WHICH HAS OS
NON-ROOT/DATA VOL: WHICH WILL NOT HAVE OS

SERVER MIGRATION:

1. CREATE A SERVER ON US-EAST-1A AND DEPLOY APP
USE MY AMAZON SCRIPT

2. TAKE A SNAPSHOT OF SERVER
SELECT SERVER -- > STORAGE -- > VOLUME -- > ACTIONS -- > CREATE SNAPSHOT -- > CREATE

3. COPYING THE SNAPSHOT:
SELECT SNAPSHOT -- > ACTIONS -- > COPY SNAPSHOT -- > DESTINATION: AP-SOUTH-1 -- > COPY

4. CREATE EBS FROM SNAPSHOT
SELECT SNAPHSOT -- > ACTIONS -- > CREATE A VOLUME -- > AZ: US-EAST-2B -- > CREATE

5. CREATE A SERVER ON AP-SOUTH-1 & DETACH EXISTING VOLUME
CREATE A SERVER -- > SELECT -- > STOP

VOLUME -- > SELECT -- > ACTIONS -- > DETACH

6. ATTACH THE VOLUME CREATED FROM SNAPSHOT
VOLUME -- > ACTIONS -- > ATTACH -- > SELECT SERVER -- > DEVICENAME: FIRST OPTION -- > ATTACH

Q1. HOW DATA WILL BE STORED ON EC2 ?
A. EBS

Q2. HOW TO TAKE SERVER BACKUP ?
A. SNAPSHOTS

Q3. ARE SNAPSHOTS & EBS REGION AND AZ SPECIFIC ?
A. YES

Q4. MIN AND MAX SIZE OF EBS ?
A. 8 GB - 16 TB

Q5. CAN WE ATTACH MULTIPLE EBS TO SERVERS AT A TIME ?
A. YES

Q6. HOW TO SAVE COST OF SNAPSHOTS >
A. ARCHIVE THEM (RECOVERY PERIOD IS 24 TO 72)

Q7. CAN WE ATTACH SNAPSHOT TO EC2 DIRECTLTY ?
A. NO

Q8. TYPES OF VOLUMES IN EBS ?
A. SSD & HDD

AMI VS SNAPSHOTS:

FIRST TIME WE USE AMI TO LAUNCH SERVERS
SNAPSHOTS WE USE TO TAKE BACKUP DAILY.

1 -- > 10 files -- > ami
10 -- > 100 files -- > snapshot
==========

IAM : PART-1 : 21-07-2025

IAM: IDENTITY & ACCESS MANAGEMENT
WHY TO USE: TO PROVIDE PERMISSIONS AND RESTRICTIONS TO USERS IN AWS.

AUTHENTICATION : PERMISSION TO LOGIN
AUTHORIZATION : PERMISSION TO WORK
POLICY: SET OF PERMISSIONS

IN REAL TIME WE DONT USE ROOT USER FOR ROUTINE WORKS
WE USE AWS IAM USER IN REAL TIME. (FOR DAILY WORKS)

IAM USER:

IAM -- > USERS -- > CREATE USER -- > NAME: ABC -- > Provide user access to the AWS Management Console -- > I want to create an IAM user -- > Next -- > Attach policies directly -- > AmazonReadOnlyAccess -- >

IAM GROUP:
USED TO GIVE PERMISSION FOR MULTIPLE USERS.
A SINGLE USER CAN BE PART OF MAX 10 GROUPS.
GROUP CANNOT BE NESTED.
IF USER HAVE PERMISSION ON GROUP LEVEL AND USER LEVEL BOTH OF THE APPLIES.

create few users without permissions

IAM -- > USER GROUPS -- > NAME: DEVOPS -- > SELECT USERS -- > Attach policies directly -- > CREATE

IF U FROGOT USERNANE &PASSWORD:
SELECT USER -- > SECURITY CREDNTIALS -- > MANAGE CONSOLE ACCESS

IAM ROLES:
ROLES ARE USED BY SERVICES.
ROLE IS SIMLAR TO IAM USER.
ROLES ALSO HAVING PERMISSION LIKE USERS.
ROLE DOESNT HAVE ANY CREDS.

WE CAN SET TIME LIMIT FOR ROLES.
IF TWO SERVICES WANT TO COMMUNICATE OR WORK TOGETHER WE USE ROLES.
EX: EC2 -- > S3

http://54.242.59.171/

ROLE -- > CREATE ROLE -- > AWS SERVICE -- > EC2 -- > PERMISSIONS: S3 FULL ACCESS -- > NAME: S3 ROLE -- > CREATE

ATTACH ROLE TO SRVER:
SELECT SERVER -- > ACTIONS -- > SECURITY -- > MODIFY IAM ROLE -- > S3 ROLE -- > UPDATE

DEPLOY THE APP FROM SCRIPT
GO TO SERVER

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
apt install unzip -y
unzip awscliv2.zip
sudo ./aws/install

command: aws s3 ls -- > execute this on server
aws s3 cp /var/log/ngix/access.log s3://bucketname

REVOKE SESSION: TO REMOVE THE EXISTING PERMISSION OF A ROLE.

INLINE POLICY: CREATING CUSTOM POLICY FOR RESOURCES

USER -- > ADD PERMISSION -- > CREATE INLINE POLICY -- > SERVICE: S3 -- > All S3 actions (s3:*) -- > bucket: BUCKETNAME -- > next

Q1. DIFF BLW AUTHENTICATION VS AUTHORIZATION ?
A. AUTHENTICATION : PERMISSION TO LOGIN
AUTHORIZATION : PERMISSION TO WORK

Q2. HOW TWO SERVICES COMMUNICATE WITH EACH OTHER ?
A USING ROLES

Q3. CAN WE CREATE NESTED GROUPS IN AWS ?
A NO

Q4. HOW MANY GROPUPS CAN A SINGLE USER CAN BE PART OF ?
A. 10

Q5. DIFF BLW IAM USER & ROLES >
A   USER WILL HAVE CREDS, ROLES WILL NOT HAVE
    USER WONT HAVE TIME LIMIT BY DEFAULT, ROLES WILL HAVE
    USER: USER -- SERVIVE ROLE: SERVIVE -- SERVIVE

Q6. CAN WE ATTACH MULTIPPLE ROLES TO USER ?
A   YES

Q7. HOW DO YOU RESTRICT A SPECIFIC PERMISSION TO USER ?
A. PERMISSION BOUNDARY

Q8. BY DEFAULT FOR A USER HOW MANY KEYS WE CAN CREATE ?
A. 2

Q9. IF YOU LOST YOUR KEYS WHAT YOU ARE GOING TO DO ?
A. DEACTIVE THE KEYS

Q10. I HAVE A SPECIAL BUCKET WHERE MY CUSTOMRE NEED TO ACCESS FILES HOW DO YOU PROVIDE PERMISSION TO THAT CUSTOMER ?
A. INLINE POLICYUSER -- > KEYS (PERMISSIONS) -- > SERVER

Q11. WHAT IS THE POLICY FORMAT ?
A. JSON(JAVA SCRIPT OBJECT NOTATION)

==========
==
ACCESS KEY & SECRET ACCESS KEY = USED TO ASSIGN PERMISSION TO SERVER
BY DEFAULT WE CAN CREATE ONLY 2 KEYS.
WHAT WILL YOU DO WHEN YOU LOST THE KEYS?
ANS: DEACTIVATE THE KEYS
NOTE: PLS DONT DELETE THEM WITHOUT DEACTIVATE

FLOW: USER -- > KEYS -- > ATTACH TO SERVER -- > SERVER

CREATE A USER WTH ADMIN ACCESS

CREATE A USER -- > SECURITY CREDENTIALS -- > CREATE ACCESS KEY -- >
Command Line Interface (CLI)-- > create

LINUX:
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

WINDOWS:
msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi

aws configure -- > Run this command on server

AWS Access Key ID [None]: ********** [pls put your own keys :(]
AWS Secret Access Key [None]: **********
Default region name [None]: ap-east-1
Default output format [None]: table

CLI -- > COMMAND LINE INTERFACE
WHY TO USE: TO OPERATE/CONTROL AWS SERVICES THROUGH COMMANDS

aws s3 ls : to list the buckets
aws s3 ls s3://bucketname : to list files inside the bucket
aws s3 mb s3://newbucket : to create a bucket
aws s3 rb s3://newbucket : to delete a bucket
aws s3 cp file s3://bucket : to copy file1 to bucket
aws s3 cp copy-s3-uri . : to copy files from bucket to server
aws s3 cp aws s3://bucket --recursive : to copy folder to bucket
aws s3 sync s3://bucket1 s3://bucket2 : to copy all files from one bucket to another.

IAM:
aws iam list-users : TO LIST IAM USERS
aws iam list-groups : TO LIST IAM GROUPS
aws iam list-roles : TO LIST IAM ROLES

aws iam create-group --group-name devops
aws iam create-user --user-name rahamabc
aws iam create-role --role-name demorole

aws iam delete-group --group-name devops
aws iam delete-user --user-name rahamabc
aws iam delete-role --role-name demorole

EC2:

aws ec2 run-instances --image-id ami-05fcfb9614772f051 --instance-type t2.micro --count 1
aws ec2 describe-instances : to show complete info of all the servers
aws ec2 stop-instances --instance-ids i-054a200a4effc917c
aws ec2 start-instances --instance-ids i-054a200a4effc917c
aws ec2 terminate-instances --instance-ids i-054a200a4effc917c

CUSTOM PASSWORD POLICY:
IAM -- > Account Settings -- > Password policy -- > Edit -- > save
Needs to be between 6 and 128.

USER CREDS REPORT:
The credentials report lists all your IAM users in this account and the status of their various credentials. After a report is created, it is stored for up to four hours.

IAM
Credential Report

PERMISSION BOUNDARY: TO RESTRICT A USER TO NOT USE A SPECIFIC SERVICE.

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

FIRST EVER DOMAIN NAME : https://symbolics.com/ -- > 1985

NAME RESOLUTION : PROCESS OF CONVERTING IP TO DOMAIN (1.2.3.4 = WWW.SWIGGY.COM)
DNS: DOMAIN NAME SYSTEM : 53

ROUTE 53

DNS: DOMAIN NAME SYSTEM
PORT: 53
PURPOSE: CONVERTING IP TO DOMAIN (192.168.1.0 = www.swiggy.com)
DOMAIN PROVIDERS: GODADDY, BIGROCK, HOSTINGER, AWS, --------

TYPES OF RECORDS:

A : IPV4
AAAA : IPV6
CNAME : HOSTNAME --> HOSTNAME
MX : MAIL SERVER

```
#! /bin/bash
sudo -i
apt update
apt install nginx  -y
cd /var/www/html/
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* .
```

NAME SERVER: IT TELLS BROWSER TO CONNECT EXACT SERVER WHERE YOUR APP RUNS
EX: GUIDE FOR A TOUR

PRACTICAL PART:
STEP-1: BUY A DOMAIN FROM DOMAIN FROM BELOW SITES (GODADDY,HOSTINGER, BIG ROCK)
STEP-2: CREATE A SERVER IN AWS AND LAUNCH AMAZON APP
STEP-3: GO TO ROUTE 53 -- > HOSTED ZONES -- > CREATE -- > GIVE DOMAIN -- > PUBLIC -- > CREATE
STEP-4: UPDATE NAME SERVER VALUES IN DOMAIN REGISTRAR (BIG ROC * : : K)
STEP-5: CREATE RECORD -- > GIVE IP -- > SAVE

NOW LETS CONVERT LOAD BALANCER DNS TO OUR OWN DNS
CREATE ONE MORE SERVER AND ATTACH BOTH SERVERS TO LOAD BALANCER

STEP-6: SELECT RECORD -- > EDIT --> ALIAS --> SELECT APP LB -- > REGION -- > SELECT LB -- >

==========

SESSION-19: CLOUD TRAIL : 21-02-2025

CLOUD TRAIL: IT WILL SHOW SERVICES ACCESSED BY USER
BY DEFAULT IT STORES LAST 90 DAYS OF ACTIVITIES.
$2.00 per 1,00,000 management events delivered.
$0.10 per 1,00,000 network activity events delivered

BY DEFAULT CLOUD TRAIL WILL RECORD ALL EVENTS
BUT I WANT ON SPECIFI EVENTS (S3, EC2) WE NEED TO CREATE CLOUD TRAIL

Trail name: my-users-events
Log file SSE-KMS encryption: Disable
NEXT
Events: Management & Data
Resource: s3
Create trail

Q1. HOW DO YOU TRACK USERATVITIES IN AWS ?
A: CLOUD TRAIL

Q2. ONE OF THE USER DELETED ON SERVER IN AWS ACCOUNT HOW DO YOU FIND THEM ?
A: CLOUD TRAIL

Q3. BY DEFAULT HOW MANY DAYS EVENTS SHOULD BE STORED ?
A: 90 DAYS

Q4. CAN WE FILER EVENTS SEPERATELY FOR A RESOURCE ?
A: YES

CDN: CONTENT DELIVERY NETWORK
USED TO DELIVER APP FROM EDGE LOCATION
IT GIVES FAST RESPONSE
EX: CRICKET MATCHES, E-COMMERCE SALES, ------

WE NEED TO CREATE ORIGIN
ORIGIN: FROM WHERE YOU ORIGINAL APPLICATION IS COMMING

EXAMPLE OF ORIGINS: S3, ELB, API GATEWAY

Origin Shield
Its an additional caching layer that can help reduce the load on your origin and help protect its availability.

ADVANTAGES:
1. REDUCE LATENCY
2. CUT COST
3. CUSTOMIZE DELIVERY
4. SECURITY

FREE TIER:
1 TB of data transfer out
10,000,000 HTTP or HTTPS requests
2,000,000 CloudFront Function invocations
Each month, always free

SCRIPT:

```
#! /bin/bash
sudo -i
apt update
apt install nginx  -y
cd /var/www/html/
git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
mv online-clone-amazon/* .
```

NOTE: FROM LB WE NEED TO ACCESS APP FROM HTTP NOT HTTPS

STEP-1: CREATE 2 SERVERS AND DEPLOY AMAZON APP
STEP-2: CREATE A LOAD BALANCER
SETP-3: CLOUD FRONT -- > Name: -->
Single website or app -- > ORIGIN DOMAIN: ELB (SELECT YOUR LB) -- > Protocol: HTTP only (ORIGIN PROTOCOL)
-- > Enable Origin Shield: US-EAST-1 -- > Protocol: HTTPS(CLOUD FRONT PROTOCOL) -- > SELECT WAF -- > IPv6: OFF -- > CREATE

NOTE: IT WILL TAKE 2 MINS TO ACTIVATE

HOW TO BLOCK USERS FROM DIFF LOCATIONS:
SECURITY -- > CloudFront geographic restrictions

Q1. HOW DO YOU INCREASE SPEED OF APP TO USRES ?
A. USING CLOUD FRONT

Q2. WHAT IS EDGE LOCATION ?
A. FROM WHERE YOU ORIGINAL APPLICATION IS COMMING

Q3. TELL ME FEW ORIGINS FOR CDN ?
A. S3, ELB, API GATEWAY

Q4. CAN WE REDIRECT HTTP TO HTTPS AUTOMATICALLY IN CDN ?
A. YES

Q5. WHAT IS TTL ?
A. TIME TO LIVE

==========

WALL : VPC
PUB-SUB : HALL
MAIN GATE : IGW

CIDR: CLASSLESS INTER DOMAIN ROUTING

VPC = WALL

STEP-1: CREATE VPC
VPC -- > CREATE VPC -- > NAME: SWIGGY -- > CIDR: 10.0.0.0/16 -- > CREATE

SUBNET = ROOM

STEP-2: CREATE SUBNET [PUBLIC = HALL]
SUBNET -- > CREATE -- > VPC: SWIGGY -- > NAME: WEB-SUBNET -- > CIDR: 10.0.0.0/24 -- > CREATE

STEP-3: CREATE A INTERNET GATEWAY [IGW = MAIN GATE]
INTERNET GATEWAY -- > CREATE -- > NAME: SWIGGY-IGW -- > CREATE -- > ATTACH TO VPC -- > SWIGGY

STEP-4: CREATE ROUTE TABLE
ROUTE TABLE -- > NAME: SWIGGY-WEB-RTB -- > VPC: SWIGGY -- > CREATE
EDI ROUTES -- > ADD -- > 0.0.0.0/0 -- > IGW : SWIGGY -- > ADD

ASSOCIATE THE WEB SERVER TO THE ROUTE TABLE.
SUBNET ASSOCIATION -- > SELECT WEBSUBNET -- > ASSOCIATE.

CREATE A WEB SERVER WITH SWIGGY VPC AND WEB SUBNET.
NOTE: SELECT EDIT OPTION IN NETWORK
VPC   : SWIGGY
SUBNET   : SWIGY-WEB-SUBNET

NOTE: ENABLE AUTO ASSIGN PUBLIC IP FOR WEB SERVER WHILE CREATING.
CREATE NEW SG

http://13.233.224.47/

#! /bin/bash
sudo -i
apt update
apt install nginx -y
cd /var/www/html/
git clone https://github.com/karishma1521success/swiggy-clone.git
mv swiggy-clone/* .

STEP-5: CREATE SUBNET (PRIVATE =APP)
SUBNET -- > CREATE -- > VPC: SWIGGY -- > NAME: APP-SUBNET -- > CIDR: 10.0.1.0/24 -- > CREATE

STEP-6: CREATE A NAT GATEWAY
NAT GATEWAY -- > CREATE -- > NAME: SWIGGY-NAT -- > SUBNET: WEB SUBNET -- > Elastic IP allocation -- > CREATE

STEP-7: CREATE ROUTE TABLE
ROUTE TABLE -- > NAME: APP-RTB -- > VPC: SWIGGY -- > CREATE
EDI ROUTES -- > ADD -- > 0.0.0.0/0 -- > NAT : SWIGGY -- > ADD
ASSOCIATE THE APP SERVER TO THE ROUTE TABLE.
SUBNET ASSOCIATION -- > SELECT APPSUBNET -- > ASSOCIATE.

CREATE A APP SERVER WITH SWIGGY VPC AND APP SUBNET.
NOTE: SELECT EDIT OPTION IN NETWORK
VPC   : SWIGGY
SUBNET   : SWIGY-APP-SUBNET

NOTE: DISABLE AUTO ASSIGN PUBLIC IP FOR APP SERVER WHILE CREATING.
CREATE NEW SG

NOW CONNECTING TO APP SERVER:

OPEN WEB SERVER & sudo -i
vim pemfile -- > copy & paste the pem file content -- > chmod 400 pemfile
ssh "pemfile" ec2-user@public-ip

PEERING: ESTABLISHING CONNECTION BLW 2 VPCS

TRANSIT GATEWAY:
networking service that allows you to connect multiple VPCs, on-premises networks, and other AWS resources through a central hub.

VPC Endpoints: provide a secure and private connection between your VPC & AWS services, without requiring access over the public internet.
This is useful for scenarios where you want to avoid exposing sensitive data to the internet while enabling private communication between resources in your VPC and services like Amazon S3, DynamoDB, or other AWS services.

==========
RDS
DATABASE
CREATE DATABASE
STANDARD
ENGINE: MYSQL
TEMPLATE: 3 INSTANCES
CREDS: AWS SECRET MANAGER
INSTANCE CONFIG: DB.M5.2XLARGE
CONNECT TO EC2: SELECT YOUR DB SERVER

INSTALL MYSQL IN AMAMZON LINUX:
sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm
sudo dnf install mysql80-community-release-el9-1.noarch.rpm -y
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2023
sudo dnf install mysql-community-client -y
sudo dnf install mysql-community-server -y

INSTALLING ON UBUNTU:
sudo apt update
sudo apt install mysql-server -y

mysqld --version
sudo systemctl status mysql

AFTER CREATING DB TO GET THE CREDS

VIEW CREDS
MANAGE CREDS
SECERET VALUE
REVEAL

CONNECTION TO DATABASE:
DATABASE -- > CONNECTIVITY -- > END POINT
database-1-instance-1.us-east-1.rds.amazonaws.com

NOW GO TO EC2 AND RUN BELOW COMMAND TO CONNECT
mysql -u admin -h endpoint -p

GETTING PASSWORD:
DATABASE -- > CONFIGUARTION -- > Master credentials ARN -- > Manage in Secrets Manager
Retrieve secret value

```
create database raham;
show databases;
use raham;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    password VARCHAR(20),
    city VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO users (name, email, password, city) VALUES ('ramu', 'ramu@gmail.com', 'ramu123', 'rajavarm' );
INSERT INTO users (name, email, password, city) VALUES ('remo', 'remo@gmail.com', 'remo143', 'vizag');
```

INSERT INTO users (name, email, password, city) VALUES ('aparichith',
'aparichith@gmail.com', 'kumbipakam', 'hell');

SELECT * FROM users;
exit

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

CFT: CLOUD FORMATION TEMPLATE
WE CEATE RESOURCES THROUGH CODE
CODE:YAML/JSON

YAML = YET ANOTHER MARKUP LANGUAGE
JSON = JAVA SCRIPT OBJECT NOTATION

NAME : RAHAM
LOC : HYD
COMPANY : NIT
PRO : TECHIE
AGE : *****

ADVANTAGES:
1. WRITE CODE FOR ONCE AND USE MULTIPLE TIMES
2. WE CAN SAVE TIME
3. WE CAN AVOID MANUAL WORK
4. WE CAN LIMIT THE MISTAKES

TEMPLATE: IT IS A FILE WHICH CONSIST OF RESOURCE INFROMATION.
STACK: GROUP OF RESOURCE

Specifying templates:
1. S3
2. LOCAL
3. GITHUB
4. BUILD COMPOSERS

STEPS FOR PRACTICAL PART:
1. SELECT CFT -- > Create Stack
2. Build from Infrastructure Composer
3. Drag and drop the ec2 resource and copy code from chatgpt
4. validate -- > create template -- > next

5. CREATE SNS TOPIC FOR SENDING EMAIL.

PROMPT-1: SIMPLE PROMPT
generate a cft code for aws vpc

PROMPT-2: HIGH LEVEL PROMPT
as an experienced cloud engineer generate a powerful cft script to create vpc subnet igw and routetable in a simplified way that need to have very less code

STATUS:
CREATION
DELETION
UPDATION
ROLLBACK: IF ONE RESOURCE IN STACK GOT FAILED, IT WILL DELETE ALL RESOURCES.

```
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-020cba7c55df1f615
      InstanceType: t2.micro
      KeyName: Rkeypair
      SecurityGroups:
        - all
      Tags:
        - Key: Name
          Value: raham
```

CHANGE SET: it is created when we want to modify the existing resource configuration.
create changeset
execute changeset.

update stack
create changeset

Edit in Infrastructure Composer
template
t2.medium=t2.large
validate

create changeset
next
next
next
submit
select changeset - > execute

VPC CODE:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Minimal VPC setup with subnet, IGW, and route table

Resources:

  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.10.0.0/16
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags: [{ Key: Name, Value: MyMinimalVPC }]

  Subnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: 10.10.1.0/24
      MapPublicIpOnLaunch: true
      Tags: [{ Key: Name, Value: PublicSubnet }]

  IGW:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags: [{ Key: Name, Value: IGW }]

  AttachIGW:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref VPC
      InternetGatewayId: !Ref IGW

  RouteTable:
```

```
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags: [{ Key: Name, Value: PublicRouteTable }]

  DefaultRoute:
    Type: AWS::EC2::Route
    DependsOn: AttachIGW
    Properties:
      RouteTableId: !Ref RouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref IGW

  SubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref Subnet
      RouteTableId: !Ref RouteTable

Outputs:
  VPCId:
    Value: !Ref VPC
  SubnetId:
    Value: !Ref Subnet
  InternetGatewayId:
    Value: !Ref IGW
```

ASSIGNMENT-1:
ALL OF THE BELOW ACTIONS YOU NEED TO DO IN IAM USER

1. CREATE A VPC WITH 2 PUBLIC SUBNETS AND 2 PRIVATE SUBNETS, ROUTE TABLES, NAT, IGW ---

2. CREATE 2 SERVERS ON PUBLIC SUBNETS AND LAUCH AMAZON APP
3. ATTACH A LOAD BALANCER TO THAT SERVER
4. CREATE A CLOUD FRONT FOR LOAD BALANCER
5. STORE THE LOGS IN AWS S3 BUCKETS (/var/log/httpd/access_log)
6. S3 BUCKET MUST CREATE BY USING CLI COMMAND & LOGS ALSO COPIED BY CLI COMMAND
7. CREATE A CLOUD WATCH DASHBOARD FOR WEB SERVERS
8. CREATE SNAPSHOT AND AMI FOR THE SERVERS

OPTINAL : TRY TO CREATE EVERYTHING USING CFT

==========

GIT: GLOBAL INFORMATION TRACKER
VCS: VERSION CONTROL SYSTEM
SCM: SOURCE CODE MANAGEMENT

VCS: TO STORE EACH VERSION OF CODE SEPERATELY.

V1 : 1 SERVICES : 100 LINES -- > REPO-1
V2 : 2 SERVICES : 200 LINES -- > REPO-2
V3 : 3 SERVICES : 300 LINES -- > REPO-3

WHY WE NEED TO STORE THEM SEPERATELY >
TO DO ROLLBACK: GOING BACK TO PREVIOUS VERSION OF APPLICATION
V3 -- > V2

V1 : INDEX.HTML : 10
V2 : INDEX.HTML : 20
V3 : INDEX.HTML : 30

WHAT IS GIT?

Git is used to track the files.
It will maintain multiple versions of the same file.
It is platform-independent.
It is free and open-source.
They can handle larger projects efficiently.
It is 3rd generation of vcs.
it is written on c programming
it came on the year 2005

CVCS: CENTRALIZED VERSION CONTROL SYSTEM
STORES CODE ON SINGLE REPO
Ex: SVN

DVCS: DISTRIBUTED VERSION CONTROL SYSTEM
STORES CODE ON MULTIPLE REPO
Ex: GIT

STAGES/ARCHITECTURE OF GIT:

IN GIT WE HAVE TOTAL 3 STAGES:

1. WORKING DIRECTORY : WHERE WE WRITE THE CODE
2. STAGING AREA : WHERE WE TRACK THE CODE
3. REPOSITORY : WHERE WE STORE THE TRACKED THE CODE

PRACTICAL PART:

GIT DOWNLOADING LINK: (COPY PASTE BELOW LINK IN BROWSER TO DOWNLOAD GIT)
https://github.com/git-for-windows/git/releases/download/v2.48.1.windows.1/Git-2.48.1-64-bit.exe

OPEN GITBASH IN DOWNLOAD FROM CHROME
NEXT --> NEXT -- > UNTILL IT INSTALL
OPEN GITBASH ON YOUR WINDOWS

cd paytm
mkdir paytm

ISSUE: fatal: not a git repository (or any of the parent directories): .git
SOL: run git init command

git init : used to Initialize the empty repository
with out .git commands will not work.

touch index.html : to create a file
git status : to show the file status
git add index.html : to track the file

NOTE: WITHOUT CONFIGURNING USER NAME AND EMAIL COMMITS WILL NOT HAPPEN.

git config --global user.name "raham"
git config --global user.email "raham@gmail.com"

git commit -m "commit-1" index.html : to commit a file
git log   : to show commits history
git log --oneline   : commits on single line
git log --oneline -2   : to show last 2 commits

create -- > track -- > commit (CTC)
touch -- > git add -- > git commit

Note: here every dev works on the local laptop
at the end we want all dev codes to create an application.
so here we use GitHub to combine all dev codes together.

Create a GitHub account and create Repo

[NOTE: PLEASE I KINDLY REQUEST TO COPY AND PASTE YOUR ACCOUNT LINK :( ]

git remote add origin https://github.com/devopsbyraham/paytm.git
git push origin master
IT WILL ASK AUTHENTICATON IN OTHER TAB PLS GIVE THE DETAILS
Authentication Succeeded

git show: to show the file that committed for specific commit id.
git show commit_id

==========

BRANCHES:
To Push the code from git to GitHub we need to have a branch.
It's an individual line of development for code.
we create different branches in real-time.
each developer will work on their own branch.
At the end we will combine all branches together on GitHub.
Default branch is Master. {git=master, github=main}
Common Branches: Master, Main, Release, Hot-fix -------

git branch   : to list the branches

git branch movies : to create a new branch
git checkout movies : to switch from one branch to another.
git checkout -b recharge: to create and switch from one branch to another.
git branch -m old new : to rename a branch
git branch -D movies : to delete a branch

we cant delete the current branch

PROCESS:

mkdir Paytm
cd Paytm
git init
touch index.html
git add index.html
git commit -m "commit-1" index.html

git branch movies
git checkout movies
touch movies{1..5}
git status
git add movies*
git commit -m "dev-1 commits" movies*
git push origin movies

git branch dth
git checkout dth
touch dth{1..5}
git status
git add dth*
git commit -m "dev-2 commits" dth*

git push origin dth

git checkout -b train
touch train{1..5}
git add train*
git commit -m "dev-3 commits" train*

git push origin train

git checkout -b recharge
touch recharge{1..5}
git add recharge*
git commit -m "dev-4 commits" recharge*

git push origin recharge

In real time we take all the code from one branch to deploy the app
so we need to add all branches in github
TO MERGE BRANCHES IN GITHUB WE NEED TO CREATE PULL REQUEST

PULL REQUEST -- > CREATE NEW PULL REQUEST -- > SELECT MOVIES -- > CREATE --> MERGE -- > CONFIRM
NOTE: FOLLOW GREEN BUTTON

GIT MERGE: it will merge files blw two different branches in git

git checkout master
git merge movies
git merge train

GIT REBASE: used to add files blw two different branches in git
git checkout master
git rebase dth
git rebase recharge

MERGE VS REBASE:
merge for public repos, rebase for private
merge stores history, rebase will not store the entire history
merge will show files, rebase will not show files

in GitHub we use Pull Request (PR) to do merging.

GIT REVERT: used to revert(get back) the files we merge.
to undo merge we can use revert.

git revert dth
git revert recharge

GITHUB REVERT:
SELECT MASTER BRANCH -- > VIEW ALL BRANCHES -- > CLICK ON PR -- > REVERT

GIT CLONE: it download code from github(Remote) to git(Local).
git clone https://github.com/anitalluri00/paytm.git
NOTE: IT WILL DOWNLOAD CODE FROM ALL BRANCHES
CHECKOUT TO ALL BRANCHES AND VERIFY

GIT FORK: it download code from github account to another.
For git clone and git fork repos must be public.

NOTE: IF WE WANT TO CLONE/FORK FOR PUBLIC REPO WE CAN DO IT DIRECTLY.
FOR PRIVTE REPOSE PREMISSION IS MUST.

PUBLIC REPO: source code will be visible from internet.
PRIVATE REPO: source code will be hidden from internet.

==========

BRANCH PROTECTION RULES:TO RESTRICT THE USER TO NOT DELETE THE BRANCH
SETTINGS -- > BRANCH -- > ADD BRANCH RULE SET -- > Enforcement status: Active
Target branches -- add target -- > include all brances --> Create

MERGE CONFLICTS:
it will rise when we merge 2 different branches with same files.
How to resolve: Manually

GIT PULL:
used to download the changed files from github to git.
git pull origin master

GIT FETCH:
used to show the changed files from github to git.
git fetch

CHERRY-PICK: Merging the specific files based on commits.
git cherry-pick commit_id
NOTE: THINK OF APPLE EXAMPLE

Process:

mkdir raham
cd raham/

git init
touch index.htmll
git add index.htmll
git commit -m "c-1" index.htmll

git checkout -b branch2
touch java{1..3}
git init
git add java*
git commit -m "java-commits" java*

touch python{1..3}
git add python*
git commit -m "python commits" python*
touch php{1..3}
git add php*
git commit -m "php commits" php*
git log --oneline
git checkout master
ll
git cherry-pick commit_id_of python
ll
git cherry-pick commit_id_of php

.gitignore: this file will ignore the files of being tracked.
if you write any filename on this .gitingore it wont track that file.
USECASE: cred

Note: . should be mandatory

GIT STATSH: use to hide the files temporarily.
Note: files need to tracked but not committed.

touch file{1..3}
git add file*
git stash -- > files will be hidden
git stash apply -- > to bring the files back
git stash list -- > to list stashes
git stash clear -- > to delete all stashes

==========
INFRASTRUCTURE:
resources used to run our application on cloud.
ex: ec2, s3, elb, vpc, Asg ----------

in general we used to deploy infra on manual

Manual:
1. time consume
2. Manual work
3. committing mistakes

Automate -- > Terraform -- > code -- > hcl (Hashicorp configuration languge)

WHAT IS TERRAFORM:
its a tool used to make infrastructure automation.
its a free and not open source.
its platform independent.
it comes on the year 2014.
who: Mitchel Hashimoto
owned: Hashicorp -- > recently IBM is maintaining.
terraform is written on the go language.
We can call terraform as IAC TOOL.
it is Cloud Agonistic (it can used to work with any cloud and even on prem)
it can manage things from onprem also.

HOW IT WORKS:
terraform uses code to automate the infra.

we use HCL : HashiCorp Configuration Language.

IAC: Infrastructure as a code.

Code --- > execute --- > Infra

ADVANTAGES:
1. Reusable
2. Time saving
3. Automation
4. Avoiding mistakes
5. Dry run (Dont Repeat Yourself)

CLOUD ALTERNATIVES:
CFT = AWS
ARM = AZURE
GDE = GOOGLE

TERRAFROM = ALL CLOUDS

SOME OTHER ALTERNATIVES:
PULUMI
OpenTofu
ANSIBLE
CHEF
PUPPET

TERRAFORM VS ANSIBLE:
Terraform will create server
and these servers will be configure by ansible.

INSTALLING TERRAFORM:

STEP-1: INSTALLING TERRAFORM

sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform
terraform --version

NOTE: TERRAFORM IS 3RD PARTY TOOL, SO WE NEED TO GIVE PERMISSION TO TERRAFORM TO WORK WITH AWS
CREATE A ROLE AND ATTACH TO EC2

STEP-2: GIVING IAM ROLE

aws configure (or) give a role
check ll .aws/ for the configuration

MAIN ITEMS IN FILE:
blocks
lables (name, type of resource)
arguments

Configuration files:
it will have resource configuration.
here we write inputs for our resource
based on that input terraform will create the real world resources.
extension is .tf

mkdir terraform
cd terraform

vim main.tf

```
provider "aws" {
region = "us-east-1"
}

resource "aws_instance" "one" {
ami = "ami-03eb6185d756497f8"
instance_type = "t2.micro"
}
```

I : INIT
P : PLAN
A : APPLY
D : DESTROY

TERRAFORM COMMANDS:
terraform init : initialize the provider plugins on backend

it will store information of plugins in .terraform folder
without plugins we cant create resources.
each provider will have its own plugins.
we can get every provider from terraform registry.
once plugins are downloaded we should not need to run init every time.

PLUGINS: ARE RESPONSIBLE TO CREATE RESOURCES IN CLOUD.

terraform plan : to create an execution plan
it will take inputs given by users and plan the resource creation
if we haven't given inputs for few fields it will take default values.

terraform apply : to create resources
as per the given inputs on configuration file it will create the resources in real word.

terrafrom destroy : to delete resources

```
provider "aws" {
region = "us-east-1"
}

resource "aws_instance" "one" {
count = 5
ami = "ami-03eb6185d756497f8"
instance_type = "t2.micro"
}
```

terraform apply --auto-approve
terraform destroy --auto-approve

STATE FILE: used to store the resource information which is created by terraform
to track the resource activities
in real time entire resource info is on state file.
we need to keep it safe & Secure
if we lost this file we cant track the infra.
Command:
terraform state list

terraform target: used to destroy the specific resource
terraform state list
single target: terraform destroy -auto-approve -target="aws_instance.one[3]"

multi targets: terraform destroy -auto-approve -target="aws_instance.one[1]"
-target="aws_instance.one[2]"

TERRAFORM FMT:
used to give allignment and indentation for terraform files.
rewrite configuration files to a canonical format and style.
terraform fmt

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

META ARGUMENTS:

PARALLELISM: by default terraform follows parallelism.
it will execute all the resources at a time.
by default parallelism limit is 10.
Terraform can create maximum 10 resources at a time.
terraform apply -auto-approve -parallelism=1
NOTE: IT WILL APPLICALBLE FOR BOTH APPLY & DESTROY.

DEPENDS_ON: one resource creation will be depending on other.
this is called explicit dependency.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  ami           = "ami-046d18c147c36bef1"
  instance_type = "t2.micro"
  tags = {
    Name = "raham-server"
  }
}

resource "aws_s3_bucket" "two" {
  bucket     = "terraformbucketabcd123"
  depends_on = [aws_instance.one]
}
```

2. COUNT: used to create similar objects.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  count         = 3
  ami           = "ami-046d18c147c36bef1"
  instance_type = "t2.micro"
  tags = {
    Name = "dev-server-${count.index+1}"
  }
}
```

NOTE: it wont create resources with different configs.

3. FOR_EACH: it is a loop used to create resources.
we can pass different configuration to same code.
it will create resource with less code.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  for_each      = toset(["dev-server", "test-server", "prod-server"])
  ami           = "ami-046d18c147c36bef1"
  instance_type = "t2.micro"
  tags = {
    Name = "${each.key}"
  }
}
```

4. LIFECYCLE:

PREVENT DESTROY: used to prevent the resource to not be deleted.
it is bool.

```
provider "aws" {
  region = "us-east-1"
}
```

```
resource "aws_instance" "one" {
  ami           = "ami-046d18c147c36bef1"
  instance_type = "t2.micro"
  tags = {
    Name = "raham-server"
  }
  lifecycle {
    prevent_destroy = true
  }
}
```

NOTE: CHECK WITH TERRAFORM APPLY

IGNORE CHANGES:
when user modify anything on current state manually changes will be ignored.

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  ami           = "ami-046d18c147c36bef1"
  instance_type = "t2.micro"
  tags = {
    Name = "raham-server"
  }
  lifecycle {
    ignore_changes = all
  }
}
```

NOTE: MODIFY NAME MANUALLY AND CHECK WITH TERRAFORM APPLY

CREATE BEFORE DESTROY:
by default when we modify some properties terraform will first destroy old server and create new server later.

if we follow create before destroy lifecycle the new resource will create first
and later old resource is going to be destroyed.

```
provider "aws" {
  region = "us-east-1"
```

```
}

resource "aws_instance" "one" {
  ami           = "ami-0208b77a23d891325"
  instance_type = "t2.micro"
  tags = {
    Name = "raham-server"
  }
  lifecycle {
    create_before_destroy = true
  }
}
```

NOTE: CHNAGE AMI ID AND GIVE APPLY

==========

TERRAFORM IMPORT
when we create resource manually terraform wont track that resource.
Import command can used to import the resource which is created manually.
can only import one resource at a time (FROM CLI)
it can import both code to config file and state file.
FOR STATE FILE: terraform import aws_instance.one (not prefarable)

FIRST CREATE A MANUAL SERVER.

cat main.tf

```
provider "aws" {
  region = "us-east-1"
}

import {
  to = aws_instance.one
  id = "i-0c23bdc1b7b73d61c"
}
```

FOR STATEFILE & CODE:
terraform plan -generate-config-out=ec2.tf
NOTE: DELETE LINE 20 AND 21 FROM ec2.tf

terraform apply -auto-approve
terraform state list

TERRAFORM TAINT
It allows for you to manually mark a resource for recreation
in real time some times resources fails to create so to recreate them we use taint
in new version we use -replace option
TO TAINT: terraform taint aws_instance.one[0]
TO UNTAINT: terraform untaint aws_instance.one[0]
terraform state list
terraform taint aws_instance.one[0]
terraform apply --auto-approve

TERRAFORM REPLACE:
terraform apply --auto-approve -replace="aws_instance.one[0]"

CODE:
statefilebucketfroterraform

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  count         = 2
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "raham"
  }
}
```

TERRAFORM VALIDATE:
used validates the configuration files in your working directory.
it will show error when we havent given the values for variables.
command: terraform validate

TERRAFORM PLAN:

used to save plan in a file for future reference.
command: terraform plan -out myplan
  to apply   : terraform apply myplan

to destroy: terraform plan -destroy

TERRAFORM REFRESH

This command will be use to refresh the state file.
terraform compares the current state to desired state, it it found any changes
on the current state it will update values to state file.
when we run plan, apply or destroy refresh will perform automatically.
if a server is manually created running terraform apply -refresh-only would detect those changes and update the state file to reflect the current state of the resource, but it won't attempt to change the infrastructure to match the Terraform configuration.
if you dont want to refresh while apply & destroy use
terraform apply/destroy -refresh=false

TERRAFORM DEBUGGING
Terraform automates the infrastructure.
issues like misconfigurations or dependency errors can occur.
Debugging helps understand issues and fix them efficiently.
You can set TF_LOG to one of the log levels TRACE, DEBUG, INFO, WARN or ERROR to change the verbosity of the logs, with TRACE being the most verbose.

```
export TF_LOG=TRACE
export TF_LOG_PATH="logs.txt"
terraform init
terraform plan
```

ALIAS & PROVIDERS:
we can map resource blocks to particular provider blocks.
used to create resources on different regions.

CODE:

```
provider "aws" {
region = "us-east-1"
}

resource "aws_instance" "three" {
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
```

```
    Name = "dev-server"
  }
}


provider "aws" {
region = "ap-south-1"
alias = "south"
}

resource "aws_instance" "abcd" {
  provider      = aws.south
  ami           = "ami-08bf489a05e916bbd"
  instance_type = "t2.micro"
  tags = {
    Name = "dev-server"
  }
}
```

==========

1. what is default backend: local
2. command to change the backend : terraform init -migrate-state
3. why to lock state file: to prevent from corruption
4. when state file will lock: when two people work on same state file parallely
5. does local backend support state lcoking: no
6. why use s3 : to lock state file automatically.

TERRAFORM STATE FILE

Terraform Stores the infrastructure infromation on state file.
it will automatically refreshed when we run plan, apply & destroy.
In Terraform, a backend is a configuration that determines how and where Terraform stores its state file and how it manages operations like apply, plan, and destroy.
By default Terraform uses local backend.
it stores state file in terraform.tfstate in local folder.
it stores information in json format.
if we delete any resource it stores infromation in terraform.tfstate.backup.

TERRAFORM STATE FILE LOCKING
in Real time once we complete our work we need to lock state file.
it ensures that only one operation can be executed at a time.
once you lock state file you cant modify the infrastructure anymore.
When two people working on state file at a time it will be locked automatically.
unfortunately if two people runs apply at same time unpredictable results, like creating duplicate resources or destroying the wrong infrastructure.
Not all Terraform backends support locking.

Terraform used local backend to manage state file.
But in that local backend only one person can able to access it.
in Real time it's often necessary to have a centralized, consistent, and secure storage mechanism for the state file.
Amazon S3 is a popular choice for this, and when combined with DynamoDB for locking, it ensures safe, consistent operations.
TERRAFORM S3 BACKEND

ADVANTAGES
Global Access
Team Collaboration
Secure and Scalable
Centralized State Management
State File Versioning
Disaster Recovery

WHY LOCKING HAPPEND

when 2 developers work on the same project with same state file then the locking will be happend.
if state file is locked only first operation execute and second operation waits.
to remove state lock use: terraform force-unlock <LOCK_ID>
after adding dynamodb run: terraform init -reconfigure

Add that block to existing code and run terraform init -upgrade
dynamodb -- > create table -- > Partition key: LockID -- > create
now after apply state file will go to s3 bucket
dev-1 type destroy and dev-2 type apply now state file locked.
you can check lock-id in new items of table.
once destroy done for dev-1 state file will be unlocked and dev-2 can work.

CODE:

```
provider "aws" {
  region = "us-east-1"
}

terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
    use_lockfile = true
  }
}

resource "aws_instance" "one" {
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "dev-server"
  }
}
```

MIGRATING FROM S3 TO LOCAL BACKEND

if we want state file to back on local use below method.
remove backend code from main.tf
run terraform init -migrate-state

CODE:

```
provider "aws" {
  region = "us-east-1"
}


resource "aws_instance" "one" {
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "dev-server"
  }
```

}

TERRAFORM REFRESH

This command will be use to refresh the state file.
terraform compares the current state to desired state, it it found any changes
on the current state it will update values to state file.
when we run plan, apply or destroy refresh will perform automatically.
if a server is manually created running terraform apply -refresh-only would detect those changes and update the state file to reflect the current state of the resource, but it won't attempt to change the infrastructure to match the Terraform configuration.
if you dont want to refresh while apply & destroy use
terraform apply/destroy -refresh=false

TERRAFORM BACKEND BLOCK

By default there is no backend configuration block within Terraform configuration Because Terraform will use it's default backend - local
This is why we see the terraform.tfstate file in our working directory.
FOR PARTIAL BACKEND:
path = "state_data/terraform.dev.tfstate"
FROM CLI:
terraform init -backend-config="path=state_data/terraform.prod.tfstate" -migrate-state
If want we can specify multiple partial backends too.

CODE:

```
provider "aws" {
  region = "us-east-1"
}

terraform {
  backend "local" {
    path = "/tmp/abc.tfstate"
  }
}


resource "aws_instance" "one" {
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
```

```
  tags = {
    Name = "dev-server"
  }
}
```

TERRAFORM OUTPUTS:
Whenever we create a resource by Terraform if you want to print any output of that resource we can use the output block this block will print the specific output as per our requirement.

```
provider "aws" {
}

resource "aws_instance" "one" {
ami = "ami-00b8917ae86a424c9"
instance_type = "t2.micro"
tags = {
Name = "raham-server"
}
}

output "raham" {
value = [aws_instance.one.public_ip, aws_instance.one.private_ip,
aws_instance.one.public_dns]
}
```

TO GET COMPLTE OUTPUS:

```
output "raham" {
value = aws_instance.one
}
```

Note: when we change output block terraform will execute only that block
remianing blocks will not executed because there are no changes in those blocks.

==========

PROVISOIONERS: to execute commands or scripts on servers.

LOCAL EXEC:
executes command/script on local machine (where terraform is installed)
it will execute the command when resource is created.

CODE:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "three" {
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "abc-server"
  }
  provisioner "local-exec" {
    command = "echo this is my local server"
  }
}
```

REMOTE EXEC:

executes command/script on remote machine.
once the server got created it will execute the commands and scripts for
installing the softwares and configuring them and deploying app also.

CODE:

```
provider "aws" {
}

resource "aws_instance" "one" {
  ami                    = "ami-000ec6c25978d5999"
  instance_type          = "t2.micro"
  key_name               = "Rkeypair"
  vpc_security_group_ids = ["sg-0329cd46bd67094a1"]
  tags = {
    Name = " rahaminstance"
  }
```

```
  provisioner "remote-exec" {
    inline = [
      "sudo yum install httpd git -y",
      "sudo systemctl start httpd",
      "sudo cd /var/www/html",
      "sudo git clone https://github.com/karishma1521success/swiggy-clone.git",
      "sudo mv swiggy-clone/* .",
      "sudo mv /home/ec2-user/* /var/www/html"
    ]

    connection {
      type        = "ssh"
      user        = "ec2-user"
      private_key = file("~/.ssh/id_rsa")
      host        = self.public_ip
    }
  }
}
```

FILE PROVISONER: COMMANDS WE CAN EXECUTE THROUGH FILE
NOTE: CREATE A SCRIPT ON YOUR LOCAL FOLDER

```
provider "aws" {
}

resource "aws_instance" "one" {
  ami                    = "ami-01816d07b1128cd2d"
  instance_type          = "t2.micro"
  key_name               = "swikp"
  vpc_security_group_ids = ["sg-0c656c667bc0861e0"]
  user_data              = "${file("apache.sh")}"

  tags = {
    Name = " rahaminstance"
  }
}
```

apache.sh

```
#! /bin/bash
sudo -i
```

```
sudo yum install httpd git -y
sudo systemctl start httpd
sudo systemctl status httpd
sudo git clone https://github.com/Ironhack-Archive/online-clone-amazon.git
sudo mv online-clone-amazon/* /var/www/html
```

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_instance" "one" {
  count                  = 3
  ami                    = "ami-0144277607031eca2"
  instance_type          = "t2.micro"
  tags = {
    Name = "dev-server-${count.index+1}"
  }
}

output "server" {
value = aws_instance.one[*].public_ip
}
```

==========

PROVIDER BLOCK:

By default provider plugins in terraform change version for every few weeks.
when we run init command, it download latest plugins always.
some code will not work with old plugins, so we need to update them.
To get latest provider plugins : https://registry.terraform.io/browse/providers.
when you add a new provider terraform init is must.
terraform providers: to list the providers which required to run code.
to create infra on any cloud all we need to have is provider.

TYPES:
1. OFFICIAL : MANAGED BY TERRAFORM
2. PARTNER : MANAGE BY 3RD PATRY COMPANY
3. COMMUNITY: MANAGED BY INDIVIDUALS

CODE:

```
provider "aws" {
  region = "us-east-1"
}


terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">5.70.0"
    }
  }
}
```

MULTI PROVIDERS:

```
terraform {
  required_version = ">=1.9.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">5.70.0"
    }
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "4.9.0"
    }
    google = {
      source  = "hashicorp/google"
      version = "6.10.0"
    }
  }
}
```

TERRAFORM BLOCK:

This block is used to set global configurations and settings for the Terraform.
It usually includes details such as required providers, backend configuration, and version constraints.

terraform -v
terraform -version
terraform --version

TERRAFORM LOCAL BLOCK:

A local block is used to define values.
if a value is repeating multiple times we can define it here.
This makes our code cleaner and easier to understand.
simply define value once and use for mutiple times.

CODE:

```
provider "aws" {
  region = "us-east-1"
}

locals {
env = "test"
}

resource "aws_vpc" "one" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "${local.env}-vpc"
  }
}

resource "aws_subnet" "two" {
  vpc_id     = aws_vpc.one.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "${local.env}-subent"
  }
}

resource "aws_instance" "three" {
  subnet_id     = aws_subnet.two.id
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "${local.env}--server"
```

```
  }
}
```

TERRAFORM COMMENTS:
We use comments to make others code to understand easily.
Terraform supports three different syntaxes for comments.
-- > single line comment
// -- > single line comment
/* */ -- > multi line comment
Note: if we put comments for code, terraform thinks code is not existed and it will destroy the resource.

TLS PROVIDER:

it provides utilities for working with Transport Layer Security keys & certificates.
It provides resources that allow private keys, certificates & CSR.
Add tls on your own and try this below code.

CODE:

```
provider "aws" {
  region = "us-east-1"
}

resource "tls_private_key" "rsa-4096-example" {
  algorithm = "RSA"
  rsa_bits  = 4096
}


resource "local_file" "private_key_pem" {
  content  = tls_private_key.rsa-4096-example.private_key_pem
  filename = "devops.pem"
}
```

==========
TERRAFORM VARIABLES

Terraform variables are values which we can change without editing the whole Terraform code every time.

It makes your code reusable (same code, different ENV— dev, test, prod)
It makesyour code clean (no hardcoded values)
It makes it flexible (you can pass values at runtime)

TERRAFORM VARIABLES TYPE:
Input Variables : Values Given By the user
Output Variables : Values Given By the Terraform
Local Variables : Temporary variables within Terraform

DATA TYPES:

Type	Description	Example
string	Text value	"""dev"", ""us-east-1"""
number	Numeric value	"2, 300"
bool	True or False	"true, false"
list	Ordered collection of values	"[""t2.micro"", ""t3.micro""]"
map	Key-value pairs	"{ env = ""dev"", tier = ""1"" }"
object	Group of named attributes	"{ name = ""app"", size = 2 }"
tuple	Ordered list of diff data types	"[""app"", 2, true]"

STRING & NUMBER:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "raham" {
  count         = var.instance_count
  ami           = "ami-04aa00acb1165b32a"
  instance_type = var.instance_type
  tags = {
    Name = "Raham"
  }
}

variable "instance_count" {
default = 3
}

variable "instance_type" {
default = "t2.micro"
```

```
}
```

LIST:

Main.tf

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "one" {
  count         = length(var.instance_type)
  ami           = "ami-0de716d6197524dd9" # Example AMI ID, replace with a valid one
  instance_type = var.instance_type[count.index]
  tags = {
    Name = var.instance_name[count.index]
  }
}
```

Varibale.tf

```
variable "instance_type" {
  type    = list(string)
  default = ["t2.micro", "t2.small", "t2.medium", "t2.large"]
}

variable "instance_name" {
  type    = list(string)
  default = ["dev-server", "test-server", "uat-server", "prod-server"]

}
```

MAP:

Main.tf

```
provider "aws" {
  region = "us-east-1"
```

```
}

resource "aws_instance" "one" {
  ami           = "ami-0de716d6197524dd9" # Example AMI ID, replace with a valid one
  instance_type = "t2.micro"              # Default instance type, can be overridden
  tags          = var.tags
}
```

Varibale.tf

```
variable "tags" {
  type = map(string)
  default = {
    Name        = "MyInstance"  # Default tag value, can be overridden
    Environment = "Development" # Example additional tag
    Client      = "TCS"
  }
}
```

TERRAFORM TFVARS:

When we have multiple configurations for terraform to create resource
we use tfvars to store different configurations.
Pass the tfvars While apply Command to take the values of that file.
NOTE: BY DEFAULT TERRAFORM WILL PICK VALUES FROM TERRAFORM.TFVARS
if you don't pass any tfvars files terraform will pick values from terraform.tfvars.
ALTERNATE NAME: terraform.tfvars.json

ALTERNATE NAME:
terraform.tfvars.json

cat main.tf.

```
provider "aws" {
region = "us-east-1"
}

resource "aws_instance" "one" {
count = var.instance_count
ami = "ami-0e001c9271cf7f3b9"
instance_type = var.instance_type
tags = {
Name = var.instance_name
```

```
}
}

cat variable.tf
variable "instance_count" {
}

variable "instance_type" {
}

variable "instance_name" {
}

cat dev.tfvars
instance_count = 1

instance_type = "t2.micro"

instance_name = "dev-server"

cat test.tfvars
instance_count = 2

instance_type = "t2.medium"

instance_name = "test-server"

cat variable.tfvars
instance_count = 3

instance_type = "t2.large"

instance_name = "prod-server"

terraform apply -auto-approve -var-file="dev.tfvars"
terraform apply -auto-approve -var-file="test.tfvars"
terraform apply -auto-approve -var-file="prod.tfvars"


==========================================================================
==========================================================
```

WORKSPACES:
Terraform workspaces allow you to maintain multiple environments
(e.g., development, testing, production)
using the same Terraform configuration but with different states.
Each workspace has its own state file (terraform.tfstate.d folder)
resources created/managed in one workspace do not affect other workspaces.
the default workspace in Terraform is default
Each workspace is isloated (seperated with each other)

terraform workspace list : to list the workspaces
terraform workspace new dev : to create workspace
terraform workspace show : to show current workspace
terraform workspace select dev : to switch to dev workspace
terraform workspace delete dev : to delete dev workspace

NOTE:
1. we need to empty the workspace before delete
2. we cant delete current workspace, we can switch and delete
3. we cant delete default workspace

EXECUTION:
NOTE: TAKE CODES FROM TFVARS CONCEPT.

terraform workspace new dev
terraform apply -auto-approve -var-file="dev.tfvars"

terraform workspace new test
terraform apply -auto-approve -var-file="test.tfvars"

terraform workspace new prod
terraform apply -auto-approve -var-file="prod.tfvars"

FOR DELETION:

terraform destroy -auto-approve -var-file="prod.tfvars"
terraform workspace select test
terraform workspace delete prod

terraform destroy -auto-approve -var-file="test.tfvars"
terraform workspace select dev
terraform workspace delete test

terraform destroy -auto-approve -var-file="dev.tfvars"
terraform workspace select default
terraform workspace delete dev

TERRAFORM CLI:

cat main.tf
provider "aws" {
}

resource "aws_instance" "one" {
count = var.var.instance_name
ami = "ami-00b8917ae86a424c9"
instance_type = var.instance_type
tags = {
Name = var.instance_name
}
}

cat variable.tf
variable "instance_count" {
}

variable "instance_type" {
}

variable "instance_name" {
}

METHOD-1:
terraform apply --auto-approve
terraform destroy --auto-approve

METHOD-2:
terraform apply --auto-approve -var="instance_type=t2.micro"
terraform destroy --auto-approve -var="instance_type=t2.micro"

NOTE: If you want to pass single variable from cli you can use -var or if you want to pass multiple variables from cli create terraform .tfvars files and use -var-file.

TERRAFORM ENV VARS:

A Terraform environment variable is a key-value pair set within the environment where Terraform is running.
Used to specify the location of the Terraform configuration files, set provider credentials, or define backend configurations.

LINUX & MAC:
```
export TF_VAR_region=us-east-1
export TF_VAR_instance_type='{ "t2.micro", "t2.medium" }'
export TF_VAR_tagmap='{ Environment = "dev", Project = "demo" }'
```

WINDOWS:
```
$env:TF_VAR_instance_type = "t2.micro"
$env:TF_VAR_instance_count = 1
$env:TF_VAR_instance_name = "abcd"
```

DYNAMIC BLOCK:
it is used to reduce the length of code and used for reusabilty of code in loop.

```
provider "aws" {
  region = "us-east-1"
}

locals {
  ingress_rules = [{ port = 443 }, { port = 80 }, { port = 22 }]
}


resource "aws_security_group" "allow_tls" {
  name        = "terra sg"
  description = "Allow TLS inbound traffic"

  dynamic "ingress" {
    for_each = local.ingress_rules
    content {
      description = "*"
      from_port   = ingress.value.port
      to_port     = ingress.value.port
```

```
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
   }
  }

  tags = {
   Name = "terra sg"
  }
}
```

==========

MODULES:
it divides the code into folder structure.
Modules are group of multiple resources that are used together.
This makes your code easier to read and reusable across your organization.
we can publish modules for others to use.
each module will be having sperate plugins.
modules plugins will be store on .terraform/modules/

TYPES:
Root Module: This is the main directory where Terraform commands are run.
All Terraform configurations belong to the root module.

Child Modules: These modules are called by other modules.

yum install tree -y

```
.
├── main.tf
├── modules
│   ├── my_instance
│   │   └── main.tf
│   ├── s3_module
│   │   └── main.tf
│   └── vpc_module
│       └── main.tf
```

CODE:

cat main.tf

```
provider "aws" {
  region = "us-east-1"
}


module "vpc" {
  source = "./modules/vpc_module"
}

module "ec2" {
  source = "./modules/my_instance"
}

module "s3" {
  source = "./modules/s3_module"
}


mkdir -p modules/my_instance/
vim modules/my_instance/main.tf
resource "aws_instance" "one" {
  ami           = "ami-0ddc798b3f1a5117e"
  instance_type = "t2.micro"
  tags = {
    Name = "module-server"
  }
}


modules/s3_module/
vim modules/s3_module/main.tf
resource "aws_s3_bucket" "example" {
  bucket = "rahamdemo-tf-test-bucket"
}

cat modules/vpc_module/main.tf
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "module-vpc"
  }
}
```

MODULE INPUT & OUTPUTS:
We can add Input and Output Blocks for Terraform Modules
Root module can refer both variables & values of child modules.
Child modules cant refer variables, but it can refer its values.

```
├── main.tf
├── modules
│   └── myec2
│       ├── main.tf
│       ├── output.tf
│       └── variable.tf
```

```
cat main.tf
provider "aws" {
region = "us-east-1"
}

module "server" {
source = "./modules/myec2"
}

output "module_output" {
value = module.server.public_ip
}
```

```
cat modules/myec2/main.tf
resource "aws_instance" "one" {
  ami           = var.ami
  instance_type = var.instance_type
  tags = {
    Name = var.instance_name
  }
}

cat modules/myec2/variable.tf
variable "ami" {
  default = "ami-0ddc798b3f1a5117e"
}
```

```
variable "instance_type" {
  default = "t2.micro"
}

variable "instance_name" {
  default = "module-server"
}

cat modules/myec2/output.tf
output "public_ip" {
  value = aws_instance.one.public_ip
}
```

PUBLIC MODULES:

The module must be on GitHub and must be a public repo.
NAMING FORMAT: terraform-<provider>-<name> (terraform-aws-ec2-instance)
must have a description.
 module structure will be main.tf, variables.tf, outputs.tf.
x.y.z tags for releases.

TERRAFORM GRAPH:
it shows the relationships between objects in a Terraform configuration.
using the DOT language.
once create infra run the command: terraform graph
Go to Google -- > Graphwiz online & copy paste the code

CODE:

```
provider "aws" {
region = "us-east-1"
}

resource "aws_vpc" "one" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "dev-vpc"
  }
}

resource "aws_subnet" "two" {
```

```
  vpc_id     = aws_vpc.one.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "dev-subent"
  }
}

resource "aws_instance" "three" {
  subnet_id     = aws_subnet.two.id
  ami           = "ami-0866a3c8686eaeeba"
  instance_type = "t2.micro"
  tags = {
    Name = "dev-server"
  }
}
```

==========

HCP CLOUD:

HCP means HashiCorp Cloud Platform
it is a managed platform to automate cloud infrastructure.
it provide privacy, security and isloation.
it supports multiple providers like AWS, Azure, and Google Cloud.
it offers a suite of open-source tools for managing infrastructure, including Terraform, Vault, Consul, and Nomad.
We can use Different Code Repos for a Project.
We can use Variable sets to apply same variables to all the workspaces.

ACCOUNT CREATION:

ACCONT-1: GITHUB -- > TO STORE CODE
CREATE A GITHUB ACCOUNT (https://github.com/ -- > singup -- > username,password,email)
create repo -- > name -- > create -- > add new file -- > write terraform code -- > commit

ACCONT-2: HCP

Go to google & type : HCP CLOUD ACCOUNT SIGNIN

COTINUE WITH GITHUB
AUHORIZE
CLICK CHECK BOXES -- > CONTINUE

CREATE AN ORGINIZATION
CLICK ON TERRAFORM
CONTINUE TO HCP
CONTINUE WITH HCP ACCOUNT

CREATE ORG : NAME: SWIGGY

CREATE YOUR WORKSPACE
Version Control Workflow
INTEGRATE YOUR VCS -- > GITHUB -- > SELECET REPO -- > NEXT -- > CONTINUE TO WORKSPACE
CONFIGURE VARAIBLES -- > ADD VARIABLE

AWS_ACCESS_KEY_ID : VALUE: Environment variable -- > SENSITIVE -- > SAVE
AWS_SECRET_ACCESS_KEY: Environment variable -- > SENSITIVE -- > SAVE
NOTE: MARK THEM AS ENV VARIBALE AND MAKE SURE NO SPACES ARE GIVEN

RUNS -- > NEW RUN -- > START -- > confirm and apply
IT WILL AUTOMATICALLY PLAN & WE NEED TO APPLY BY MANUAL
SECOND TIME WHEN WE CHANGE CODE IT WILL AUTOMATICALLY PLAN
PLAN & APPLY
DESTROY

Cost Estimation policy:
Cost Estimation policy: we can estimate the cost of infra before we create it.
by default this feature is disable, we need to enable
click on terraform logo -- > orginization -- > settings -- > Cost Estimation -- > enable

SENTINEL POLICY: ITS A POLICY AS A CODE.
BY THIS SENTINEL POLICY WE CAN WRITE OUR OWN CONDITIONS.
IT WILL CHECK THE CONDITION OF RESOURCE BEFORE IT CREATED.
IF CONDITION IS SATISFIED IT WILL CREATE RESOURE, OTHERWISE IT WONT.
EX: TAGS, VERIFIED AMIS, SG --

TERRAFORM LOGO -- > ORG -- > SETTINGS -- > POLICY

TERRAFORM CLOUD FEATURES:
1. Workspaces
2. Projects
3. Runs
4. Variables and Variable Sets
5. Policies and Policy Sets
6. Run Tasks
7. Single Sign-On (SSO)
8. Remote State
9. Private Registry
10. Agents
11. Role-Based Access Control
12. Version Control Integration
13. Observability

TERRAFORM CLOUD ENTERPISE FEATURES:
Private Module Registry: Includes a private registry for sharing modules securely across teams.
Policy as Code: Integrates Sentinel for enforcing policies during provisioning.
Enhanced Automation: Offers advanced run triggers, notifications, and support for custom workflows.

Multi-Organization Support: Allows multiple teams or departments to manage their own infrastructure within a single account.
Advanced Collaboration: Provides role-based access controls and team management features, allowing for fine-grained permissions.
Enhanced Security: Features like SSO (Single Sign-On), audit logs, and compliance tools.

TO DELETE RESOURCES:
SETTINGS -- > DESTRUCTION AND DELETION -- > DELETE QUEUE

Cost Estimation policy:
Cost Estimation policy: we can estimate the cost of infra before we create it.
by default this feature is disable, we need to enable
click on terraform logo -- > orginization -- > settings -- > Cost Estimation -- > enable

SENTINEL POLICY: ITS A POLICY AS A CODE.
BY THIS SENTINEL POLICY WE CAN WRITE OUR OWN CONDITIONS.
IT WILL CHECK THE CONDITION OF RESOURCE BEFORE IT CREATED.
IF CONDITION IS SATISFIED IT WILL CREATE RESOURE, OTHERWISE IT WONT.
EX: TAGS, VERIFIED AMIS, SG --
click on terraform logo -- > orginization -- > settings -- > Policy -- > name: Hard

POLICY: NOE: TAKE POLICY FORM OFFICAL DOCS OR CHATGPT

```
# Enforce that all resources must have tag "Env" set to "Prod"

import "tfplan/v2" as tfplan

# Function to check tags
validate_tags = func(resource) {
    # Some resources may not support tags
    if not ("tags" in resource.applied) {
        return true
    }

    # Check if "Env" tag exists and equals "Prod"
    return resource.applied.tags["Env"] is "Prod"
}

# Collect all resources from the plan
resources = filter tfplan.resource_changes as _, rc {
    rc.mode is "managed" and rc.type is not null
}

# Run validation
all_tags_valid = all resources as _, rc {
    validate_tags(rc)
}

# Main Rule
main = rule {
    all_tags_valid
}
```

POLICTY SET -- > CONNECT TO POLICY SET -- > NAME -- > SELECT POLICY -- > SAVE

TERRAFORM LOGO -- > ORG -- > SETTINGS -- > POLICY
TERRAFORM CLOUD FEATURES:
1. Workspaces
2. Projects
3. Runs

4. Variables and Variable Sets
5. Policies and Policy Sets
6. Run Tasks
7. Single Sign-On (SSO)
8. Remote State
9. Private Registry
10. Agents
11. Role-Based Access Control
12. Version Control Integration
13. Observability

TERRAFORM CLOUD ENTERPISE FEATURES:
Private Module Registry: Includes a private registry for sharing modules securely across teams.
Policy as Code: Integrates Sentinel for enforcing policies during provisioning.
Enhanced Automation: Offers advanced run triggers, notifications, and support for custom workflows.

Multi-Organization Support: Allows multiple teams or departments to manage their own infrastructure within a single account.
Advanced Collaboration: Provides role-based access controls and team management features, allowing for fine-grained permissions.
Enhanced Security: Features like SSO (Single Sign-On), audit logs, and compliance tools.

TO DELETE RESOURCES:
SETTINGS -- > DESTRUCTION AND DELETION -- > DELETE QUEUE

OPA is a more flexible, open-source policy engine that works across a much broader cloud-native stack – Terraform, Kubernetes, APIs, and more. Sentinel, on the other hand, is developed by HashiCorp and is tightly coupled with Terraform and the HashiCorp ecosystem.

REMOTE ENHANCED BACKEND:
CREATE THIS FILE ON LOCAL:

```
provider "aws" {
region = "us-east-1"
}

terraform {
  backend "remote" {
    hostname = "app.terraform.io"
    organization = "my-terraform-batch-001"
```

```
    workspaces {
      name = "muthurepo"
    }
  }
}

resource "aws_instance" "three" {
  count         = 1
  ami           = "ami-00ca32bbc84273381"
  instance_type = "t2.micro"

  tags = {
    Name = "dev-server"
  }
}
```

COMMANDS:

terraform login
add token
terraform init
terraform plan
==========
========

OPENTOFU:
its a Forked version of terraform.
it comes into market on 2023.

TERRAFORM is FREE & NOT-OPEN SOURCE & OPENTOFU is FREE & OPEN SOURCE.
TERRAFORM IS MAINTAINED BY IBM BUT OPEN TOFU MAINTED BY COMMUNITY.
TERRAFORM HAS LOT OF PROVIDERS & PLUGINS, BUT OPEN TOFU HAVING LIMTED.
TERRAFORM HAS MORE FEATURES, OPENTOFU HAS LESS FEATURES.
TERRAFORM HAS HCP, BUT OPENTOFU DONT HAVE ANY CLOUD.

wget
https://github.com/opentofu/opentofu/releases/download/v1.9.0-rc2/tofu_1.9.0-rc2_linux_amd64.zip
unzip tofu_1.9.0-rc2_linux_amd64.zip
mv tofu /usr/local/bin/tofu

tofu version

AFTER INSTALLNG OPENTOFU RUN THE FOLLOWING COMMAND:

vim /root/.bashrc
export PATH=$PATH:/usr/local/bin/
:wq
source /root/.bashrc

```
resource "aws_instance" "one" {
  ami           = "ami-0208b77a23d891325"
  instance_type = "t2.micro"
  tags = {
    Name = "raham-server"
  }
}
```

tofu init
tofu plan
tofu apply
tofu destroy

TERRAFORMER:

STEP-1: INSTALL TERRAFORM

sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform
aws configure

STEP-2: INSTALLING TERRAFORMER

wget
https://github.com/GoogleCloudPlatform/terraformer/releases/download/0.8.24/terraformer-all-lin
ux-amd64
chmod +x terraformer-all-linux-amd64
sudo mv terraformer-all-linux-amd64 /usr/local/bin/terraformer
echo "export PATH=$PATH:/usr/local/bin/" >> .bashrc

source .bashrc

terraformer --version

STEP-3: IMPORTING EXISTING INFRA TO TERRAFORM

vi main.tf
provider "aws" {
region = "us-east-1"
}

terraform init

COMMAND: terraformer import aws --resources=sg,ec2_instance,elb --regions=us-east-1
UPDATE PROVIDERS IN STATE FILE:
terraform state replace-provider -- -/aws hashicorp/aws
terraform init
terraform plan
terraform apply

TERRASCAN
Terrascan is a open-source static code analysis tool.
It is designed for scanning infrastructure as code (IaC) templates and configurations.
It helps identify security vulnerabilities and compliance violations
it provides best practices, security standards, and compliance requirements.
It can also be used in pipelines with CI/CD systems

ADVANTAGES
Terrascan can be Integrated into CI/CD pipelines, IDEs and deployment workflows.
Terrascan provides a library of predefined security and compliance rules.
It can be set up to regularly scan and check for any drifts.
Generates detailed reports that highlight issues found during the analysis.
Supports mutiple cloud providers (AWS, Azure, GCP) cross-cloud infrastructure.
Tools K8S, ArgoCD, Atlantis, GitHub, and Docker, making it versatile for
ADVANTAGES

commands
curl -L "$(curl -s https://api.github.com/repos/tenable/terrascan/releases/latest | grep -o -E "https://.+?_Linux_x86_64.tar.gz")" > terrascan.tar.gz

tar -xf terrascan.tar.gz terrascan && rm terrascan.tar.gz
install terrascan /usr/local/bin && rm terrascan
sudo install terrascan /usr/local/bin
installation
terrascan init
terrascan scan
commands

NORMAL CODE

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t3.micro"
  }
}
```

ENHANCED CODE

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "example" {
  ami                    = "ami-0c55b159cbfafe1f0"
  instance_type = "t3.micro"
  monitoring       = true
  metadata_options {
    http_endpoint = "disabled"
  }

  tags = {
    Name = "IMDS-Disabled-Monitoring-Off"
  }
}
```

==========

==========
STEP-1: UNDERSTAND THE ARCHITECTURE
STEP-2: WRITE THE CODE (in Real time code will be there)
STEP-3: CONFIGURE THE VALUES AS PER REQUIRMETS
STEP-4: USE HCP CLOUD TO DEPLOY INFRA

CODE : https://github.com/devopsbyraham/new-infra-3-tier.git
NOTE : PLESE FORK TO YOUR ACCOUNT (FORK=DOWLOAD IN GITHUB)
REPLACE MY NAME devopsbyraham WITH YOUR GITHUB NAME
==========
==========

raw mutton -- > clean --> marnet -- > ingredients -- > Mutton Biryani
raw code -- > build -- > test -- > artifact -- > Deployment

ARTIFACT: its final product of our code.
developers will give raw code that code we are going to convert into artifact.

TYPES:
1. jar : Java Archive : Backend code
2. war : Web Archive : Frontend code + Backend code
3. ear : Enterprise Archive : jar + war

JAR FILE:

.java -- > compile -- > .class -- > .jar file

.java : basic raw
.class : executable file
.jar : artifact

all the artifacts are going to created by a build too1.

MAVEN:
Maven is the build tool.
build: process of adding the libs & dependencies to code.
its a free and opensource.
its a platform independent tool.
its is also called as Project management too1.
it will manage the complete structure of the project.
the main file in the maven tool is POM.XML

POM.XML: its a file which consist of complete project information.
Ex: name, artifact, tools, libs, dep --------

POM: PROJECT OBJECT MODEL
XML: EXTENSIBLE MARKUP LANGUAGE

WHO GIVE POM.XML : DEVELOPERS
dev will give both code and pom.xml in github

maven is written on java by apache software foundation.
supports: JAVA-1.8.0
year: 2004
home path: .m2

PRATCICAL PART:
1. CREATE AN UBUNTU EC2 INSTANCE AND CONNECT

2. apt update
sudo apt install openjdk-8-jdk maven tree -y
mvn -v
update-alternatives --config java (when u dont see java 1.8.0 for maven use the command)

3. git clone https://github.com/devopsbyraham/jenkins-java-project.git
4. cd jenkins-java-project

[ERROR] Source option 5 is no longer supported. Use 7 or later.
[ERROR] Target option 5 is no longer supported. Use 7 or later.
SOL:
yum remove java* maven* -y

ERROR: [ERROR] The goal you specified requires a project to execute but there is no POM in this directory (/root/jenkins-java-project).
SOL: CHECK POM.XML IS THERE OR NOT

MAVEN LIFECYCLE:
GOALS : a command used to run a task.
Goals refers pom.xml to execute.

NOTE: Without Pom.xml Goals will Not Execute.

PLUGIN: its a small software with makes our work automated.

instead of downloading tools we can download plugins.
this plugins will download automatically when we run goals.

1. mvn compile : used to compile the code (.java [src] -- > .class [target])
2. mvn test : used to test the code (.java [test] -- > .class [target])
3. mvn package : used to create artifact
4. mvn install : used to copy artifact to .m2 (project folder -- > .m2)
5. mvn clean : to delete the target folder

6. mvn clean package: compile -- > package

mvn clean : remove old war files
mvn package : compile -- > test -- > artifact

PROBLEMS WITHOUT MAVEN:
1. we cant create artifacts.
2. We cant create project structure. [mvn archetype:generate]
3. we cant build and deploy the apps.

ALTERNATIVIES:
MAVEN, ANT, GRADLE, NPM

PROGRAMMING VS BUILD:

JAVA : MAVEN
PYTHON : GRADLE
.NET : VS CODE
C, C# : MAKE FILE
node.js: npm

ALTERNETIAVES:
ANT, GRADLE for java projects.

MAVEN VS ANT:
1. MAVEN IS BUILD & PROJECT MANAGEMNT, ANT IS ONLY BUILD TOOL
2. MAVEN HAS POM.XML, ANT HAS BUILD.XML
3. MAVEN HAS A LIFECYCLE, ANT WILL NOT HAVE LIFECYCLE
4. MAVEN PLUGINS ARE REUSABLE, ANT SCRIPTS ARE NOT RESUEABLE.
5. MAVEN IS DECLARATIVE, ANT IS PROCEDURAL.

SYNOPSIS:

maven is a build tool -- > artifacts -- > java & python -- > mvn clean package
remove old artifact and compile, test and create artifact to deploy the app
==========

JENINS IS A CI/CD TOOL.
REALITY: JENKINS IS ONLY FOR CI.

CI : CONTINOUS INTEGRATION : CONTINOUS BUILD + CONTINOUS TEST (OLD CODE WITH NEW CODE)

DAY-1: 100 LINES : BUILD + TEST
DAY-2: 200 LINES : BUILD + TEST
DAY-3: 300 LINES : BUILD + TEST

BEFORE CI:
MANUAL PROCESS
TIME WASTE

AFTER CI:
AUTOMATED PROCESS
TIME SAVING

CD: CONTINOUS DELIVERY/DEPLOYMENT

ENV:

PRE-PROD/NON-PROD:
DEV : developers
QA : testers
UAT : clients

LIVE/PROD ENV:
PROD : users

CONTINOUS DELIVERY: Deploying the application to production in manual.
CONTINOUS DEPLOYMENT: Deploying the application to production in automatic.

PIPELINE:

WAKEUP -- > DAILY ACTIVITIES -- > BREAKFAST -- > LUNCH -- > CLASS
CODE -- > BUILD -- > TEST -- > ARTIFACT -- > DEPLOY

SETP BY STEP EXECUTION OF A PROCESS.
SERIES OF EVENTS INTERLINKED WITH EACHOTHER.

JENKINS:
ITS A FREE AND OPEN-SOURCE TOOL.
IT IS PLATFORM INDEPENDENT.
IT CAN AUTOMATE ENTIRE SDLC.

JENKINS WRITTEN ON JAVA.
IT CONSIST OF PLUGINS.
WE HAVE COMMUNITY SUPPORT.
IT IS OWNED BY SUN MICRO SYSTEM AS HUDSON.
HUDSON IS PAID VERSION.
LATER ORACLE BROUGHT HUDSON AND MAKE IT FREE.
LATER HUDSON WAS RENAMED AS JENKINS.
INVENTOR: Kohsuke Kawaguchi
PORT NUMBER: 8080
JAVA: JAVA-17
DEFAULT PATH: /var/lib/jenkins

ALTERNATIVES:
GITLAB, BAMBOO, GO CI, CIRCLE CI, TARVIS, SEMAPHORE, BUDDY BUILD MASTER, HARNESS, ARGOCD -----

CLOUD: AWS CODEPIPELINE, AZURE PIPLEINE ----------

SETUP: Create an EC2 with UBUNTU 24.04 AMI and Include all traffic/8080 in sg

```
#! /bin/bash
sudo apt update -y
sudo apt install -y openjdk-17-jdk
java -version
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update -y
sudo apt install -y jenkins
sudo systemctl start jenkins
```

sudo systemctl status jenkins

54.172.145.179

CONNECT:
copy-public-ip:8080 (browser)
cat /var/lib/jenkins/secrets/initialAdminPassword (server)
paster password on browser -- > installing plugins --- > user details -- > save --> save -- > start using jenkins

MANAGE JENKINS (SETTINGS SYMBOL) -- > TOOLS -- > ADD MAVEN -- > name: maven -- > SAVE

JOB: it is used to perform task.
to do any work or task in jenkins we need to create a job.

to run commands we need to select execute shell on build steps.

build now: to run job
workspace: place where our job outputs will store

CREATING A JOB:
NEW ITEM -- > NAME: ci-job -- > FREESTYLE -- > OK -- > SCM -- > GIT -- > REPOURL: https://github.com/devopsbyraham/jenkins-java-project.git -- >Build Steps -- > ADD Build Steps -- > Invoke top-level Maven targets -- > Maven Version: maven --> clean package -- > save -- > build now

WORKSPACE: where your job output is going to be stored
Default: /var/lib/jenkins/workspace

CUSTOM WORKSPACE:
GO TO JOB -- > CONFIGURE -- > Advance Options -- > Use custom workspace -- > /root/raham

cd
mkdir raham
chown jenkins:jenkins /root
chown jenkins:jenkins /root/raham

Now we can see output on root folder after building.

CUSTOM PORT NUMBER:
IF A TOOL IS HAVING A PORT NUMBER WE NEED TO CALL IT AS SERVICE
FOR EVERY SERVICE WE HAVE A CONFIGURATION FILE.
TO CHANGE ANY PROPERTY OF A SERVICE WE NEED TO MODIFY THE CONFIG FILE.

find / -name jenkins.service

vim /usr/lib/systemd/system/jenkins.service
LINE 72: 8080=8090
systemctl daemon-reload
systemctl restart jenkins

==========

INSTALLING MAVEN ON JENKINS
Dashboard
Manage Jenkins
Tools
Maven installations
name: maven
save

CRON JOB: We can schedule the jobs that need to be run at particular intervals.
here we use cron syntax
cron syntax has * * * * *
each * is separated by space

* : minutes
* : hours
* : date
* : month
* : day of week (sun=0, mon=1 ----)

6:32 PM 15 JULY 2025

32 18 15 7 2

create a ci job -- > Build Triggers -- > Build periodically -- > * * * * * -- > save

CRONTAB-GENERATOR: https://crontab-generator.org/

limitation: it will not check the code is changed or not.

POLL SCM:
in pollscm we will set time limit for the jobs.
if dev commit the code it will wait until the time is done.
in given time if we have any changes on code it will generate a build

create a ci job -- > Build Triggers -- > poll scm -- > * * * * * -- > save
commit the changes in GitHub then wait for 1 min.

LIMITATION:
1. in pollscm, we need to wait for the time we set.
2. we will get the last commit only.

WEBHOOK: it will trigger build the moment we change the code.
here we need not to wait for the build.

create ci job -- > Build Triggers: GitHub hook trigger for GITScm polling -- > save

repo on github -- > settings -- > webhooks -- > add webhook -- > Payload URL (jenkins url) -- > http://35.180.46.134:8080/github-webhook/ -- > Content type -- > application/json -- > add

BUILD SCRIPTS: to make jenkins builds from remote loc using script/
give token
give url on other browser.
create ci job -- > Build Triggers: Trigger builds remotely -- > token -- > save

THROTTLE BUILD:

To restrict the builds in a certain time or intervals.
if we dont restrict due to immediate builds jenkins might crashdown.

POINT-1:
by default jenkins will not do concurrent builds.
we need to enable this option in configuration.

Execute concurrent builds if necessary -- > tick it

POINT-2:
MAX BY DEFAULT JENKINS CAN DO 2 BUILDS
IF I WANT MAKE 4 BUILDS AT A TIME

Jenkins -- > Build Executor Status -- >
Built-In Node
Configure
Number of executors :4
save

create a ci job -- > configure -- > Throttle builds -- > Number of builds: 3 -- > time period : hours -- > save

now it will take 20 minutes of gap for each build.

IF JENKINS GOT CRASHED
STOP THE SERVER AND START IT AFTER 2 MINS

NOTE: chkconfig jenkins on
it will restart Jenkins automatically

CUSTOM THEMES:

dashboard -- > manage Jenkins -- > PLUGINS -- > Simple Them & Customizable Header -- > SELECT AND INSTALL ALL OF THEM -- >
manage Jenkins -- > Appearance -- Customizable theme -- > ADD & CSS
https://github.com/alefnode/jenkins-themes (TO GET DIFFERENT THEMS)

https://cdn.rawgit.com/afonsof/jenkins-material-theme/gh-pages/dist/material-cyan.css

ADD HEADERS OF NARESHIT AND USE LOGO FROM IMAGES.

==========

==========

MASTER AND SLAVE:
it is used to distribute the builds.
it reduce the load on jenkins server.
communication blw master and slave is ssh.
Here we need to install agent (java).
SLAVE IS your instance.
slave can use any platform.
label = way of assigning work for slave.

SETUP:
#STEP-1 : Create a server and install JAVA-1.8.0 maven
sudo apt update
sudo apt install -y wget gnupg
wget -qO - https://repos.azul.com/azul-repo.key | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/azul.gpg
echo "deb https://repos.azul.com/zulu/deb stable main" | sudo tee /etc/apt/sources.list.d/zulu.list
sudo apt update
sudo apt install -y zulu8-jdk
sudo apt install -y maven

sudo apt install -y openjdk-17-jdk
update-alternatives --config java
set java to version17

#STEP-2: SETUP THE SLAVE SERVER
Dashboard -- > Manage Jenkins -- > Nodes -- > New node -- > nodename: abc -- > permanent agent -- > save

CONFIGURATION OF SALVE:

Number of executors : 3 #Number of Parallel builds
Remote root directory : /tmp #The place where our output is stored on slave sever.
Labels : one #place the op in a particular slave
useage: last option
Launch method : last option
Host: (your privte ip)
Credentials -- > add -- >jenkins -- > Kind : ssh username with privatekey -- > username: ubuntu
privatekey : pemfile of server -- > save -- >
Host Key Verification Strategy: last option

STEP-3: BUILD JOB ON SLAVE

DASHBOARD -- > JOB -- > CONFIGURE -- > RESTRTICT WHERE THIS JOB RUN -- >
LABEL: SLAVE1 -- > SAVE

TOMCAT:

WEBSITE: FRONTEND -- > DB IS OPT
WEBAPP: FRONTEND + BACKEND -- > DB IS MANDATORY

WEB APPLICATION SERVER/APPLICATION SERVER/APP SERVER = TOMCAT

ITS A WEB APPLICATION SERVER USED TO DEPLOY JAVA APPLICATIONS.
IT IS WRITTEN ON JAVA LANGUAGE.
AGENT: JAVA
PORT: 8080
WE CAN DEPLOY OUR ARTIFACTS.
ITS FREE AND OPENSOURCE.
Its Platform Independent.
YEAR: 1999

war : tomcat/webapps
jar : tomcat/lib

ALTERNATIVES: NGINX, IIS, WEBSPHERE, JBOSS, GLASSFISH, WEBLOGIC

NOTE: USE SLAVE AS TOMCAT FOR TEMP
SETUP: CREATE A NEW SERVER
INSTALL JAVA: sudo apt install -y openjdk-17-jdk

STEP-1: DOWNLOAD TOMCAT (dlcdn.apache.org)
wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.109/bin/apache-tomcat-9.0.109.tar.gz

STEP-2: EXTRACT THE FILES
tar -zxvf apache-tomcat-9.0.109.tar.gz

STEP-3: CONFIGURE USER, PASSWORD & ROLES
vim apache-tomcat-9.0.109/conf/tomcat-users.xml

```
 56   <role rolename="manager-gui"/>
 57   <role rolename="manager-script"/>
 58   <user username="tomcat" password="raham123" roles="manager-gui, manager-script"/>
```

STEP-4: DELETE LINE 21 AND 22
vim apache-tomcat-9.0.109/webapps/manager/META-INF/context.xml

STEP-5: STARTING TOMCAT
sh apache-tomcat-9.0.109/bin/startup.sh

CONNECTION:
COPY PUBLIC IP:8080
manager apps -- > username: tomcat & password: raham123

SCRIPT:

```
#! /bin/bash
wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.109/bin/apache-tomcat-9.0.109.tar.gz
tar -zxvf apache-tomcat-9.0.109.tar.gz
sed -i '56  a\<role rolename="manager-gui"/>' apache-tomcat-9.0.109/conf/tomcat-users.xml
sed -i '57  a\<role rolename="manager-script"/>' apache-tomcat-9.0.109/conf/tomcat-users.xml
sed -i '58  a\<user username="tomcat" password="raham123" roles="manager-gui,
manager-script"/>' apache-tomcat-9.0.109/conf/tomcat-users.xml
sed -i '59  a\</tomcat-users>' apache-tomcat-9.0.109/conf/tomcat-users.xml
sed -i '56d' apache-tomcat-9.0.109/conf/tomcat-users.xml
sed -i '21d' apache-tomcat-9.0.109/webapps/manager/META-INF/context.xml
sed -i '22d'  apache-tomcat-9.0.109/webapps/manager/META-INF/context.xml
sudo sh apache-tomcat-9.0.109/bin/startup.sh
```

DEPLOYING THE APP:
WORKSPACE IN JENKINS DASHBOARD
TARGET
CLICK ON
NETFLIX-1.2.2.war

GO TO TOMCAT
CLICK ON WAR file to deploy - > CHOOSE FILE -- > UPLOAD NETFLIX WAR -- > DEPLOY

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

PIPELINE: STEP BY EXECUTION OF A PROCESS
SERIES OF EVENTS INTERLINKED WITH EACH OTHER.

code -- > build -- > test -- > artifact -- > deployment
CI-PIPELINE CD-PIPELINE
|----------||----------|

IF PIPELINE FAILS ON STAGE-1 IT WONT GO FOR STAGE-2.

plugin download:
Dashboard
Manage Jenkins
Plugins
available
pipeline stage view
install
Restart

why to use ?
to automate the work.
to have clarity about the stage.

TYPES:
1. DECLARATIVE
2. SCRIPTED

pipeline syntax:
We use Groovy Script for jenkins Pipeline.
it consists of blocks that include stages.

SHORTCUT: PASSS

P : PIPELINE
A : AGENT
S : STAGES
S : STAGE
S : STEPS

SINGLE STAGE: this pipeline will have only one stage.

EX-1:

```
pipeline {
    agent any

    stages {
        stage('abc') {
            steps {
                sh 'touch file1'
            }
        }
    }
}
```

EX-2:

```
pipeline {
    agent any

    stages {
        stage('raham') {
            steps {
                sh 'touch file2'
            }
        }
    }
}
```

MULTI STAGE: this pipeline will have more than one stage.

```
pipeline {
    agent any

    stages {
        stage ('two') {
            steps {
                sh 'lsblk'
            }
        }
        stage ('three') {
            steps {
                sh 'lscpu'
            }
        }
        stage ('four') {
            steps {
```

```
            sh 'lsmem'
        }
    }
  }
}
```

CI PIPELINE:

CODE + BUILD + TEST + ARTIFACT

```
pipeline {
  agent any


  tools {
    maven 'maven'
  }

  stages {
    stage ('checkout') {
      steps {
        git 'https://github.com/devopsbyraham/jenkins-java-project.git'
      }
    }
    stage ('build') {
      steps {
        sh 'mvn compile'
      }
    }
    stage ('test') {
      steps {
        sh 'mvn test'
      }
    }
    stage ('artifact') {
      steps {
        sh 'mvn package'
      }
    }
  }
}
```

PIPELINE AS A CODE: Running more than one command/action inside a single stage.
to reduce the length of the code.
to save the time.

```
pipeline {
    agent any


    tools {
        maven 'maven'
    }

    stages {
        stage ('checkout') {
            steps {
                git 'https://github.com/devopsbyraham/jenkins-java-project.git'
                sh 'mvn compile'
                sh 'mvn test'
                sh 'mvn package'
            }
        }
    }
}
```

MULTI STAGE PIPELINE AS A CODE: Running more than one command/action in multiple stages.

```
pipeline {
    agent any


    tool {
        maven 'maven'
    }

    stages {
        stage ('one') {
            steps {
                git 'https://github.com/devopsbyraham/jenkins-java-project.git'
                sh 'mvn compile'
            }
```

```
        }
        stage ('two') {
            steps {
                sh 'mvn test'
                sh 'mvn package'
            }
        }
    }
}
```

PAAC OVER SINGLE SHELL: Running all the shell commands on a single shell.

```
pipeline {
    agent any

    tools {
        maven 'maven'
    }

    stages {
        stage ('checkout') {
            steps {
                git 'https://github.com/devopsbyraham/jenkins-java-project.git'
                sh '''
                mvn compile
                mvn test
                mvn clean package
                '''
            }
        }
    }
}
```

INPUT PARAMETERS: BASED ON USER INPUT THE PIPELINE IS GOING TO EXECUTE.
used to avoid the mistakes.
pipeline will wait until we provide the input.

```
pipeline {
    agent any

    tools {
        maven 'maven'
```

```
    }

    stages {
        stage ('checkout') {
            steps {
                git 'https://github.com/devopsbyraham/jenkins-java-project.git'
            }
        }
        stage ('build') {
            steps {
                sh 'mvn compile'
            }
        }
        stage ('test') {
            steps {
                sh 'mvn test'
            }
        }
        stage ('artifact') {
            steps {
                sh 'mvn package'
            }
        }
        stage ('deploy') {
            input {
                message "is your inputs correct ?"
                ok "yes"
            }
            steps {
                echo "my code is deployed"
            }
        }
    }
}
```

NOTE: In real time providing manual approval is best practise for Jenkins pipelines.

NOTE: if we have syntax issue none of the stages will execute.

TERRAFORM PIPELINE:

CREDENTIALS ADDING

MANAGE JENKINS
CREDENTIALS
GLOBAL
ADD CREDENTIALS
KIND: SECERTE TEXT
AWS_ACCESS_KEY_ID: VALUE -- > id: accesskey
AWS_SECRET_ACCESS_KEY: VALUE -- > id: secretkey

```
pipeline {
    agent any

    environment {
        AWS_ACCESS_KEY_ID = credentials('accesskey')
        AWS_SECRET_ACCESS_KEY = credentials('secretkey')
    }

    stages {
        stage('Code') {
            steps {
                git branch: 'main', url: 'https://github.com/RAHAMSHAIK007/rahamterrarepo.git'
            }
        }
        stage('init') {
            steps {
                sh 'terraform init'
            }
        }
        stage('validate') {
            steps {
                sh 'terraform validate'
            }
        }
        stage('plan') {
            steps {
                sh 'terraform plan'
            }
        }
        stage('apply') {
            input {
                message "is everything fine ?"
                ok "yes"
```

```
        }
        steps {
            sh 'terraform $action -auto-approve'
        }
      }

  }
}
```

DIFF BLW SCRIPTED VS DECLARATVE

SCRIPTED: DECLARATIVE:
SHORT LONG
NO STAGES IT HAS STAGES
START NODE START WITH PIPELINE

==========
ARTIFACTORY ON NEXUS AND S3

NEXUS:
Its an Artifactory storage service.
used to store artifacts on repo. (.JAR, .WAR, .EAR)
Nexus server -- > Repo -- > Artifact
we can use this server to rollback in real time.
it req t2.medium & 30 GB EBS
nexus uses java-17
PORT: 8081
NOTE: ALLOW REDPLOY POLICY TO STORE WAR FILE MULTIPLE TIMES.

ALTERTAVIVES: JFROG, S3, -----

STEP-1: CREATE A UBUNTU SERVER WITH T2.MEDIUM AND 30 GB EBS

STEP-2: EXECUTE THE BELOW SCRIPT ON SERVER
wget https://download.sonatype.com/nexus/3/nexus-unix-x86-64-3.79.0-09.tar.gz
tar -zxvf nexus-unix-x86-64-3.79.0-09.tar.gz
yum install java-17-amazon-corretto -y
sudo useradd nexus
chown -R nexus:nexus nexus-3.79.0-09
sudo sh nexus-3.79.0-09/bin/nexus start

SETUP SCIPT:
https://github.com/RAHAMSHAIK007/all-setups.git

STEP-3: CONFIGURE NEXUS SERVER
copy-public-ip:8081 (browser)

STEPS: signin -- > username: admin & password: /app/sonatype-work/nexus3/admin.password -- > next -- > set password -- > disable anonymous access -- > save

CREATING REPO:
settings symbol -- > repositories -- > new -- > maven2(hosted) -- > name -- > save

NOTE: to integrate any tool with Jenkins we need to download the plugin.

STEP-4: INTEGRATING NEXUS TO PIPELINE

NEXUS INTEGRATION TO PIPELINE:
1. Download the plugin (Nexus Artifact Uploader)
Manage Jenkins -- > plugins -- > Available Plugins -- > Nexus Artifact Uploader -- > install.
2. Configure it to pipeline by using pipeline syntax
search for nexus

NOTE: All the information will be available on pom.xml file.

ADDING CREDS:
Dashboard
Manage Jenkins
Credentials
global
Add creds

PIPELINE:

```
pipeline {
    agent any

    stages {
        stage('checkout') {
            steps {
                git 'https://github.com/devopsbyraham/jenkins-java-project.git'
```

```
        }
    }
    stage('build') {
        steps {
            sh 'mvn compile'
        }
    }
    stage('test') {
        steps {
            sh 'mvn test'
        }
    }
    stage('artifact') {
        steps {
            sh 'mvn package'
        }
    }
    stage('Artifact upload') {
        steps {
            nexusArtifactUploader artifacts: [[artifactId: 'NETFLIX', classifier: '', file:
'target/NETFLIX-1.2.2.war', type: '.war']], credentialsId:
'968c23dd-b648-4f15-91bf-7d76981a1218', groupId: 'in.RAHAM', nexusUrl:
'100.25.197.110:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'netflix', version:
'1.2.2'
        }
    }
  }
}
```

INTEGRTAING S3 WITH JENKINS:
1. installing s3 publisher plugin.
2. configure s3profile
(Manage Jenkins -- > System -- > s3 profile -- > give creds -- > test -- > save)

3. use pipeline syntax to give details(select s3 upload)
PROFILE: raham
SOURCE: target/NETFLIX-1.2.2.war
Destination bucket: BUCKET NAME
Bucket Region: REGION

NOTE: DONT GIVE SPACE FOR BUCKET NAME

```
aws s3api put-bucket-versioning --bucket my_bucket --versioning-configuration Status=Enabled
```

```
pipeline {
    agent any

    stages {
        stage('code') {
            steps {
                git branch: '$branch', url:
'https://github.com/RAHAMSHAIK007/jenkins-java-project.git'
            }
        }
        stage('build') {
            steps {
                sh 'mvn compile'
            }
        }
        stage('test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('code quality') {
            steps {
                echo " my code is done with code quality testing"
            }
        }
        stage('artifact') {
            steps {
               sh 'mvn clean package'
            }
        }
        stage('s3') {
            steps {
                s3Upload consoleLogLevel: 'INFO', dontSetBuildResultOnFailure: false,
dontWaitForConcurrentBuildCompletion: false, entries: [[bucket: 'netflixwarfiles3bucket',
excludedFile: '', flatten: false, gzipFiles: false, keepForever: false, managedArtifacts: false,
noUploadOnFailure: false, selectedRegion: 'us-east-1', showDirectlyInBrowser: false,
sourceFile: 'target/NETFLIX-1.2.2.war', storageClass: 'STANDARD', uploadFromSlave: false,
useServerSideEncryption: false]], pluginFailureResultConstraint: 'FAILURE', profileName:
```

```
'raham', userMetadata: []
            }
        }
        stage('deploy') {
            steps {
                deploy adapters: [
                    tomcat9(credentialsId: 'a333c89f-d56f-49d7-8630-eeceaf165b5f', path: '', url:
'http://13.217.161.148:8080/')
                ],
                contextPath: 'raham',
                war: 'target/*.war'
            }
        }
    }
}
```

In summary: S3 wins for minimal cost and maintenance, especially for storing WAR files for real-time access.

DOWNLOAD PLUGIN CALLED PERIDOIC BACKUP

Dashboard
Manage Jenkins
Periodic Backup Manager
Configure

Temporary Directory : /tmp
Backup schedule (cron) : 0 2 * * *
Maximum backups in location : 10
Store no older than (days) : 7
File Management Strategy : FullBackup
Storage Strategy : zip Format

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

RBAC:

RBAC: ROLE BASE ACCESS CONTROL.
TO restrict the user PERMISSIONS in jenkins.

fresher1= fresher
raham = exp

STEP-1: USER CREATION
manage jenkins -- > users -- > create users -- > suresh: fresher

STEP-2: PLUGIN DOWNLOADING
Dashboard
Manage Jenkins
Plugins
Available plugin
Role-based Authorization Strategy

STEP-3: CONFIGURE THE PLUGIN
Dashboard
Manage Jenkins
Security
Authorization
Role-based Strategy
SAVE

STEP-4: MANAGE AND ASSIGN ROLES TO USERS
manage roles -- > add -- > fresher -- > fresher: overall read & job:read -- > save
assign roles -- > add user -- > yesudas: fresher -- > save

RESTRICTING TO SPECIFIC JOB:
MANAGE JENKINS -- > SECURITY --> AUTHORIZATION -- > PROJECT BASED MATRIX
AUTHORIZATION STRATEGY -- > ADD USER -- > YESUDAS -- > Overall Read

GO TO JOB -- > Enable project-based security
ADD USER -- > YESUDAS
JOB: BUILD & READ
SAVE

HOW TO ADD PARAMETERS:

PARAMETERS: Used to pass inputs for jobs

CHOICE: to pass single input at a time. (99%)
STRING: to pass multiple inputs at a time. (1%)

MULTI-LINE STRING: to pass multiple inputs on multiple lines at a time.
FILE: to pass the file as input.
BOOL: to pass input either yes or no.

This project is parameterized
Name: branch
Choices:
dev
test
save

BACKUP:

TOMCAT INEGRATION:
STEP-1: INSTALL TOMCAT ON SERVER
STEP-2: DOWNLOAD PLUGIN (Deploy to Container)
STEP-3: CREDS ADDING:
Dashboard
Manage Jenkins
Credentials
System
Global credentials (unrestricted)
Add credentials
username & password
save

```
pipeline {
    agent any

    tools {
        maven 'maven'
    }
```

```
    stages {
        stage('code') {
            steps {
                git branch: '$branch', url:
'https://github.com/RAHAMSHAIK007/jenkins-java-project.git'
            }
        }
        stage('build') {
            steps {
                sh 'mvn compile'
            }
        }
        stage('test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('artifact') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('s3') {
            steps {
                s3Upload consoleLogLevel: 'INFO', dontSetBuildResultOnFailure: false,
dontWaitForConcurrentBuildCompletion: false, entries: [[bucket: 'mynetflixartifactbucket007',
excludedFile: '', flatten: false, gzipFiles: false, keepForever: false, managedArtifacts: false,
noUploadOnFailure: false, selectedRegion: 'us-east-1', showDirectlyInBrowser: false,
sourceFile: 'target/NETFLIX-1.2.2.war', storageClass: 'STANDARD', uploadFromSlave: false,
useServerSideEncryption: false]], pluginFailureResultConstraint: 'FAILURE', profileName:
'raham', userMetadata: []
            }
        }
        stage('deploy') {
            steps {
                deploy adapters: [
                    tomcat9(
                        credentialsId: '6d2b07d2-3027-435c-b243-8ff6067d9e63',
                        url: 'http://98.85.254.168:8080/'
                    )
                ],
                    contextPath: 'netflix',
                    war: 'target/NETFLIX-1.2.2.war'
```

```
        }
      }
   }
}
```

==========

CREDS ADDING:
Dashboard
Manage Jenkins
Credentials
System
Global credentials (unrestricted)
Add credentials
username & pasword

STEPS FOR PROJECT:

STEP-1: WE CREATE THE CLIENT ACCOUNT IN AWS. [aws orginization]

STEP-2: CREATE THE INFRASTRUCTURE FROM HCP.
Link: https://github.com/devopsbyraham/netflixdemoinfra.git

```
provider "aws" {
region = "us-east-1"
}

resource "aws_instance" "one" {
count = 4
ami = "ami-04aa00acb1165b32a"
instance_type = "t2.medium"
key_name = "argocd"
vpc_security_group_ids = ["sg-06785bc60f40dceda"]
tags = {
Name = var.instance_names[count.index]
}
}

variable "instance_names" {
default = ["jenkins", "APPSERVER-1", "APPSERVER-2", "Monitoring server"]
}
```

AWS_ACCESS_KEY_ID : VALUE: Environment variable -- > SENSITIVE -- > SAVE

AWS_SECRET_ACCESS_KEY: Environment variable -- > SENSITIVE -- > SAVE
NOTE: MARK THEM AS ENV VARIBALE AND MAKE SURE NO SPACES ARE GIVEN

STEP-3: CONFIGURE THE SEVERS [SCRIPTS] [JENKINS, TOMCAT, S3, MONITORING]
Link: https://github.com/RAHAMSHAIK007/all-setups.git

SETP-4: WRITE PIPELINE FOR CI/CD.
A. Configure Slave
B. Download Plugins [pipeline stage view, s3, Deploy to container, slack]
C. Write Pipeline Code

SETP-5: INTEGRATE TO SLACK

STEP-6: MONITOR THE APPLICATION FROM GRAFANA

TOOLS:
GIT
GITHUB
MAVEN
JENKINS
NEXUS
TOMCAT
HCP
TERRAFORM
SLACK
PROMETHEUS
GRAFANA

PIPELINE CODE:

```
pipeline {
    agent any

    stages {
        stage('checkout') {
            steps {
                git branch: '$branch', url: 'https://github.com/devopsbyraham/jenkins-java-project.git'
            }
        }
        stage('build') {
            steps {
```

```
            sh 'mvn compile'
        }
    }
    stage('test') {
        steps {
            sh 'mvn test'
        }
    }
    stage('artifact') {
        steps {
            sh 'mvn package'
        }
    }
    stage('Nexus') {
        steps {
            nexusArtifactUploader artifacts: [[artifactId: 'NETFLIX', classifier: '', file:
'target/NETFLIX-1.2.2.war', type: '.war']], credentialsId:
'35cc5458-5e5a-4992-8e90-394d9a3c9ab1', groupId: 'in.RAHAM', nexusUrl:
'ec2-3-80-145-80.compute-1.amazonaws.com:8081', nexusVersion: 'nexus3', protocol: 'http',
repository: 'abcd', version: '1.2.2'
        }
    }
    stage('deploy') {
        steps {
            deploy adapters: [
                tomcat9(
                    credentialsId: '85e5e800-4740-461a-868d-519b51fa90ed',
                    path: '',
                    url: 'http://ec2-54-234-53-68.compute-1.amazonaws.com:8080/'
                )
            ],
            contextPath: 'netflix',
            war: 'target/*.war'
        }
    }
  }
  post {
    always {
        echo "slack notify"
        slackSend (
             channel: '#netflix', message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME}
\n build ${env.BUILD_NUMBER} \n More info at: ${env.BUILD_URL}"
        )
```

```
    }
  }
}
```

PATH FOR PROMTHEUS: vim /etc/prometheus/prometheus.yml

==========

LEVEL-1: JENKINS
PLUGINS
SYNTAX
CONFIGURATION

LEVEL-2: SERVER
GIT MAVEN JAVA PROMETHEUS (TOOL ISSUES)
FIREWALL & NETWORK
HARDWARE

LEVEL-3: CODE
LIBS
CODE SMELLS
SYNTAX OR ALLIGNEMT

==========

Automated: Deployment,

Installation & DEPLOYMENT [ONE AT A TIME]

To perform end-to-end automation we can use Ansible.
Creating servers
configure servers
deployment application on servers
Not with single, it can deal with multiple server.
ANSIBLE ALTERNATIVES: CHEF, PUPPET, SALTSTACK ----

ANSIBLE:
its a Configuration Management Tool.
Configuration: Hardware and Software

Management: Pkgs update, installing, remove ----
Ansible is used to manage and work with multiple servers together.
its a free and Opensource.
it is used to automate the entire deployment process on multiple servers.
We install Python on Ansible.
we use a key-value format for the playbooks.

PLAYBOOK:
create servers
install packages & software
deploy apps

Jenkins = pipeline = groovy
ansible = playbooks = yaml
YAML: YET ANOTHER MARKUP LANGUAGE

HISTORY:
in 2012 dev called Maichel Dehaan who developed ansible.
After few years RedHat taken the ansible.
it is platform-independent & will work on all linux flavours.

ARCHITECTURE:
PLAYBOOK: its a file which consist of code
INVENTORY: its a file which consist ip of nodes
SSH: used to connect with nodes
Ansible is Agent less.
Means no need to install any software on worker nodes.

SETUP:
CREATE 5 SERVERS [1=ANSIBLE, 2=DEV, 2=TEST]

EXECUTE THE BELOW COMMANDS ON ALL 5 SERVERS:
sudo -i
hostnamectl set-hostname ansible/dev-1/dev-2/test-1/test-2
sudo -i

passwd root -- > to login to other servers
vim /etc/ssh/sshd_config [40 = uncomment line and put yes 65 (no = yes)]
systemctl restart sshd
systemctl status sshd

hostname -i

THE BELOW STEPS NEED TO BE RUN ON ANSIBLE SERVER:

yum install ansible -y
yum install python3 python-pip python-dlevel -y (optional)

vim /etc/ansible/hosts
[dev]
172.31.20.40
172.31.21.25
[test]
172.31.31.77
172.31.22.114

NOTE: PLEASE I KINDLY REQUEST GIVE YOUR PRIVATE IPS :(

ssh-keygen -- > enter 4 times
ssh-copy-id root@private ip of dev-1 -- > yes -- > password -- > ssh private ip -- > ctrl d
ssh-copy-id root@private ip of dev-2 -- > yes -- > password -- > ssh private ip -- > ctrl d
ssh-copy-id root@private ip of test-1 -- > yes -- > password -- > ssh private ip -- > ctrl d
ssh-copy-id root@private ip of test-2 -- > yes -- > password -- > ssh private ip -- > ctrl d

ansible -m ping all : To check worker node connection with ansible server.

1. ADHOC COMMANDS:
these are simple Linux commands.
these are used for temp works.
these commands will be over ridden.

ansible all -a "yum install git -y"
ansible all -a "yum install maven -y"
ansible all -a "mvn --version"
ansible all -a "touch file1"
ansible all -a "touch raham.txt"
ansible all -a "ls"
ansible all -a "yum install httpd -y"
ansible all -a "systemctl status httpd"
ansible all -a "systemctl start httpd"
ansible all -a "useradd raham"
ansible all -a "cat /etc/passwd"
ansible all -a "yum remove git* maven* httpd* -y"

2. MODULES:
its a key-value pair.
modules are reusable.
module flag is -m
we can use different modules for different purposes.

ansible all -m yum -a "name=git state=present"
ansible all -m yum -a "name=maven state=present"
ansible all -m yum -a "name=maven state=present" [present=installed]
ansible all -m service -a "name=httpd state=started"[started=restart]
ansible all -m service -a "name=httpd state=stopped" [stopped=stop]
ansible all -m yum -a "name=http state=absent" [absent=uninstall]
ansible all -m user -a "name=vikram state=present"
ansible all -m group -a "name=devops state=absent"
ansible all -m copy -a "src=raham.txt dest=/tmp"

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

3. PLAYBOOKS:
playbooks used to execute multiple modules.
we can reuse the playbook multiple times.
in real time we use a playbook to automate our work.
for Server Creation, pkg installation, deployment ----
here we use key-value pairs.
Key-Value can also be called as Dictionary.
ansible-playbook will be written on YAML syntax.
YAML = YET ANOTHER MARKUP LANGUAGE
extension for playbook is .yml or .yaml
playbook start with --- and end with ... (opt)

EX-1:

```
- hosts: all
  tasks:
    - name: installing git
      yum: name=git state=present

    - name: installing httpd
```

```
    yum: name=httpd state=present

  - name: starting httpd
    service: name=httpd state=started

  - name: create user
    user: name=jayanth state=present

  - name: copy a file
    copy: src=index.html dest=/root
```

TO EXECUTE: ansible-playbook playbok.yml

Gather facts: it will get information of worker nodes
its by default task performed by ansible.

ok=total number of tasks
changed= no.of tasks successfully executed

sed -i 's/present/absent/g' playbook.yml

BY DEFAULT ANSIBLE WILL EXECUTE PLAYBOOK ON SEQUENTIAL.
IF TASK-3 GOT FAILED IT WILL STOP EXECUTING REMAING TASKS.
Put ignore_errors: true IT WILL IGNORE THE FAILED TASK AND RUN REMAINING TASKS.

EX-2:

```
- hosts: all
  ignore_errors: true
  tasks:
    - name: installing git
      yum: name=git state=absent

    - name: installing httpd
      yum: name=httpd state=absent

    - name: starting httpd
      service: name=httpd state=started

    - name: create users
      user: name=pushpa state=absent
```

```
    - name: copying a file
      copy: src=raham.txt dest=/root
```

TAGS: by default ansible will execute all tasks sequentially in a playbook.
we can use tags to execute a specific tasks or to skip a specific tasks.

EX-1:

```
- hosts: all
  ignore_errors: yes
  tasks:
    - name: installing git
      yum: name=git state=present
      tags: a

    - name: installing httpd
      yum: name=httpd state=present
      tags: b

    - name: starting httpd
      service: name=httpd state=started
      tags: c

    - name: create a user
      user: name=kohli state=present
      tags: d

    - name: copy a file
      copy: src=index.html dest=/tmp
      tags: e
```

SINGLE TAG: ansible-playbook raham.yml --tags d
MULTI TAGS: ansible-playbook raham.yml --tags b,c

EX-2:

```
- hosts: all
  ignore_errors: yes
  tasks:
    - name: uninstalling git
      yum: name=git* state=absent
```

```
    tags: a

  - name: uninstalling httpd
    yum: name=httpd state=absent
    tags: b

  - name: starting httpd
    service: name=httpd state=started
    tags: c

  - name: delete a user
    user: name=kohli state=absent
    tags: d

  - name: copy a file
    copy: src=index.html dest=/tmp
    tags: e
```

SKIP A SINGLE TASK: ansible-playbook raham.yml --skip-tags c
SKIP MULTIPLE TASK: ansible-playbook raham.yml --skip-tags a,c

SETUP MODULE: used to print the complete info of worker nodes
ansible all -m setup

ansible all -m setup | grep -i family
ansible all -m setup | grep -i pkg
ansible all -m setup | grep -i cpus
ansible all -m setup | grep -i mem

SHORTCUTS FOR LINUX:
vim .bashrc
alias ar='ansible-playbook raham.yml'
alias info='ansible all -m setup'

source .bashrc

VARIABLES:

STATIC VARS: we can define these vars inside the playbook and use for multiple times, once a variable is defined here it will not change untill we change.

```
- hosts: all
  vars:
    a: git
    b: maven
  tasks:
    - name: installing {{a}}
      yum: name={{a}} state=present

    - name: installing {{b}}
      yum: name={{b}} state=present
```

TO EXECUTE: ansible-playbook playbbok.yml

DYNAMIC VARS: these vars will be defined outside the playbook and these will change as per our requirments.

```
- hosts: all
  vars:
  tasks:
    - name: installing maven
      yum: name={{a}} state=absent
    - name: installing httpd
      yum: name={{b}} state=absent
```

ansible-playbook raham.yml --extra-vars "a=docker b=httpd"

CREATING EC2 FROM PLAYBOOK:

NOTE: If Ansible want to create an Ec2 instance in the cloud it need to have permission so to allocate the permission for our Ansible we can create IAM user

IAM -- > create user -- > name: Ansible -- > next -- > Attach policies directly -- > ec2fullAccess -- > next -- > create user

Ansible -- > Security Credentials -- > Create access key -- > CLI -- > checkbox -- > create access key --> download .csv file

COME TO ANSIBLE SERVER:
aws configure -- > giving ansible user permissions to server

AWS Access Key ID [None]: xxxxxxxxxxxx
AWS Secret Access Key [None]: xxxxxxxxxxx
Default region name [None]: us-east-1
Default output format [None]: table

BOTO: it provides access for developers to access the aws services for python.

sudo yum install pip -y
sudo pip install boto3

```
- hosts: localhost
  tasks:
    - name: Create an instance
      amazon.aws.ec2_instance:
        name: "public-withebs-instance"
        image_id: "ami-052064a798f08f0d3"
        instance_type: "t3.micro"
```

==========

HANDLERS:
when we have two tasks in a single playbook if task 2 is depending upon task 1 so then we can use the concept called handlers .
once task one is executed successfully it will notify task 2 to perform the operation.
the name of the notify and the name of the task two must be same.
if task-1 haven't executed successfully or even skipped, it wont executed the task-2

```
- hosts: all
  tasks:
    - name: installing httpd
      yum: name=httpd state=present
      notify: starting httpd
  handlers:
    - name: starting httpd
      service: name=httpd state=started
```

sed -i 's/present/absent/g' raham.yml

```
- hosts: all
  tasks:
    - name: installing httpd
```

```
    yum: name=httpd state=absent
    notify: starting httpd
  handlers:
    - name: starting httpd
      service: name=httpd state=started
```

SETUP MODULE: used to print the complete info of worker nodes

```
ansible all -m setup
```

```
ansible all -m setup  | grep -i family
ansible all -m setup  | grep -i pkg
ansible all -m setup  | grep -i cpus
ansible all -m setup  | grep -i mem
```

CONDITIONS:
CLUSTER: Group of servers
HOMOGENIUS: all servers have having same OS and flavour.
HETROGENIUS: all servers have different OS and flavour.

used to execute this module when we have different Clusters.

RedHat=yum
Ubuntu=apt

```
- hosts: all
  tasks:
    - name: installing git on RedHat
      yum: name=git state=present
      when: ansible_os_family == "RedHat"

    - name: installing git on Debian
      apt: name=git state=present
      when: ansible_os_family == "Debian"
```

```
- hosts: all
  tasks:
    - name: installing httpd
      yum: name=httpd state=present
      when: ansible_nodename == "dev-1"

    - name: installing mysql
```

```
      yum: name=mysql state=present
      when: ansible_nodename == "dev-2"

    - name: installing python
      yum: name=python state=present
```

VALIDATORS:
1. YAMLINT
2. YAML FORMATTER
3. YAML CHECKER
4. CHATGPT

SHELL VS COMMAND VS RAW:

```
- hosts: all
  tasks:
    - name: installing maven
      shell: yum install maven -y

    - name: installing httpd
      command: yum install httpd -y

    - name: installing docker
      raw: yum install docker -y
```

raw >> command >> shell.

```
ansible all -a "mvn -v"
ansible all -a "htppd -v"
ansible all -a "docker -v"
```

L : LINUX
A : APACHE
M : MYSQL
P : PYTHON

```
- hosts: all
  tasks:
    - name: installing apache
      yum: name=httpd state=present
```

```
    - name: installing mysql
      yum: name=mysql state=present

    - name: installing python
      yum: name=python3 state=present
```

```
 ansible all -a "httpd --version"
 ansible all -a "python3 --version"
 ansible all -a "mysql --version"
```

PIP: its a pkg manager used to install python libs/modules

Redhat: yum
ubuntu: apt
python: pip

```
- hosts: all
  tasks:
    - name: install pip
      yum: name=pip state=present

    - name: installing NumPy
      pip: name=NumPy state=present

    - name: installing Pandas
      pip: name=Pandas state=present
```

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

DEBUG: to print the messages from a playbook.

```
- hosts: all
  tasks:
    - name: printing a msg
```

```
    debug:
      msg: hai all welcome to my session
```

ansible all -m setup

NAME : ansible_nodename
FAMILY : ansible_os_family
PKG : ansible_pkg_mgr
CPU : ansible_processor_vcpus
MEM : ansible_memtotal_mb
FREE : ansible_memfree_mb

```
- hosts: all
  tasks:
    - name: print a msg
      debug:
        msg: "my node name is: {{ansible_nodename}}, the os is: {{ansible_os_family}}, the package manager is: {{ansible_pkg_mgr}}, total cpus is: {{ansible_processor_vcpus}}, the total ram: {{ansible_memtotal_mb}}, free ram is: {{ansible_memfree_mb}}"
```

JINJA2 TEMPLATE:
used to get the customized outputs.
its a text file which can extract the variables and these values will change as per time.

INLINE MODULE: Used to change specific line in a files on worker node.

```
- hosts: all
  tasks:
    - name: Replace the first line with 'hello all'
      lineinfile:
        path: /root/file1
        line: "hello everyone"
        insertafter: '^.*$'
        state: present
```

FETCH: Used to fetch data in files from remote servers to local.
Note: Create a file in worker nodes

```
- hosts: all
  tasks:
    - name: fetching data
      fetch: src=/root/raham.txt dest=/tmp/abc.txt
```

LOOKUPS: this module used to get data from files, db and key values

```
- hosts: dev
  vars:
    a: "{{lookup('file', '/root/creds.txt') }}"
  tasks:
    - debug:
        msg: "hai my user name is {{a}}"
```

```
cat creds.txt
user=raham
```

CREATING EC2 FROM PLAYBOOK:

NOTE: If Ansible want to create an Ec2 instance in the cloud it need to have permission so to allocate the permission for our Ansible we can create IAM user

IAM -- > create user -- > name: Ansible -- > next -- > Attach policies directly -- > ec2fullAccess -- > next -- > create user

Ansible -- > Security Credentials -- > Create access key -- > CLI -- > checkbox -- > create access key --> download .csv file

COME TO ANSIBLE SERVER:
aws configure -- > giving ansible user permissions to server

```
AWS Access Key ID [None]: xxxxxxxxxxxxx
AWS Secret Access Key [None]: xxxxxxxxxxxx
Default region name [None]: us-east-1
Default output format [None]: table
```

BOTO: it provides access for developers to access the aws services for python.

```
sudo yum install pip -y
sudo pip install boto3
```

```
- hosts: localhost
```

```
  tasks:
    - name: start an instance and Add EBS
      amazon.aws.ec2_instance:
        name: "public-withebs-instance"
        image_id: "ami-0150ccaf51ab55a51"
        instance_type: "t2.micro"
```

STRATAGIES: Way of executing the playbook.

LINEAR: execute tasks sequentially
if task-1 is executed on server-1 it will wait till task-2 execution
FREE: execute all tasks without sequential
if task-1 is executed on server-1 it wont wait till task-2 execution
ROLLING:
BATCH:

==========

SETUP MODULE: used to print the complete info of worker nodes
ansible all -m setup

ansible all -m setup | grep -i family
ansible all -m setup | grep -i pkg
ansible all -m setup | grep -i cpus
ansible all -m setup | grep -i mem

CONDITIONS:
CLUSTER: Group of servers
HOMOGENIUS: all servers have having same OS and flavour.
HETROGENIUS: all servers have different OS and flavour.

used to execute this module when we have different Clusters.

RedHat=yum
Ubuntu=apt

- hosts: all

```
  tasks:
    - name: installing git on RedHat
      yum: name=git state=present
      when: ansible_os_family == "RedHat"

    - name: installing git on Debian
      apt: name=git state=present
      when: ansible_os_family == "Debian"


- hosts: all
  tasks:
    - name: installing httpd
      yum: name=httpd state=present
      when: ansible_nodename == "dev-1"

    - name: installing mysql
      yum: name=mysql state=present
      when: ansible_nodename == "dev-2"

    - name: installing python
      yum: name=python state=present
```

VALIDATORS:
1. YAMLINT
2. YAML FORMATTER
3. YAML CHECKER
4. CHATGPT

SHELL VS COMMAND VS RAW:

```
- hosts: all
  tasks:
    - name: installing maven
      shell: yum install maven -y

    - name: installing httpd
      command: yum install httpd -y

    - name: installing docker
      raw: yum install docker -y
```

raw >> command >> shell.

```
ansible all -a "mvn -v"
ansible all -a "htppd -v"
ansible all -a "docker -v"
```

```
L       : LINUX
A       : APACHE
M       : MYSQL
P       : PYTHON
```

```
- hosts: all
  tasks:
    - name: installing apache
      yum: name=httpd state=present

    - name: installing mysql
      yum: name=mysql state=present

    - name: installing python
      yum: name=python3 state=present
```

```
 ansible all -a "httpd --version"
 ansible all -a "python3 --version"
 ansible all -a "mysql --version"
```

PIP: its a pkg manager used to install python libs/modules

Redhat: yum
ubuntu: apt
python: pip

```
- hosts: all
  tasks:
    - name: install pip
      yum: name=pip state=present

    - name: installing NumPy
```

```
    pip: name=NumPy state=present

  - name: installing Pandas
    pip: name=Pandas state=present
```

```
.
└── roles
    ├── pkgs
    │   └── tasks
    │       └── main.yml
    ├── users
    │   └── tasks
    │       └── main.yml
    └── webserver
        └── tasks
            └── main.yml
```

ROLES:
roles is a way of organizing playbooks in a structured format.
main purpose of roles is to encapsulate the data.
we can reuse the roles multiple times.
length of the playbook is decreased.
it contains on vars, templates, task -----
in real time we use roles for our daily activities.
yum install tree -y

mkdir playbooks
cd playbooks/

mkdir -p roles/pkgs/tasks
vim roles/pkgs/tasks/main.yml

```
- name: installing pkgs
  yum: name=git state=present
- name: install maven
  yum: name=maven state=present
- name: installing docker
  yum: name=docker state=present
```

mkdir -p roles/users/tasks
vim roles/users/tasks/main.yml

```
- name: create users
  user: name={{item}} state=present
  with_items:
    - uday
    - naveen
    - rohit
    - lokesh
    - saipallavi
    - supriya

mkdir -p roles/webserver/tasks
vim roles/web/tasks/main.yml

- name: installing httpd
  yum: name=httpd state=present

- name: starting httpd
  service: name=httpd state=started

cat master.yml

- hosts: all
  roles:
    - pkgs
    - users
    - webserver

find . -type f -exec sed -i 's/present/absent/g' {} \;
```

ANSIBLE GALAXY:

Ansible Galaxy is a website where users can share roles and to a command-line tool for installing, creating, and managing roles.
Ansible Galaxy gives greater visibility to one of Ansible's most exciting features, such as application installation or reusable roles for server configuration.
Lots of people share roles in the Ansible Galaxy.
Ansible roles consist of many playbooks, which is a way to group multiple tasks into one container to do the automation in a very effective manner with clean, directory structures.

ANSIBLE VAULT:
it is used to encrypt the files, playbooks ----
Technique: AES256 (USED BY FACEBOOK, AWS)

vault will store our data very safely and securely.
if we want to access any data which is in the vault we need to give a password.
Note: we can restrict the users to access the playbook also.

cat creds.txt
user=raham
passowrd=test123

ansible-vault create creds1.txt : to create a vault
ansible-vault edit creds1.txt : to edit a vault
ansible-vault rekey creds1.txt : to change password for a vault
ansible-vault decrypt creds1.txt : to decrypt the content
ansible-vault encrypt creds1.txt : to encrypt the content
ansible-vault view creds1.txt : to show the content without decrypt

ASYNCHRONOUS & POLLING ACTIONS:
for every task in ansible we can set time limit
if the task is not performed in that time limit ansible will stop playbook execution
this is called as asynchronous and polling.

```
- hosts: all
  ignore_errors: yes
  tasks:
    - name: sleeping
      command: sleep 30
      async: 20
      poll: 10

    - name: install git
      yum: name=git state=present
```

async: time we set for task to complete
poll: it will check if task is completed or not for every 10 sec

WEB SERVER : TO SHOW THE APP : httpd : 80 : /var/www/html
frontend code

APP SERVER : TO USE THE APP : Tomcat : 8080 : tomcat/webapps
frontend code + backend code

```
- hosts: all
  tasks:
    - name: installing httpd
      yum: name=httpd state=present

    - name: starting httpd
      service: name=httpd state=started

    - name: installing git
      yum: name=git state=present

    - name: checkout
      git:
        repo: https://token@github.com/RAHAMSHAIK007/netflix-clone.git
        dest: /var/www/html
      tags: a
```

==========

LINK FOR SCRIPTS & PLAYBOOKS : https://github.com/RAHAMSHAIK007/all-setups.git
LINK FOR PROJECT: https://github.com/devopsbyraham/jenkins-java-project.git

Install Jenkins on ansible server and connect to dashboard

SETP-1: PUSHING THE CODE FROM GIT TO GITHUB : N0
STEP-2: PERFORM CI ON JENKINS : NO
STEP-3: STORE ARTIFACT ON S3 : YES
STRP-4: INSTALL TOMCAT ON NODES : YES
SETP-5: COPY WAR FILE TO TOMCAT

JENKINS:

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum upgrade
# Add required dependencies for the jenkins package
yum install java-17-amazon-corretto jenkins -y
sudo yum install jenkins -y
sudo systemctl restart jenkins
sudo systemctl daemon-reload
```

HOW TO INSTALL TOMAT:
yum install git -y
git clone https://github.com/RAHAMSHAIK007/all-setups.git
cd all-setups
ansible-playbook tomcat.yml

MAVEN INTEGRATION:
Manage Jenkins
Tools
Maven
name: maven
save

INTEGRTAING S3 WITH JENKINS:
1. installing s3 publish plugin.
2. configure s3profile
(Manage Jenkins -- > System -- > s3 profile -- > give creds -- > test -- > save)
3. use pipeline syntax to give details
PROFILE:
SOURCE: target/NETFLIX-1.2.2.war
Destination bucket: BUCKET NAME
Bucket Region: REGION

NOTE: MAKE SURE YOU DONT HAVE ANY SPACE FOR BUCKET NAME

INTEGRTAING ANSIBE WITH JENKINS:

1. install ansible plugin

2. Configure ansible to jenkins
manage jenkins -- > tools -- > ansible -- > name: ansible & Path to ansible executables directory: /usr/bin -- > save (NOTE: /usr/bin is a folder where all Linux commands will store)

3. SAMPLE STEP: ANSIBLE PLAYBOOK
pipeline syntax -- > sample steps -- > select ansible playbook
Ansible tool: ansible -- >
Playbook file path in workspace: /etc/ansible/deploy.yml -- >
Inventory file path in workspace: /etc/ansible/hosts -- >
SSH CREDS: give creds of ansible & worker nodes -- > type: username & password
hostsubnet: dev
Disable the host SSH key check -- >

generate script

DECLARAIVE PIPELINE:

```
pipeline {
    agent any

    stages {
        stage('code') {
            steps {
                git branch: '$branch', url:
'https://github.com/RAHAMSHAIK007/jenkins-java-project.git'
            }
        }
        stage('build') {
            steps {
                sh 'mvn compile'
            }
        }
        stage('test') {
            steps {
                sh 'mvn test'
            }
        }
        stage('sonarqube') {
            steps {
                echo "my code is passed the sonar gate qualities"
            }
        }
        stage('artifact') {
            steps {
                sh 'mvn clean package'
            }
        }
        stage('s3') {
            steps {
                s3Upload consoleLogLevel: 'INFO', dontSetBuildResultOnFailure: false,
dontWaitForConcurrentBuildCompletion: false, entries: [[bucket: 'rahamshaik007netflixbucket',
excludedFile: '', flatten: false, gzipFiles: false, keepForever: false, managedArtifacts: false,
noUploadOnFailure: false, selectedRegion: 'us-east-1', showDirectlyInBrowser: false,
sourceFile: 'target/NETFLIX-1.2.2.war', storageClass: 'STANDARD', uploadFromSlave: false,
useServerSideEncryption: false]], pluginFailureResultConstraint: 'FAILURE', profileName: 's3',
userMetadata: []
```

```
            }
        }
        stage('deploy') {
            steps {
                ansiblePlaybook credentialsId: 'd9713c79-8d68-41fa-a6bd-3d7d70c92b64',
disableHostKeyChecking: true, installation: 'ansible', inventory: '/etc/ansible/hosts', limit:
'$server', playbook: '/etc/ansible/deploy.yml', vaultTmpPath: ''
            }
        }
    }
}
```

SCRIPTED PLIPELINE:

```
node {
    stage('code') {
        git 'https://github.com/devopsbyraham/jenkins-java-project.git'
    }
    stage('build') {
        sh 'mvn compile'
    }
    stage('test') {
        sh 'mvn test'
    }
    stage('artifact') {
        sh 'mvn package'
    }
    stage('s3') {
        s3Upload consoleLogLevel: 'INFO', dontSetBuildResultOnFailure: false,
dontWaitForConcurrentBuildCompletion: false, entries: [[bucket: 'netflixartifact0099',
excludedFile: '', flatten: false, gzipFiles: false, keepForever: false, managedArtifacts: false,
noUploadOnFailure: false, selectedRegion: 'ap-south-1', showDirectlyInBrowser: false,
sourceFile: 'target/NETFLIX-1.2.2.war', storageClass: 'STANDARD', uploadFromSlave: false,
useServerSideEncryption: false]], pluginFailureResultConstraint: 'FAILURE', profileName:
's3-bucket', userMetadata: []
    }
    stage('deploy') {
        ansiblePlaybook credentialsId: 'ansible', disableHostKeyChecking: true, installation:
'ansible', inventory: '/etc/ansible/hosts', limit: '$server', playbook: '/etc/ansible/deploy.yml',
vaultTmpPath: ''
    }
}
```

cat playbook.yml

```
- hosts: all
  tasks:

    - name: task1
      copy:
        src: /var/lib/jenkins/workspace/pipeline/target/NETFLIX-1.2.2.war
        dest: /root/tomcat/webapps
```

PROJECT:

Q: TELL ME ABOUT YOUR CURRENT PROJECT YOU ARE WORKING ON ?
INTRO: THANKS FOR GIVING THE OPPURTUNITY TO EXPLAIN MY PROJECT.

1. IN MY COMPANY WE USE MONOLITHIC ARCHITECTURE FOR OUR PROJECT.
2. OUR REAL TIME WORK AS A DEVOPS ENGINEER WILL START WITH CREATING INFRA.
3. FOR INFRA CREATION WE USE TERRAFORM IN OUR COMPANY.
4. WE USE MODULES TO CREATE 3 TIER ARCHITECTURE IN AWS CLOUD.

5. ONCE INFRA GOT CREATED WE CONFIGURE THE SERVERS USING ANSIBLE.
6. WE INSTALL SOFTWARES ON WEB, APP, DB SERVERS BY USING PLAYBOOKS
7. WE CREATE ROLES AND WE EXECUTE THEM WHENEVER WE WANT.

8. OUR DEV WILL PUSH THE CODE TO GITHUB WE INTEGRATE CODE FROM GITHUB TO JENKINS
9. IN JENKINS WE DO CI (BUILD + TEST) AS RESULT WE GET ARTIFACT(WAR)
10. WE STORE ARTIFACTS ON S3 BUCKETS FOR ROLLBACK PURPOSE.
11. WE RUN THE PIPELINE THAT WILL DEPLOY THE APP AUTOMATICALLY TO THE DEV SERVERS.
12. AFTER DEV WE DEPLOY ON TEST AND UAT IF EVERY THING IS FINE WE DEPLOY THE APP TO PROD.

PIPELINE FLOW:
CODE -- > BUILD -- > TEST -- > SONARQUBE -- > ARTIFACT -- > S3 -- > DEPLOY -- > SLACK

13. AFTER DEPLOYING THE APP WE VERIFY THE APP IS RUUNING OR NOT FROM ELB DNS.

14. FINALLY ONCE EVERYTHING IS DONE WE MONITOR THE SERVERS FROM GRAFANA.

TOOLS:
CLOUD : AWS
INFRA : TERRAFORM
CODE : GIT & GITHUB
CI : JENKINS
CONFIG MANAGEMENT: ANSIBLE
MONITORING : PROMETHEUS & GARFANA
APP SERVER : TOMCAT

ISSUES YOU FACED IN THE PROJECT:

SIR IN OUR PROJECT ONE OF THE MOST COMMON THING I FACED IS PIPELINE FAILURES, BUT HOW I RESLOVED IS

1. PIPELINE FAILURES: DEVOPS TEAM
WE NEED TO CHECK LOGS
WE NEED TO CHECK CONFIGURATION
WE NEED TO CHECK CREDS
WE NEED TO CHECK PLUGINS
WE NEED TO CHECK DEPENDENCIES

2. INFRA FAILURES: DEVOPS TEAM
WE NEED TO CHECK THE PLUGINS
WE NEED TO THE CONFIG AND SYNTAX ISSUES
WE NEED TO CHECK THE PROVIOSINERS
MODULE STRUCTURE

3. CODE FAILURES: DEV TEAM
WE NEED TO CHECK SONARQUBE DASHBOARD
WE NEED TO CHECK COMPILE STAGE
WE NEED TO SEND LOGS TO DEV

IF YOUR PROD APP IS NOT WORKING PROPERLY WHAT YOU GONNA DO : ROLLBACK
IF YOUR PROD APP IS WORKING SLOWLY WHAT YOU GONNA DO : SCALING, EDGE LOC
IF YOUR PROD APP IS NOT SHOWING APP TO USER : CHECK THE SERVICE HTTPD/TOMCAT

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

APPLICATION: Collection of services

MONOLITHIC: multiple services are deployed on single server with single database.
MICRO SERVICES: multiple services are deployed on multiple servers with multiple database.

BASED ON USERS AND APP COMPLEXITY WE NEED TO SELECT THE ARCHITECTURE.

FACTORS AFFECTING FOR USING MICROSERVICES:
F-1: COST
F-2: MAINTAINANCE
F-3: TIME

CONTAINERS:
its free of cost and can create multiple containers.
its same as a server/vm.
it will not have any operating system.
os will be on images.
(SERVER=AMI, CONTAINER=IMAGE)

CONTAINER TOOLS: DOCKER, CONTAINERD, ROCKET, CRI-O, PODMAN, KATA
CONTAINER ----

DOCKER:
Its an free & opensource tool.
it is platform independent.
used to create, run & deploy applications on containers.
it is introduced on 2013 by solomenhykes & sebastian phal.
We used GO language to develop the docker.
here we write files on YAML.
Docker will use host resources (cpu, mem, n/w, os).
Docker can run on any OS but it natively supports Linux distributions.
before docker user faced lot of problems, but after docker there is no issues with the application.

CONTAINERIZATION/DOCKERIZATION:
Process of packing an application with its dependencies.
ex: PUBG

APP= PUBG & DEPENDECY = MAPS
APP= CAKE & DEPENDECY = KNIFE

os level of virtualization.

VIRTUALIZATION:
able to create resource with our hardware properties.

ARCHITECTURE & COMPONENTS:
client: it will interact with user
user gives commands and it will be executed by docker client

daemon: manages the docker components(images, containers, volumes)

host: where we install docker (ex: linux, windows, macos)

Registry: manages the images.
INSIDE THE IMAGE WE HAVE OS TO RUN THE CONTAINER.

ARCHITECTURE OF DOCKER:
yum install docker -y #client
systemctl start docker #client,Engine
systemctl status docker

COMMANDS:
docker pull ubuntu : pull ubuntu image
docker images : to see list of images
docker run -it --name cont1 ubuntu : to create a container
-it (interactive) - to go inside a container
cat /etc/os-release : to see os flavour

apt update -y : to update
redhat=yum
ubuntu=apt
without update we cant install any pkg in ubuntu

apt install git -y
apt install apache2 -y

ctrl p q : to exit container
docker ps -a : to list all containers
docker attach cont_name : to go inside container

docker stop cont_name : to stop container
docker start cont_name : to start container
docker pause cont_name : to pause container
docker unpause cont_name: to unpause container
docker inspect cont_name: to get complete info of a container
docker rm cont_name : to delete a container

STOP: will wait to finish all process running inside container
KILL: wont wait to finish all process running inside container
==========
=

OS LEVEL OF VIRTUALIZATION:
ability to take backup of complete os and reuse it.

docker pull ubuntu
docker run -it --name cont1 ubuntu
apt update -y
apt install mysql-server apache2 python3 -y
touch file{1...5}
apache2 -v
mysql-server --version
python3 --version
ls

ctrl p q

docker commit cont1 raham:v1
docker run -it --name cont2 raham:v1
apache2 -v
mysql-server --version
python3 --version
ls

DOCKERFILE:
it is an automation way to create image.
here we use components to create image.
in Dockerfile D must be Capiatl.
Components also capital.
To write our instructions we need to use components in Dockerfile
This Dockerfile will be Reuseable.
here we can create image directly without container help.

Name: Dockerfile

```
docker kill $(docker ps -qa)
docker rm $(docker ps -qa)
docker rmi -f $(docker images -qa)
```

COMPONENTS:

```
FROM        : used to 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
ENV         : to set env variables (inside container)
ARGS        : to pass env variables (outside containers)
LABEL       : to add labels for docker images
EXPOSE              : to indicate port number
VOLUME
NETWORK
```

ENTRYPOINT VS CMD:

```
FROM ubuntu
ENTRYPOINT ["sleep"]
CMD ["1000"]
```

EX-1:

```
FROM ubuntu
RUN  apt update
RUN apt install apache2 -y
RUN apt install python3 -y
CMD apt install mysql-server -y
```

```
docker build -t raham:v1 .
docker run -it --name cont1 raham:v1
```

EX-2:

```
FROM ubuntu
```

COPY index.html /tmp
ADD https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.102/bin/apache-tomcat-9.0.102.tar.gz /tmp

docker build -t raham:v2 .
docker run -it --name cont2 raham:v2

EX-3:
FROM ubuntu
COPY index.html /tmp
ADD https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.107/bin/apache-tomcat-9.0.107.tar.gz /tmp
WORKDIR /tmp
ENV user raham
ENV client swiggy

docker build -t raham:v3 .
docker run -it --name cont3 raham:v3

EX-4:

FROM ubuntu
LABEL author raham
EXPOSE 8080

docker build -t raham:v4 .
docker run -it --name cont4 raham:v4

INDEX.HTML LINK: https://www.w3schools.com/howto/tryit.asp?
filename=tryhow_css_form_icon

NETFLIX-DEPLOYMENT:

apt install git -y
git clone https://github.com/RAHAMSHAIK007/netflix-clone.git
cd netflix-clone/

Vi Dockerfile

FROM ubuntu
RUN apt update
RUN apt install apache2 -y
COPY * /var/www/html/
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]

docker build -t netflix:v1 .
docker run -it --name netflix1 -p 80:80 netflix:v1

MULTI-STAGE BUILD:
With multi-stage builds, you use multiple FROM statements in your Dockerfile
if we build image-1 from docker file and use that image-1 to build other image.

Dockerfile
FROM ubuntu

----

build image1

FROM image1 -- > multi-stage build

ADV:
less time
less work
less size
less complexity

==========

vim Dockerfile

FROM ubuntu
RUN apt update -y
RUN apt install apache2 -y
COPY index.html /var/www/html
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]

Index.html: take form w3 schools
https://www.w3schools.com/howto/tryit.asp?filename=tryhow_css_form_icon

docker build -t movies:v1 .
docker run -itd --name movies -p 81:80 movies:v1

docker build -t train:v1 .
docker run -itd --name train -p 82:80 train:v1

docker build -t dth:v1 .
docker run -itd --name dth -p 83:80 dth:v1

docker build -t recharge:v1 .
docker run -itd --name recharge -p 84:80 recharge:v1

docker ps -a -q : to list container ids
docker kill $(docker ps -a -q) : to kill all containers
docker rm $(docker ps -a -q) : to remove all containers

Note: In the above process all the containers are managed and created one by one in real time we manage all the containers at same time so for that purpose we are going to use the concept called Docker compose.

DOCKER COMPOSE:
It's a tool used to manage multiple containers in single host.
we can create, start, stop, and delete all containers together.
we write container information in a file called a compose file.
compose file is in YAML format.
inside the compose file we can give images, ports, and volumes info of containers.
we need to download this tool and use it.

INSTALLATION:
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose version

In Linux majorly you are having two type of commands first one is inbuilt commands which come with the operating system by default
second one is download commands we are going to download with the help of yum, apt or Amazon Linux extras.
some commands we can download on binary files.

NOTE: linux will not give some commands, so to use them we need to download seperately

once a command is downloaded we need to move it to /usr/local/bin
because all the user-executed commands in linux will store in /usr/local/bin
executable permission need to execute the command

vim docker-compose.yml

```
version: '3.8'
services:
  movies:
    image: movies:v1
    ports:
      - "81:80"
  train:
    image: train:v1
    ports:
      - "82:80"
  dth:
    image: dth:v1
    ports:
      - "83:80"
  recharge:
    image: recharge:v1
    ports:
      - "84:80"
```

COMMANDS:

```
docker-compose up -d                : to create and start all containers
docker-compose stop         : to stop all containers
docker-compose start        : to start all containers
docker-compose kill         : to kill all containers
docker-compose rm           : to delete all containers
docker-compose down                 : to stop and delete all containers
docker-compose pause                : to pause all containers
docker-compose unpause              : to unpause all containers
docker-compose ps -a                : to list the containers managed by compose file
docker-compose images               : to list the images managed by compose file
docker-compose logs         : to show logs of docker compose
docker-compose top          : to show the process of compose containers
docker-compose restart              : to restart all the compose containers
docker-compose scale train=10       : to scale the service
```

CHANGING THE DEFULT FILE:

by default the docker-compose wil support the following names
docker-compose.yml, docker-compose.yaml, compose.yml, compose.yaml

mv docker-compose.yml raham.yml
docker-compose up -d : throws an error

docker-compose -f raham.yml up -d
docker-compose -f raham.yml ps
docker-compose -f raham.yml down

images we create on server.
these images will work on only this server.

git (local) -- > github (internet) = to access by others
image (local) -- > dockerhub (internet) = to access by others

CREATE A DOCKERHUB ACCOUNT
CLICK ON DOTS -- > DOCKER HUB -- > CREATE REPOSITORY

Replace your username

STEPS:
create dockerhub account
create a repo

docker tag movies:v1 vijaykumar444p/movies
docker login -- > username and password
docker push vijaykumar444p/movies

docker login -- > username and password

docker tag train:v1 vijaykumar444p/train
docker push vijaykumar444p/train

docker tag dth:v1 vijaykumar444p/dth
docker push vijaykumar444p/dth

docker tag recharge:v1 vijaykumar444p/recharge

docker push vijaykumar444p/recharge

docker rmi -f $(docker images -q)
docker pull vijaykumar444p/movies:latest

INSTALLING MULTIPLE TOOLS:
docker run -it --name jenkins -p 8080:8080 jenkins/jenkins:lts
docker run -d --name prometheus-container -e TZ=UTC -p 9090:9090
ubuntu/prometheus:2-24.04_stable
docker run -d --name=grafana -p 3000:3000 grafana/grafana

DOCKER SAVE:
docker image save swiggy:v1 > swiggy:v1.tar :covert image to file
docker image history swiggy:v1
docker rmi swiggy:v1
docker images
docker image load < swiggy\:v1.tar

COMMAND TO ZIP: gzip dummy:v5.tar abc.zip
DECOMPRESS COMMAND: gzip movies:latest.gz -d

COMPRESSING DOCKER IMAGE SIZE:
1. push to dockerhub
2. use multi stage docker build
3. reduce layers
4. use tar balls

==========

High Avaliabilty: more than one server
why: if one server got deleted then other server will gives the app

DOCKER SWARM:
its an orchestration tool for containers.
used to manage multiple containers on multiple servers.
here we create a cluster (group of servers).
in that cluster, we can create same container on multiple servers.
here we have the manager node and worker node.
manager node will create & distribute the container to worker nodes.
worker node's main purpose is to maintain the container.
without docker engine we cant create the cluster.

Port: 2377
worker node will join on cluster by using a token.
manager node will give the token.

docker swarm join --token
SWMTKN-1-2hlbbmgpw65ciunzowk7692vigc7d74arft7xpg35a72fkn9lu-577e38366sinrwo9heo9
pn0wn 172.31.83.232:2377

SETUP:
create 3 servers
install docker and start the service
hostnamectl set-hostname manager/worker-1/worker-2
Enable 2377 port

docker swarm init (manager) -- > copy-paste the token to worker nodes
docker node ls

Note: individual containers are not going to replicate.
if we create a service then only containers will be distributed.

SERVICE: it's a way of exposing and managing multiple containers.
in service we can create copy of conatiners.
that container copies will be distributed to all the nodes.

service -- > containers -- > distributed to nodes

http://3.84.44.179:81/
http://54.163.208.40:81/
http://35.175.222.164:81/

docker service create --name movies --replicas 3 -p 81:80 vijaykumar444p/movies:latest
docker service ls : to list services
docker service inspect movies : to get complete info of service
docker service ps movies : to list the containers of movies
docker service scale movies=10 : to scale in the containers
docker service scale movies=3 : to scale out the containers
docker service rollback movies : to go previous state
docker service logs movies : to see the logs
docker service rm movies : to delete the services.

when scale in it follows lifo pattern.
LIFO MEANS LAST-IN FIRST-OUT.

Note: if we delete a container it will recreate automatically itself.
it is called as self healing.

CLUSTER ACTIVIES:
docker swarm leave (worker) : to make node inactive from cluster
To activate the node copy the token.
docker node rm node-id (manager): to delete worker node which is on down state
docker node inspect node_id : to get comple info of worker node
docker swarm join-token manager : to generate the token to join

Note: we cant delete the node which is ready state
if we want to join the node to cluster again we need to paste the token on worker node

DOCKER NETWORKING:
Docker networks are used to make communication between the multiple containers that are running on same or different docker hosts.

We have different types of docker networks.
Bridge Network : SAME HOST
Overlay network : DIFFERENT HOST
Host Network
None network

BRIDGE NETWORK: It is a default network that container will communicate with each other within the same host.

OVERLAY NETWORK: Used to communicate containers with each other across the multiple docker hosts.

HOST NETWORK: When you Want your container IP and ec2 instance IP same then you use host network

NONE NETWORK: When you don’t Want The container to get exposed to the world, we use none network. It will not provide any network to our container.

To create a network: docker network create network_name
To see the list: docker network ls

To delete a network: docker network rm network_name
To inspect: docker network inspect network_name
To connect a container to the network: docker network connect network_name container_id/name
docker exec -it cont1 /bin/bash
apt update
apt install iputils-ping -y : command to install ping checks
ping ip-address of cont2
crtl pq

To disconnect from the container: docker network disconnect network_name container_name
To prune: docker network prune

PORTAINER:
it is a container organizer, designed to make tasks easier, whether they are clustered or not.
abel to connect multiple clusters, access the containers, migrate stacks between clusters
it is not a testing environment mainly used for production routines in large companies.
Portainer consists of two elements, the Portainer Server and the Portainer Agent.
Both elements run as lightweight Docker containers on a Docker engine

Must have swarm mode and all ports enable with docker engine
curl -L https://downloads.portainer.io/ce2-16/portainer-agent-stack.yml -o portainer-agent-stack.yml
docker stack deploy -c portainer-agent-stack.yml portainer
 docker ps
public-ip of swamr master:9000

DATABASE SETUP:
docker run -itd --name dbcont -e MYSQL_ROOT_PASSWORD=raham123 mysql:9.0.1
docker exec -it dbcont /bin/bash
mysql -u root -p

docker run -it --rm -p 8888:8080 tomcat:9.0

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

DOCKER VOLUMES:

It is used to store data inside container.

volume is a simple directory inside container.
containers uses host resources (cpu, ram, rom).
single volume can be shared to multiple containers.
ex: cont-1 (vol1) --- > cont2 (vol1) & cont3 (vol1) & cont4 (vol1)
at a time we can share single volume to single container only.
every volume will store under /var/lib/docker/volumes

METHOD-1:
DOCKER FILE:

```
FROM ubuntu
VOLUME ["/volume1"]
```

```
docker build -t raham:v1 .
docker run -it --name cont1 raham:v1
cd volume1/
touch file{1..5}
cat>file1
ctrl p q

docker run -it --name cont2 --volumes-from cont1  ubuntu
cd /volume1
ll
touch file{6..10}
ctrl pq

docker attach cont1
cd volume1
ll

docker run -it --name cont3 --volumes-from cont1 ubuntu
```

METHOD-2:
FROM CLI:

```
docker run -it --name cont4 -v volume2 ubuntu
cd volume2/
touch java{1..5}
ctrl p q

docker run -it --name cont5 --volumes-from cont4  ubuntu
cd volume2
ll
```

touch java{6..10}
ctrl p q
docker attach cont4
ls

METHOD-3: VOLUME MOUNTING

docker volume ls : to list volumes
docker volume create name : to create volume
docker volume inspect volume3 : to get info of volume3
cd /var/lib/docker/volumes/volume3/_data
touch python{1..5}
docker run -it --name cont5 --mount source=volume3,destination=/abc ubuntu
docker volume rm : to delete volumes
docker volume prune : to delete unused volumes

HOST -- > CONTAINER:

cd /root
touch raham{1..5}
docker volume inspect volume4
cp * /var/lib/docker/volumes/volume4/_data
docker exec cont5 ls /volume4

RESOURCE MANAGEMENT:
By default, docker containers will not have any limits for the resources like cpu ram and memory so we need to restrict resource use for container.

By default docker containers will use host resources(cpu, ram, rom)
Resource limits of docker container should not exceed the docker host limits.

docker stats --> to check live cpu and memory

docker run -it --name cont7 --cpus="0.1" --memory="300mb" ubuntu
docker update cont7 --cpus="0.7" --memory="300mb"

JENKINS SETUP BY DOCKER:
docker run -it --name jenkins -p 8080:8080 jenkins/jenkins:lts

PROMETHEUS SETUP BY DOCKER:
docker run -it --name prometheus -p 9090:9090 bitnami/prometheus:latest

GRAFANA SETUP BY DOCKER:
docker run -it --name grafana -p 3000:3000 grafana/grafana

TRIVY:
ITS A TOOL USED TO SCAN IMAGES AND FILE SYSTEMS
IT ENSURE OUR IMAGES ARE SAFE AND SECURE BEFORE WE USE THEM

wget
https://github.com/aquasecurity/trivy/releases/download/v0.18.3/trivy_0.18.3_Linux-64bit.tar.gz
tar zxvf trivy_0.18.3_Linux-64bit.tar.gz
sudo mv trivy /usr/local/bin/
echo "export PATH=$PATH:/usr/local/bin/" >> .bashrc
source .bashrc

COMMANDS:

trivy image raham:v1
trivy image ubuntu:latest
docker pull ubuntu:22.04
trivy image ubuntu:22.04
docker pull nginx:1.19.6 #1.19.6 specifies the vulnerable version of the nginx image
trivy image nginx:1.19.6
trivy filesystem /
==========
==========

STEP-1: LAUNCH AN INSTANCE WITH T2.LARGE AND EBS 30 ON UBUNTU 24.04
STEP-2: INSTALL JENKINS, GIT, DOCKER & TRIVY

JENKINS:
#! /bin/bash
sudo apt update -y
sudo apt install -y openjdk-17-jdk
java -version
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023 .key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update -y

sudo apt install -y jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins

DOCKER:
apt install docker.io -y
chmod 777 ///var/run/docker.sock

TRIVY:
wget
https://github.com/aquasecurity/trivy/releases/download/v0.18.3/trivy_0.18.3_Linux-64bit.tar.gz
tar zxvf trivy_0.18.3_Linux-64bit.tar.gz
sudo mv trivy /usr/local/bin/
echo "export PATH=$PATH:/usr/local/bin/" >> .bashrc
source .bashrc

STEP-3: INSTALL THE FOLLOWING JENKINS PLUGINS

PROJECT PIPELINE CODE:

```
pipeline {
    agent any

    tools {
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME = tool 'mysonar'
    }
    stages {
        stage("Clean WS") {
            steps {
                cleanWs()
            }
        }
        stage("Code") {
            steps {
                git "https://github.com/devops0014/Zomato-Project.git"
            }
        }
        stage("Sonarqube Analysis") {
```

```
            steps {
                withSonarQubeEnv('mysonar') {
                    sh """$SCANNER_HOME/bin/sonar-scanner \
                        -Dsonar.projectName=zomato \
                        -Dsonar.projectKey=zomato"""
                }
            }
        }
        stage("Quality Gates") {
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }
        stage("Install Dependencies") {
            steps {
                sh 'npm install'
            }
        }
        stage("OWASP") {
            steps {
                dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit
--disableNodeAudit', odcInstallation: 'DP-Check'
                dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }
        stage("Trivy") {
            steps {
                sh 'trivy fs . > trivyfs.txt'
            }
        }
        stage("Build") {
            steps {
                sh 'docker build -t image1 .'
            }
        }
        stage("Tag & Push") {
            steps {
                script {
                    withDockerRegistry(credentialsId: 'docker-password') {
                        sh 'docker tag image1 rahamshaik/mydockerprojectfor5pm:myzomatoimage'
                        sh 'docker push rahamshaik/mydockerprojectfor5pm:myzomatoimage'
```

```
                }
            }
        }
    }
    stage("Scan the Image") {
        steps {
            sh 'trivy image rahamshaik/mydockerprojectfor5pm:myzomatoimage'
        }
    }
    stage("Container") {
        steps {
            sh 'docker run -d --name cont1 -p 3000:3000
rahamshaik/mydockerprojectfor5pm:myzomatoimage'
        }
    }
  }
}
```

Key Differences

Feature	OWASP	SonarQube
Focus	Web app security risks	Code quality & security
Type	Security framework/tools	Static code analysis tool
Approach	Scanning live apps, guidelines	Scanning source code
Use Cases	Finding web vulnerabilities	Improving code quality, security

CODE LEVEL SECURITY : SONARQUBE
AP LEVEL SECURITY : OWASP
IMAGE LEVEL SECURITY : TRIVY

When to Use What?
Use OWASP when you need security best practices, penetration testing, and live application security assessments.

Use SonarQube when you need automated code analysis for vulnerabilities, bugs, and maintainability in your source code.

WHAT IS TRIVY:

Trivy is an open-source security scanner for containers, Kubernetes, infrastructure as code (IaC), and dependencies. It helps detect vulnerabilities, misconfigurations, and secrets in various environments.

How Trivy Works
It fetches vulnerability data from sources like NVD, Red Hat, Debian, and GitHub Security Advisories.
Scans images, filesystems, repositories, or Kubernetes clusters.
Outputs results in various formats like JSON, table, and SARIF (for security tools).

Why Use Trivy?
✅ Fast & lightweight – Minimal setup and quick scanning.
✅ Comprehensive – Covers multiple security aspects (CVEs, misconfigurations, secrets).
✅ Easy integration – Works with Docker, Kubernetes, and CI/CD pipelines.

DEPENENCY CHECK:
It is an open source tool performing analysis of 3rd party dependencies;
NOTE: false positives and false negatives may exist in the analysis performed by the tool.

Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

NOTE: PLEASE CHECK DOCKER PROJECT PDF WHICH IS UPLOADED TO GOOGLE CLASS ROOM

==========

ECS & ECR:

ECS:

Amazon Elastic Container Service (Amazon ECS) is a highly scalable and fast container management service.
Used to create, run, stop, and manage containers on a cluster.
ECS doesn't have any server and compute power (CPU & Ram).
Can be able to do roll back.
With Amazon ECS, your containers are defined in a task definition that you use to run an

individual task or task within a service.
Deploy & load balance application across multiple servers.
It will do auto scaling to handle the large traffic.
You can run your tasks and services on a serverless infrastructure that's managed by AWS Fargate.
Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.

DOCKER-COMPOSE VS ECS:
When we use the docker-compose file to deploy it contains all the configuration of containers.
For example if we have 3 servers we cannot deploy to all the containers at the same time, and if you do the same deployment on all three servers individually it cannot able to know all the 3 servers are manging the same application which is major disadvantage.
Docker compose cannot able to do automatic load balancing and Auto scaling.
Instead of using K8S, Swarm, Apahe Mesos and NoMad we can use ECS which is alternative for those
Apache Mesos is the opposite of virtualization because in virtualization one physical resource is divided into multiple virtual resources, while in Mesos multiple physical resources are clubbed into a single virtual resource.

ECS LAUNCH TYPES
WORKING WITH EC2:
Here we are going to manage the infrastructure ie instances.
We need to install ECS agent to communicate.
Need to set Firewall.
We need to make sure all the patches is done for recent updates.
Finally you can manage the containers and configure it as per your requirments.

WORKING WITH FARGET:
It follow serverless Architecture.
We don’t have EC2 instances so need not maintain them.
It will create servers on demand.
We will pay for what we use here.

ECS TASKS:
A task definition is required to run Docker containers in Amazon ECS. Simply it is the blue print for containers and how it is deployed. It will contain
The Docker image to use with each container in your task for Application.
How much CPU and memory to use with each task or each container within a task
The infrastructure that your tasks are hosted on

The Docker networking mode to use for the containers in your task
The logging configuration to use for your tasks
Whether the task continues to run if the container finishes or fails
The command that the container runs when it's started
Any data volumes that are used with the containers in the task
The IAM role that your tasks use

ECS SERVICES:
A service definition defines how to run your Amazon ECS service.
It will ensure that certain tasks are running at all times.
It will restart the containers that is crashed or exited.
For example if you have an application we want 2 instances/containers tp run our application all time, we say this to service then it will create 2 instances/containers and start running our application.
If any instance fail it will restart the task.
LOAD BALANCERS:
The main intention is to distribute the traffic here.
If we have app is deployed we can assign the LB and route the traffic to resources.
If we scale our instance the LB is here able to drive the traffic to newly created instance.

HOSTING AN APPLICATION ON ECS:
UPLOADING IMAGE TO ECR:
Amazon Elastic Container Registry (ECR) is a fully managed container registry.
Easy to store, manage, share, and deploy your container images and artifacts anywhere.
It supports private repositories with resource-based permissions using AWS IAM.
Specified users or EC2 instances can access your container repositories and images.
You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts.

STEPS: WORK ON MUMBAI REGION

1. CREATE EC2 AND INSTALL GIT & DOCKER
yum install git docker -y
systemctl start docker
systemctl status docker

2. GET THE CODE AND CREATE A IMAGE
git clone https://github.com/RAHAMSHAIK007/netflix-clone.git
cd netflix-clone

vim Dockefile

```
FROM ubuntu
RUN apt update && apt install apache2 -y
COPY * /var/www/html
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```

docker build -t swiggy:v1 .

3. CREATE A REPO ON ECR AND PUSH THE SIMAGE

ECR -- > CREATE -- > NAME: SWIGGY -- > SCAN IMAGE -- > CREATE REPO

View push commands
NOTE: USE ACCESS KEY AND SECRET KEYS FOR AUTHENTICATION
RUN ALL THE 4 COMMANDS SETP BY STEP TO PUSH IMAGE TO ECR.

EXPLORE FOLLOWING OPTIONS:
SCANNING
PERMISSION
REPLICATION
CACHE PULL
LIFECYCLE

4. CREATE TASK DEFINATION.
TASK DEFINATAION -- > CREATE -- > Task definition family: Netflix -- > AWS Fargate -- > ceate a new role -- > VALUES UPTO YOUR CHOICE -- > CREATE

5. CREATE A CLUSTER BY USING FARGETE.
ECS -- > CLUSTER -- > CREATE -- > NAME -- >
AWS Fargate (serverless) -- > CREATE

6. CREATE SERVICE

ECS =-- > CLUSTER -- > SERVICE -- > CREATE -- > FARGATE -- > TASK -- > NETFLIX -- > DESIRED TASKS: 3 -- > LOAD BALANC AND ASG -- > CREATE

NOTE: DELETE TASKS BEFORE DELETE CLUSTER.

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

DOCKER ALTERNATIVES: containerd, rocket, cri-o, Podman, Kata Containers

CHIRU : CLUSTER
NAGABABU: NODE
PAWAN : POD
CHARAN : CONTAINER
ALLU : APP

K8S:

LIMITATIONS OF DOCKER SWARM:
1. CANT DO AUTO-SCALING AUTOMATICALLY
2. CANT DO LOAD BALANCING AUTOMATICALLY
3. CANT HAVE DEFAULT DASHBOARD
4. CANT DO LOAD ADVANCE NETWORKING.
5. USED FOR EASY APPS.

HISTORY:
Initially Google created an internal system called Borg (later called as omega) to manage its thousands of applications [JOE, BURNS]
later they donated the borg system to cncf and they make it as open source.
initial name is Borg but later cncf rename it to Kubernetes
the word Kubernetes originated from Greek word called pilot or Hailsmen.
Borg: 2014
K8s first version came in 2015.
IT is an free and open-source container orchestration platform.
It is used to automates many of the manual processes like deploying, managing, and scaling containerized applications.

ARCHITECTURE:

DOCKER : CNCA
K8S: CNPCA

C : CLUSTER
N : NODE

P : POD
C : CONTAINER
A : APPLICATION

COMPONENTS:
MASTER:

1. API SERVER: communicate with user (takes command execute & give op)
2. ETCD: database of cluster (stores complete info of a cluster ON KEY-VALUE pair)
3. SCHEDULER: select the worker node to schedule pods (depends on hw of node)
4. CONTROLLER: control the k8s objects (n/w, service, Node)

WORKER:

1. KUBELET : its an agent (it will inform all activites to master) It create containers.
2. KUBEPROXY: it deals with nlw (ip, networks, ports)
3. POD: group of conatiners (inside pod we have app)

Note: all components of a cluster will be created as a pod.

API SERVER : FATHER
ETCD : MARRIAGE BROKER
SCHEDULER : BRIDE
CONTROLLER : MOTHER

KUBLET : SIBBLINGS
KUBE PROXY : RELATIVES
POD : FUNCTION HALL

API-SERVER : FATHER : COMMUNICATION TO K8S CLUSTER
ETCD : MARRIAGE BROKER : STORE CLUSTER-INFORAMTION
SCHEDULER : BRIDE : TO SELECT THE NODE TO PLACE POD
CONTROLLER : MOTHER : NODE, RC, SC, NC ----------

KUBELET : SIBBLINGS : INFORM ALL ACTIVITES TO API-SERVER
KUBE-PROXY : RELATIVES : DEAL WITH NETWORK
POD : FUNCTION HALL : GROUP OF CONTAINER

CLUSTER TYPES:

1. SELF MANAGED: WE NEED TO CREATE & MANAGE THEM

minikube = single node cluster
kubeadm = multi node cluster (manual)
kops = multi-node cluster (automation)
kind:
k9s:
kubespray:

2. CLOUD-BASED: CLOUD PROVIDERS WILL MANAGE THEM

AWS = EKS = ELASTIC KUBERNETES SERVICE
AZURE = AKS = AZURE KUBERENETS SERVICE
GOOGLE = GKE = GOOGLE KUBERENETS ENGINE

MINIKUBE:
It is a tool used to setup single node cluster on K8's.
Here Master and worker runs on same server.
It contains API Servers, ETDC database and container runtime
It is used for development, testing, and experimentation purposes on local.
It is a platform Independent.
Installing Minikube is simple compared to other tools.

NOTE: But we don't implement this in real-time Prod

REQUIREMENTS:

2 CPUs or more
2GB of free memory
20GB of free disk space
Internet connection
Container or virtual machine manager, such as: Docker.

Kubectl is the command line tool for k8s
if we want to execute commands we need to use kubectl.

SETUP:
sudo apt update -y
sudo apt upgrade -y

```
sudo apt install curl wget apt-transport-https -y
sudo curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo mv minikube-linux-amd64 /usr/local/bin/minikube
sudo chmod +x /usr/local/bin/minikube
sudo minikube version
sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
sudo echo "$(cat kubectl.sha256) kubectl" | sha256sum --check
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
sudo minikube start --driver=docker --force
```

NOTE: When you download a command as binary file it need to be on /usr/local/bin
because all the commands in linux will be on /usr/local/bin
and need to give executable permission for that binary file to work as a command.

POD:
It is a smallest unit of deployment in K8's.
It is a group of containers.
Pods are ephemeral (short living objects)
Mostly we can use single container inside a pod but if we required, we can create multiple containers inside a same pod.
when we create a pod, containers inside pods can share the same network namespace, and can share the same storage volumes .
While creating pod, we must specify the image, along with any necessary configuration and resource limits.
K8's cannot communicate with containers, they can communicate with only pods.
We can create this pod in two ways,
1. Imperative(command)
2. Declarative (Manifest file)

IMPERATIVE:

```
kubectl run pod1 --image nginx
kubectl get pods/pod/po
kubectl get pod -o wide
kubectl describe pod pod1
```

kubectl delete pod pod1

DECRALATIVE: by using file called manifest file

MANDATORY FEILDS: without these feilds we cant create manifest

apiVersion:
kind:
metadata:
spec:

vim pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
    - image: vinodvanama/paytmtrain:latest
      name: cont1
```

execution:
kubectl create -f pod.yml
kubectl get pods/pod/po
kubectl get pod -o wide
kubectl describe pod pod1
kubectl delete -f raham.yml

DRAWBACK: once pod is deleted we can't retrieve the pod.

TIP: GENERATING MANIFEST FILE FOR EXISTING K8S OBJECTS
kubectl get po
kubectl get po pod1 -o yaml
kubectl get po pod1 -o yaml > abc.yml

TROBLESHOOT:
ERROR: ErrImagePull
SOL : DELETE THE POD AND CREATE WITH PROPER IMAGE NAME

==========

==
MINIKUBE ALTERNATIVES:
KUBEADM, KOPS, KIND, RANCHER, EKS, AKS, GKE, KIND9

KOPS:
INFRASTRUCTURE: Resources used to run our application on cloud.
EX: Ec2, VPC, ALB, ASG----------

Minikube -- > single node cluster
All the pods on single node
if that node got deleted then all pods will be gone.

KOPS:
kOps, also known as Kubernetes operations.
it is an free, open-source and Platform Independent tool.
used to create, destroy, upgrade, and maintain a highly available, production-grade Kubernetes cluster.
Depending on the requirement, kOps can also provide cloud infrastructure.
AWS (Amazon Web Services) and GCE (Google Cloud Platform) are currently officially supported, with DigitalOcean, Hetzner and OpenStack in beta support, and Azure in alpha.

ADVANTAGES:
- Automates the provisioning of AWS and GCE Kubernetes clusters
- Deploys highly available Kubernetes masters
- Supports rolling cluster updates
- Autocompletion of commands in the command line
- Generates Terraform and CloudFormation configurations
- Manages cluster add-ons.
- Supports state-sync model for dry-runs and automatic idempotency
- Creates instance groups to support heterogeneous clusters

ALTERNATIVES:
Amazon EKS , MINIKUBE, KUBEADM, RANCHER, TERRAFORM.

STEP-1: GIVING PERMISSIONS (IAM)

KOps Is a third party tool if it want to create infrastructure on aws
aws need to give permission for it so we can use IAM user to allocate permission for the kops tool

IAM -- > USER -- > CREATE USER -- > NAME: KOPS -- > Attach Polocies Directly -- > AdministratorAccess -- > NEXT -- > CREATE USER
USER -- > SECURTITY CREDENTIALS -- > CREATE ACCESS KEYS -- > CLI -- > CHECKBOX -- > CREATE ACCESS KEYS -- > DOWNLOAD

aws configure (run this command on server)

SETP-3: INSTALL KUBECTL AND KOPS

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
wget https://github.com/kubernetes/kops/releases/download/v1.25.0/kops-linux-amd64

chmod +x kops-linux-amd64 kubectl
mv kubectl /usr/local/bin/kubectl
mv kops-linux-amd64 /usr/local/bin/kops

#IF COMMANDS ARE NOT WORKING FOLLOW THIS
vim .bashrc

source .bashrc

SETP-4: CREATING BUCKET

aws s3api create-bucket --bucket 1030amrahamsdevopsbatchnewkopsclass00709.k8s.local --region us-east-1
aws s3api put-bucket-versioning --bucket 1030amrahamsdevopsbatchnewkopsclass00709.k8s.local --region us-east-1 --versioning-configuration Status=Enabled
export KOPS_STATE_STORE=s3://1030amrahamsdevopsbatchnewkopsclass00709.k8s.local

SETP-4: CREATING THE CLUSTER
kops create cluster --name rahamss.k8s.local --zones us-east-1a --control-plane-image ami-0360c520857e3138f --control-plane-count=1 --control-plane-size t2.large --image ami-0360c520857e3138f --node-count=2 --node-size t2.medium

kops update cluster --name rahams.k8s.local --yes --admin

Suggestions:
 * list clusters with: kops get cluster
 * edit this cluster with: kops edit cluster rahams.k8s.local
 * edit your node instance group: kops edit ig --name=rahams.k8s.local nodes-us-east-1a

* edit your master instance group: kops edit ig --name=rahams.k8s.local master-us-east-1a

ERROR:
Error: State Store: Required value: Please set the --state flag or exp
SOL: export KOPS_STATE_STORE=s3://rahamsdevopsbatchmay14102024am.k8s.local

NOTE: TO GET INFO OF DEPLOYMENT TO FILE
kubectl create deployment movies --image=name --replicas=4 --dry-run=client -o yaml > movies-deployment.yml

ADMIN ACTIVITIES:
To scale the worker nodes:
kops edit ig --name=rahams.k8s.local nodes-us-east-1a
kops update cluster --name rahams.k8s.local --yes --admin
kops rolling-update cluster --yes

ADMIN ACTIVITIES:
kops edit ig --name=rahams.k8s.local master-us-east-1a
kops update cluster --name rahams.k8s.local --yes
kops rolling-update cluster

SCALING IN GUI: ASG -- > MASTER/WORKER NODE -- > EDIT -- > DESIRED: 4 -- > SAVE

NOTE: In real time we use five node cluster two master nodes and three worker nodes.

NOTE: its My humble request for all of you not to delete the cluster manually and do not delete any server use the below command to delete the cluster.

TO DELETE: kops delete cluster --name rahams.k8s.local --yes
==========
==========

SERVICE:
Used to expose Pods to the users.
If Front end Pod need to communicate with backend pod we use service for it.

COMMAND: kubectl api-resources

TYPES:
CLUSTER-IP
NODE PORT

LOAD BALANCER

COMPONENTS OF SERVICES:
Selector: To select pods
Port: Associated to Service
TargetPort: Associated to Pod
nodePort: Associated to Node
Type: Type of the service

TYPES:
1. CLUSTERIP: It will work inside the cluster.
it will not expose to outer world.
Ideal for internal communication within a cluster.
Suitable for backend services like databases, caches, or internal APIs
Preferred in Development and Testing Envs.
Not accessible from outside the cluster.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: movies
  name: movies-deploy
spec:
  replicas: 10
  selector:
    matchLabels:
      app: movies
  template:
    metadata:
      labels:
        app: movies
    spec:
      containers:
      - name: cont1
        image: rahamshaik/moviespaytm:latest
        ports:
          - containerPort: 80
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: service1
spec:
  type: ClusterIP
  selector:
    app: movies
  ports:
    - port: 80
```

DRAWBACK:
We cannot use app outside.

2. NODEPORT:

Node Port Range= 30000 - 32767
NodePort expose Pod on a static port on each node.
if Port is not given it will assign automatically.
if target Port is not Given it takes Port value.
NodePort services are typically used for smaller applications with a lower traffic volume.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: movies
  name: movies-deploy
spec:
  replicas: 10
  selector:
    matchLabels:
      app: movies
  template:
    metadata:
      labels:
        app: movies
    spec:
      containers:
      - name: cont1
        image: rahamshaik/moviespaytm:latest
        ports:
          - containerPort: 80
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: service1
spec:
  type: NodePort
  selector:
    app: movies
  ports:
    - port: 80
      nodePort: 31111
```

NOTE: UPDATE THE SG (REMOVE OLD TRAFFIC AND GIVE ALL TRAFFIC)
DRAWBACK:
EXPOSING PUBLIC-IP & PORT
PORT RESTRICTION.

3. LOADBALACER:
In LoadBalaner we can expose application externally with the help of Cloud Provider LoadBalancer.
it is used when an application needs to handle high traffic loads and requires automatic scaling and load balancing capabilities.
After the LoadBalancer service is created, the cloud provider will created the Load Balancer.
REPLACE NodePort with LoadBalancer

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: swiggy
  name: swiggy-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: swiggy
  template:
    metadata:
      labels:
        app: swiggy
```

```
    spec:
      containers:
      - name: cont1
        image: rahamshaik/trainservice:latest
        ports:
          - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: abc
spec:
  type: LoadBalancer
  selector:
    app: swiggy
  ports:
    - port: 80
```

==========

NAMESPACE:

It is used to divide the cluster to multiple teams on real time.
Used for isolating groups of resources within cluster.
By Default we work on Default Name space in K8's.
We create NameSpaces when we work for Prod level Workloads.
If we create pod on one namespace it cant be access by other namespaces.
We can set access limits by RBAC and Limits of Cpu, RAM by Quotas.
It is applicable only for namespaced objects (e.g. Deployments, Services, etc.)
It wont apply for cluster-wide objects (e.g. StorageClass, Nodes, PV).

CLUSTER: HOUSE
NAMESPACES: ROOM
TEAM MATES: FAMILY MEM

Each namespace is isolated.
if your are room-1 are you able to see room-2.
If dev team create a pod on dev ns testing team cant able to access it.

we cant access the objects from one namespace to another namespace By Default.

TYPES:

default : Is the default namespace, all objects will create here only
kube-node-lease : it will store object which is taken from one node to another.
kube-public : all the public objects will store here.
kube-system : default k8s will create some objects, those are storing on this ns.

NOTE: Every component of Kubernetes cluster is going to create in the form of pod
And all these pods are going to store on kUBE-SYSTEM ns.

kubectl get pod -n kube-system : to list all pods in kube-system namespace
kubectl get pod -n default : to list all pods in default namespace
kubectl get pod -n kube-public : to list all pods in kube-public namespace
kubectl get po -A : to list all pods in all namespaces
kubectl get po --all-namespaces

kubectl create ns dev : to create namespace
kubectl config set-context --current --namespace=dev : to switch to the namespace
kubectl config view : to see current namespace
kubectl run dev1 --image nginx
kubectl run dev2 --image nginx
kubectl run dev3 --image nginx
kubectl get po
kubectl create ns test : to create namespace
kubectl config set-context --current --namespace=test : to switch to the namespace
kubectl config view --minify | grep namespace : to see current namespace
kubectl get po -n dev
kubectl delete pod dev1 -n dev
kubectl delete ns dev : to delete namespace
kubectl delete pod --all: to delete all pods

NOTE: BY DEFAULT K8S NAMESPACE WILL PROVIDE ISOLATION BUT NOT RESTRICTION.
TO RESTRICT THE USER TO ACCESS A NAMESPACE IN REAL TIME WE USE RBAC.
WE CREATE USER, WE GIVE ROLES AND ATTACH ROLE.

alias switch="kubectl config set-context --current"
switch --namespace=default
switch --namespace=dev

==========

DAY-5: MERTIC SERVER, DAEMONSET, NODE SCALING, POD SCALING, KUBECOLOR

METRIC SERVER INSTALLATION:

kubectl apply -f
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/high-availability-1.21+.yaml

kubectl top nodes
kubectl top pods

Metrics Server offers:

METRIC SERVER: Metrics of pods/nodes (cpu, ram, disk ---)
its not a built in feature in k8s (HEAPSTER)
we install metric server in all the nodes
it creates like a pod -- > (DaemonSet)
Req: 1 millcore , 2 Mb Ram
it collects metrics for every 15 sec
Scalable support up to 5,000 node clusters.

You can use Metrics Server for:
CPU/Memory based horizontal autoscaling (Horizontal Autoscaling)
Automatically adjusting/suggesting resources needed by containers (Vertical Autoscaling)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movies
  labels:
    app: movies
spec:
  replicas: 3
  selector:
    matchLabels:
      app: movies
  template:
```

```
    metadata:
      labels:
        app: movies
    spec:
      containers:
        - name: cont1
          image: yashuyadav6339/movies:latest
          resources:
            requests:
              cpu: "100m"
              memory: "100Mi"
```

```
kubectl apply -f hpa.yml
kubectl get all
kubectl get deploy
kubectl autoscale deployment movies --cpu-percent=20 --min=3 --max=10
kubectl get hpa
kubectl desribe hpa movies
kubectl get al1

open second termina and give
kubectl get po --watch

come to first terminal and go inside pod
kubectl exec mydeploy-6bd88977d5-7s6t8 -it -- /bin/bash

apt update -y
apt install stress -y
stress

check terminal two to see live pods

KUBECOLOR: USED TO ADD COLOR TO TERMINAL

wget
https://github.com/hidetatz/kubecolor/releases/download/v0.0.25/kubecolor_0.0.25_Linux_x86_
64.tar.gz
tar -zxvf kubecolor_0.0.25_Linux_x86_64.tar.gz
./kubecolor
chmod +x kubecolor
mv kubecolor /usr/local/bin/
kubecolor get po
```

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

CSI:
CSI stands for container storage interface.
It provides an interface for integrating storage systems with Kubernetes
Before CSI, Kubernetes had a limited set of built-in storage options.
CSI was Developed to Support Multiple Storage Options in K8s.
After CSI any one can Write Drivers for Own Storage.
What ever K8s instructed our Storage Drivers need to Do it.
Below storage Solutions we can use for k8s from Cloud Services.

VOLUMES IN K8S:
1. EMPTYDIR:
Its a empty Volume and exists as long as the pod exists.
Data in an emptyDir volume is lost when the pod is deleted.

2. HostPath:
its a volume type where data is store in particular path of node.
If the pod is deleted, the volume will remain on the host.
but data will store on single node only

3. Persistent Volume & Persistent Volume Claim:
PV represents an actual storage resource in the cluster.
PVC is a request for storage by a user or a pod.

PV & PVC:
PV its a cluster wide Group of volumes created by Admin.
User Can select volumes from the Group as per Requirements.
If user want to use Volume he need to create Persistent Volume Claim.
Once PVC is Created k8s will Bind PV based on request and Properties.
To Bound Specific PVC to PV use Labels and Selector.
Each PVC is Bound to only One PV, even if we have Additional storage.
If we create PVC and No PV is available so PVC will be on Pending state.

RESTRICTIONS:
1. Instances must be on same az as the ebs
2. EBS supports only a sinlge EC2 instance mounting

pv.yml

apiVersion: v1

```
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: vol-04e624118b7aa6835
    fsType: ext4
```

pvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

dep.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pvdeploy
spec:
  replicas: 1
  selector:
    matchLabels:
     app: swiggy
  template:
    metadata:
      labels:
        app: swiggy
    spec:
```

```
    containers:
    - name: raham
      image: ubuntu:rolling
      command: ["/bin/bash", "-c", "sleep 10000"]
      volumeMounts:
      - name: my-pv
        mountPath: "/tmp/persistent"
    volumes:
      - name: my-pv
        persistentVolumeClaim:
          claimName: my-pvc
```

```
kubectl exec pvdeploy-86c99cf54d-d8rj4 -it -- /bin/bash
cd /tmp/persistent/
touch file{1..5}
ls
vim file1
exit
```

now delete the pod and new pod will created then in that pod you will see the same content.

RECLAIM POLICY:
Reclaim Policy determines what happens to a PV once PVC Deleted.
RETAIN: PV will Available, but not Reuseable by any PVC. [Devadas-Paru]
RECYCLE: PV will be available But Data will be deleted. [Bharath-telidu]
DELETE: PV will be Deleted Automatically. [Ramcharan-Kajol]

ACCESS MODES:
RWO: single node mount the volume as read-write at a time.
ROX: multiple nodes mount the volume as read-only simultaneously.
RWX: multiple nodes mount the volume as read-write simultaneously.
RWOP: volume to be mounted as read-write by a single pod.

STORAGE CLASSES:
In PV & PVC storage is Created and increased Manually.
Storage Classses will automatically Provisions the storage to Pods.
This is Called as Dynamic Provisioning.
By using SC no need to Maintain PV.

vim sc.yml

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
```

```
kubectl create -f sc.yml
kubectl get sc
```

EXTRACTING MANIFEST FROM EXSTING OBJECTS:

```
kubectl run pod1 --mage nginx
kubectl get po pod1 -o yaml > pod.yml
```

==========

ENV VARS:
In k8s we can set env vars using evn feild.
env is array means we can set multiple values.
it has key-value fromat.
to set env vars we use configmaps and secrets.

CONFIG MAPS:
it is used to pass configuration data to pods in key-value fromat.
we pod is created inject configmap, so data is used as env varibles.
First create configmap and later inject to pod.
But the data should be non confidential data ()
But it does not provider security and encryption.
Limit of config map data in only 1 MB (for more data use volumes).
To pass values from cli use literal.

COMMAND:

```
kubectl create cm dbcreds --from-literal=password=raham123

kubectl get cm
kubectl describe cm dbcreds
kubectl describe cm dbcreds


kubectl create cm dbcreds --from-literal=user=raham --from-literal=password=test123
--dry-run=client -o yaml > cm.yml
kubectl create -f cm.yml
```

kubectl get cm

CODES:

cat config.yml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dbcreds
data:
  user: "raham"
  password: "test123"
```

cat pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: env-configmap
spec:
  containers:
    - name: app
      image: nginx
      envFrom:
        - configMapRef:
            name: dbcreds
```

kubectl get po
kubectl exec -it env-configmap -- /bin/bash

root@env-configmap:/# echo $user
raham
root@env-configmap:/# echo $password
test123

SECERETS:
Used to store sensitive information like passwords, keys ---
it wont encrypt data but it will encode them in base64.

when pod is created inject Secret.
Dont push to github becuse they are not encrypted but encode.
it wont encrypt in etcd also.
Data feild indicates number of secrets in secret.
USE BELOW COMMAND TO ENCODE: echo -n "raham123" | base64
SE BELOW COMMAND TO DECODE: echo -n "raham123" | base64 -d

COMMAND:
kubectl create secret generic dbcreds --from-literal=user=raham
--from-literal=password=raham123
kubectl get secret raham -o yaml
kubectl describe secret dbcreds
kubectl delete secret dbcreds

kubectl create secret generic dbcreds --from-literal=user=raham
--from-literal=password=test123 --dry-run=client -o yaml > secrets.yml

kubectl create -f secrets.yml

cat secret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: dbcreds
data:
  user: "cmFoYW0="
  password: "dGVzdDEyMw=="
```

cat pod1.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: env-secret
spec:
  containers:
    - name: app
      image: nginx
      envFrom:
        - secretRef:
```

```
      name: dbcreds
```

kubectl create -f pod1.yml

kubectl get po
kubectl exec -it env-secret -- /bin/bash

root@env-configmap:/# echo $user
raham
root@env-configmap:/# echo $password
test123

NOTE:
Dont push to github becuse they are not encrypted but encode.
it wont encrypt in etcd also.
Enable Encryption at Rest for Secrets so they stored as encrypted in ETCD
secret is not written to disk storage, Kubelet stores them to tmpfs.
Once the Pod that depends on the secret is deleted, kubelet will delete its local copy of the secret data as well.

kubectl create deploy swiggydb --image=mariadb
kubectl get pods
kubectl logs swiggydb-5d49dc56-cbbqk

It is crashed why because we havent specified the password for it

kubectl set env deploy swiggydb MYSQL_ROOT_PASSWORD=Raham123
kubectl get pods
now it will be on running state
kubectl delete deploy swiggydb

MULTI-CONTAINER POD:
It will have more than one container in a pod.
each container have different purpose to work on.
They created and destroyed together and share same n/w and volume.
If any of them fails, the POD restarts.

SIDE CAR:

It creates a helper container to main container.
main container will have the app and helper container Helps main container.

Init Container:
it initialize the first work and exits later.

Ambassador Design Pattern:
used to connect containers with the outside world

Adapter Design Pattern:
enable communication and coordination between containers.

CODE:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: cont1
    image: nginx:1.14.2
  - name: cont2
    image: nginx:1.18
```

CHECKING LOGS FOR MULTI CONTAINER PODS:

```
kubectl logs nginx -c cont1
kubectl logs nginx -c cont2
```

==========

STEP-1: CREATE 2 SERVER (1 MASTER & 1 WORKER) (STEP 1 TO STEP 6 COMMON FOR ALL NODES)

STEP-2: UPDATE
sudo apt update -y && sudo apt upgrade -y

STEP-3: Then import GPG key and configure APT repository.

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.33/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.33/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

STEP-4: INSTALL KUBECTL, KUBELET, KUBEADM
(KUBECTL : to communicate with k8s)
(KUBELET : to create pods)
(KUBEADM : to create cluster)

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
kubeadm version

STEP-5: INSTALL CONTAINER RUNTIME (to create containers)

sudo apt-get update
sudo apt install containerd -y

STEP-6:

Cgroup drivers (short for Control Group drivers) are components of the Linux kernel that manage how system resources (like CPU, memory, disk I/O, etc.) are allocated to containers/process.

mkdir -p /etc/containerd
containerd config default
containerd config default >> /etc/containerd/config.toml
change line 139 false to true
sed -i '139s/false/true/' /etc/containerd/config.toml
systemctl restart containerd.service
systemctl status containerd.service
echo "1" >> /proc/sys/net/ipv4/ip_forward (0=no connetion to servers) (1=connect to other servers)

STEP-7: CREATE CLUSTER WITH KUBEADM (RUN COMMAND ON MASTER NODE)

kubeadm init --apiserver-advertise-address 172.31.28.87 --pod-network-cidr "10.0.0.0/16" --upload-certs --ignore-preflight-errors=all

```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

kubectl get no

GIVE THIS TOKEN ON ALL WORKER NODES:

kubeadm join 172.31.24.115:6443 --token shns2s.akxuwrq6yvo43u6o \
        --discovery-token-ca-cert-hash
sha256:647e837c76c8c2eeac4ddd82ac9e2c8ea93679d81e44d697c65fa4b84d8a2da0
--ignore-preflight-errors=all

kubectl get no

STEP-8: DEPLOY A NETWORK PLUGIN (WEAVENET)
GO TO ADDONS PAGE IN K8S DOCS AND FIND IT

kubectl apply -f https://reweave.azurewebsites.net/k8s/v1.29/net.yaml


STEP-9: COPY PASTE TOKEN ON WORKER NODE
kubeadm join 172.31.27.84:6443 --token u97a1o.2e46o337yico1pov \
        --discovery-token-ca-cert-hash
sha256:2206382552dff3b58277a4010ad96657d3b2bba4ceae221e0f048fcbeae5c310

STEP-10: RUN A POD
kubectl run pod1 --image nginx
kubectl get po


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

CLUSTER UPGRADING:
REF:
https://v1-31.docs.kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

```
sudo apt update
sudo apt-cache madison kubeadm


sudo apt-mark unhold kubeadm && \
sudo apt-get update && sudo apt-get install -y kubeadm='1.33.5-1.1' && \
sudo apt-mark hold kubeadm


kubeadm version
kubeadm upgrade plan --ignore-preflight-errors=CreateJob
kubeadm upgrade apply v1.31.4  --ignore-preflight-errors=CreateJob

kubectl get no

NOW UPDATE KUBELETE:
STEP-1: MOVE THE EXISTING PODS
kubectl drain <node-to-drain> --ignore-daemonsets


sudo apt-mark unhold kubelet kubectl && \
sudo apt-get update && sudo apt-get install -y kubelet='1.31.4-1.1' kubectl='1.31.4-1.1' && \
sudo apt-mark hold kubelet kubectl

sudo systemctl daemon-reload
sudo systemctl restart kubelet

kubectl uncordon <node-to-uncordon>


================================================================================
=======
REASONS TO UPGRADE:
1. PERFOMANCE IMPROVE
2. BUGS WILL BE REMOVED
3. NEW FEATURES

DRAIN:  marking node as unschedulable and move the pods from one node to another
UNCORDON: Marking the node as to schedule the new pods.

TAKE AWAYS:
```

1. WHILE UPGRADING K8S VERSION WE CAN UPGRADE ONE VERSION AT A TIME
1.30 ---- > 1.34 (NOT POSSIBLE)
1.30 ---- > 1.31 --- > 1.32 --- > 1.33 --- > 1.34 (POSSIBLE)

2. ALL K8S COMPONENTS (KUBEADM, KUBECTL, KUBELET) NEED NOT HAVE SAME VERSION

3. TYPES OF STRATAGIES (ALL AT ONCE, ROLLING UPDATE, BLUE-GREEN)

KEPS:
STEERING COMMUNITY
IMPLEMNET
RELEASE
FEATURE

==========

KOPS	KUBEADM
SERVER	SERVER
NO KEYS	KEYS
NO REP	REPO IS MANDATORY
NO KUBELET	KUBELET IS REQ
NO CONTAINER RUNTIME	CONTAINER RUN TIME IS REQ
NO CGROUP	CGROUP
NO IPV4 FORWARD	IPV4 FORWARD
NO TOKEN	TOKEN
NO CNI	CNI

==========

BEST EXAMPLE: PET DOG LOYALITY

SCHEDULER:
Scheduler main work is to select the node to place the pod.
while a pod is created it will select a node based on pod requirment.

if scheduler cant find node then pod will be on pending state.
Initially we can place a pod on node by writing node name on manifest.
scheduler runs as a pod if it stop/delete scheduling wont happen.

TAINTS & TOLERATIONS:
By Default scheduler can place any pod on any Node.
if we want to place Pod-1 on Node-1 then we use Taints & Tolerations.
First we taint the Node-01 so that no pod will be placed on any node.
Then we can tolerate the Particular pod what we want to place on Node-01.
We set Taint for Nodes & Tolerations for Pods.
EX: Kubectl taint node01 app=paytm:NoSchedule
NOTE: it wont guarantee that pod-1 definitely place on Node-01 all the time
By Default k8s master is tainted thats why no pod is placed on master.
To schedule pod on a master add - at end by running taint command.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movies
  labels:
    app: paytm
spec:
  replicas: 4
  selector:
    matchLabels:
      app: paytm
  template:
    metadata:
      labels:
        app: paytm
    spec:
      containers:
        - name: cont1
          image: nginx
      tolerations:
        - key: "dog"
          operator: "Equal"
          value: "pitbull"
```

effect: "NoSchedule"

kubectl taint node i-04a90983debd1c8ec dog=pitbull:NoSchedule

NODE SELECTOR:
Kubernetes can select nodes to place pods based on the labels of a node.
First we can label the node & later same labels we can add to pod manifest.
Now all pods will be set on same node.
NOTE: It can place pod on node with single label, But if we want to place pod on a node with multiple lables this wont work.
kubectl label node node_id key=value

kubectl label nodes node-1 meat=mutton

NOTE: UNTAINT THE NODE
kubectl taint node node_id app=swiggy:NoSchedule-

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movies
  labels:
    app: paytm
spec:
  replicas: 10
  selector:
    matchLabels:
      app: paytm
  template:
    metadata:
      labels:
        app: paytm
    spec:
      containers:
        - name: cont1
          image: nginx
      nodeSelector:
        meat: "mutton"
```

NOTE: IF NODE IS HAVING ANY TAINTS THE PODS WILL NOT SCHEDULE
kubectl taint node i-06c09c90c2768ceba app=swiggy:NoSchedule-

kubcetl create -f dep.yml

TO REMOVE LABEL: kubectl label node i-0b17bfc7885c1c9f3 hero=diwakar --overwrite

NODE AFFINITY:
its a feature in Kubernetes that facilitates us to specify the rules for scheduling the pod based on the node labels.
we can use operators with multiple values too.
OPERATORS: [In, NotIn, Exists, DoesNotExist, Gt, Lt]
TYPES:
requiredDuringSchedulingIgnoredDuringExecution: While scheduling matching the label is mandatory, after placing pod optional.

PreferredDuringSchedulingIgnoredDuringExecution: While scheduling matching the label is not mandatory, after placing pod optional.

kubectl label node node-1-id meat=mutton (node-1)
kubectl label node node-2-id meat=fish (node-2)

GIVE LABELS AND TRY THIS

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movies
spec:
  replicas: 10
  selector:
    matchLabels:
      app: paytm
  template:
    metadata:
      labels:
        app: paytm
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      affinity:
```

```
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: meat
              operator: In
              values:
              - chicken
              - mutton
```

POD AFFINITY   : PLACES POD-2 ON WHERE POD-1 IS PLACED
POD ANTI AFFINIT: PLACES POD-2 ON WHERE POD-1 IS NOT PLACED

RESOURCE QUOTAS:

Scheduler checks the CPU, RAM of node before we Place a Pod
If Node CPU, RAM is not matching for Pod, it select another Node in Cluster.
If any node is not having Sufficient CPU, RAM then pod will go Pending state.
Without limits a container in pod can consume all CPU, RAM of Node.
So we need to set limits in Real time to restrict the Containers.
Note: Limits are set in container level, if we have 2 containers in a pod then set values individually.
A container cant use more Cpu's than Specified limit, But Not Memory.
Pod will be terminated when it use more than limits and we get OOM error.
To set limits and Request we use Quotas and Limit Ranges in Real time.
LimitRange applies to individual containers or pods.
ResourceQuota applies to the entire namespace.

cat dev-quota.yml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: dev-quota
  namespace: dev
spec:
  hard:
    pods: "10"
    limits.cpu: "1"
```

```
    limits.memory: 1Gi
```

cat dep.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: movies
  labels:
    app: movies
spec:
  replicas: 7
  selector:
    matchLabels:
      app: movies
  template:
    metadata:
      labels:
        app: movies
    spec:
      containers:
        - name: cont1
          image: yashuyadav6339/movies:latest
          resources:
            limits:
              cpu: "0.1"
              memory: 100Mi
```

```
kubectl get events
kubectl get events --field-selector type=Warning
kubectl get events --field-selector type=Warning --sort-by='.lastTimestamp'
```

STATIC POD:

Kubelet Create Pods without API Server is called as Static Pods.
we put Pod manifest in /etc/kubernetes/manifests [on worker node]
Kubelet Read files and Create Pods and Manages it.
If we made any change to Manifest File Kubelet Recreates the Pod.
If we remove file then pod will be deleted automatically.

We can change path of location in kubelet.service file.
use [--config=abc.yml] and in abc.yml [staticPodPath: /etc/xyz]
Even if we create Pod from Kubelet API Server knows about that Pod.
Because API will have a Read only Mirror of a Pod but cant edit and delete.
Kubelet can create static and dynamic pods at a same time.
USE CASES: used to Deploy control-plane components as a pods in Master.
NOTE: RS, DEPLOYMENT cant create because they required Controllers.

PRACTICAL:
cd /etc/kubernetes/manifests [Run this on Worker node]
Create a manifest file and run it.
Go to Kops server and Check

HOW TO CHANGE STATIC POD PATH:
vim /var/lib/kubelet/config.yaml
staticPodPath: /root
systemctl restart kubelet.service

kubectl run pod2 --image nginx --dry-run=client -o yaml > pod2.yml
kubectl get po

==========

FILE TO CHECK FOR USER, CLUSTER, NAMESPACE DETAILS: /root/.kube/config
RBAC:

role : set of permissions for one ns
role binding: adding users to role
these will work on single namespace

cluster role: set of permissions for entire Cluster
cluster role binding: adding users to cluster role
these will work on all namespaces

when we run kubectl get po k8s api server will authenticate and check authorization
authentication: permission to login
authorization: permission to work on resources

To authenticate API requests, k8s uses the following options:
client certificates,

bearer tokens,
authenticating proxy,
or HTTP basic auth.

Kubernetes doesn't have an API for creating users.
Though, it can authenticate and authorize users.

We will choose the client certificates as it is the simplest among the four options.

why certs needed on k8s: for authentication purpose.
certs will have users & keys for login.

LINK: https://kubernetes.io/docs/tasks/administer-cluster/certificates/

1. Create a client certificate
We'll be creating a key and certificate sign request (CSR) needed to create the certificate. Let's create a directory where to save the certificates. I'll call it cert:

mkdir dev1 && cd dev1

1. Generate a key using OpenSSL: (admission)
openssl genrsa -out dev1.key 2048

2. Generate a Client Sign Request (CSR) : (learning)

openssl req -new -key dev1.key -out dev1.csr -subj "/CN=dev1/O=group1"
ls ~/.minikube/

3. Generate the certificate (CRT): (CCC)

openssl x509 -req -in dev1.csr -CA ~/.minikube/ca.crt -CAkey ~/.minikube/ca.key
-CAcreateserial -out dev1.crt -days 500

Now, that we have the .key and the .crt, we can create a user.

Create a user
1. Set a user entry in kubeconfig
kubectl config set-credentials dev1 --client-certificate=dev1.crt --client-key=dev1.key

2.Set a context entry in kubeconfig

kubectl config set-context dev1-user --cluster=minikube --user=dev1
kubectl config view

3.Switching to the created user
kubectl config use-context dev1-context
$ kubectl config current-context # check the current context
dev1-context
But, now, dev1 doesn’t have any access privileges to the cluster. For that we’ll have access denied if we try to create any resource:

$ kubectl create namespace ns-test
kubectl get po
Error from server (Forbidden): namespaces is forbidden: User "dev1" cannot create resource "namespaces" in API group "" at the cluster scope

3. Grant access to the user
To give access to manage k8s resources to dev1, we need to create a Role and a BindingRole.

kubectl config use-context minikube
kubectl create ns dev

3.1. Create a Role

vim role.yml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-role
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "create", "watch", "list", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-rolebind
  namespace: dev
subjects:
```

```
- kind: User
  name: dev1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: dev-role
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl apply -f role.yml

kubectl config use-context dev1-context
kubectl config view | grep -i namespace
kubectl config set-context --current --namespace=dev

kubectl run pod1 --image nginx : work
kubectl get po : work
kubectl delete po pod1 : error -- > no permissions

TO UPDATE PERMISSION GO TO MINIKUBE USER
kubectl config use-context minikube

ADD THIS IN FILE:
verbs: ["get", "create", "watch", "list", "delete"]
kubectl apply -f role.yml
kubectl config use-context dev1-context
kubectl delete po pod1 : work


kubectl create deploy raham --image nginx --replicas 2
kubectl create cm raham --from-literal=user=raham

NOW UPATE ROLE WITH DEPLOYMEN AND CM

kubectl config use-context minikube

ADD THIS IN FILE:
- apiGroups: ["*"] #
resources: ["pods", "deployments", "configmaps"]
kubectl apply -f role.yml
kubectl config use-context dev1-context
```

BY DEFAULT API VERSION IN ROLE IS: v1 ("")
SO TO WORK WITH ALL RESOURCES PUT *

CLUSTER ROLE: GIVES PERMISSION TO WORK WITH ENTIRE CLUSTER

vim clusterrole.yml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dev-role
rules:
- apiGroups: ["*"] # "" indicates the core API group
  resources: ["pods", "deployments", "configmaps", "namespaces"]
  verbs: ["get", "create", "watch", "list", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dev-rolebind
subjects:
- kind: User
  name: dev1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: dev-role
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl create ns test
kubectl config use-context dev1-context
kubectl config view | grep -i namespace
kubectl config set-context --current --namespace=dev
```

SOME OBJECT IN K8S CREATE IN CLUSTER LEVEL SO WE NEED TO CREATE CLUSTERROLE AND CLUSTER ROLEBIND
PV, PVC, NS

CLI: BEST WAY TO WORK

kubectl create role -h
kubectl create role test --verb=create,delete,list --resource=pods -n test

kubectl create rolebinding -h
kubectl create rolebinding testbinding --role=test --user=dev1 -n test

EXAMPLE: kubectl get deploy raham -n test -o yaml > dep.yml

CLUTSTER-ROLE: IT WORKS WITH MULTIPLE NAMESPACES
kubectl create clusterrole dev-clusterrole --verb=create,list --resource=pods --dry-run=client -o yaml > clusterrole.yml
kubectl create -f clusterrole.yml

kubectl create clusterrolebinding dev-clusterrolebinding --clusterrole=dev-clusterrole --user=dev1 --dry-run=client -o yaml > dev-clusterrolebinding.yml
kubectl create -f dev-clusterrolebinding.yml

VERIFY:
kubectl config use-context dev1-context
kubectl get po
kubectl get po

KUBECTL AUTH: TO CHECK PERMISSIONS:

kubectl auth can-i get pods --user=dev1 -n dev
kubectl auth can-i get pv --user=dev1
kubectl auth can-i get deploy --user=dev1