Docker Assignments

1) **Difference between Virtualization and Containerization.**

Virtualization creates VMs. Virtual version of independent machines or computer resources created on basis of predefined configuration. Ex: separate images of Ubuntu, RHEL, AWS Linux etc. Performed using hypervisor.

Containerization is a lightweight process. Application is encapsulated within a container providing a minimalistic operating system. So, application uses OS of container rather than reaching out to VM/machine where container is hosted

2) **Difference between Docker attach and Exec.**

Docker Attach Command - The docker attach command allows you to attach to a running container using the container's ID or name, either to view its ongoing output or to control it interactively. The command docker attach is for attaching to the existing process. It does not start any new process.

docker attach ubuntu (docker attach command using container name)

docker attach be2848539d76 (docker attach command using container id)

Docker Exec Command - The docker exec is specifically for running new things in an already started container. The docker exec command runs a new command in a running container, be it starting an interactive shell in a running container, or to execute arbitrary commands in the container environment.

docker exec ubuntu sh -c "echo hello && echo world"
(It runs the command by running a one-liner shell script by prefixing them with the executable *sh -c* with container name)

3) Difference between Docker Stop and Kill.

Docker Stop Command - The docker stop command stops the container gracefully and provides a safe way out.

docker stop be2848539d76 (docker stop command using containerId)

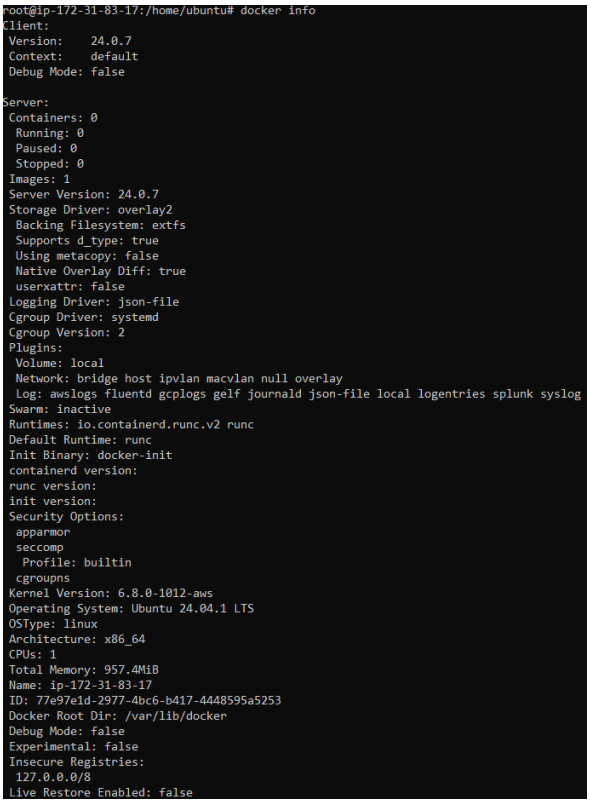
docker stop hello-world (docker stop command using the container name)

Docker Kill Command - The docker kill command terminates the entry point process abruptly. The docker kill command causes an unsafe exit.

docker kill be2848539d76 (docker kill command using containerId)

4) Docker Commands List

- docker info (Display system-wide information)



- docker --help (Get help with Docker)

```
root@ip-172-31-83-17:/home/ubuntu# docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Swarm Commands:
  swarm       Manage Swarm

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
```

- docker pull <image_name> (Pull an image from a Docker Hub)

```
root@ip-172-31-83-17:/home/ubuntu# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
e4fff0779e6d: Pull complete
2a0cb278fd9f: Pull complete
7045d6c32ae2: Pull complete
03de31afb035: Pull complete
0f17be8dcff2: Pull complete
14b7e5e8f394: Pull complete
23fa5a7b99a6: Pull complete
Digest: sha256:447a8665cc1dab95b1ca778e162215839ccbb9189104c79d7ec3a81e14577add
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
root@ip-172-31-83-17:/home/ubuntu#
```

- docker search <image_name> (Search Hub for an image)

```
root@ip-172-31-83-17:/home/ubuntu# docker search ubuntu
NAME                              DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
ubuntu                            Ubuntu is a Debian-based Linux operating sys…   17232     [OK]
ubuntu/chiselled-jre              [MOVED TO ubuntu/jre] Chiselled JRE: distrol…   3
ubuntu/jre                        Distroless Java runtime based on Ubuntu. Lon…   15
ubuntu/traefik                    Ubuntu ROCK for Traefik, a modern HTTP rever…   0
ubuntu/mimir                      Ubuntu ROCK for Mimir, a horizontally scalab…   0
ubuntu/dotnet-deps                Chiselled Ubuntu for self-contained .NET & A…   16
ubuntu/grafana-agent              Ubuntu ROCK for Grafana Agent, an open-sourc…   0
ubuntu/python                     A chiselled Ubuntu rock with the Python runt…   8
ubuntu/cassandra                  Cassandra, an open source NoSQL distributed …   2
ubuntu/dotnet-runtime             Chiselled Ubuntu runtime image for .NET apps…   20
ubuntu/dotnet-aspnet              Chiselled Ubuntu runtime image for ASP.NET a…   22
ubuntu/alertmanager               Ubuntu Rock for Alertmanager. It handles ale…   0
ubuntu/grafana                    Grafana, a feature rich metrics dashboard & …   11
ubuntu/zookeeper                  ZooKeeper maintains configuration informatio…   13
ubuntu/postgres                   PostgreSQL is an open source object-relation…   40
ubuntu/redis                      Redis, an open source key-value store. Long-…   22
ubuntu/kafka                      Apache Kafka, a distributed event streaming …   50
ubuntu/cortex                     Cortex provides storage for Prometheus. Long…   4
ubuntu/memcached                  Memcached, in-memory keyvalue store for smal…   5
ubuntu/prometheus-pushgateway     Allows ephemeral and batch jobs to expose th…   0
ubuntu/prometheus-alertmanager    Alertmanager handles client alerts from Prom…   9
ubuntu/bind9                      BIND 9 is a very flexible, full-featured DNS…   96
ubuntu/squid                      Squid is a caching proxy for the Web. Long-t…   94
ubuntu/apache2                    Apache, a secure & extensible open-source HT…   74
ubuntu/nginx                      Nginx, a high-performance reverse proxy & we…   116
root@ip-172-31-83-17:/home/ubuntu#
```

- docker images (List the images that have been downloaded)

```
root@ip-172-31-83-17:/home/ubuntu# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    5ef79149e0ec   13 days ago   188MB
```

- docker image rm <image_name> (Removing a Docker Image)

```
root@ip-172-31-83-17:/home/ubuntu# docker image rm nginx
Untagged: nginx:latest
Untagged: nginx@sha256:447a8665cc1dab95b1ca778e162215839ccbb9189104c79d7ec3a81e14577add
Deleted: sha256:5ef79149e0ec84a7a9f9284c3f91aa3c20608f8391f5445eabe92ef07dbda03c
Deleted: sha256:aa557aaf0b93b5b9af247beb198be89632444af2d52b37f2b67bdf2227194625
Deleted: sha256:85dee7d6fa4b70eaa6b43b5dede5255b6794fac35d2aa6a80f8590dad223c1a6
Deleted: sha256:9996f8429bfc2f1e10f561e17bdc0c66eb82e30088725fbdc634a774933ba066
Deleted: sha256:d861b1c34411369feaaaf4921018baa607e89f1c393b8ed950f676dd3349b9ff
Deleted: sha256:01b9716819d1d6863f886e1adcd8bb2bf936dcacb16e1750e873bfd8e1667014
Deleted: sha256:8d8c7099ecd8343e6700c1d348211ab159c0eda2a9d318c5cf600a8eb2588f5e
Deleted: sha256:9853575bc4f955c5892dd64187538a6cd02dba6968eba9201854876a7a257034
```

- docker run <image_name> (Pulls the image from docker hub and runs a container from the image)

```
root@ip-172-31-83-17:/home/ubuntu# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:53cc4d415d839c98be39331c948609b659ed725170ad2ca8eb36951288f81b75
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

- docker ps -a (Show all containers)

```
root@ip-172-31-83-17:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
3949f64b00ed   hello-world   "/hello"   2 minutes ago   Exited (0) 2 minutes ago             sad_albattani
root@ip-172-31-83-17:/home/ubuntu#
```

- docker version (Echoes Client's and Server's Version of Docker)

```
root@ip-172-31-83-17:/home/ubuntu# docker version
Client:
 Version:           24.0.7
 API version:       1.43
 Go version:        go1.22.2
 Git commit:        24.0.7-0ubuntu4.1
 Built:             Fri Aug  9 02:33:20 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          24.0.7
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.22.2
  Git commit:       24.0.7-0ubuntu4.1
  Built:            Fri Aug  9 02:33:20 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.12
  GitCommit:
 runc:
  Version:          1.1.12-0ubuntu3.1
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
root@ip-172-31-83-17:/home/ubuntu#
```

- docker start <container_Id> (Starts one or more stopped containers)

```
root@ip-172-31-83-17:/home/ubuntu# docker start 8949f64b00ed
8949f64b00ed
```

- docker ps -l (Show latest container)

```
root@ip-172-31-83-17:/home/ubuntu# docker ps -l
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                          PORTS     NAMES
8949f64b00ed   hello-world   "/hello"   12 minutes ago   Exited (0) About a minute ago             sad_albattani
```

5) Docker Swarm Commands

- docker swarm init –advertise-addr 192.168.10.1 (Initializing the Swarm)

```
root@m:/home/ubuntu# docker swarm init --advertise-addr 54.161.156.162
Swarm initialized: current node (8vpuygcs875423ahh35j0r1ih) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0x13gqyebm2qssi6fltfp0562zbcwq5wfyvakf7jim5c9jj9ut-d1mzbmqu0tff53xx4scp8iye1 54.1
61.156.162:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

- docker swarm join-token worker (Getting a Worker to Join the Swarm)

```
ubuntu@ip-172-31-25-32:~$ sudo su
root@w:/home/ubuntu# docker swarm join --token SWMTKN-1-0x13gqyebm2qssi6fltfp0562zbcwq5wfyvakf7jim5c9jj9ut-d1mzbmqu0tff5
3xx4scp8iye1 54.161.156.162:2377
This node joined a swarm as a worker.
```

- docker service ls (Listing Services)

```
root@m:/home/ubuntu# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
p3vvkvuvllof   web       replicated   3/3        nginx:latest   *:80->80/tcp
root@m:/home/ubuntu#
```

- docker node ls (Listing Nodes)

```
root@m:/home/ubuntu# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
8vpuygcs875423ahh35j0r1ih *   m          Ready     Active         Leader           24.0.7
l8bbi4c6s6r0jdd3zab80mo74     w          Ready     Active                          24.0.7
root@m:/home/ubuntu#
```

- docker service create –name vote -p 8080:80 instavote/vote (Creating a Service)

```
root@m:/home/ubuntu# docker service create --replicas 3 -p 80:80 --name web nginx
p3vvkvuvllofvwk2fx8o37tne
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@m:/home/ubuntu#
```

- docker service scale vote=3 (Scaling a Service)

```
root@m:/home/ubuntu# docker service scale web=5
web scaled to 5
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
root@m:/home/ubuntu#
```

6) Docker monitoring commands.

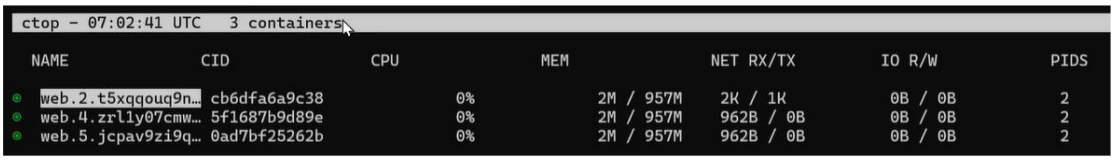
- wget https://github.com/bcicen/ctop/releases/download/v0.7.1/ctop-0.7.1-linux-amd64 -O /usr/local/bin/ctop (Installation of ctop monitoring tool)

```
root@m:/home/ubuntu# wget https://github.com/bcicen/ctop/releases/download/v0.7.1/ctop-0.7.1-linux-amd64 -O /usr/local/bin/ctop
--2024-08-17 07:01:54--  https://github.com/bcicen/ctop/releases/download/v0.7.1/ctop-0.7.1-linux-amd64
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response...
```

chmod +x /usr/local/bin/ctop

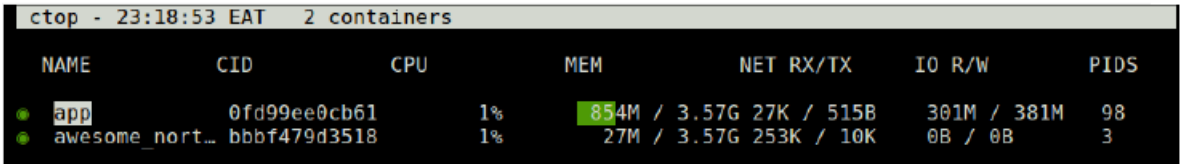
```
root@m:/home/ubuntu# chmod +x /usr/local/bin/ctop
root@m:/home/ubuntu#
```

- ctop (ctop command to view the usage of docker swarm)



- ctop -a (To display active containers only)

```
$ ctop -a
```



- ctop -scale-cpu (To display CPU as **%** of system total)

```
$ ctop -scale-cpu
```

- ctop -f app (Filter any containers)

```
$ ctop -f app
```

- ctop -h (This command is for ctop help)

```
$ ctop -h
```

7) Docker container log file and home path location.

- Docker container log file location

/var/lib/docker/container

```
root@ip-172-31-91-134:/home/ubuntu# cd /var/lib/docker/containers/
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers# ls -ltr
total 8
drwx--x--- 4 root root 4096 Aug 29 13:15 7aaa84922d010cf980b3e71698af2c3cf73702487ae932c9f5523acd95894a9b
drwx--x--- 4 root root 4096 Aug 29 13:38 0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers#
root@ip-172-31-91-134:/var/lib/docker/containers# cd 0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4/
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4# ls -ltr
total 36
drwx------ 2 root root 4096 Aug 29 13:38 checkpoints
-rw-r--r-- 1 root root   71 Aug 29 13:38 resolv.conf.hash
-rw-r--r-- 1 root root  796 Aug 29 13:38 resolv.conf
-rw-r--r-- 1 root root  174 Aug 29 13:38 hosts
drwx--x--- 2 root root 4096 Aug 29 13:38 mounts
-rw-r--r-- 1 root root   13 Aug 29 13:38 hostname
-rw------- 1 root root 1464 Aug 29 13:38 hostconfig.json
-rw------- 1 root root 2675 Aug 29 13:38 config.v2.json
-rw-r----- 1 root root 2261 Aug 29 13:39 0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4-json.log
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
root@ip-172-31-91-134:/var/lib/docker/containers/0fd0de001f8fccd40677aeb0a21821599ff91c18d2a1a2bb69b72cf0686e41c4#
```

- Docker home path location

/var/lib/docker

```
root@ip-172-31-91-134:/var/lib/docker# ls -ltr
total 44
drwx------  2 root root 4096 Aug 29 13:02 runtimes
drwx------  4 root root 4096 Aug 29 13:02 plugins
drwx-----x  2 root root 4096 Aug 29 13:02 volumes
-rw-------  1 root root   36 Aug 29 13:02 engine-id
drwx------  3 root root 4096 Aug 29 13:02 image
drwxr-x---  3 root root 4096 Aug 29 13:02 network
drwx------  2 root root 4096 Aug 29 13:02 swarm
drwx--x--x  4 root root 4096 Aug 29 13:02 buildkit
drwx------  2 root root 4096 Aug 29 13:38 tmp
drwx--x--- 17 root root 4096 Aug 29 13:38 overlay2
drwx--x---  4 root root 4096 Aug 29 13:38 containers
root@ip-172-31-91-134:/var/lib/docker#
```

8) Image has been downloaded in docker we need to know where it is referenced/maintained.

/var/lib/docker/image/overlay2

```
root@ip-172-31-91-134:/var/lib/docker/image/overlay2# ls -ltr
total 16
drwx------ 4 root root 4096 Aug 29 13:02 imagedb
drwx------ 4 root root 4096 Aug 29 13:06 distribution
drwx------ 5 root root 4096 Aug 29 13:15 layerdb
-rw------- 1 root root  801 Aug 29 13:38 repositories.json
root@ip-172-31-91-134:/var/lib/docker/image/overlay2#
```

9) Install docker old version and upgrade to new version

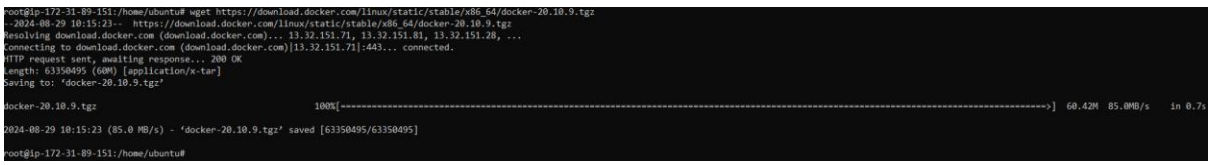
- Create a new EC2 instance with ubuntu image and run

  apt update -y && apt upgrade -y

```
root@ip-172-31-83-227:/home/ubuntu# apt update -y && apt upgrade -y
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

- Download the old version static binary archive by running the below wget command and choose your hardware platform, and download the .tgz file relating to the version of Docker Engine you want to install.

  wget https://download.docker.com/linux/static/stable/x86_64/docker-20.10.9.tgz



  ls -ltr to check if the file is downloaded

```
root@ip-172-31-89-151:/home/ubuntu# ls -ltr
total 61872
-rw-r--r-- 1 root root 63350495 Jul  5  2023 docker-20.10.9.tgz
root@ip-172-31-89-151:/home/ubuntu#
```

- Extract the archive using the tar utility. The dockerd and docker binaries are extracted.

  tar xzvf docker-20.10.9.tgz

```
root@ip-172-31-89-151:/home/ubuntu# tar xzvf docker-20.10.9.tgz
docker/
docker/containerd-shim-runc-v2
docker/dockerd
docker/docker-proxy
docker/ctr
docker/docker
docker/runc
docker/containerd-shim
docker/docker-init
docker/containerd
root@ip-172-31-89-151:/home/ubuntu#
```

  ls -ltr to check if the tar file is extracted.

```
root@ip-172-31-89-151:/home/ubuntu# ls -ltr
total 61876
drwxrwxr-x 2 ubuntu ubuntu     4096 Oct  4  2021 docker
-rw-r--r-- 1 root   root   63350495 Jul  5  2023 docker-20.10.9.tgz
root@ip-172-31-89-151:/home/ubuntu#
```

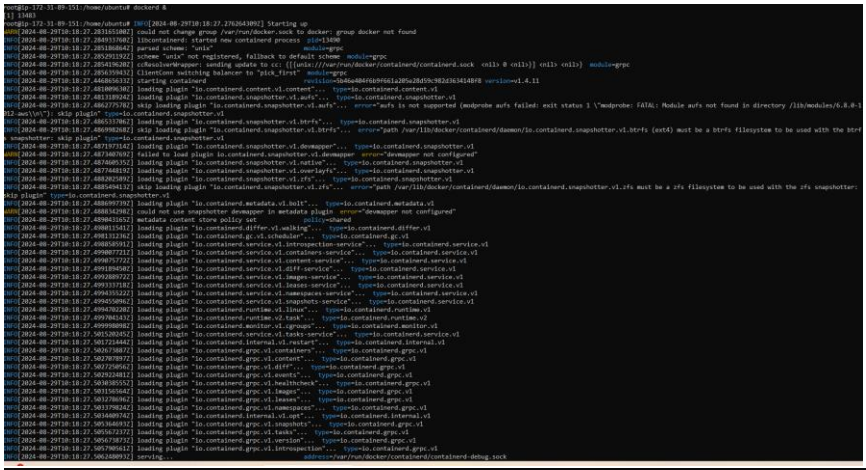
- Move the binaries to a directory on your executable path, such as /usr/bin/

  cp docker/* /usr/bin/

```
root@ip-172-31-83-227:/home/ubuntu# cp docker/* /usr/bin/
root@ip-172-31-83-227:/home/ubuntu#
```

- Start the Docker daemon

  dockerd &



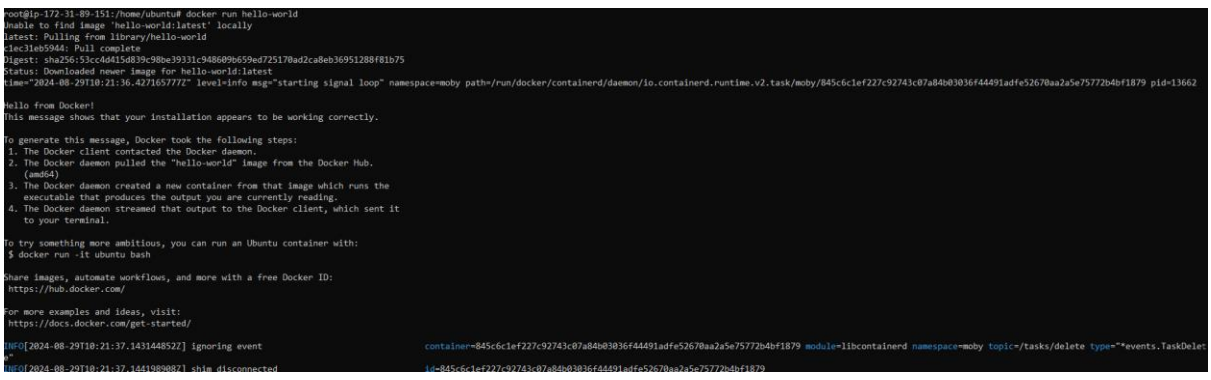
- We will verify the version of docker installed, shows 20.10.9

  docker --version

```
root@ip-172-31-89-151:/home/ubuntu# docker --version
Docker version 20.10.9, build c2ea9bc
root@ip-172-31-89-151:/home/ubuntu#
```

- Also, we will verify that docker is installed correctly by running the hello-world image. It ran successfully.

  docker run hello-world



- Verify that the hello-world container got created.

  docker ps -a

```
root@ip-172-31-89-151:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
845c6c1ef227   hello-world   "/hello"   13 seconds ago   Exited (0) 12 seconds ago             elastic_heyrovsky
```

- Now before upgrading the docker version, we need to kill the dockerd (docker daemon) process id.

- For that we need to find out the process id of docker daemon using the below command.

  pidof dockerd

```
root@ip-172-31-89-33:/home/ubuntu# pidof dockerd
15615
```

- Now will kill the pid of docker daemon to stop dockerd by using below command.

  kill 15615 (process id of dockerd)

  On successful execution, we will see the below screenshot.

```
root@ip-172-31-89-33:/home/ubuntu# kill 15615
INFO[2024-08-30T06:59:06.978506538Z] Processing signal 'terminated'
root@ip-172-31-89-33:/home/ubuntu# INFO[2024-08-30T06:59:06.981770511Z] Daemon shutdown complete
INFO[2024-08-30T06:59:06.981980905Z] stopping event stream following graceful shutdown  error="context canceled" module=libcontainerd namespace=moby
INFO[2024-08-30T06:59:06.982090115Z] stopping healthcheck following graceful shutdown  module=libcontainerd
INFO[2024-08-30T06:59:06.982113626Z] stopping event stream following graceful shutdown  error="context canceled" module=libcontainerd namespace=plugins.moby

[1]+  Done                    dockerd
root@ip-172-31-89-33:/home/ubuntu#
```

- Now we need to setup the docker repository and then we can install it from that repository.

- Adding Docker's official GPG key using the below commands

  apt-get update

```
root@ip-172-31-89-33:/home/ubuntu# sudo apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu noble-security InRelease
Fetched 126 kB in 1s (230 kB/s)
Reading package lists... Done
```

apt-get install ca-certificates curl

```
root@ip-172-31-89-33:/home/ubuntu# sudo apt-get install ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20240203).
curl is already the newest version (8.5.0-2ubuntu10.3).
The following packages were automatically installed and are no longer required:
  bridge-utils dns-root-data dnsmasq-base pigz ubuntu-fan
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
```

install -m 0755 -d /etc/apt/keyrings

```
root@ip-172-31-89-33:/home/ubuntu# install -m 0755 -d /etc/apt/keyrings
root@ip-172-31-89-33:/home/ubuntu#
```

curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

```
root@ip-172-31-89-33:/home/ubuntu# curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
root@ip-172-31-89-33:/home/ubuntu#
```

chmod a+r /etc/apt/keyrings/docker.asc

```
root@ip-172-31-89-33:/home/ubuntu# chmod a+r /etc/apt/keyrings/docker.asc
root@ip-172-31-89-33:/home/ubuntu#
```

- Now we will add the repository to Apt sources by using the below commands.

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
root@ip-172-31-89-33:/home/ubuntu# echo \
>   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
>   $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
>   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
root@ip-172-31-89-33:/home/ubuntu#
```

apt-get update

```
root@ip-172-31-89-33:/home/ubuntu# apt-get update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [13.2 kB]
Fetched 62.1 kB in 1s (85.1 kB/s)
Reading package lists... Done
root@ip-172-31-89-33:/home/ubuntu#
```

- Now we will install the latest version of docker and components using the below command.

apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

```
root@ip-172-31-89-33:/home/ubuntu# sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bridge-utils dns-root-data dnsmasq-base ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  docker-ce-rootless-extras libltdl7 libslirp0 slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 slirp4netns
0 upgraded, 9 newly installed, 0 to remove and 3 not upgraded.
Need to get 121 MB of archives.
After this operation, 437 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libltdl7 amd64 2.4.7-7build1 [40.3 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 slirp4netns amd64 1.2.1-1build2 [34.9 kB]
Get:4 https://download.docker.com/linux/ubuntu noble/stable amd64 containerd.io amd64 1.7.21-1 [29.5 MB]
Get:5 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-buildx-plugin amd64 0.16.2-1~ubuntu.24.04~noble [29.9 MB]
Get:6 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-cli amd64 5:27.2.0-1~ubuntu.24.04~noble [14.8 MB]
Get:7 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce amd64 5:27.2.0-1~ubuntu.24.04~noble [25.3 MB]
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-ce-rootless-extras amd64 5:27.2.0-1~ubuntu.24.04~noble [9318 kB]
Get:9 https://download.docker.com/linux/ubuntu noble/stable amd64 docker-compose-plugin amd64 2.29.2-1~ubuntu.24.04~noble [12.5 MB]
Fetched 121 MB in 2s (49.4 MB/s)
Selecting previously unselected package containerd.io.
(Reading database ... 98483 files and directories currently installed.)
Preparing to unpack .../0-containerd.io_1.7.21-1_amd64.deb ...
Unpacking containerd.io (1.7.21-1) ...
Selecting previously unselected package docker-buildx-plugin.
Preparing to unpack .../1-docker-buildx-plugin_0.16.2-1~ubuntu.24.04~noble_amd64.deb ...
Unpacking docker-buildx-plugin (0.16.2-1~ubuntu.24.04~noble) ...
Selecting previously unselected package docker-ce-cli.
Preparing to unpack .../2-docker-ce-cli_5%3a27.2.0-1~ubuntu.24.04~noble_amd64.deb ...
Unpacking docker-ce-cli (5:27.2.0-1~ubuntu.24.04~noble) ...
Selecting previously unselected package docker-ce.
Preparing to unpack .../3-docker-ce_5%3a27.2.0-1~ubuntu.24.04~noble_amd64.deb ...
Unpacking docker-ce (5:27.2.0-1~ubuntu.24.04~noble) ...
Selecting previously unselected package docker-ce-rootless-extras.
Preparing to unpack .../4-docker-ce-rootless-extras_5%3a27.2.0-1~ubuntu.24.04~noble_amd64.deb ...
Unpacking docker-ce-rootless-extras (5:27.2.0-1~ubuntu.24.04~noble) ...
Selecting previously unselected package docker-compose-plugin.
Preparing to unpack .../5-docker-compose-plugin_2.29.2-1~ubuntu.24.04~noble_amd64.deb ...
Unpacking docker-compose-plugin (2.29.2-1~ubuntu.24.04~noble) ...
Selecting previously unselected package libltdl7:amd64.
Preparing to unpack .../6-libltdl7_2.4.7-7build1_amd64.deb ...
Unpacking libltdl7:amd64 (2.4.7-7build1) ...
Selecting previously unselected package libslirp0:amd64.
Preparing to unpack .../7-libslirp0_4.7.0-1ubuntu3_amd64.deb ...
Unpacking libslirp0:amd64 (4.7.0-1ubuntu3) ...
Selecting previously unselected package slirp4netns.
Preparing to unpack .../8-slirp4netns_1.2.1-1build2_amd64.deb ...
Unpacking slirp4netns (1.2.1-1build2) ...
Setting up docker-buildx-plugin (0.16.2-1~ubuntu.24.04~noble) ...
```

```
Setting up libltdl7:amd64 (2.4.7-7build1) ...
Setting up docker-ce-cli (5:27.2.0-1~ubuntu.24.04~noble) ...
Setting up libslirp0:amd64 (4.7.0-1ubuntu3) ...
Setting up docker-ce-rootless-extras (5:27.2.0-1~ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.2.0-1~ubuntu.24.04~noble) ...
Could not execute systemctl:  at /usr/bin/deb-systemd-invoke line 148.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Pending kernel upgrade!
Running kernel version:
  6.8.0-1012-aws
Diagnostics:
  The currently running kernel version is not the expected kernel version 6.8.0-1014-aws.

Restarting the system to load the new kernel will not be handled automatically, so you should consider rebooting.

Restarting services...

Service restarts being deferred:
 /etc/needrestart/restart.d/dbus.service
 systemctl restart getty@tty1.service
 systemctl restart networkd-dispatcher.service
 systemctl restart serial-getty@ttyS0.service
 systemctl restart systemd-logind.service
 systemctl restart unattended-upgrades.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

- Now we will restart the docker daemon service by using the below command.

  dockerd &

```
root@ip-172-31-89-33:/home/ubuntu# dockerd &
[1] 16945
root@ip-172-31-89-33:/home/ubuntu# INFO[2024-08-30T07:05:10.897194406Z] Starting up
INFO[2024-08-30T07:05:10.898411176Z] detected 127.0.0.53 nameserver, assuming systemd-resolved, so using resolv.conf: /run/systemd/resolve/resolv.conf
INFO[2024-08-30T07:05:11.070661683Z] [graphdriver] using prior storage driver: overlay2
INFO[2024-08-30T07:05:11.085815598Z] Loading containers: start.
INFO[2024-08-30T07:05:11.328504187Z] Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be used to set a preferred IP address
WARN[2024-08-30T07:05:11.401310537Z] error locating sandbox id b3ec8ec9b820b36f7e070a87c6a3a0edba24bba370320cf49eb535d4b9b0ab24: sandbox b3ec8ec9b820b36f7e070a87c6a3a0edba24bba370320cf49eb535d4b9b0ab24 not found
INFO[2024-08-30T07:05:11.401760896Z] Loading containers: done.
INFO[2024-08-30T07:05:11.424213104Z] Docker daemon                                 commit=3ab5c7d containerd-snapshotter=false storage-driver=overlay2 version=27.2.0
INFO[2024-08-30T07:05:11.424792506Z] Daemon has completed initialization
INFO[2024-08-30T07:05:11.476058882Z] API listen on /var/run/docker.sock

root@ip-172-31-89-33:/home/ubuntu#
```

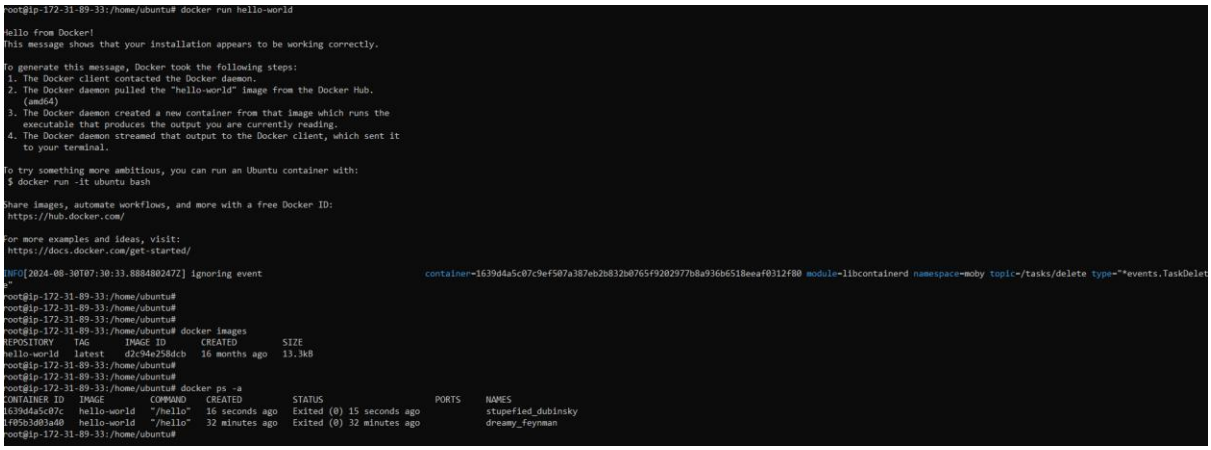
- The docker version is now updated to the latest one.

- We will verify the version by running the below command.

  docker --version

```
root@ip-172-31-89-33:/home/ubuntu# docker --version
Docker version 27.2.0, build 3ab4256
```

- We can see that the image and containers from old and new version are present.

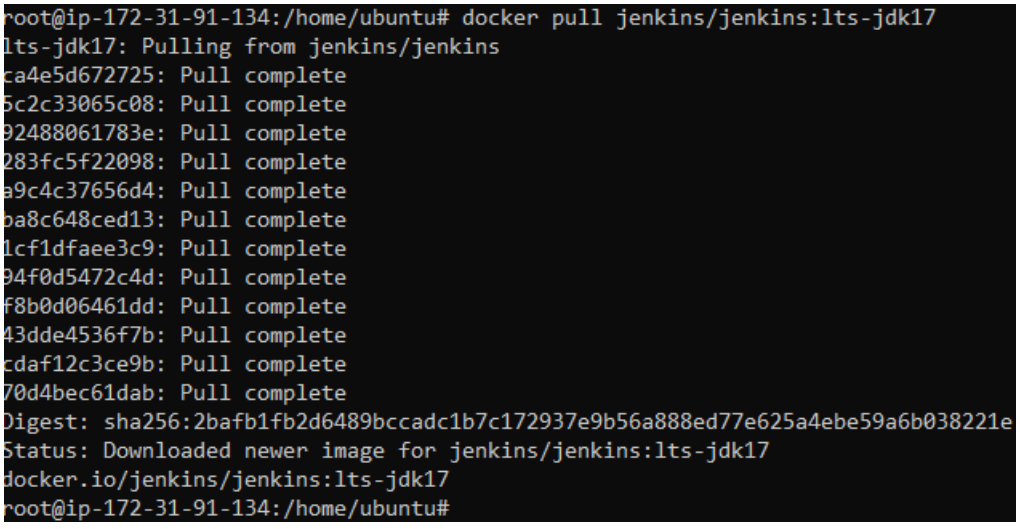


10) Install Jenkins in docker

Creating a container with Jenkins installed inside it.

- We will pull the jenkins image from dockerhub.

  docker pull jenkins/jenkins:lts-jdk17



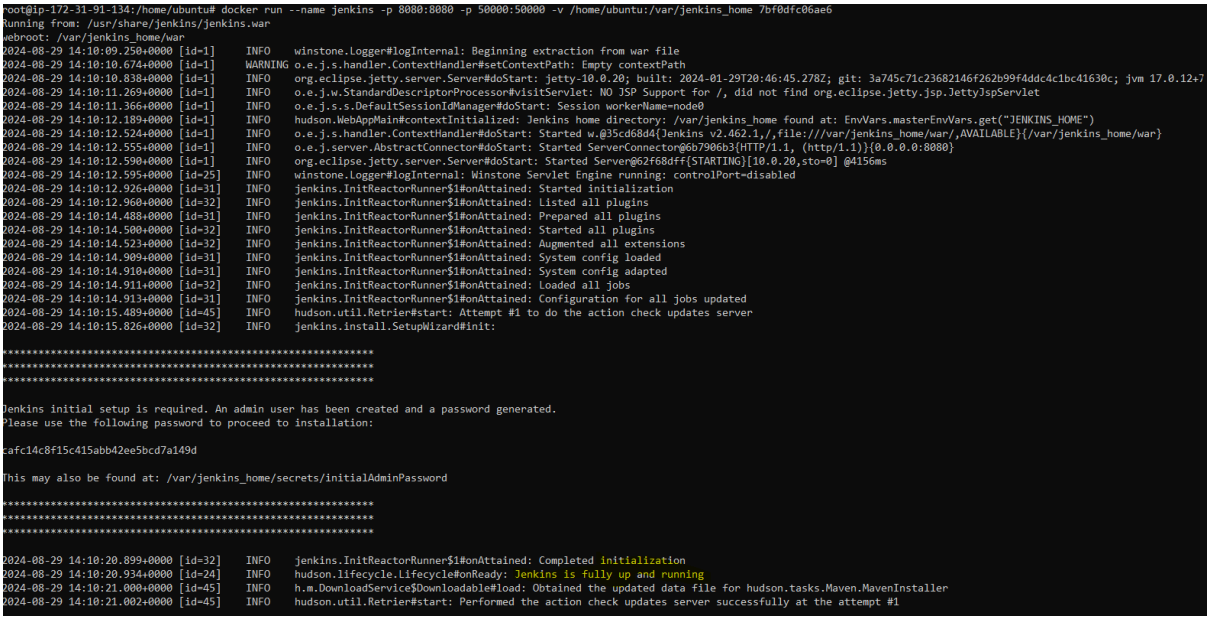
- Verify the image is downloaded.

  docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY        TAG         IMAGE ID       CREATED         SIZE
jenkins/jenkins   lts-jdk17   7bf0dfc06ae6   3 weeks ago     469MB
tomcat            latest      c2a444ea6cd7   3 weeks ago     508MB
ubuntu            latest      edbfe74c41f8   4 weeks ago     78.1MB
hello-world       latest      d2c94e258dcb   16 months ago   13.3kB
```

- Now we will Run a container using the image you just pulled.

  docker run --name jenkins -p 8080:8080 -p 50000:50000 -v /home/ubuntu:/var/jenkins_home 7bf0dfc06ae6



docker run: This is the docker command for starting a container.

-- name jenkins: This command will give the container the name jenkins after it has been started.

-p 8080:8080: The first 8080 is the port you will use to access the Jenkins service on your own machine(localhost:8080). The second 8080 after the first colon is the Jenkins default port. What we have done with this command is bound port 8080 on Jenkins to port 8080 on our local machine.

-p 50000:50000: Same as the case of the Jenkins port above, we are binding the ports to each other. However, in this case, the port is meant

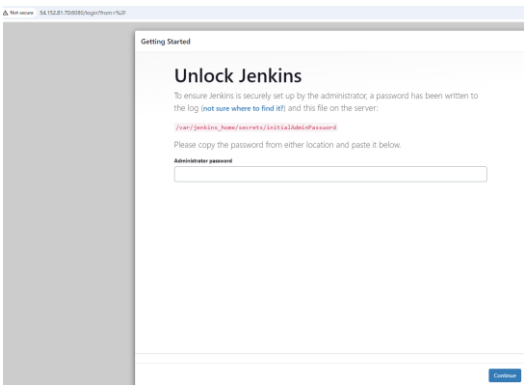
for the Jenkins master to be able to communicate with the slaves. The Jenkins master coordinates and does the administrative tasks of your Jenkins infrastructure while the purpose of the slaves is to run the tasks. It is also important to note that the master can also run jobs like the Jenkins slaves. But you can **have** your Jenkins setup without the slaves.

-v /home/ubuntu:/var/jenkins_home: The -v flag is a command for mapping or creating volumes in Jenkins. Data that is generated while a container is running is ephemeral in nature. The image that is used to generate your container is read-only, so any data that is generated while the container is running will exist only inside the container. Once the container is destroyed, all that data is lost. In our case, we do not want to lose all our configurations and settings after our Jenkins container is destroyed. We have to create a volume on our local machine and map it to the Jenkins directory in the container in other to save our data. This way, any files generated while the container is running can still be accessed after the container is destroyed and when we start another container. The first part before the colon(/your/home) is the path on your computer where you want the Jenkins files to be saved.

7bf0dfc06ae6 - This last part of our command, specifies the id of the image you want to run, which is the id Jenkins container we just pooled. You can run docker images to get the id of the image pulled on your computer.

- Setup Jenkins

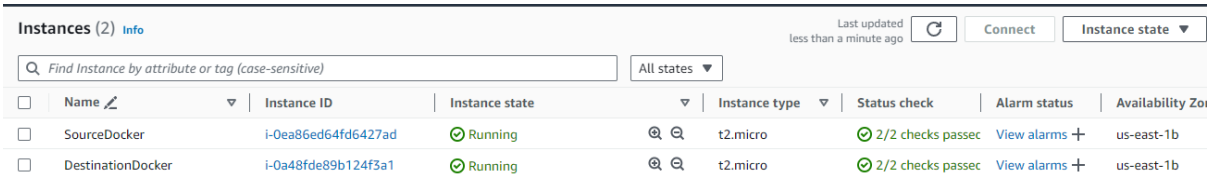
Visit **Public IP of EC2:8080** on your browser. You should see a web page like the one below: If this comes up then we are good



11) How to copy/move a docker container from one host to another host.

- The Task is that we need to create a container from the image and create an index file inside it and then save the container and run on different server or EC2 instance.

- For the above tasks, we need to have 2 EC2 instances with docker installed on them, one on which we need to create the container and index file inside it and other EC2 instance is where we will run the saved container copied from first EC2 instance.



- Now we will perform the task of running a nginx container from nginx latest image in an interactive mode, so that we can create an index.html file on the **SourceDocker** EC2 Instance.

- **SourceDocker**

  - We pulled and run a nginx image which created a container by running the below command.

    docker run -d -p 80:80 --restart always --name nginxtest nginx

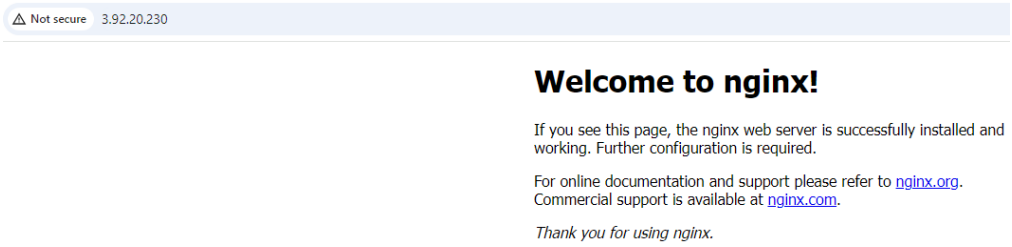
```
root@ip-172-31-29-95:/home/ubuntu# docker run -d -p 80:80 --restart always --name nginxtest nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Downloaded newer image for nginx:latest
bdc4bfa351999802e264fb7b8e2a8c157b8cfe1e335d4c47d619945eeaa2e4e1
root@ip-172-31-29-95:/home/ubuntu#
```

    We will verify the container details by running the below command

docker ps

```
root@ip-172-31-29-95:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED             STATUS             PORTS
        NAMES
bdc4bfa35199   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp, :::80->
80/tcp   nginxtest
root@ip-172-31-29-95:/home/ubuntu#
```

- We checked on Web URL to confirm that nginx was running properly.



- The nginx container already has default index.html file in it. We will change the content of the index.html and confirm if it's functioning properly.

- We will run the below command to get inside the running container's /bin/bash using the container id

docker exec -it bdc4bfa35199 /bin/bash

```
root@ip-172-31-29-95:/home/ubuntu# docker exec -it bdc4bfa35199 /bin/bash
root@bdc4bfa35199:/#
root@bdc4bfa35199:/#
```

- Now we will go to the location where the index.html file is kept inside the container.

cd /usr/share/nginx/html

```
root@bdc4bfa35199:/# cd /usr/share/nginx/html
root@bdc4bfa35199:/usr/share/nginx/html#
root@bdc4bfa35199:/usr/share/nginx/html#
```

ls -ltr

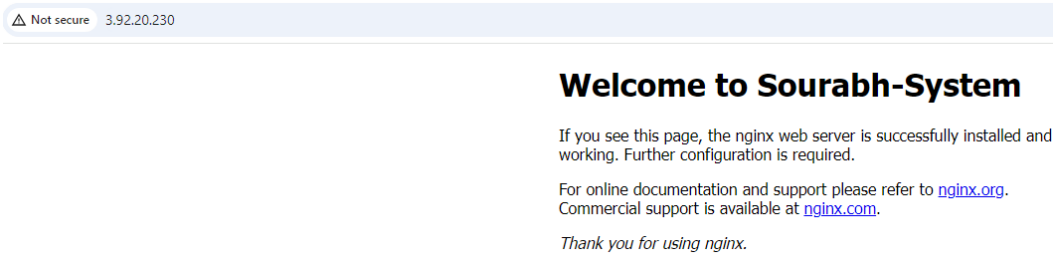
```
root@bdc4bfa35199:/usr/share/nginx/html# ls -ltr
total 8
-rw-r--r-- 1 root root 615 Aug 12 14:21 index.html
-rw-r--r-- 1 root root 497 Aug 12 14:21 50x.html
root@bdc4bfa35199:/usr/share/nginx/html#
```

- We will change the content of the index.html from the below command.

  sed -i -e 's/nginx!/Sourabh-System/g' index.html

```
root@bdc4bfa35199:/usr/share/nginx/html# sed -i -e 's/nginx!/Sourabh-System/g' index.html
root@bdc4bfa35199:/usr/share/nginx/html#
root@bdc4bfa35199:/usr/share/nginx/html#
```

- Now we will test changes on the Web URL. Changes were applied successfully.



- Now we have our customized index.html file in a running container.

- We will exit out the container and keep it running by using Ctrl p and q

- Now we will stop the container and preserve the current state of the container to a new image.

  docker stop bdc4bfa35199 (Stops the container using the container ID)

```
root@ip-172-31-29-95:/home/ubuntu# docker stop bdc4bfa35199
bdc4bfa35199
root@ip-172-31-29-95:/home/ubuntu#
```

docker commit nginxtest mynginximage (Command preserves the current state of container in a new image)

```
root@ip-172-31-29-95:/home/ubuntu# docker commit nginxtest mynginximage
sha256:f73e67514ae32890d1c485b8c8ee54f2a5867eba551e8d4d0c79f82f56886d84
root@ip-172-31-29-95:/home/ubuntu#
```

- Now we will save your newly committed Docker container image to an archive file (tar file) using the below command

docker save -o mynginximage.tar mynginximage (Command saves the committed image to tar file)

```
root@ip-172-31-29-95:/home/ubuntu# docker save -o mynginximage.tar mynginximage
root@ip-172-31-29-95:/home/ubuntu#
```

Verify the tar image

ls -ltr

```
root@ip-172-31-29-95:/home/ubuntu# ls -ltr
total 187388
-rw------- 1 root root 191883776 Sep 10 06:58 mynginximage.tar
root@ip-172-31-29-95:/home/ubuntu#
```

- For this task we need to create test user with home directory and give root level access and setup passwordless ssh as both test account and password less ssh will be used to copy our image using the scp.

sourcedocker

useradd -p abc123 -m test (Command for creating test with unencrypted password using **-p** and creating home directory using **-m**)

```
root@ip-172-31-29-95:/home/ubuntu# useradd -p abc123 -m test
root@ip-172-31-29-95:/home/ubuntu#
```

We need to reset the password once for the test user to give it an encrypted password. Since, we have setup and unencrypted password using the below command.

passwd test (command to reset the password for the user)

```
root@ip-172-31-29-95:/home/ubuntu# passwd test
New password:
Retype new password:
passwd: password updated successfully
root@ip-172-31-29-95:/home/ubuntu#
```

Now we will also verify home directory is created for the test user or not by using the below command.

cd /home/test (Command to change directory)

```
root@ip-172-31-29-95:/home/ubuntu# cd /home/test
root@ip-172-31-29-95:/home/test#
```

Home directory created and verified

Destinationdocker

useradd -p abc123 -m test (Command for creating test with unencrypted password using **-p** and creating home directory using **-m**)

```
root@ip-172-31-20-184:/home/ubuntu# useradd -p abc123 -m test
root@ip-172-31-20-184:/home/ubuntu#
```

We need to reset the password once for the test user to give it an encrypted password. Since, we have setup and unencrypted password using the below command.

passwd test (command to reset the password for the user)

```
root@ip-172-31-20-184:/home/ubuntu# passwd test
New password:
Retype new password:
passwd: password updated successfully
```

Now we will also verify home directory is created for the test user or not by using the below command.

cd /home/test (Command to change directory)

```
root@ip-172-31-20-184:/home/ubuntu# cd /home/test
root@ip-172-31-20-184:/home/test#
root@ip-172-31-20-184:/home/test#
```

Home directory created and verified

Providing “test” user root level access with a script which we need to run on all 3 EC2 instances.

SourceDocker

Create a shell script with any name, in my case I created “rootlevel.sh”. Edit the file and enter the below contents in that file.

vi rootlevel.sh (Edit the file)

```
root@ip-172-31-29-95:/home/test# vi rootlevel.sh
root@ip-172-31-29-95:/home/test#
```

Contents of the script

#!/bin/bash

Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

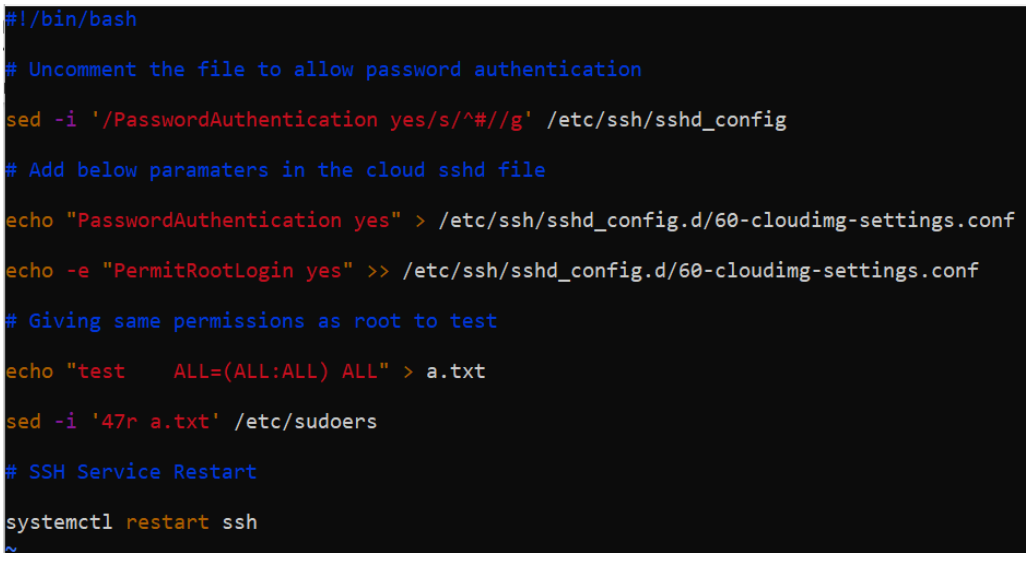
Giving same permissions as root to test

echo "test   ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

SSH Service Restart

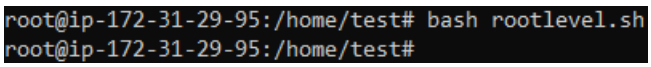
systemctl restart ssh



```
#!/bin/bash
# Uncomment the file to allow password authentication
sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config
# Add below paramaters in the cloud sshd file
echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
# Giving same permissions as root to test
echo "test    ALL=(ALL:ALL) ALL" > a.txt
sed -i '47r a.txt' /etc/sudoers
# SSH Service Restart
systemctl restart ssh
```

Run the script as bash and test user will have same level of access as root

bash rootlevel.sh

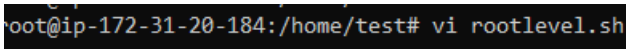


Same steps needed to done on other DestinationDocker.

Destination Docker

Create a shell script with any name, in my case I created “rootlevel.sh”. Edit the file and enter the below contents in that file.

vi rootlevel.sh (Edit the file)



Contents of the script

#!/bin/bash

Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

Giving same permissions as root to test

echo "test   ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

SSH Service Restart

systemctl restart ssh

```
#!/bin/bash
# Uncomment the file to allow password authentication
sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config
# Add below paramaters in the cloud sshd file
echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
# Giving same permissions as root to test
echo "test    ALL=(ALL:ALL) ALL" > a.txt
sed -i '47r a.txt' /etc/sudoers
# SSH Service Restart
systemctl restart ssh
```

Run the script as bash and test user will have same level of access as root

bash rootlevel.sh

```
root@ip-172-31-20-184:/home/test# bash rootlevel.sh
```

Now setup password less ssh between master and worker by following below steps.

- **Creating a temporary file “tempUserName.txt” for passing “test” user details on master EC2 instance.**

  echo "test" > tempUserName.txt (Command creates tempUserName.txt and puts test as text inside)

```
root@ip-172-31-29-95:/home/test# echo "test" > tempUserName.txt
root@ip-172-31-29-95:/home/test#
```

- **Creating a file by name “ipaddr” which will contain the Public ip address of destinationdocker on sourcedocker EC2 instance.**

  echo "34.234.63.151" > ipaddr (Command creates ipaddr file and puts the ip address as text inside)

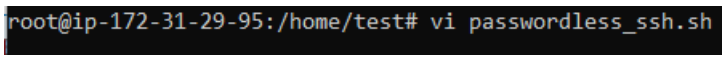
```
root@ip-172-31-29-95:/home/test# echo "34.234.63.151" > ipaddr
root@ip-172-31-29-95:/home/test#
```

- **Setting up password less authentication between master and worker nodes from the master instance using script.**

  SourceDocker

  Create a shell script with any name, in my case I created “passwordless_ssh.sh”. Edit the file and enter the below contents in that file.

vi passwordless_ssh.sh (Edit the file)



Contents of the script

```
#!/bin/bash

# Defining variable for ipaddr file

ipAddFile="./ipaddr"

# Command runs SSH-Keygen without prompting anything

echo -e "\n" | ssh-keygen -N "" &> /dev/null

# Now the Script checks the IP address file and concat the
username and IP address to copy the ssh pub key

echo "$ipAddFile"

for IP in `cat $ipAddFile`; do
        if [[ $IP == *"["* ]]; then
                echo "$IP"|cut -d "[" -f2 | cut -d "]" -f1>tempUserName.txt
                else
                        user=$(cat tempUserName.txt)
                        ssh-copy-id $user@$IP
```

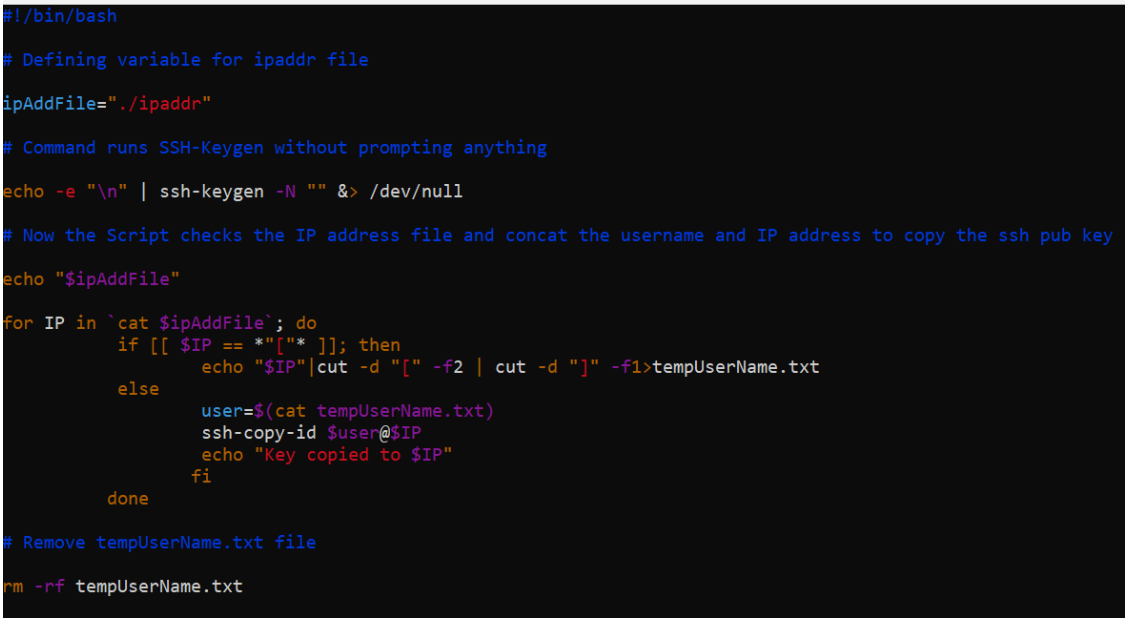
echo "Key copied to $IP"

fi

done

Remove tempUserName.txt file

rm -rf tempUserName.txt



```
#!/bin/bash

# Defining variable for ipaddr file

ipAddFile="./ipaddr"

# Command runs SSH-Keygen without prompting anything

echo -e "\n" | ssh-keygen -N "" &> /dev/null

# Now the Script checks the IP address file and concat the username and IP address to copy the ssh pub key

echo "$ipAddFile"

for IP in `cat $ipAddFile`; do
          if [[ $IP == *"["* ]]; then
                  echo "$IP"|cut -d "[" -f2 | cut -d "]" -f1>tempUserName.txt
          else
                  user=$(cat tempUserName.txt)
                  ssh-copy-id $user@$IP
                  echo "Key copied to $IP"
                 fi
         done

# Remove tempUserName.txt file

rm -rf tempUserName.txt
```

Run the script as bash and passwordless SSH will be setup between source and destination docker.

bash passwordless_ssh.sh

```
root@ip-172-31-29-95:/home/test# bash passwordless_ssh.sh
./ipaddr
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_ed25519.pub"
The authenticity of host '34.234.63.151 (34.234.63.151)' can't be established.
ED25519 key fingerprint is SHA256:95gSZj/eZC38TYOFNDjgRg/s5NDgUa7OMtE2zKWSnig.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
test@34.234.63.151's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'test@34.234.63.151'"
and check to make sure that only the key(s) you wanted were added.

Key copied to 34.234.63.151
root@ip-172-31-29-95:/home/test#
```

We need to supply password for test user to get the key copied (Above screenshot shows key copied to the IP address in ipaddr file)

- Copy the mynginximage.tar to home directory of test

  cp mynginximage.tar /home/test

```
root@ip-172-31-29-95:/home/ubuntu# cp mynginximage.tar /home/test
root@ip-172-31-29-95:/home/ubuntu#
```

- Now we will do Scp command to copy our tar file to destination docker.

  scp mynginximage.tar test@34.234.63.151:/home/test/

```
root@ip-172-31-29-95:/home/test# scp mynginximage.tar test@34.234.63.151:/home/test/
mynginximage.tar                                              100%  183MB  90.0MB/s   00:02
root@ip-172-31-29-95:/home/test#
```

- Now we will perform the task of running a nginx container from the tar file copied from the **SourceDocker** EC2 Instance on the **DestinationDocker** EC2 instance.

- **DestinationDocker**

- Verify that the myimage.tar file has been copied successfully under /home/test using the below command.

  ls -ltr

```
root@ip-172-31-20-184:/home/test# ls -ltr
total 187400
-rw-r--r-- 1 root root       349 Sep 10 07:13 rootlevel.sh
-rw-r--r-- 1 root root        26 Sep 10 07:14 a.txt
-rw------- 1 test test 191883776 Sep 10 07:26 mynginximage.tar
root@ip-172-31-20-184:/home/test#
```

- Now we will load the container image it container on the DestinationDocker EC2 Instance using the below command.

  docker load -i ./mynginximage.tar

```
root@ip-172-31-20-184:/home/test# docker load -i ./mynginximage.tar
8e2ab394fabf: Loading layer [==================================================>]  77.83MB/77.83MB
67796e30ff04: Loading layer [==================================================>]   114MB/114MB
eda13eb24d4c: Loading layer [==================================================>]  3.584kB/3.584kB
0fc6bb94eec5: Loading layer [==================================================>]  4.608kB/4.608kB
2bdf51597158: Loading layer [==================================================>]   2.56kB/2.56kB
16907864a2d0: Loading layer [==================================================>]   5.12kB/5.12kB
11de3d47036d: Loading layer [==================================================>]  7.168kB/7.168kB
a5985363531d: Loading layer [==================================================>]  13.82kB/13.82kB
Loaded image: mynginximage:latest
root@ip-172-31-20-184:/home/test#
```

  Our saved image is loaded highlighted in the above screenshot.

- We will now reinitialize our Docker container image with its original run flags from your source machine using the below command. For instance, our Nginx Docker container originally had port 80 mapped to my host machine's port 80

  docker create --name my-nginx-container -p 80:80 mynginximage
  (Command helps in reinitialize our Docker container image with its original run flags from your source machine)

```
root@ip-172-31-20-184:/home/test# docker create --name my-nginx-container -p 80:80 mynginximage
ad75cec7a41184b29c4ec6c7ece9b5894b3301a4c0c9e0eda4aa01d1906dbabf
root@ip-172-31-20-184:/home/test#
```

- We will run the newly imported docker container using the below command.

docker start my-nginx-container

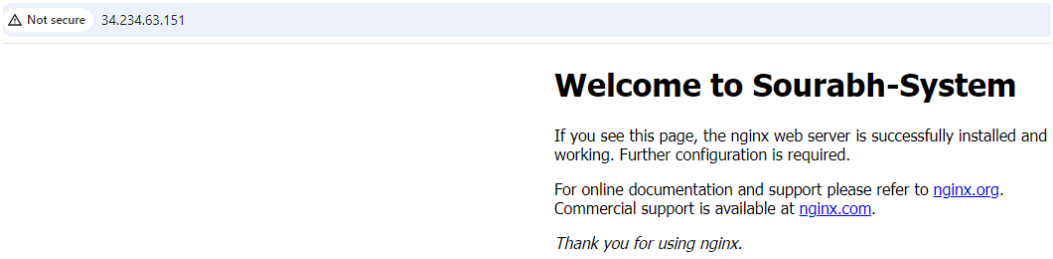
```
root@ip-172-31-20-184:/home/test# docker start my-nginx-container
my-nginx-container
root@ip-172-31-20-184:/home/test#
```

To verify, we will run

docker ps -a

```
root@ip-172-31-20-184:/home/test# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                   NAMES
ad75cec7a411   mynginximage   "/docker-entrypoint.…"   52 seconds ago   Up 20 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   my-ng
inx-container
root@ip-172-31-20-184:/home/test#
```

- Now we will test the Web URL to see if the index.html file shows the same content as shown the SourceDocker EC2 instance



- We can see it has loaded the same contents as it displayed in source EC2 instance.

- Task has been completed to Copy/Move a Docker Container to Another Host.

12) Create your own local docker registry and put an image with your created tag on it.

- To create our own docker registry we need to pull and run a registry image and create your own docker registry locally on the ec2 instance. Below command is used for the same

docker run -d -p 5000:5000 --restart always --name sourabh0905 registry

```
root@ip-172-31-91-134:/home/ubuntu# docker run -d -p 5000:5000 --restart always --name sourabh0905 registry
Unable to find image 'registry:latest' locally
latest: Pulling from library/registry
930bdd4d222e: Pull complete
a15309931e05: Pull complete
6263fb9c821f: Pull complete
86c1d3af3872: Pull complete
a37b1bf6a96f: Pull complete
Digest: sha256:12120425f07de11a1b899e418d4b0ea174c8d4d572d45bdb640f93bc7ca06a3d
Status: Downloaded newer image for registry:latest
55f920338735d126a281651ef8ac8186b32c4acb291d6bda79bb0892541b35a3
root@ip-172-31-91-134:/home/ubuntu#
```

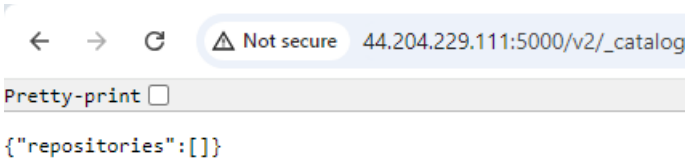
Check if the registry is created

docker ps -a

```
root@ip-172-31-91-134:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS                                       NAMES
55f920338735   registry   "/entrypoint.sh /etc…"   7 minutes ago   Up 7 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   sourabh0905
root@ip-172-31-91-134:/home/ubuntu#
```

check if registry is accessible via URL

http://44.204.229.111:5000/v2/_catalog



- Now we will pull an image from docker hub (ubuntu:latest)

  docker pull ubuntu:latest

```
root@ip-172-31-91-134:/home/ubuntu# docker pull ubuntu:latest
latest: Pulling from library/ubuntu
31e907dcc94a: Pull complete
Digest: sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@ip-172-31-91-134:/home/ubuntu#
```

Verify the image has been downloaded

docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       latest    edbfe74c41f8   4 weeks ago     78.1MB
registry     latest    cfb4d9904335   11 months ago   25.4MB
```

- We will tag the image to point to your local registry by using below command

  docker tag ubuntu:latest 44.204.229.111 (public IP of EC2):5000/ubuntu:latest

```
root@ip-172-31-91-134:/home/ubuntu# docker tag ubuntu:latest 44.204.229.111:5000/ubuntu:latest
root@ip-172-31-91-134:/home/ubuntu#
```

  Verify the image has been tagged

  docker images

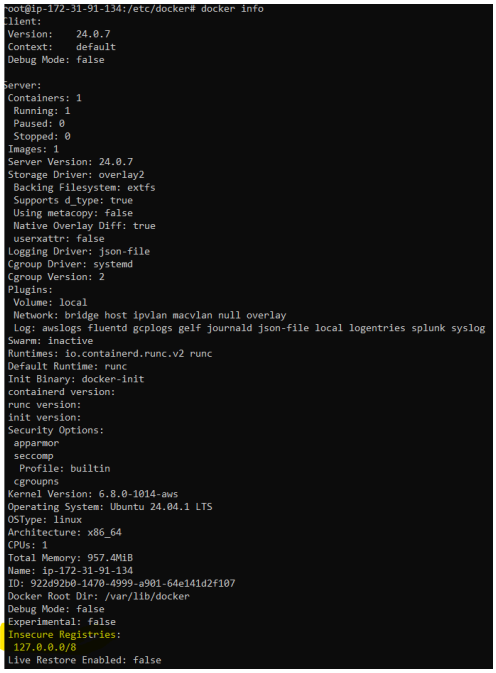
```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
44.204.229.111:5000/ubuntu   latest    edbfe74c41f8   4 weeks ago     78.1MB
ubuntu                       latest    edbfe74c41f8   4 weeks ago     78.1MB
registry                     latest    cfb4d9904335   11 months ago   25.4MB
root@ip-172-31-91-134:/home/ubuntu#
```

- Now before pushing the image to our local registry. We need to add our Public IP:5000 to the insecure registries on the ec2 instance.

- We can view the insecure registries by using the below command

  docker info

  Before adding our Public IP and port to insecure registries.



- To add our Public IP:5000 to the insecure registries on the ec2 instance, we need to create file called daemon.json under the /etc/docker

  cd /etc/docker

```
root@ip-172-31-91-134:/home/ubuntu# cd /etc/docker
root@ip-172-31-91-134:/etc/docker#
```

  vi daemon.json

```
root@ip-172-31-91-134:/etc/docker# vi daemon.json
root@ip-172-31-91-134:/etc/docker#
```

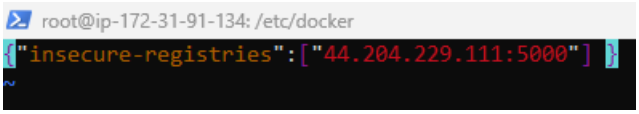
  add the lines below in the daemon.json file

  { "insecure-registries":["host:port"] }

  In our case.

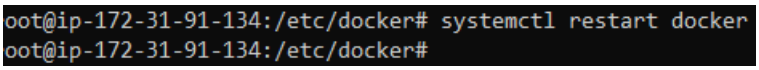
  { "insecure-registries":["22.204.229.111:5000"] }

  Esc wq!



- Need to restart docker service after creating the daemon.json file. So that changes can take effect.

  systemctl restart docker



- Now we will verify that our entry is reflecting or not by running again

  docker info

  After adding our Public IP and port to insecure registries.

```
root@ip-172-31-91-134:/etc/docker# docker info
Client:
 Version:    24.0.7
 Context:    default
 Debug Mode: false

Server:
 Containers: 1
  Running: 1
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 24.0.7
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version:
 runc version:
 init version:
 Security Options:
  apparmor
  seccomp
   Profile: builtin
  cgroupns
 Kernel Version: 6.8.0-1014-aws
 Operating System: Ubuntu 24.04.1 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 957.4MiB
 Name: ip-172-31-91-134
 ID: 922d92b0-1470-4999-a901-64e141d2f107
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Experimental: false
 Insecure Registries:
  44.204.229.111:5000
  127.0.0.0/8
 Live Restore Enabled: false
```

- Now we will push the image to our local registry running the below command.

  docker push 44.204.229.111 (public IP of EC2):5000/ubuntu:latest

  We can see that our tagged image has been pushed to the local registry.

```
root@ip-172-31-91-134:/etc/docker# docker push 44.204.229.111:5000/ubuntu:latest
The push refers to repository [44.204.229.111:5000/ubuntu]
f36fd4bb7334: Pushed
latest: digest: sha256:506f330a66a341888e5c3fbcff9f97d2b1f19a31a5c359f766bcc70a4d7bd39a size: 529
```

- Now we will verify that image is there on our local registry by running below command and going to the image location in the registry.

docker exec -it sourabh0905 (registry name) sh

-it (refers to interactive and tty stdin and stout)
sh (shell)

We can see the image located in the local registry.

```
root@ip-172-31-91-134:/home/ubuntu# docker exec -it sourabh0905 sh
/ #
/ # cd /var/lib/registry/
/var/lib/registry #
/var/lib/registry #
/var/lib/registry #
/var/lib/registry #
/var/lib/registry # ls
docker
/var/lib/registry #
/var/lib/registry #
/var/lib/registry # cd docker/
/var/lib/registry/docker #
/var/lib/registry/docker #
/var/lib/registry/docker # ls
registry
/var/lib/registry/docker #
/var/lib/registry/docker # cd registry/
/var/lib/registry/docker/registry #
/var/lib/registry/docker/registry #
/var/lib/registry/docker/registry #
/var/lib/registry/docker/registry # ls
v2
/var/lib/registry/docker/registry #
/var/lib/registry/docker/registry #
/var/lib/registry/docker/registry # cd v2/
/var/lib/registry/docker/registry/v2 #
/var/lib/registry/docker/registry/v2 #
/var/lib/registry/docker/registry/v2 #
/var/lib/registry/docker/registry/v2 # ls -ltr
total 8
drwxr-xr-x    3 root     root          4096 Aug 30 09:47 repositories
drwxr-xr-x    3 root     root          4096 Aug 30 09:47 blobs
/var/lib/registry/docker/registry/v2 #
/var/lib/registry/docker/registry/v2 #
/var/lib/registry/docker/registry/v2 # cd repositories/
/var/lib/registry/docker/registry/v2/repositories #
/var/lib/registry/docker/registry/v2/repositories #
/var/lib/registry/docker/registry/v2/repositories #
/var/lib/registry/docker/registry/v2/repositories # ls -ltr
total 4
drwxr-xr-x    5 root     root          4096 Aug 30 09:47 ubuntu
/var/lib/registry/docker/registry/v2/repositories #
/var/lib/registry/docker/registry/v2/repositories # cd ubuntu/
/var/lib/registry/docker/registry/v2/repositories/ubuntu #
/var/lib/registry/docker/registry/v2/repositories/ubuntu #
/var/lib/registry/docker/registry/v2/repositories/ubuntu # ls -ltr
total 12
drwxr-xr-x    2 root     root          4096 Aug 30 09:47 _uploads
drwxr-xr-x    4 root     root          4096 Aug 30 09:47 _manifests
drwxr-xr-x    3 root     root          4096 Aug 30 09:47 _layers
/var/lib/registry/docker/registry/v2/repositories/ubuntu #
```

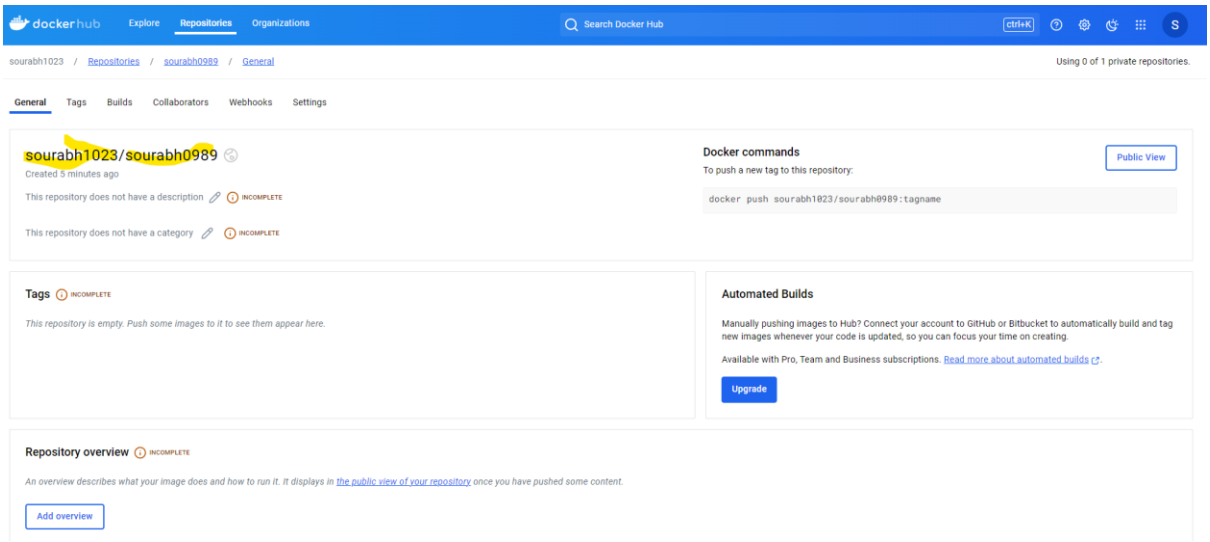
13) Push an image on Docker Hub repository. (Need to pull image, then create a tag on it and then push it back to docker)

- Before pulling the image on the ec2 instance, we need to have a dockerhub account and there we need to create a public repository.

  In my case, below are my details for dockerhub and public repository created on it.

  sourabh1023 – username for dockerhub

  sourabh0989 – public repository on docker hub



- Now we will pull an image from docker hub (ubuntu:latest)

  docker pull ubuntu:latest

```
root@ip-172-31-91-134:/home/ubuntu# docker pull ubuntu:latest
latest: Pulling from library/ubuntu
31e907dcc94a: Pull complete
Digest: sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@ip-172-31-91-134:/home/ubuntu#
```

  Verify the image has been downloaded

  docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
44.204.229.111:5000/ubuntu   latest    edbfe74c41f8   4 weeks ago     78.1MB
ubuntu                       latest    edbfe74c41f8   4 weeks ago     78.1MB
registry                     latest    cfb4d9904335   11 months ago   25.4MB
```

- We will tag the image to point to our public repository on docker hub by using below command.

  docker tag ubuntu:latest sourabh1023/sourabh0989:v1

```
root@ip-172-31-91-134:/home/ubuntu# docker tag ubuntu:latest sourabh1023/sourabh0989:v1
root@ip-172-31-91-134:/home/ubuntu#
```

  Verify the image has been tagged properly with our docker hub public repository.

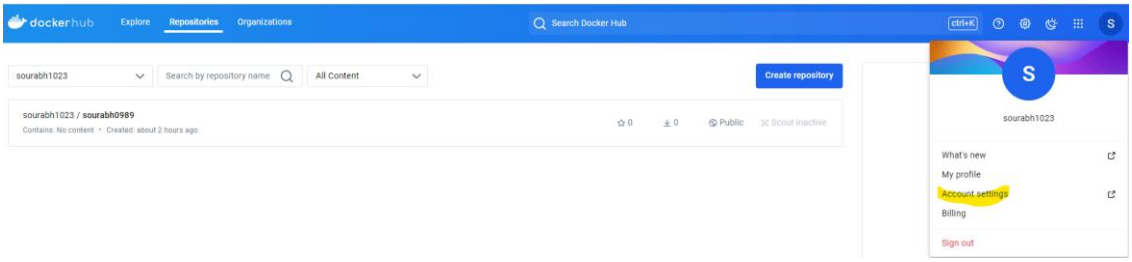
  docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
44.204.229.111:5000/ubuntu   latest    edbfe74c41f8   4 weeks ago     78.1MB
ubuntu                       latest    edbfe74c41f8   4 weeks ago     78.1MB
sourabh1023/sourabh0989      v1        edbfe74c41f8   4 weeks ago     78.1MB
registry                     latest    cfb4d9904335   11 months ago   25.4MB
root@ip-172-31-91-134:/home/ubuntu#
```

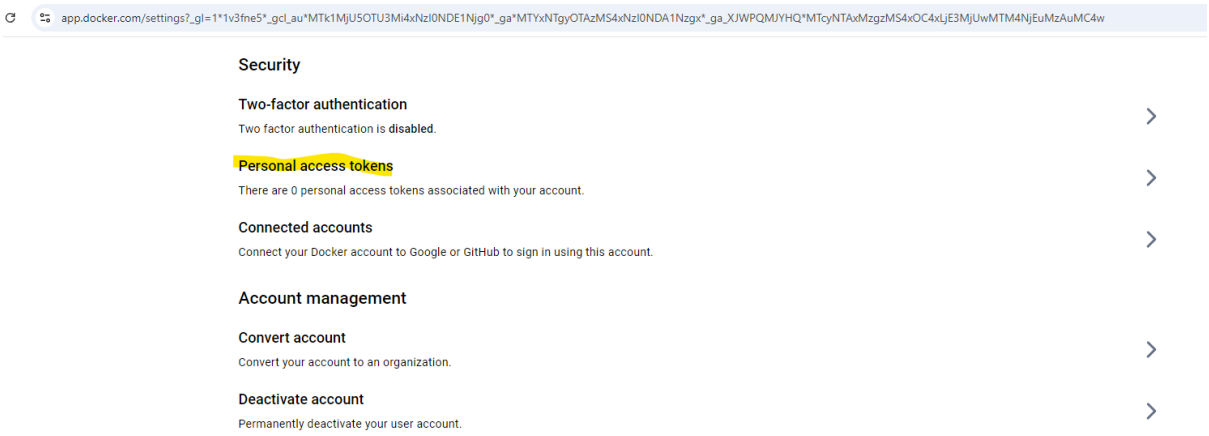
- Since its our own public repository, So to push an image, we need to enter credentials of our docker hub.

  For password, we need to generate a token from our dockerhub by following the below steps

  On your docker hub, Go to Account Settings



  Under Account Settings > Personal access tokens

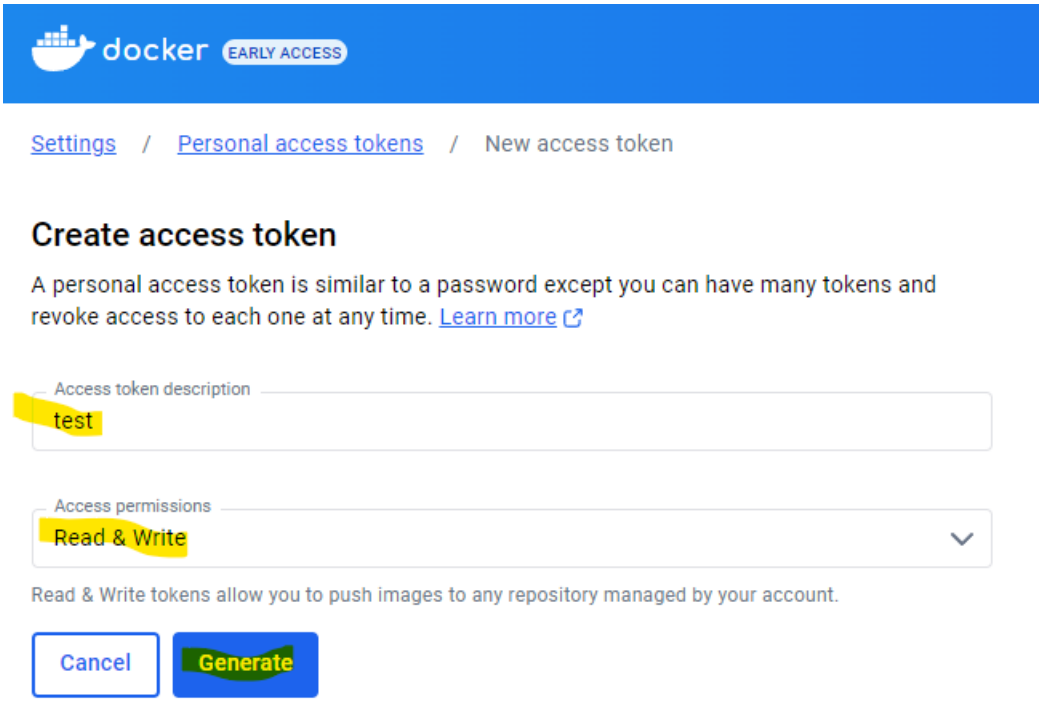


There will be new access token creation request.

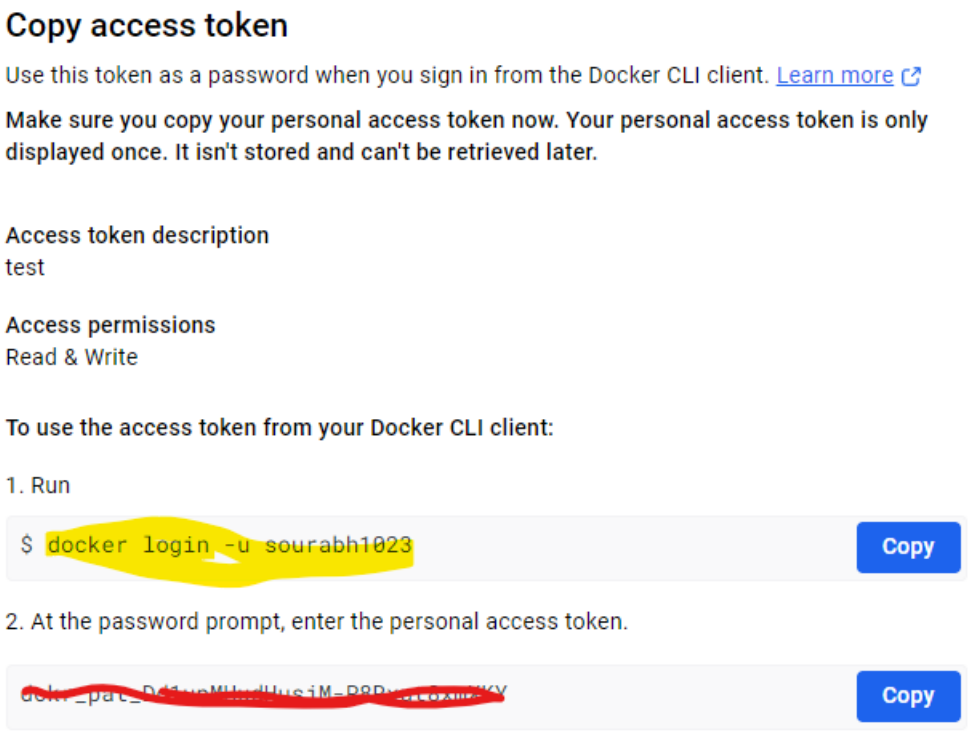
Fill the token details

Under access and permissions select **Read & Write**

click generate



You will get the below screen with token details. Copy them on ec2 instance to enter the public repository.



- Now we will proceed with docker login by using the commands in the above screenshot.

  docker login -u sourabh1023

```
root@ip-172-31-91-134:/home/ubuntu# docker login -u sourabh1023
Password:
```

  Password:

  Copy and enter the token

  Login succeeded

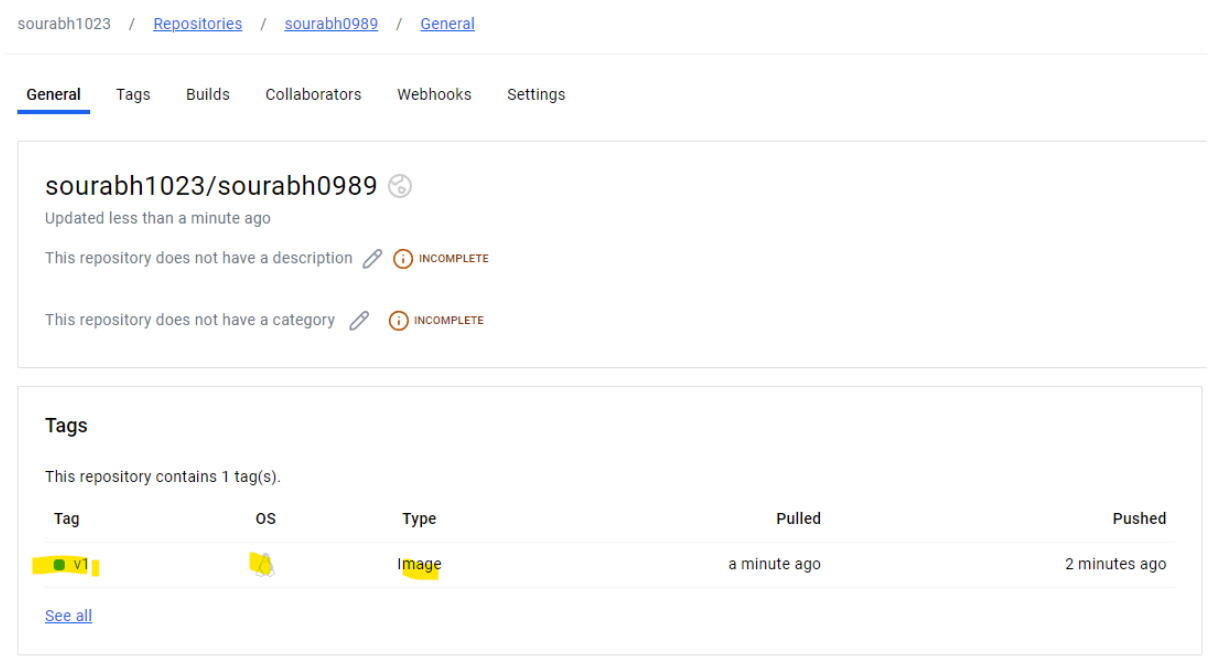
```
oot@ip-172-31-91-134:/home/ubuntu# docker login -u sourabh1023
assword:
ARNING! Your password will be stored unencrypted in /root/.docker/config.json.
onfigure a credential helper to remove this warning. See
ttps://docs.docker.com/engine/reference/commandline/login/#credentials-store

ogin Succeeded
oot@ip-172-31-91-134:/home/ubuntu#
```

- Now we will push the tagged image on our public repository using the below command.

  docker push sourabh1023/sourabh0989:v1

```
root@ip-172-31-91-134:/home/ubuntu# docker push sourabh1023/sourabh0989:v1
The push refers to repository [docker.io/sourabh1023/sourabh0989]
f36fd4bb7334: Mounted from library/ubuntu
v1: digest: sha256:49ba8c8c2821c5b894a5bcff0a7873d6c58df7a03c45ee7c3cda159795f6e80f size: 529
```

- We will now verify that the image is pushed on our repository. Below screenshot confirms the same.



14) Unused Objects or Dangling Images

Images are hanging around and it does not have a proper tag or name right now. (You can check the image ids to see that this is the same image we built previously).

Docker calls such images dangling images.

Example:

- We will create a docker file with the below entry.

  vi dockerfile

```
root@ip-172-31-91-134:/home/ubuntu# vi dockerfile
```

  FROM ubuntu:latest
  CMD ["echo", "Hello World"]

```
root@ip-172-31-91-134: /home/ubur
FROM ubuntu:latest
CMD ["echo", "Hello World"]
```

- We will build an image from the dockerfile using the below command

  docker build -t my-image .

```
root@ip-172-31-91-134:/home/ubuntu# docker build -t my-image .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  382.3MB
Step 1/2 : FROM ubuntu:latest
 ---> edbfe74c41f8
Step 2/2 : CMD ["echo", "Hello World"]
 ---> Running in 6fa2fc7d9aa5
Removing intermediate container 6fa2fc7d9aa5
 ---> 69ec524d2b88
Successfully built 69ec524d2b88
Successfully tagged my-image:latest
```

  We will verify whether our image is created.

  docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY        TAG         IMAGE ID       CREATED         SIZE
my-image          latest      69ec524d2b88   6 minutes ago   78.1MB
jenkins/jenkins   lts-jdk17   7bf0dfc06ae6   3 weeks ago     469MB
tomcat            latest      c2a444ea6cd7   3 weeks ago     508MB
ubuntu            latest      edbfe74c41f8   4 weeks ago     78.1MB
hello-world       latest      d2c94e258dcb   16 months ago   13.3kB
```

- Now we will edit the same docker file and make some changes.

  vi dockerfile

```
root@ip-172-31-91-134:/home/ubuntu# vi dockerfile
```

FROM ubuntu:latest
CMD ["echo", "Hello, World!"]

- Now we will create image from the same dockerfile but will not give it any name this time by using the below command

  docker build . [. Signifies docker file is in current folder]

```
root@ip-172-31-91-134:/home/ubuntu# docker build .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  382.3MB
Step 1/2 : FROM ubuntu:latest
 ---> edbfe74c41f8
Step 2/2 : CMD ["echo", "Hello, World!"]
 ---> Running in d0c69391b5a4
Removing intermediate container d0c69391b5a4
 ---> 30a349799024
Successfully built 30a349799024
```

We will verify whether our image is created.

docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY        TAG         IMAGE ID       CREATED          SIZE
<none>            <none>      30a349799024   9 minutes ago    78.1MB
my-image          latest      69ec524d2b88   25 minutes ago   78.1MB
jenkins/jenkins   lts-jdk17   7bf0dfc06ae6   3 weeks ago      469MB
tomcat            latest      c2a444ea6cd7   3 weeks ago      508MB
ubuntu            latest      edbfe74c41f8   4 weeks ago      78.1MB
hello-world       latest      d2c94e258dcb   16 months ago    13.3kB
```

This image which is created with <none> <none> is called a dangling image

- We can use the following command to list dangling images:

  docker images --filter "dangling=true"

```
root@ip-172-31-91-134:/home/ubuntu# docker images --filter "dangling=true"
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
<none>       <none>    30a349799024   14 minutes ago   78.1MB
root@ip-172-31-91-134:/home/ubuntu#
```

- We can use the following command to delete the dangling images:

  docker rmi $(docker images -q --filter "dangling=true")

```
root@ip-172-31-91-134:/home/ubuntu# docker rmi $(docker images -q --filter "dangling=true")
Deleted: sha256:30a3497990243cf566e1690384fd1f1bc1d2f3717f1d6012e927ece9c3f764c2
```

  We will verify whether the dangling image is removed or not.

  docker images

```
root@ip-172-31-91-134:/home/ubuntu# docker images
REPOSITORY        TAG         IMAGE ID       CREATED          SIZE
my-image          latest      69ec524d2b88   34 minutes ago   78.1MB
jenkins/jenkins   lts-jdk17   7bf0dfc06ae6   3 weeks ago      469MB
tomcat            latest      c2a444ea6cd7   3 weeks ago      508MB
ubuntu            latest      edbfe74c41f8   4 weeks ago      78.1MB
hello-world       latest      d2c94e258dcb   16 months ago    13.3kB
```

  The image which is created with <none> <none> is deleted now.

15) Create Docker Swarm with 3 workers and shutdown the master and see if leader is relected or not.

- To perform the activity, we need at least, 4 EC2 instances with docker installed on them, we will create 1 Master or Leader (Manager) and other will be workers in the docker swarm. We have defined the hostname to avoid any confusion.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IP
DM	i-008493b22e40df5df	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-44-202-64-124.co...	44.202.64
DW1	i-02e817b3928b79f7b	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-44-211-194-9.com...	44.211.19
DW2	i-0663ada3683446a4a	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-44-201-107-140.co...	44.201.10
DW3	i-07b79cba6fc192ca6	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b	ec2-54-85-111-119.co...	54.85.11

- Now setup password less ssh between master and worker by following below steps.

  On master: Type **ssh-keygen** and follow all steps as follows to generate public key and private key

```
root@DM:/home/ubuntu# ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:bHWKHxobFHSZsZQANfrIJVB9ShHTu817AfIIW2ZYz9E root@DM
The key's randomart image is:
+--[ED25519 256]--+
|     ..o=X=+= .  |
|      . .o*B.. E |
|       o.o=o+..  |
|      . B+o*o+   |
|       o S*oB .  |
|        ..*o.+ . |
|         o .  . .|
|              . .|
|               . |
+----[SHA256]-----+
root@DM:/home/ubuntu#
```

Now do **cat /root/.ssh/id_ed25519.pub** and copy all the contents inside it.

```
root@DM:/home/ubuntu# cat /root/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAv/PmXhM74IHrqg2D55re/Lz2x1KP0aol3w2lObBv8U root@DM
```

On all Workers: Do **cat >> authorized_keys** (append authorized_keys file and paste the contents copied from master server)

```
root@DW1:/home/ubuntu# cat >> /root/.ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAv/PmXhM74IHrqg2D55re/Lz2x1KP0aol3w2lObBv8U root@DM
^C
root@DW2:/home/ubuntu# cat >> /root/.ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAv/PmXhM74IHrqg2D55re/Lz2x1KP0aol3w2lObBv8U root@DM
^C
root@DW3:/home/ubuntu# cat >> /root/.ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIAv/PmXhM74IHrqg2D55re/Lz2x1KP0aol3w2lObBv8U root@DM
^C
```

Now we will ssh to all worker nodes from master using the Public IP Address.

Worker Node 1

ssh PublicIP

```
root@DM:/home/ubuntu# ssh 44.211.194.9
root@DW1:~#
root@DW1:~# exit
logout
Connection to 44.211.194.9 closed.
root@DM:/home/ubuntu#
```

Worker Node 2

ssh PublicIP

```
root@DM:/home/ubuntu# ssh 44.201.107.140
root@DW2:~#
root@DW2:~# exit
logout
Connection to 44.201.107.140 closed.
root@DM:/home/ubuntu#
```

Worker Node 3

ssh PublicIP

```
root@DM:/home/ubuntu# ssh 54.85.111.119
root@DW3:~# exit
logout
Connection to 54.85.111.119 closed.
root@DM:/home/ubuntu#
```

All three worker nodes have been verified

- Run the below command to master node to make it manager IP

docker swarm init --advertise-addr (Public IP of Master Node)

```
root@DM:/home/ubuntu# docker swarm init --advertise-addr 44.202.64.124
Swarm initialized: current node (l7u31xibq6azml6s9nawhwo7c) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-29hiy2c7xm9o9lotllwhp4s4ha2hivtfel87u3rp9ezwrtvule-bc06emfx5x1r3ye0u6tokauzn 44.202.64.124:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

root@DM:/home/ubuntu#
```

- Now run the next command highlighted the above screenshot on all the worker nodes so that worker nodes join swarm as worker.

  Command differs scenario to scenario as token will be different in different cases.

  DW1

```
root@DW1:/home/ubuntu# docker swarm join --token SWMTKN-1-29hiy2c7xm9o9lotllwhp4s4ha2hivtfel87u3rp9ezwrtvule-bc06emfx5
x1r3ye0u6tokauzn 44.202.64.124:2377
This node joined a swarm as a worker.
root@DW1:/home/ubuntu#
```

  DW2

```
root@DW2:/home/ubuntu# docker swarm join --token SWMTKN-1-29hiy2c7xm9o9lotllwhp4s4ha2hivtfel87u3rp9ezwrtvule-bc06emfx5
x1r3ye0u6tokauzn 44.202.64.124:2377
This node joined a swarm as a worker.
root@DW2:/home/ubuntu#
```

  DW3

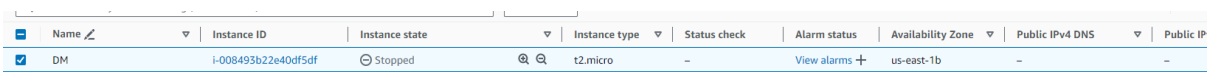
```
root@DW3:/home/ubuntu# docker swarm join --token SWMTKN-1-29hiy2c7xm9o9lotllwhp4s4ha2hivtfel87u3rp9ezwrtvule-bc06emfx5
x1r3ye0u6tokauzn 44.202.64.124:2377
This node joined a swarm as a worker.
root@DW3:/home/ubuntu#
```

- To verify whether the node is joined as worker, we will run the below command from master.

  docker node ls

```
root@DM:/home/ubuntu# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
l7u31xibq6azml6s9nawhwo7c *   DM         Ready     Active         Leader           24.0.7
i9c0brl4qfi3hvhy9r4rmrmmw     DW1        Ready     Active                          24.0.7
xv0cwovq61aovnpapckp8l99f     DW2        Ready     Active                          24.0.7
eftos3tf2e3sseo93mb1zj60z     DW3        Ready     Active                          24.0.7
root@DM:/home/ubuntu#
```

- Now we stop the EC2 instance DM, which is currently showing as the Leader node, so see if the re-election happens automatically or not.



- If we run docker node ls on any of the worker nodes e.g. on DW1, it gives the error as in the below screenshot.

  DW1

  docker node ls

```
root@DW1:/home/ubuntu# docker node ls
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify cluster state. Please run this command on a manager node or promo
te the current node to a manager.
root@DW1:/home/ubuntu#
```

  [Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify cluster state. Please run this command on a manager node or promote the current node to a manager.]

  The same result comes even if we run the command on other worker nodes DW2 and DW3

- This shows that the manager/leader re-election does not happens automatically. In order to get the new manager/leader, we need promote any one of worker nodes to be the new manager.

16) Docker Uninstallation

- To remove/ Uninstall docker completely we will follow the below sub steps.

  - Removing Docker Images, Containers, Volumes, and Configurations.

    Run the below command to remove running docker containers by first stopping them and later removing them

    docker stop $(docker ps -a -q) (Stops all Running Containers)

```
root@DM:/home/ubuntu# docker stop $(docker ps -a -q)
f5bf2d13cbaa
4c0f0f1a766f
root@DM:/home/ubuntu#
```

To verify, we run

docker ps -a

```
root@DM:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS                       PORTS     NAMES
f5bf2d13cbaa   hello-world   "/hello"      26 minutes ago   Exited (0) 26 minutes ago              flamboyant_chatelet
4c0f0f1a766f   ubuntu        "/bin/bash"   29 minutes ago   Exited (137) 59 seconds ago            elated_feistel
root@DM:/home/ubuntu#
```

docker rm $(docker ps -a -q) (Removes all the containers)

```
root@DM:/home/ubuntu# docker rm $(docker ps -a -q)
f5bf2d13cbaa
4c0f0f1a766f
root@DM:/home/ubuntu#
```

To verify, we run

docker ps -a

```
root@DM:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@DM:/home/ubuntu#
```

Now we will remove the docker images

Before Running the below command, run below command to see the images present.

docker images

```
root@DM:/home/ubuntu# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
tomcat        latest    c2a444ea6cd7   4 weeks ago      508MB
ubuntu        latest    edbfe74c41f8   4 weeks ago      78.1MB
hello-world   latest    d2c94e258dcb   16 months ago    13.3kB
root@DM:/home/ubuntu#
```

Run the below command to remove the docker images

docker rmi $(docker images -a -q) (Removes all docker images)

```
root@DM:/home/ubuntu# docker rmi $(docker images -a -q)
Untagged: tomcat:latest
Untagged: tomcat@sha256:221f95c46bbd428f2c7bce68ecc421fb68f21844f3e9e0f15cf5e7d955929b1a
Deleted: sha256:c2a444ea6cd7a0df0600cbd7ed249e1b5f8635ec7f55420a07444703f0aab70e
Deleted: sha256:ad412476353a84fb389fc0b8518e5fba5625713979278c7111952109f1e31eb8
Deleted: sha256:4713de0247283194e6dcadae2d4b56f4bb59424dbdfc6037c656652b5ccba00d
Deleted: sha256:f980689d29f2b469d030207eab9a1b28ae6134552419369f9db6d16f4a471fe4
Deleted: sha256:4e0bdee550bd42d8c365c162b98c391bddc80e2fd8519c3ba82224209f681d6c
Deleted: sha256:71623ba9f7420c5e06c17111084ba402ea512b6fac2514efa7514794badcdb9c
Deleted: sha256:35e24400f7eebd8e6c8912fb48a8b2980da4c20f97ae846503f7dc45c3ad0849
Deleted: sha256:957d5e61eb7b5905aa0c0e4899b31421853173309e9673df475c27ed0511ee9d
Deleted: sha256:b868c9ba10cbd428f15bcdc53691fea7f544a2d7a137721f8ee061b12bbe33ba
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:8a37d68f4f73ebf3d4efafbcf66379bf3728902a8038616808f04e34a9ab63ee
Deleted: sha256:edbfe74c41f8a3501ce542e137cf28ea04dd03e6df8c9d66519b6ad761c2598a
Deleted: sha256:f36fd4bb7334b7ae3321e3229d103c4a3e7c10a263379cc6a058b977edfb46de
Untagged: hello-world:latest
Untagged: hello-world@sha256:53cc4d415d839c98be39331c948609b659ed725170ad2ca8eb36951288f81b75
Deleted: sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
Deleted: sha256:ac28800ec8bb38d5c35b49d45a6ac4777544941199075dff8c4eb63e093aa81e
root@DM:/home/ubuntu#
```

To verify, we run

docker images

```
root@DM:/home/ubuntu# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
root@DM:/home/ubuntu#
```

Remove all docker networks

docker network prune (Remove all docker networks)

```
root@DM:/home/ubuntu# docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
docker_gwbridge
```

Below command cleans up unused Docker components, freeing space but also potentially resulting in data loss.

docker system prune -a

```
root@DM:/home/ubuntu# docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
root@DM:/home/ubuntu#
```

The command docker system prune -a is a powerful tool that removes:

- All stopped containers
- All networks not used by at least one container
- All volumes not used by at least one container
- All images without at least one container associated with them
- All build cache

➢ Delete Docker Packages

Below command lists out the docker packages on our system

dpkg -l | grep -i docker (lists out the docker packages on our system)

```
root@DM:/home/ubuntu# dpkg -l | grep -i docker
ii  docker.io                      24.0.7-0ubuntu4.1                amd64        Linux container runtime
root@DM:/home/ubuntu#
```

In our case, we only have docker.io

Upon identifying the docker packages, now go ahead and delete them using the below command

apt-get purge docker.io (deletes the docker packages installed on our system

```
root@DM:/home/ubuntu# apt-get purge docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  docker.io*
0 upgraded, 0 newly installed, 1 to remove and 2 not upgraded.
After this operation, 109 MB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 98762 files and directories currently installed.)
Removing docker.io (24.0.7-0ubuntu4.1) ...
'/usr/share/docker.io/contrib/nuke-graph-directory.sh' -> '/var/lib/docker/nuke-graph-directory.sh'
Stopping 'docker.service', but its triggering units are still active:
docker.socket
Processing triggers for man-db (2.12.0-4build2) ...
(Reading database ... 98551 files and directories currently installed.)
Purging configuration files for docker.io (24.0.7-0ubuntu4.1) ...

Nuking /var/lib/docker ...
  (if this is wrong, press Ctrl+C NOW!)

+ sleep 10

+ rm -rf /var/lib/docker/buildkit /var/lib/docker/containers /var/lib/docker/engine-id /var/lib/docker/image /var/lib/docker/network /var/lib/docker/nuke-graph-directory.sh
 /var/lib/docker/overlay2 /var/lib/docker/plugins /var/lib/docker/runtimes /var/lib/docker/swarm /var/lib/docker/tmp /var/lib/docker/volumes
root@DM:/home/ubuntu#
```

➢ Deleting the remaining docker files

We need execute the commands to completely remove docker and its related files from your system.

All the docker files will be in the /var/lib/docker path so you have to remove the folder.

rm -rf /var/lib/docker /etc/docker (Removes all files and directories from both the path)

```
root@DM:/home/ubuntu# rm -rf /var/lib/docker /etc/docker
root@DM:/home/ubuntu#
```

Delete the docker group that you have created in order to run the docker as non-root user or without sudo

groupdel docker (Removes the group)

```
root@DM:/home/ubuntu# groupdel docker
root@DM:/home/ubuntu#
```

Delete the docker socket using the command below

rm -rf /var/run/docker.sock (Removes the docker.sock)

```
root@DM:/home/ubuntu# rm -rf /var/run/docker.sock
root@DM:/home/ubuntu#
```

With this, the uninstallation of docker will be successful. You can verify if the docker has been uninstalled by typing docker command. Upon successful uninstallation of docker, you will see the below output.

docker

```
root@DM:/home/ubuntu# docker
bash: /usr/bin/docker: No such file or directory
root@DM:/home/ubuntu#
```

- Cleanup unused packages

  This command removes packages that were installed as dependencies and are no longer used and that's it, you have successfully removed docker from your system

  apt-get autoremove (Removes all dependencies that are no longer used)

```
root@DM:/home/ubuntu# apt-get autoremove
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
0 upgraded, 0 newly installed, 7 to remove and 2 not upgraded.
After this operation, 180 MB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 98551 files and directories currently installed.)
Removing ubuntu-fan (0.12.16) ...
ubuntu-fan: removing default /etc/network/fan configuration
Removing bridge-utils (1.7.1-1ubuntu2) ...
Removing containerd (1.7.12-0ubuntu4.1) ...
Removing dns-root-data (2023112702~willsync1) ...
Removing dnsmasq-base (2.90-2build2) ...
Removing pigz (2.8-1) ...
Removing runc (1.1.12-0ubuntu3.1) ...
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
root@DM:/home/ubuntu#
```

- Docker has been successfully uninstalled and removed from the system.

17) Docker installs without internet

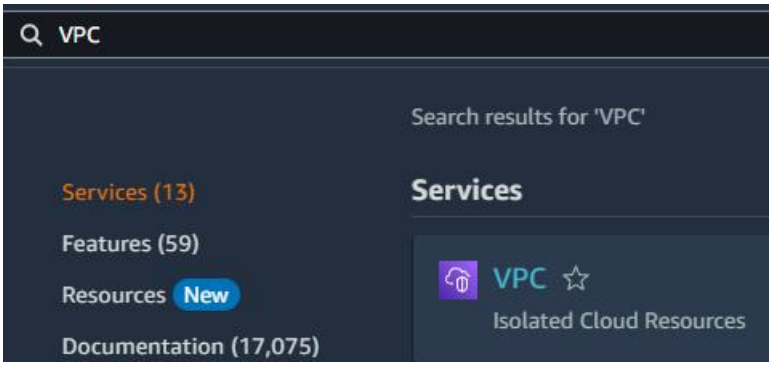
- There are multiple ways where we can install docker without internet access to the EC2 instance.

- One of the methods in AWS is to setup your own VPC, define public subnet with internet gateway attached and private subnet with NAT Gateway attached so that, EC2 connected to private subnet or network can download docker through NAT Gateway.

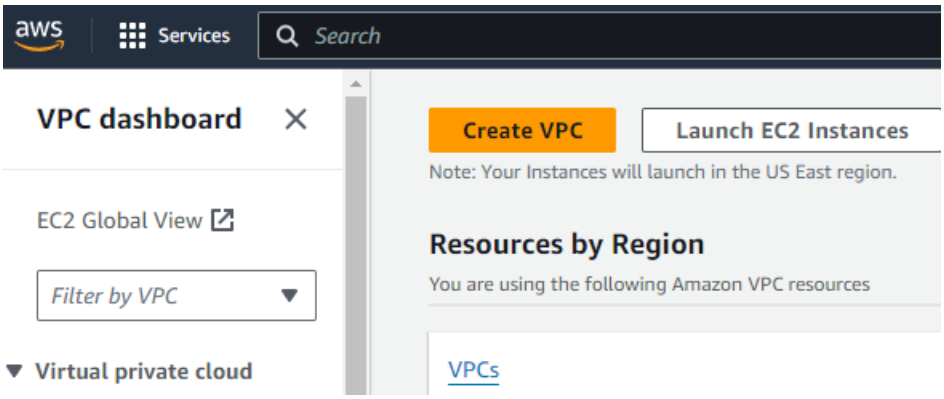
- Steps to perform the above are as follows.

  - Create your own VPC with a network range 10.0.0.0/16

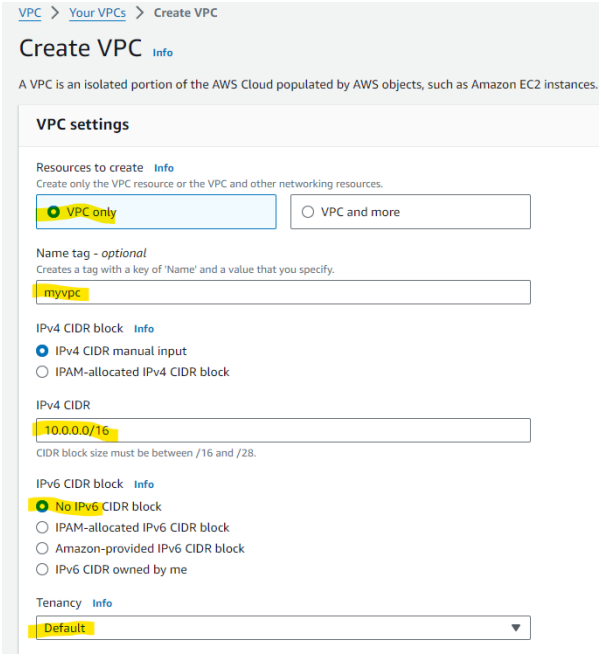
    In AWS console, search for VPC and click on it.

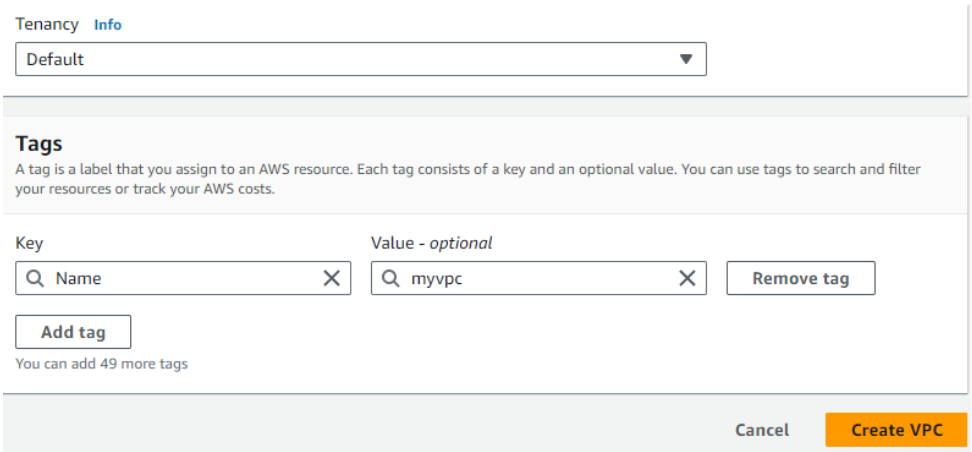


    Then click Create VPC

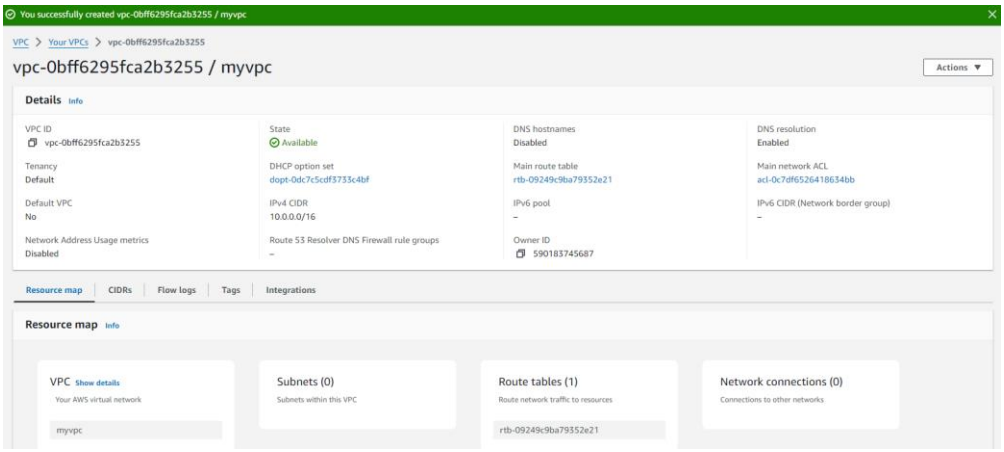


    Under Create VPC, select the below settings and provide a name for your VPC with cidr range 10.0.0.0/16. In our case, myvpc is the name of our VPC, then click create VPC.



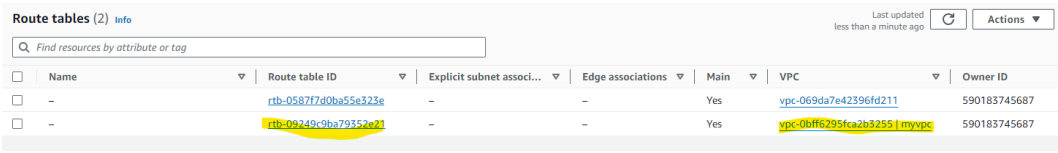


VPC is successfully created.

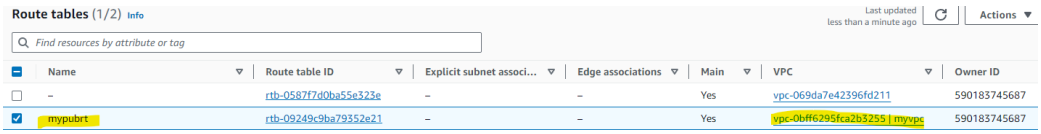


➢ Now we will go to Route Tables under VPC dashboard.

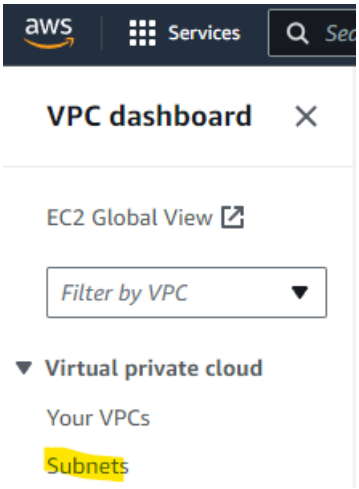
We will see there is Route table created with our VPC



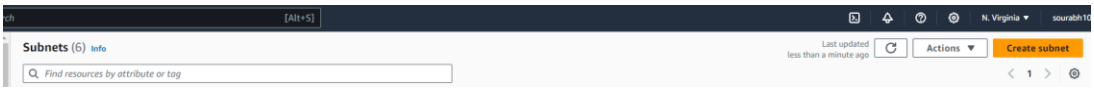
We will rename it as mypubrt (my public route table)



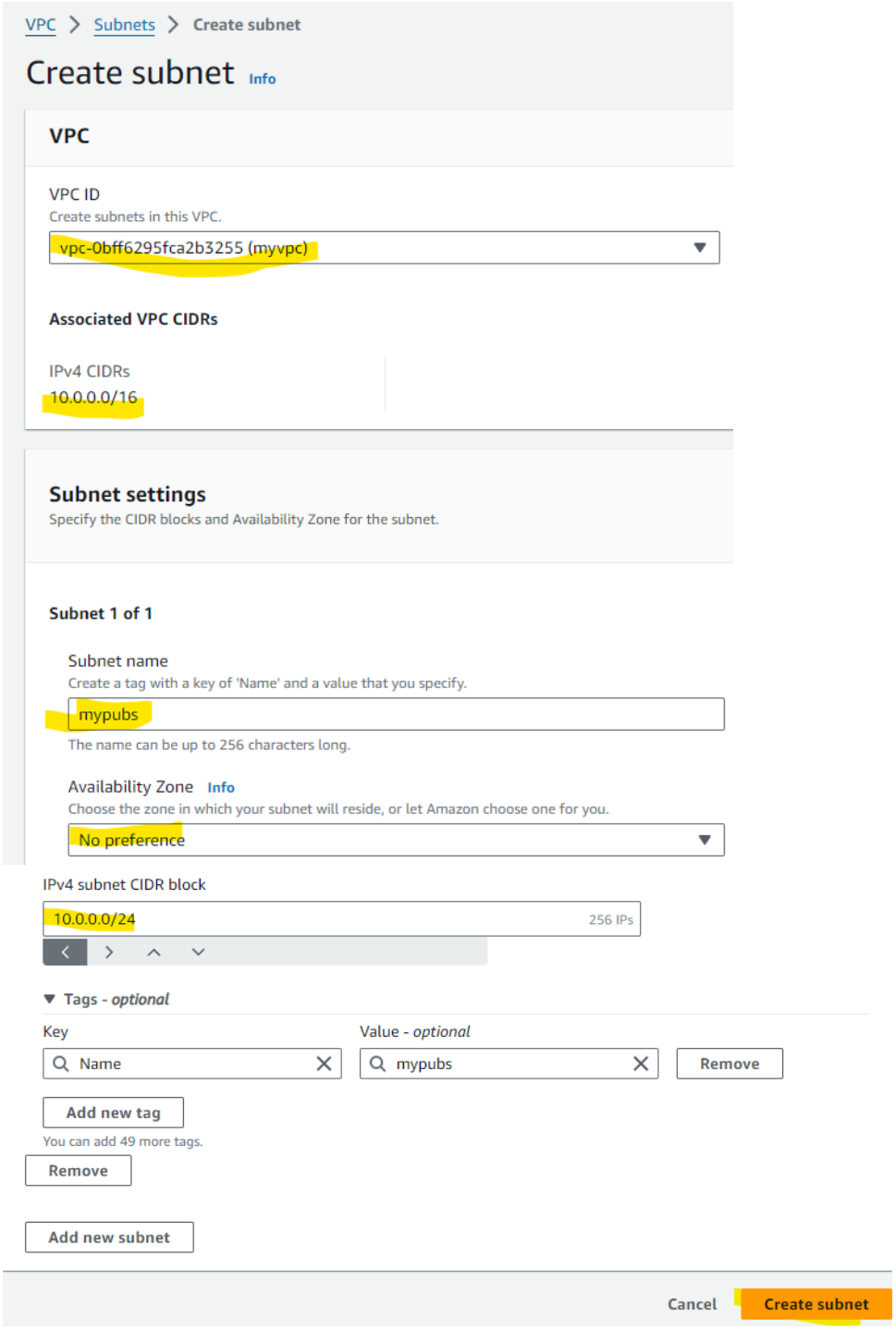
- Now we will go to subnets under VPC dashboard.



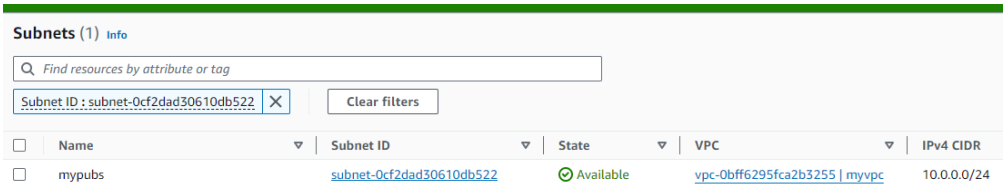
Click Create Subnet



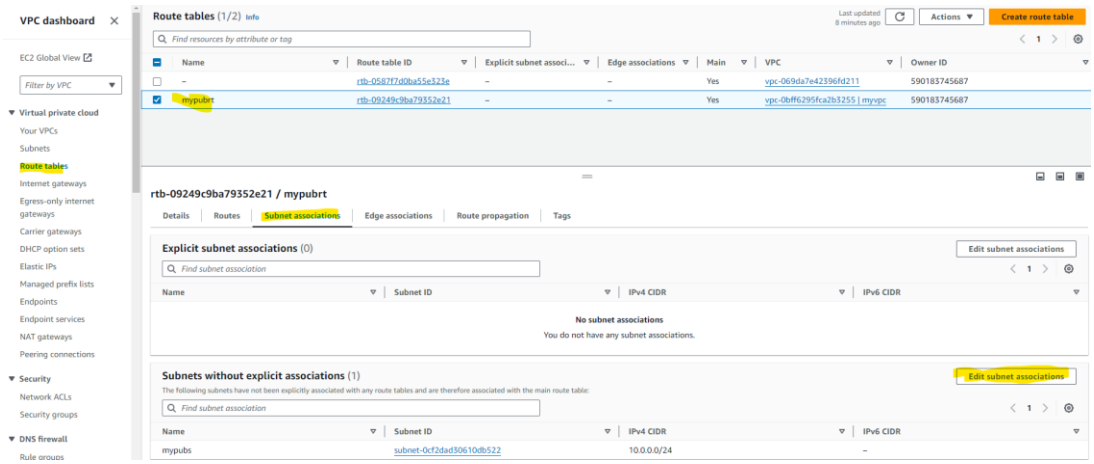
We will create a public subnet, attach it to the VPC we created and give it a name mypubs (my public subnet) and give it a cidr range 10.0.0.0/24 and click create subnet



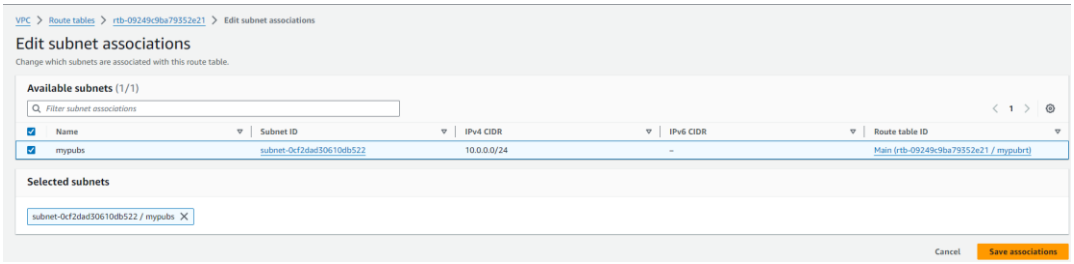
Public Subnet created successfully



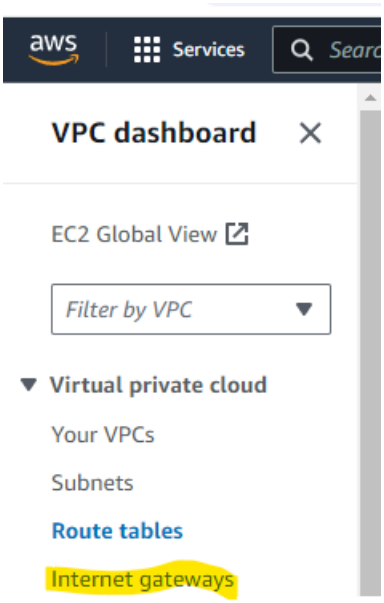
- Now again we will go to Route tables and under our Public Route Table (mypubrt) > subnet associations > Edit subnet associations



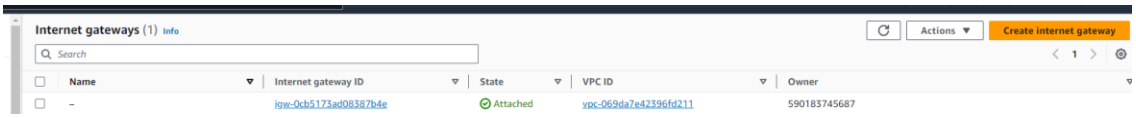
Click mypubs subnet and save associations.



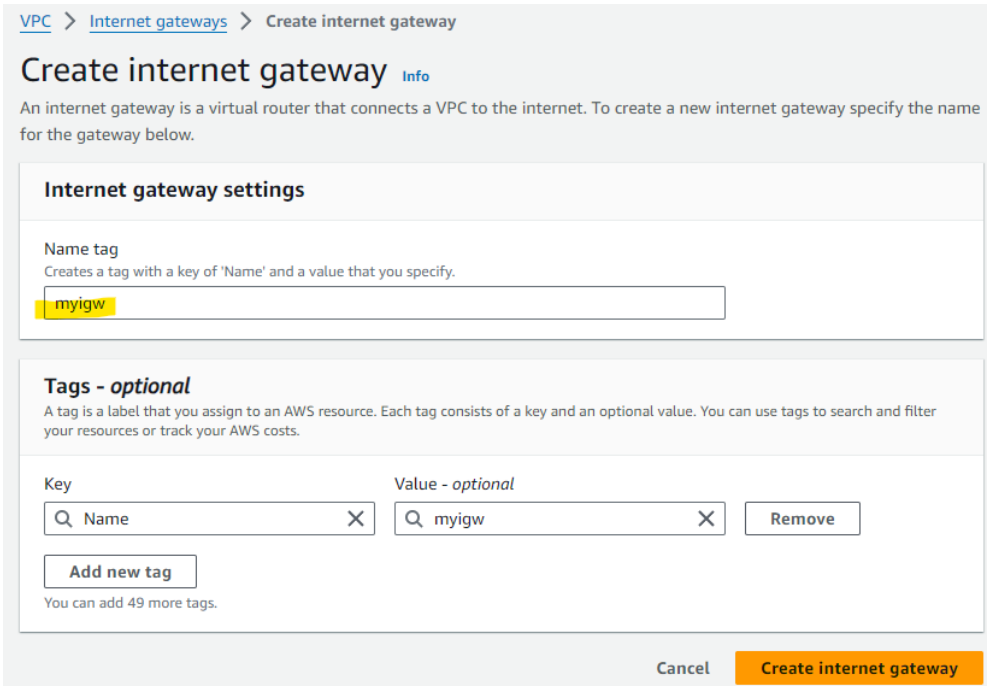
- Now we will go to Internet gateway under VPC dashboard.

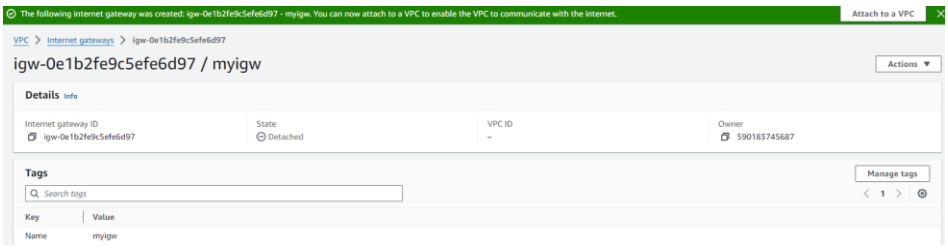


Click Create internet gateway.



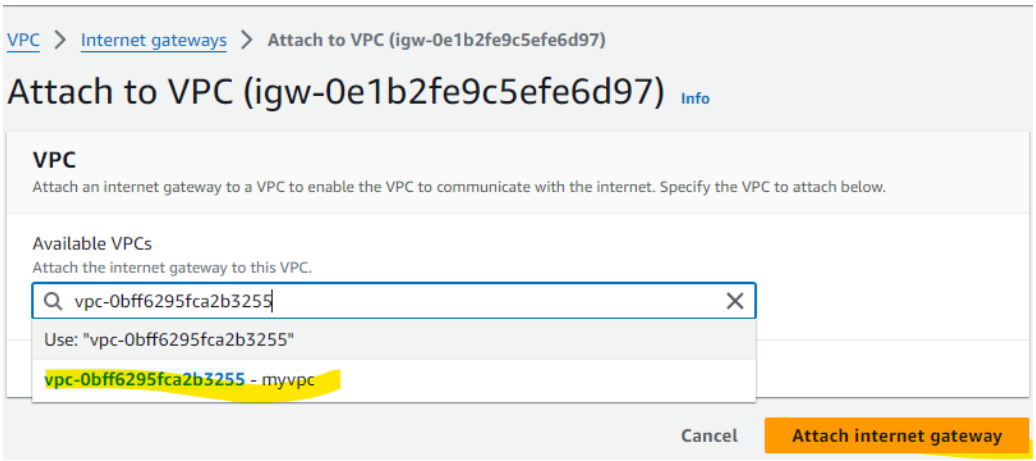
We will create an internet gateway, attach it to the VPC we created and give it a name myigw (my internet gateway) and click create internet gateway.



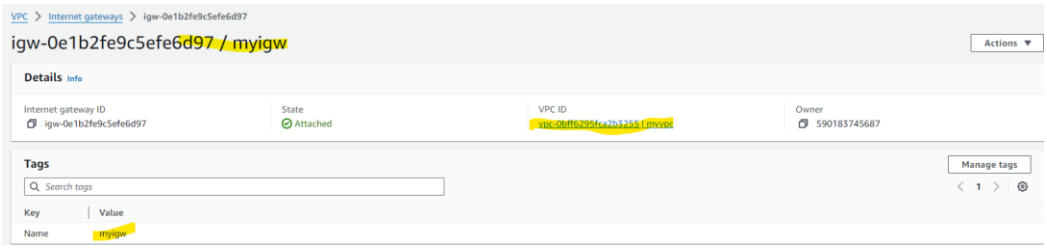


Now we attach the internet gateway to our VPC by clicking on Attach to VPC in the below screenshot.

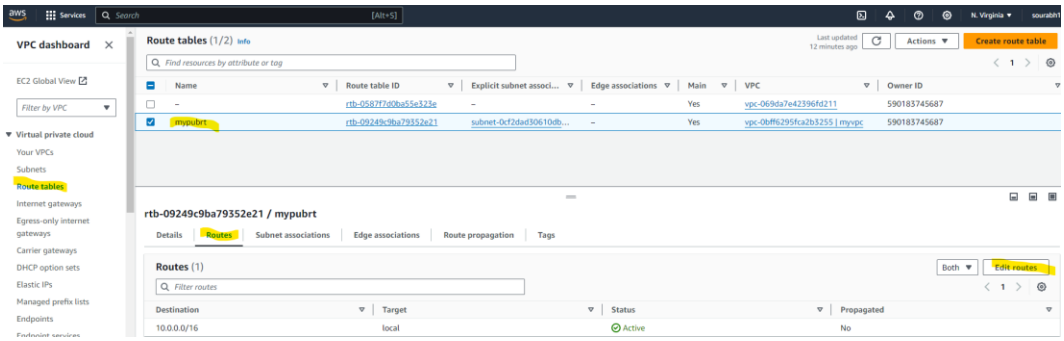
Select our VPC from the drop-down list and click attach internet gateway.



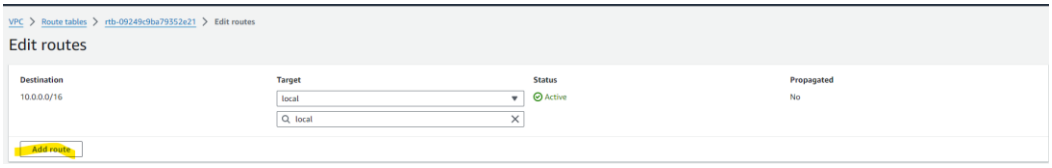
VPC attached successfully



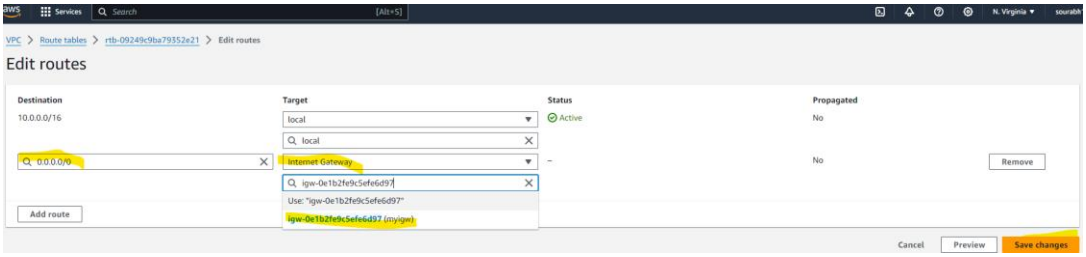
- Now again we will go to Route tables and under our Public Route Table (mypubrt) > Routes > Edit Routes



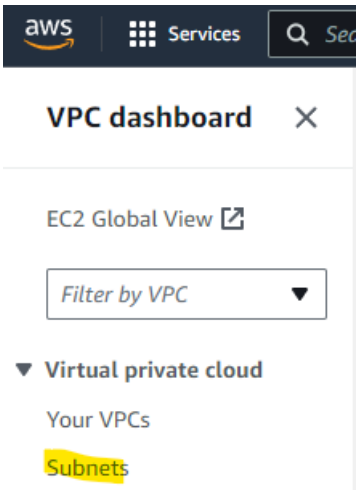
Click on Add Route



Under Destination we will give 0.0.0.0/0 and under target we will give internet gateway and attach the internet gateway (myigw) that we created and click save changes.

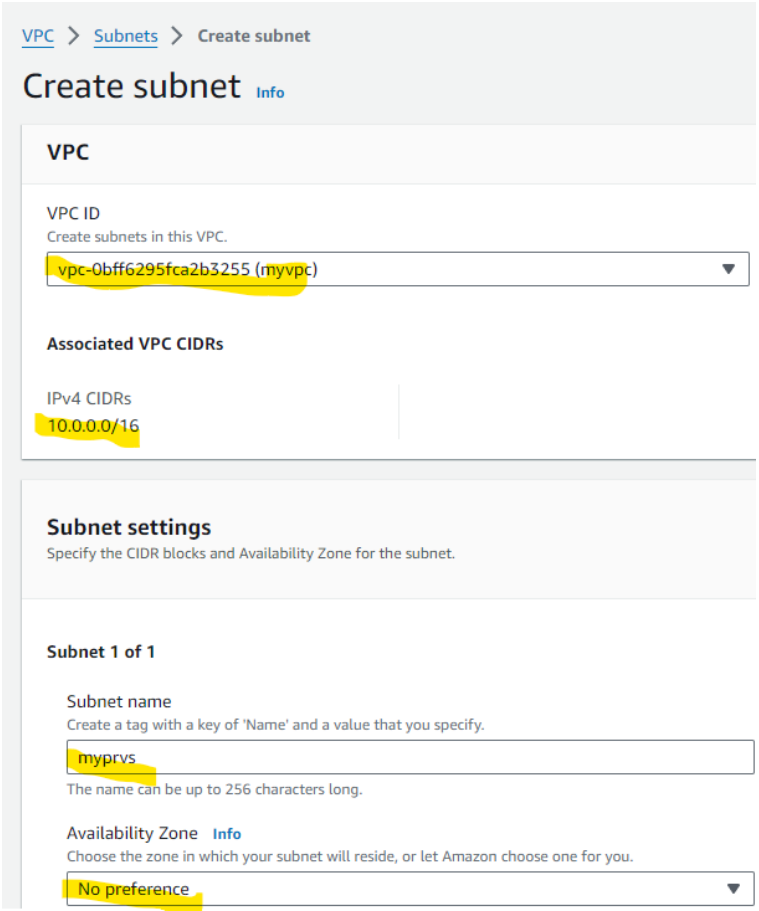


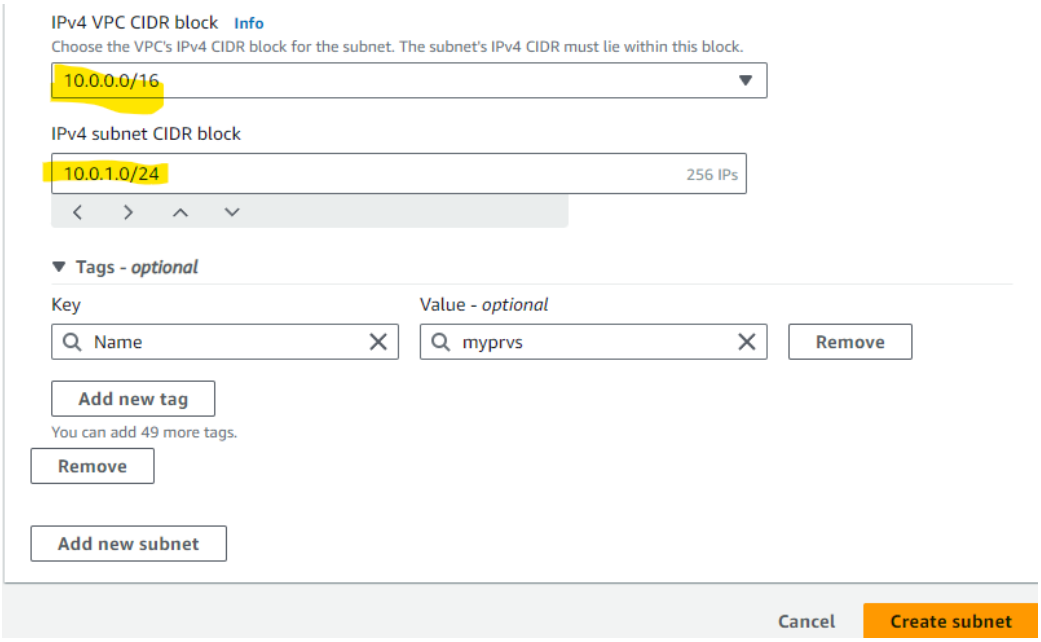
- Now we will go to subnets under VPC dashboard.



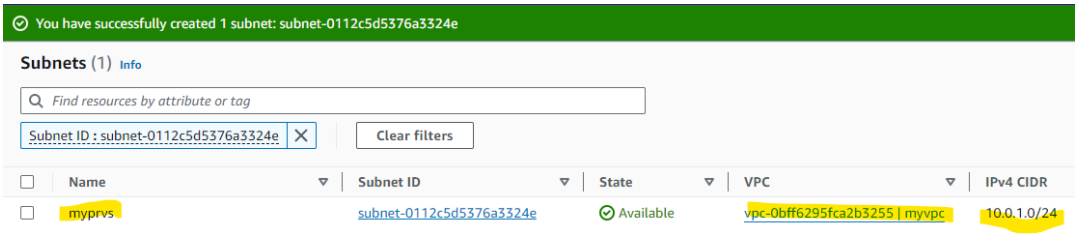
Click Create Subnet

We will create a private subnet, attach it to the VPC we created and give it a name myprvs (my private subnet) and give it a cidr range 10.0.1.0/24 and click create subnet

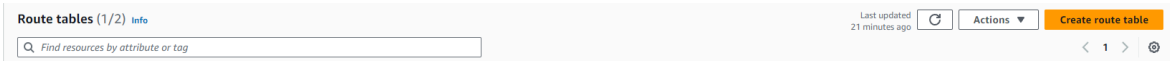




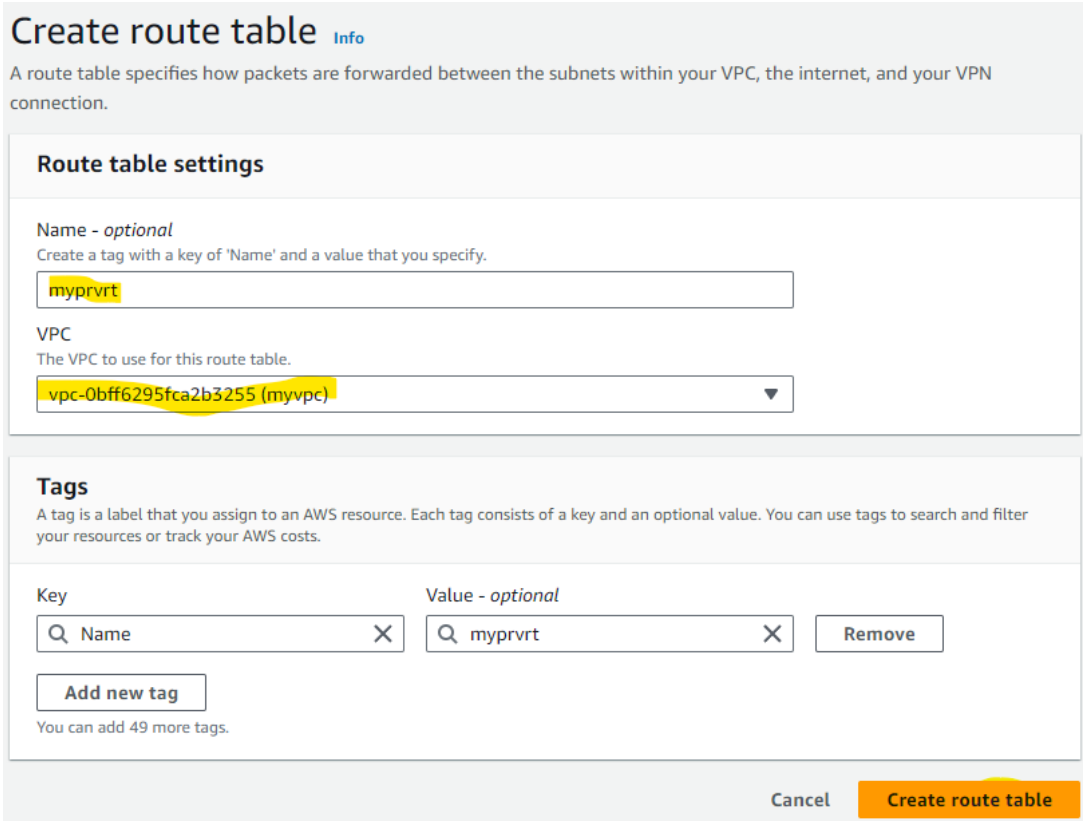
Private Subnet created successfully.



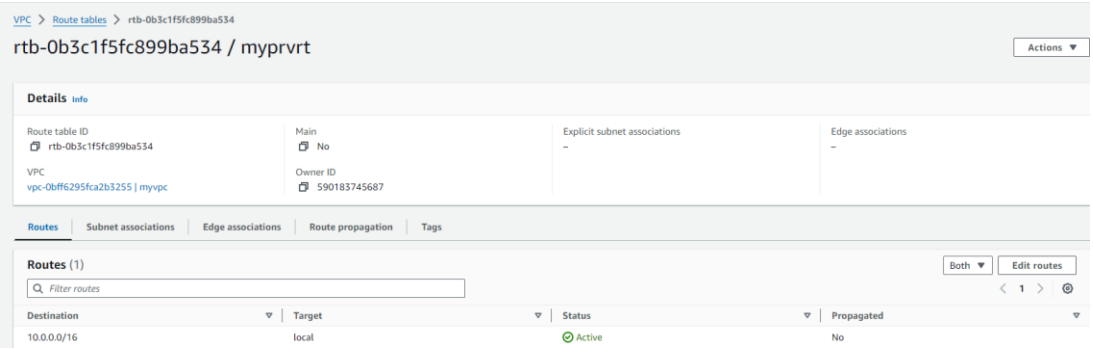
- Now again we will go to Route tables and click create route table.



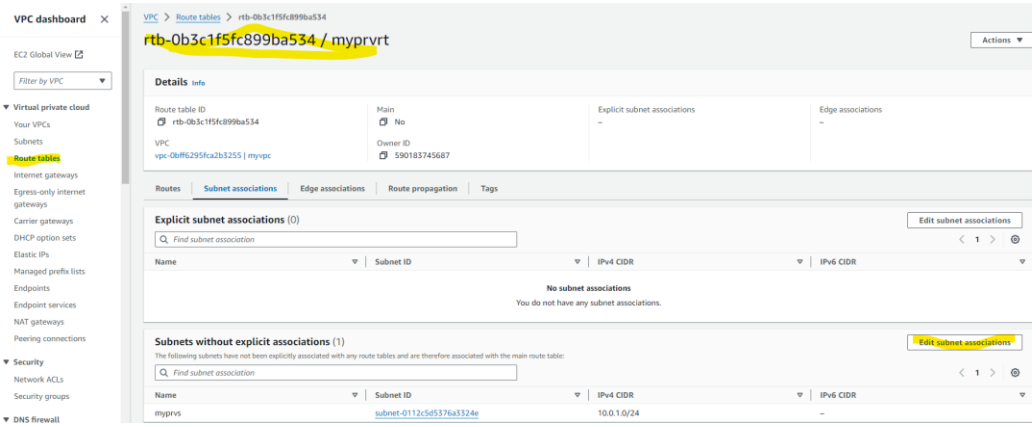
Under create route table we will give the name as myprvrt (my private route table). Attach the VPC we created to it and click create route table.



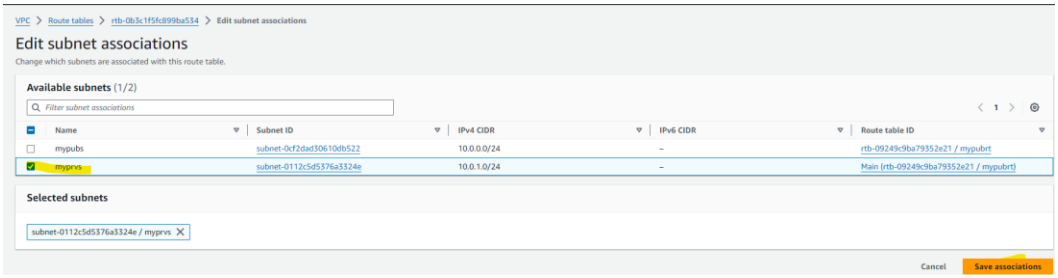
Private route table created successfully.



- Now again we will go to Route tables and under our Private Route Table (myprvrt) > subnet associations > Edit subnet associations.

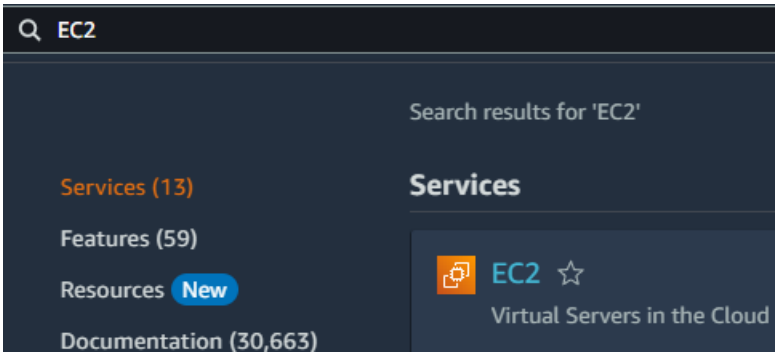


Click myprvs subnet and save associations.

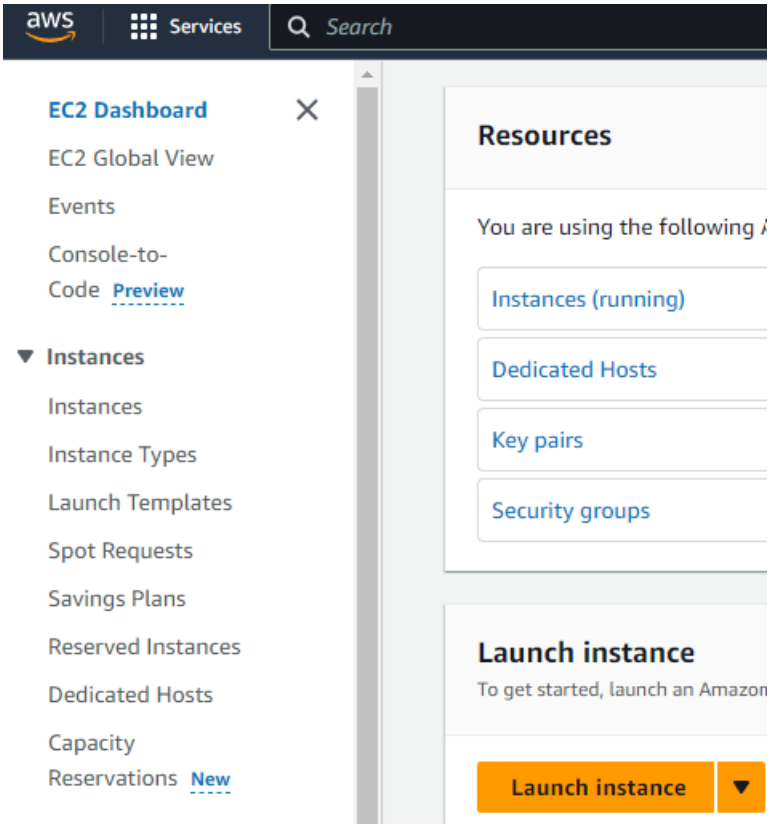


- Now we will create an EC2 instance on our VPC (myvpc) under public subnet (mypubs) along with new security group name mypubsg, where we will allow SSH and HTTPS port on the inbound rules, so that we can check the internet connectivity is working on the it or not.

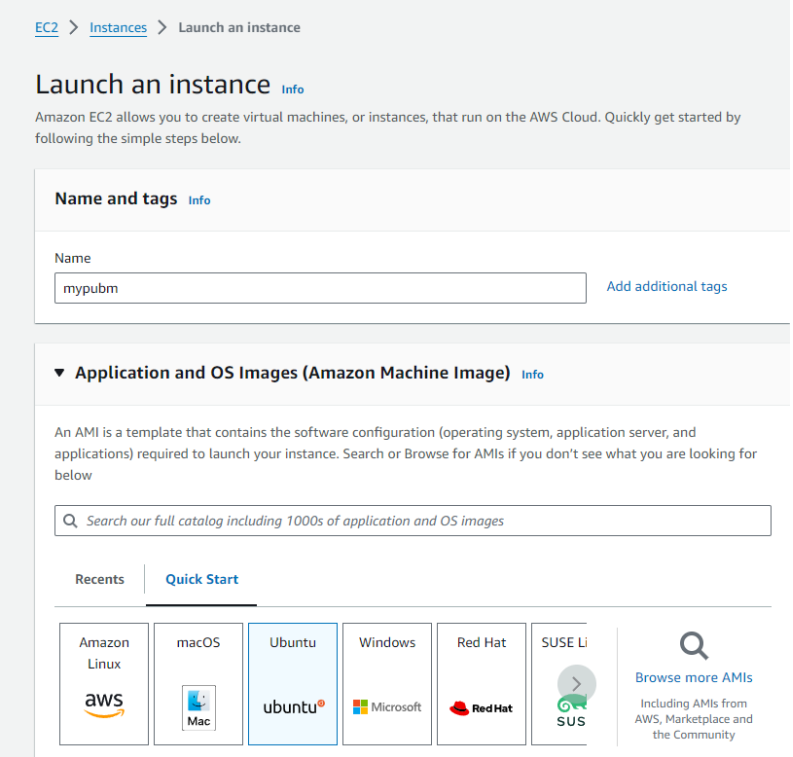
In AWS console, search for EC2 and click on it.

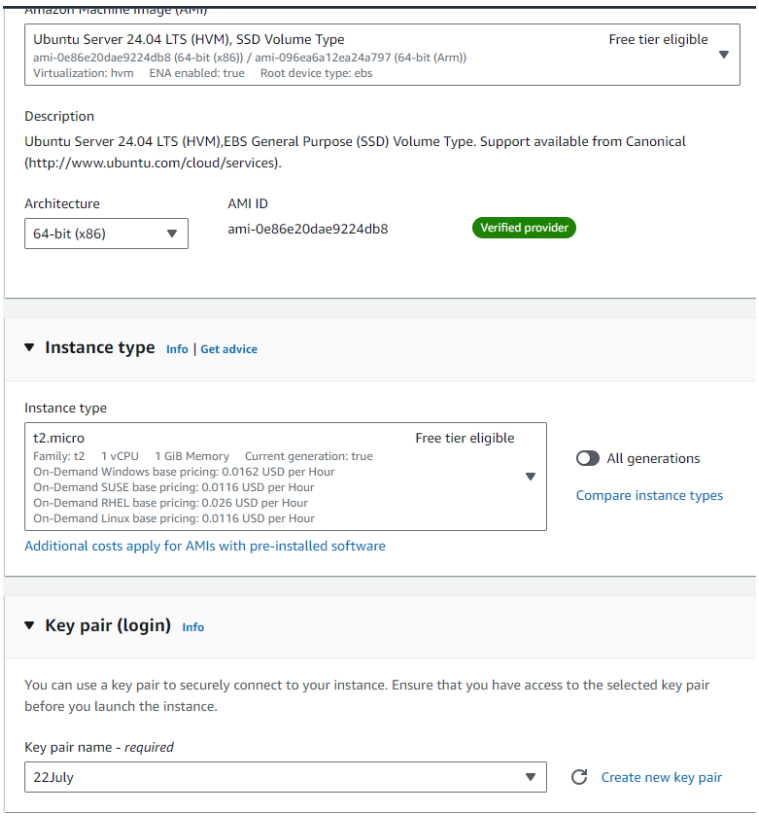


Launch an instance

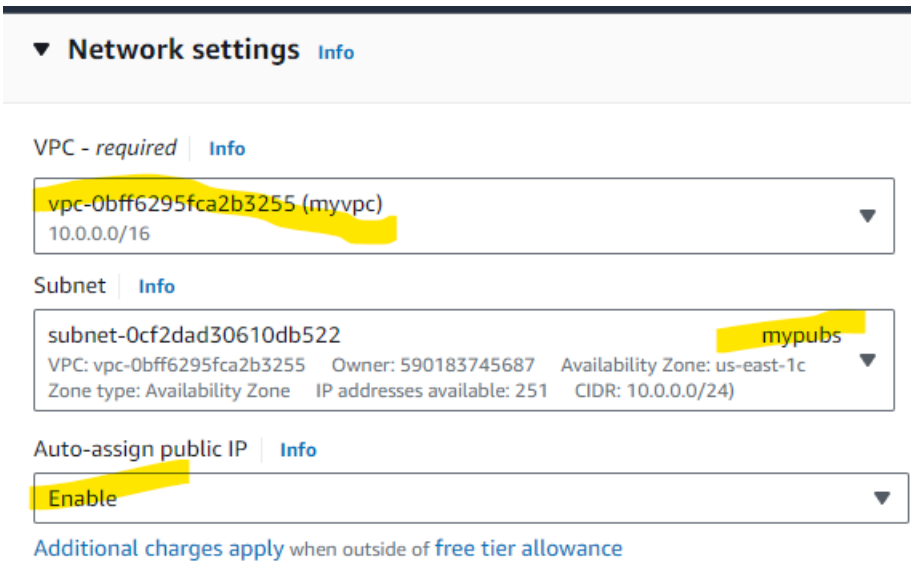


Now fill in the details with name as mypubm (my public machine), select OS as ubuntu

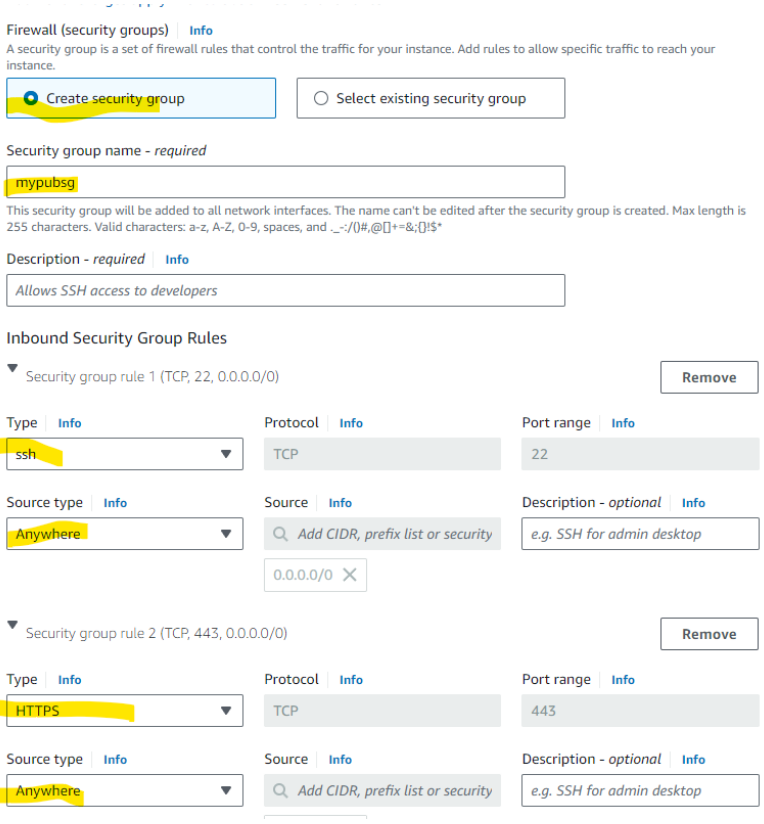




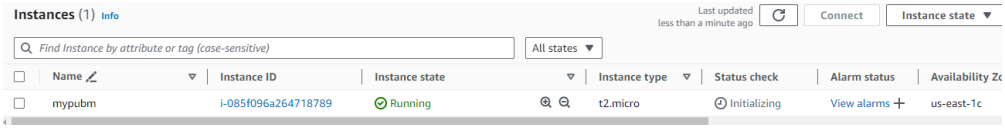
Under Network settings > Click Edit and attach our VPC and under subnet select the public subnet and auto assign public IP enable.



Under Security Groups > Create security group and give it the name mypubsg and allow HTTPS and SSH.



Rest, keep everything as default and click launch instance. We will verify the details once the instance is up.



Once the instance is up, we will connect to it via SSH and then run ping www.google.com to see if it responds

```
PS C:\Users\ssour\Downloads> ssh -i "22July.pem" ubuntu@52.90.231.242
The authenticity of host '52.90.231.242 (52.90.231.242)' can't be esta
```

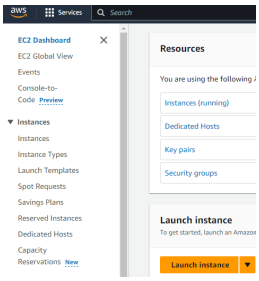
Checked and verified that ping is responding to www.google.com

```
root@ip-10-0-0-159:/home/ubuntu# ping www.google.com
PING www.google.com (142.251.111.106) 56(84) bytes of data.
64 bytes from bk-in-f106.1e100.net (142.251.111.106): icmp_seq=1 ttl=56 time=1.31 ms
64 bytes from bk-in-f106.1e100.net (142.251.111.106): icmp_seq=2 ttl=56 time=1.94 ms
64 bytes from bk-in-f106.1e100.net (142.251.111.106): icmp_seq=3 ttl=56 time=1.76 ms
64 bytes from bk-in-f106.1e100.net (142.251.111.106): icmp_seq=4 ttl=56 time=1.78 ms
64 bytes from bk-in-f106.1e100.net (142.251.111.106): icmp_seq=5 ttl=56 time=1.69 ms
^C
--- www.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
```

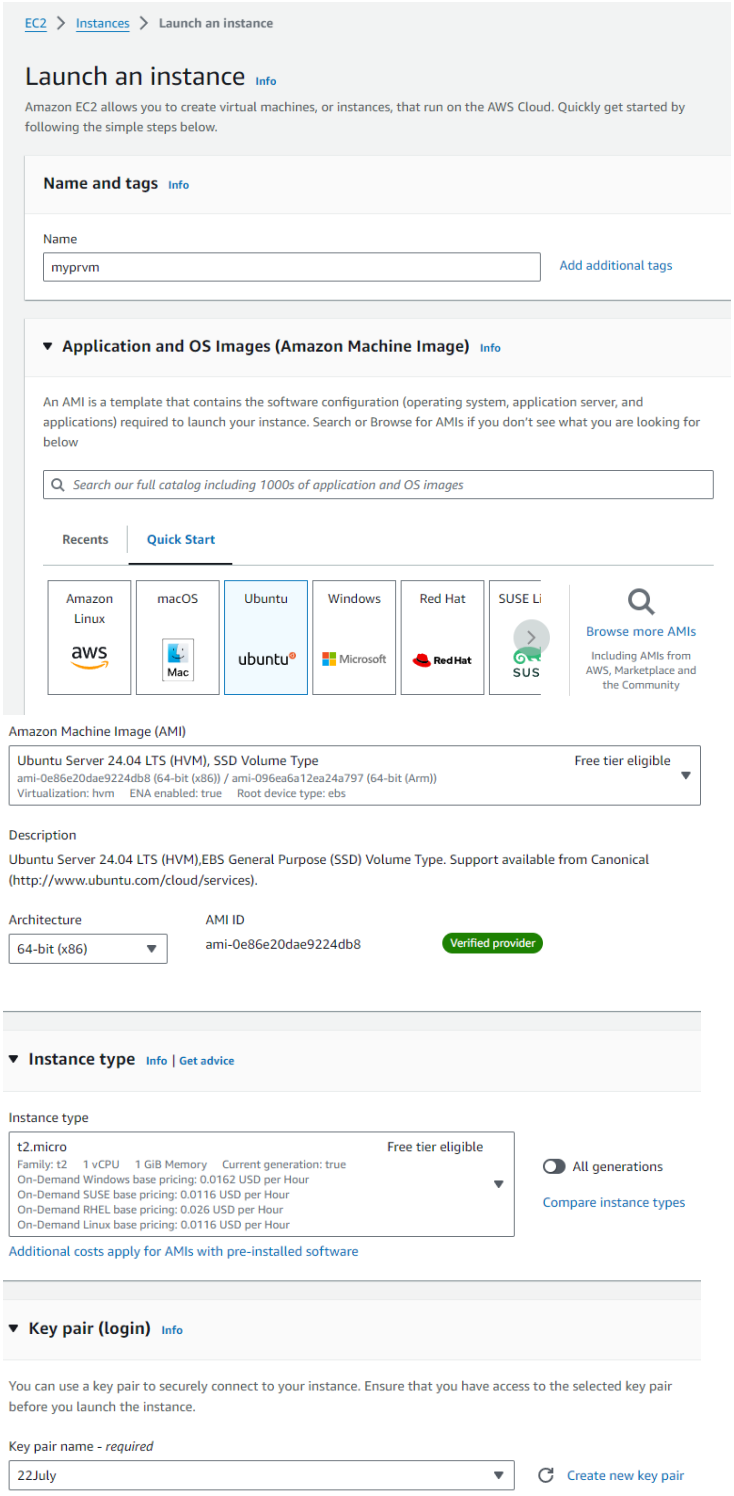
So, internet is working properly on EC2 connected to Public Subnet which is attached to Internet gateway

- Similarly, we will create an EC2 instance on our VPC (myvpc) under private subnet (myprvs) along with new security group name myprvsg, where we will allow SSH and HTTPS port on the inbound rules, so that we can check the internet connectivity is working on the it or not.

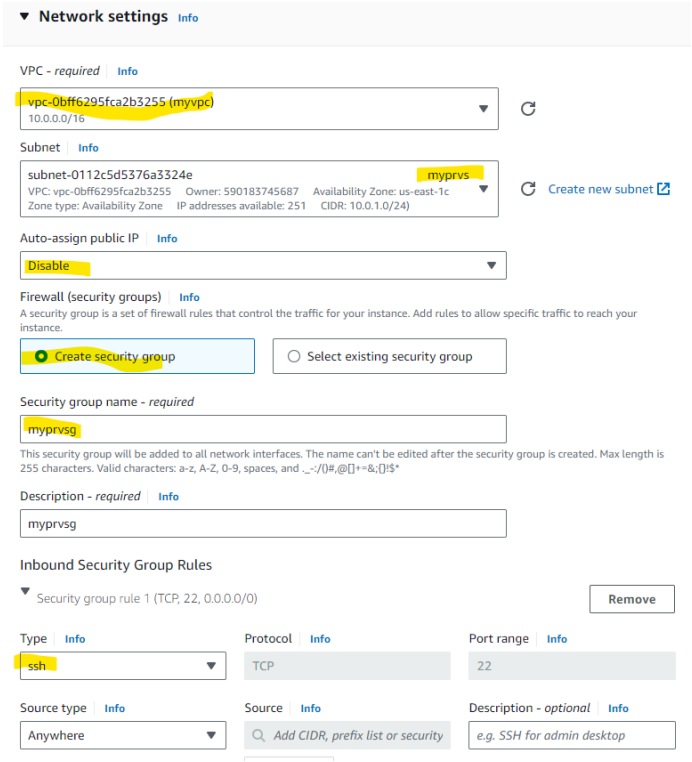
Launch an instance



Now fill in the details with name as myprvm (my private machine), select OS as ubuntu



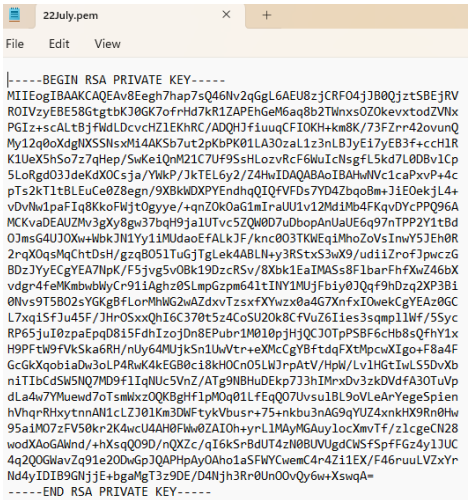
Under Network settings > Click Edit and attach our VPC and under subnet select the private subnet and auto assign public IP disable.



Rest, keep everything as default and click launch instance. We will verify the details once the instance is up.

- Now since the private EC2 instance does not have public IP, we will SSH to private EC2 instance from public EC2 instance.

To make this happen, we need to create the same .pem file i.e. in our case 22july.pem in public instance and copy the contents from the pem file on the local machine.



```
root@ip-10-0-0-159:/home/ubuntu# cat > 22July.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAv8Eegh7hap7sQ46Nv2qGgL6AEU8zjCRFO4jJB0QjztSBEjRV
ROIVzyEBE58GtgtbKJ0GK7ofrHd7kR1ZAPEhGeM6aq8b2TWnxsOZOkevxtodZVNx
PGIz+scALtBjfWdLDcvcHZlEKhRC/ADQHJfiuuqCFIOKH+km8K/73FZrr42ovunQ
My12q0oXdgNXSSNsxMi4AKSb7ut2pKbPK01LA3OzaL1z3nLBJyEi7yEB3f+ccHlR
K1UeX5hSo7z7qHep/SwKeiQnM21C7Uf9SsHLozvRcF6WuIcNsgfL5kd7L0DBvlCp
5LoRgdO3JdeKdXOCsja/YWkP/JkTEL6y2/Z4HwIDAQABAoIBAHwNVc1caPxvP+4c
oTs2kTltBLEuCe0Z8egn/9XBkWDXPYEndhqQIQfVFDs7YD4ZbqoBm+JiEOekjL4+
vDvNw1paFIq8KkoFWjtOgyye/+qnZOkOaG1mIraUU1v12MdiMb4FKqvDYcPPQ96A
MCKvaDEAUZMv3gXy8gw37bqH9jalUTvc5ZQW0D7uDbopAnUaUE6q97nTPP2Y1tBd
OJmsG4UJOXw+WbkJN1Yy1iMUdaoEfALkJF/knc0O3TKWEqiMhoZoVsInwY5JEh0R
2rqXOqsMqChtDsH/gzqBO5lTuGjTgLek4ABLN+y3RStxS3wX9/udiiZrofJpwczG
BDzJYyECgYEA7NpK/F5jvg5vOBk19DzcRSv/8Xbk1EaIMASs8FlbarFhfXwZ46bX
vdgr4feMKmbwbWyCr91iAghz0SLmpGzpm64ltINY1MUjFbiy0JQqf9hDzq2XP3Bi
0Nvs9T5BO2sYGKgBfLorMhWG2wAZdxvTzsxfXYwzx0a4G7XnfxIOwekCgYEAz0GC
L7xqiSfJu45F/JHrOSxxQhI6C370t5z4CoSU2Ok8CfVuZ6Iies3sqmpllWf/5Syc
RP65juI0zpaEpqD8i5FdhIzojDn8EPubr1M0l0pjHjQCJOTpPSBF6cHb8sQfhY1x
H9PFtW9fVkSka6RH/nUy64MUjkSn1UwVtr+eXMcCgYBftdqFXtMpcwXIgo+F8a4F
GcGkXqobiaDw3oLP4RwK4kEGB0ci8kHOCnO5LWJrpAtV/HpW/LvlHGtIwLS5DvXb
niTIbCdSW5NQ7MD9flIqNUc5VnZ/ATg9NBHuDEkp7J3hIMrxDv3zkDVdfA3OTuVp
dLa4w7YMuewd7oTsmWxzOQKBgHflpMOq01LfEqQO7UvsulBL9oVLeArYegeSpien
nVhqrRHxytnnAN1cLZJ0lKm3DWFtykVbusr+75+nkbu3nAG9qYUZ4xnkHX9Rn0Hw
95aiMO7zFV50kr2K4wcU4AH0FWw0ZAIOh+yrLlMAyMGAuylocXmvTf/zlcgeCN28
wodXAoGAWnd/+hXsqQO9D/nQXZc/qI6kSrBdUT4zN0BUVUgdCWSfSpfFGz4ylJUC
4q2QOGWavZq91e2ODwGpJQAPHpAyOAho1aSFWYCwemC4r4Zi1EX/F46ruuLVZxYr
Nd4yIDIB9GNjjE+bgaMgT3z9DE/D4Njh3Rr0UnOOvQy6w+XswqA=
-----END RSA PRIVATE KEY-----
```

We will also change the permissions of the .pem file created to 400 so that we need it for read purpose by running the command.

```
root@ip-10-0-0-159:/home/ubuntu# chmod 400 22July.pem
root@ip-10-0-0-159:/home/ubuntu#
```

Now we will connect to the private instance using the below command.

```
root@ip-10-0-0-159:/home/ubuntu# ssh -i "22July.pem" ubuntu@10.0.1.221
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Sep  5 12:42:52 UTC 2024

  System load:  0.0               Processes:             105
  Usage of /:   22.7% of 6.71GB   Users logged in:       0
  Memory usage: 19%               IPv4 address for enX0: 10.0.1.221
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-1-221:~$
```

We were able to connect to the private EC2 instance successfully.

Now we will check if ping www.google.com responds or not

Checked and verified that ping is not responding to www.google.com on the private instance.

```
root@ip-10-0-1-221:/home/ubuntu# ping www.google.com
PING www.google.com (172.253.115.105) 56(84) bytes of data.
```

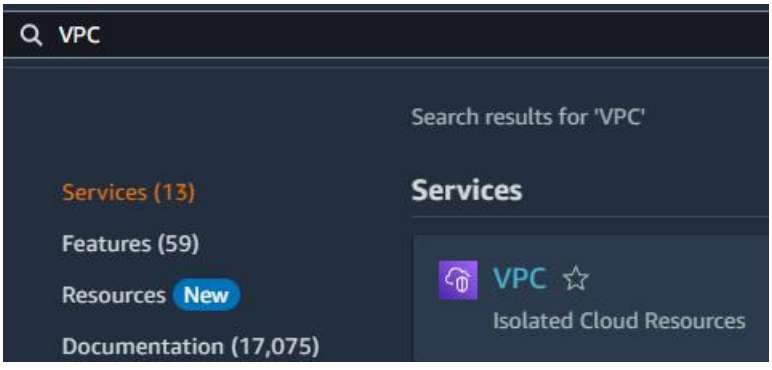
We will also try to install docker.io and see if it works or not.

Docker installation is not working as it usually does, since the machine is in private subnet and nothing open to internet.

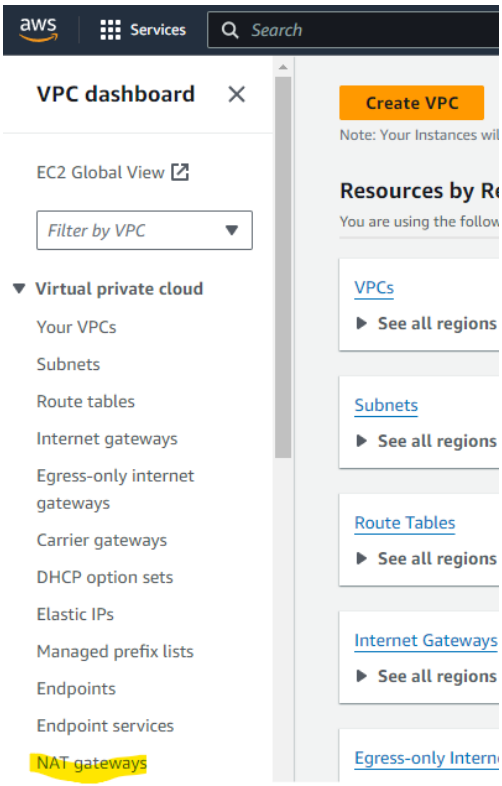
```
root@ip-10-0-1-221:/home/ubuntu# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package docker.io is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

E: Package 'docker.io' has no installation candidate
root@ip-10-0-1-221:/home/ubuntu#
```

- If we want to download and install docker on private ec2 instance without access to internet, we need to create a NAT gateway.

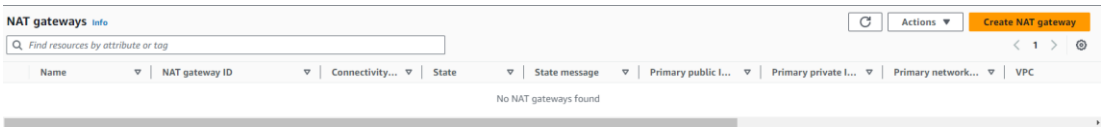
- In AWS console, search for VPC and click on it.



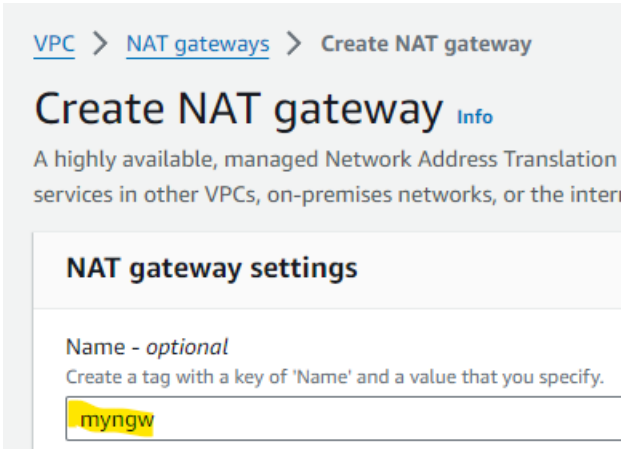
- Now we will go to NAT Gateway under VPC dashboard.



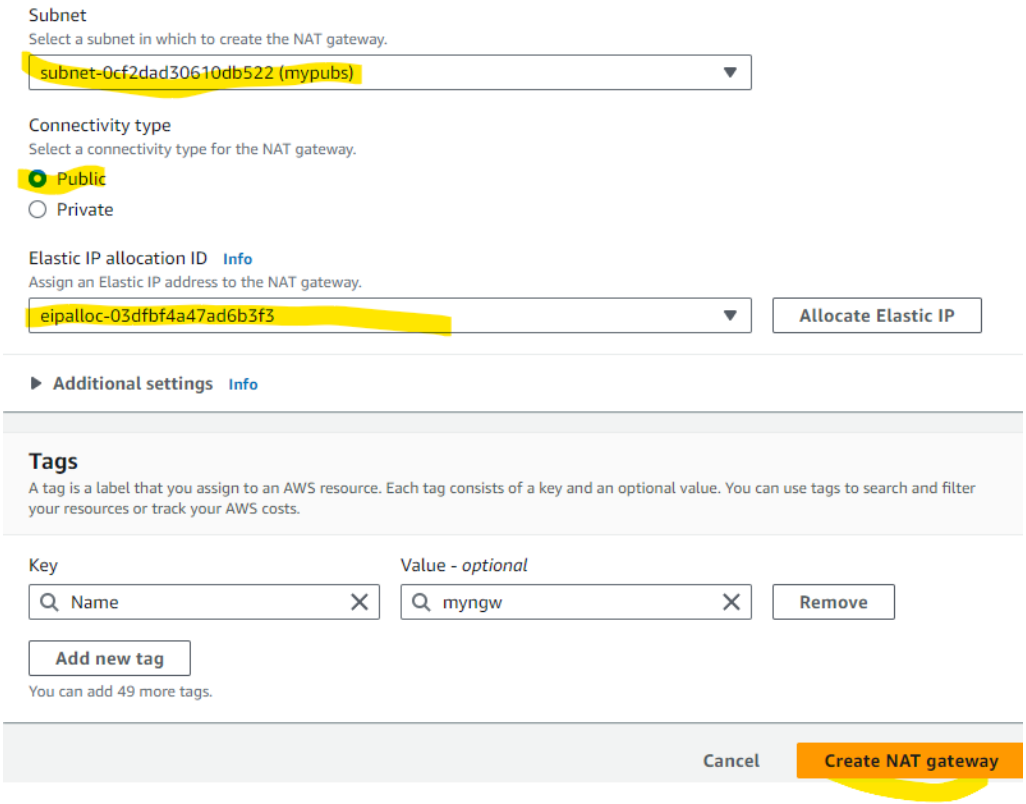
- Click Create NAT Gateway.



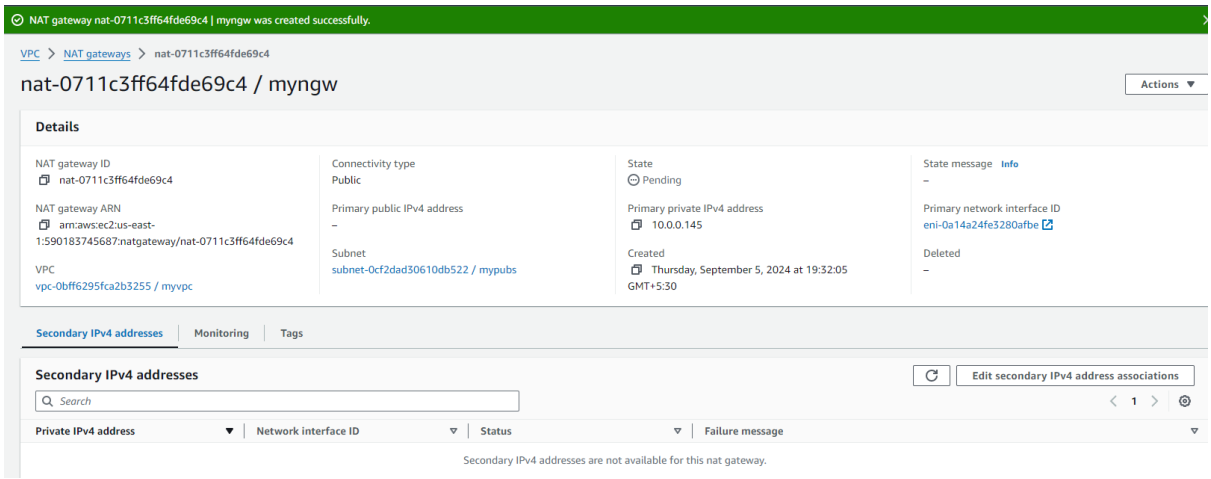
- Now under NAT Gateway Settings, give the name as myngw (My NAT Gateway)



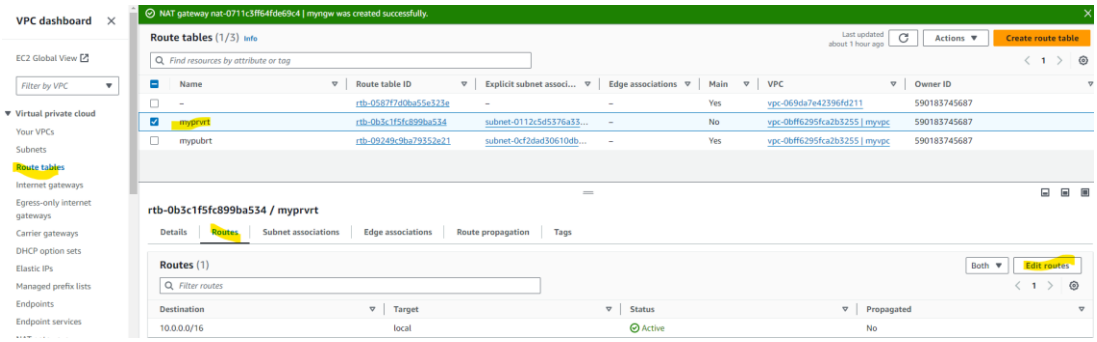
- Under Subnet, we will select our Public Subnet (mypubs) since we need internet from public subnet as it is connected to internet gateway, keep connectivity type as public and allocate Elastic IP and click create NAT Gateway.



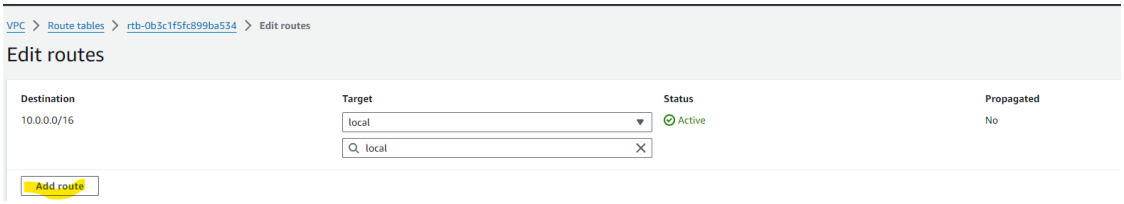
NAT Gateway is created successfully.



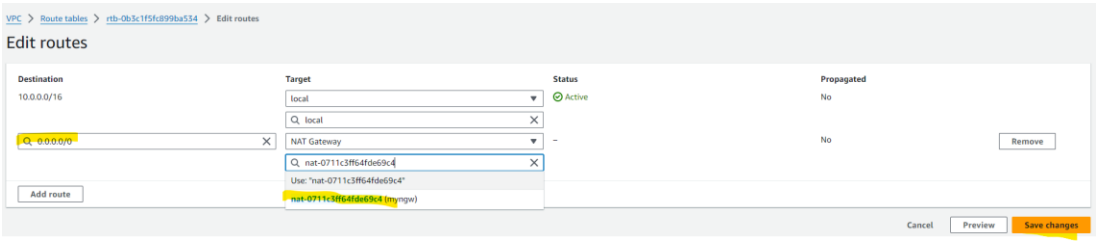
- Now again we will go to Route tables and under our Public Route Table (myprvrt) > Routes > Edit Routes



Click on Add Route



Under Destination we will give 0.0.0.0/0 and under target we will give internet gateway and attach the NAT gateway (myngw) that we created and click save changes.



- Now we will do ssh again to private instance and see if we are able to get to the internet by masking through NAT Gateway by installing docker.io on the private instance using the below command.

apt update -y && apt upgrade -y && apt install docker.io -y

Now we can see that docker.io got installed successfully.

```
root@ip-10-0-1-221:/home/ubuntu# apt update -y && apt upgrade -y && apt install docker.io -y
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [326 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [74.4 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [471 kB]
Get:16 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [4420 B]
Get:17 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [266 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [118 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 c-n-f Metadata [7864 B]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [350 kB]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [144 kB]
```

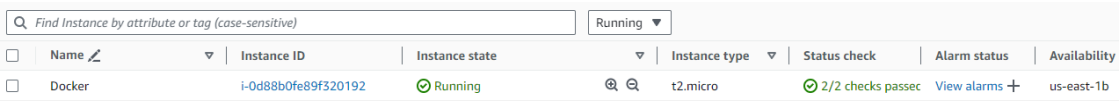
```
root@ip-10-0-1-221:/home/ubuntu# docker --version
Docker version 24.0.7, build 24.0.7-0ubuntu4.1
root@ip-10-0-1-221:/home/ubuntu#
```

- This states that docker is installed successfully without internet.

18) Create a customized container in an EC2 Instance where we need to perform below tasks

- Create a index.html file with contents (Your container created very good) on the EC2 instance.
- Create a nginx container on port 80:80.
- Location where we need to copy the index.html file from host to container so that we will get the required output on WebURL.
- Copy the index.html file to the required location and verify that we are getting the required output on WebURL.
- We also need to cat the index.html contents in the container and its output should print outside the container.

- Before we proceed for any of the tasks, we need to have an EC2 instance with docker installed on it.



- **Create a index.html file with contents (Your container created very good) on the EC2 instance.**

  Create an index.tml file by using the below command.

  vi index.html

```
root@ip-172-31-89-203:/home/ubuntu# vi index.html
root@ip-172-31-89-203:/home/ubuntu#
```

  Copy and paste the below contents in index.html file

```
<!DOCTYPE html>
<html>

    <head>
    </head>

    <body>
        <h1>Your containiner created very good !!</h1>
    </body>

</html>
```

  Esc wq! (Escape, Write Quit and save the file)

```
root@ip-172-31-89-203: /home/ubuntu
<!DOCTYPE html>
<html>

        <head>
        </head>

        <body>
                <h1>Your contaniner created very good !!</h1>
        </body>

</html>
```

Index.html file is successfully created and placed under /home/ubuntu and we verified by doing ls -ltr

```
root@ip-172-31-89-203:/home/ubuntu# ls -ltr
total 4
-rw-r--r-- 1 root root 117 Sep  6 11:28 index.html
root@ip-172-31-89-203:/home/ubuntu#
```

- **Create a nginx container on port 80:80**

We pulled and run a nginx image which created a container by running the below command.

docker run -d -p 80:80 --restart always --name nginxtest nginx

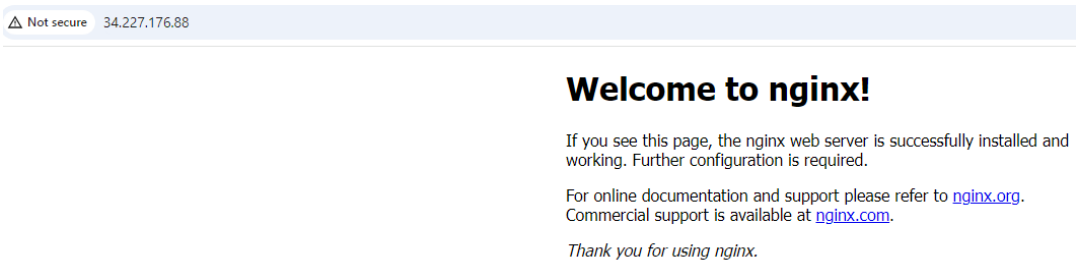
```
root@ip-172-31-89-203:/home/ubuntu# docker run -d -p 80:80 --restart always --name nginxtest nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Downloaded newer image for nginx:latest
ec4c8627fb68287959e515f693d51beb7c3fbd3e5fe980875ef192789e581dbc
root@ip-172-31-89-203:/home/ubuntu#
```

We will verify the container details by running the below command

docker ps

```
root@ip-172-31-89-203:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                   NAMES
ec4c8627fb68   nginx     "/docker-entrypoint.…"   44 seconds ago   Up 42 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp       nginxtest
root@ip-172-31-89-203:/home/ubuntu#
```

We checked on Web URL to confirm that nginx was running properly.



- **Location where we need to copy the index.html file from host to container so that we will get the required output on WebURL.**

  The location where we need to put our index.html file is as below:

  /usr/share/nginx/html

- **Copy the index.html file to the required location in the container and verify that we are getting the required output on WebURL.**

  To perform the copy from host to container, we will run the below command.

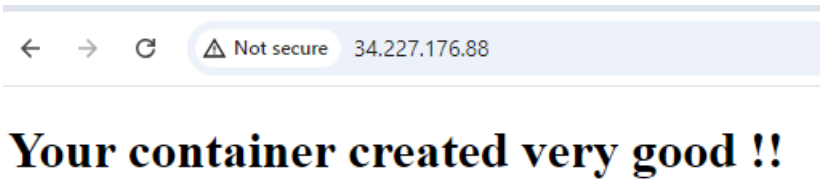
  docker cp /home/ubuntu/index.html ec4c8627fb68:/usr/share/nginx/html/

  (Command copies the index.html file from source(host) location to destination(container) location using container id)

```
root@ip-172-31-89-203:/home/ubuntu# docker cp /home/ubuntu/index.html ec4c8627fb68:/usr/share/nginx/html/
Successfully copied 2.05kB to ec4c8627fb68:/usr/share/nginx/html/
root@ip-172-31-89-203:/home/ubuntu#
root@ip-172-31-89-203:/home/ubuntu#
```

Above command shows that copy was successful.

Now we will verify that changes are reflected on WebURL after our index.html file is copied on the container location.



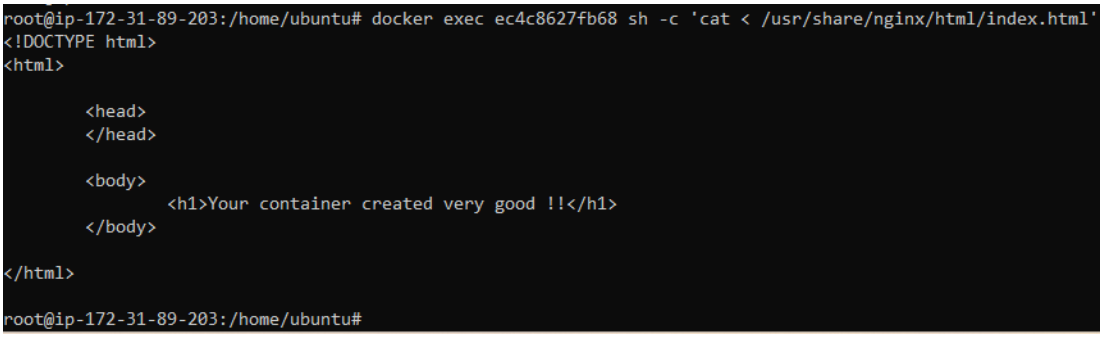
The output is as expected.

- **We also need to cat the index.html contents in the container and its output should print outside the container.**

  We will use the below command to cat the index.html inside the container and print the output on the host.

  docker exec ec4c8627fb68 sh -c 'cat < /usr/share/nginx/html/index.html'

  (Command will exec the container using container id and execute shell to cat the index.html on the mentioned location)



19) Docker swarm Installation and worker join installation script

- To run docker swarm installation and worker join script we need the below prerequisites.

- At least 3 EC2 Instances with docker installed on them by adding the commands on the user data. One instance will be called as master (leader) and other two instances will be worker nodes.

- We need to create a “test” user on all 3 EC2 instances with the command so that their home directory is also created.

- Provide “test” user root level access with a script on all 3 EC2 instances.

- We need to create a temporary file “tempUserName.txt” for passing “test” user details on master EC2 instance.

- We need to create a file by name “ipaddr” which will contain the Public ip address of worker nodes on master EC2 instance.

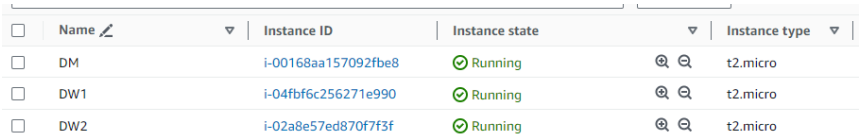
- We need to setup password less authentication between master and worker nodes from the master instance using script.

- Performing Prerequisites

  - **Creating 3 EC2 instances with ubuntu and docker installed with all the packages up to date using the commands in the user data mentioned in the below screenshot.**

```
#!/bin/bash

apt update -y && apt upgrade -y

apt install docker.io -y
```

Below are the 3 EC2 Instances. DM (Master), DW1 (Worker Node 1) and DW2 worker. We have also set the hostnames accordingly to avoid any confusion.



Name	Instance ID	Instance state	Instance type
DM	i-00168aa157092fbe8	Running	t2.micro
DW1	i-04fbf6c256271e990	Running	t2.micro
DW2	i-02a8e57ed870f7f3f	Running	t2.micro

- **Creating a "test" user on all 3 EC2 instances with the command so that their home directory is also created.**

Master Node (DM)

useradd -p abc123 -m test (Command for creating test with unencrypted password using **-p** and creating home directory using **-m**)

```
root@DM:/home/ubuntu# useradd -p abc123 -m test
root@DM:/home/ubuntu#
root@DM:/home/ubuntu#
```

We need to reset the password once for the test user to give it an encrypted password. Since, we have setup and unencrypted password using the below command.

passwd test (command to reset the password for the user)

```
root@DM:/home/ubuntu# passwd test
New password:
Retype new password:
passwd: password updated successfully
root@DM:/home/ubuntu#
```

Now we will also verify home directory is created for the test user or not by using the below command.

cd /home/test (Command to change directory)

```
root@DM:/home/ubuntu#
root@DM:/home/ubuntu# cd /home/test/
root@DM:/home/test#
```

Home directory created and verified

Same Steps needs to be done on worker nodes as well.

Worker Node 1 (DW1)

useradd -p abc123 -m test (Command for creating test with unencrypted password using **-p** and creating home directory using **-m**)

```
root@DW1:/home/ubuntu# useradd -p abc123 -m test
root@DW1:/home/ubuntu#
```

We need to reset the password once for the test user to give it an encrypted password. Since, we have setup and unencrypted password using the below command.

passwd test (command to reset the password for the user)

```
root@DW1:/home/ubuntu# passwd test
New password:
Retype new password:
passwd: password updated successfully
root@DW1:/home/ubuntu#
```

Now we will also verify home directory is created for the test user or not by using the below command.

cd /home/test (Command to change directory)

```
root@DW1:/home/ubuntu# cd /home/test/
root@DW1:/home/test#
root@DW1:/home/test#
```

Home directory created and verified

Worker Node 2 (DW2)

useradd -p abc123 -m test (Command for creating test with unencrypted password using **-p** and creating home directory using **-m**)

```
root@DW2:/home/ubuntu# useradd -p abc123 -m test
root@DW2:/home/ubuntu#
root@DW2:/home/ubuntu#
```

We need to reset the password once for the test user to give it an encrypted password. Since, we have setup and unencrypted password using the below command.

passwd test (command to reset the password for the user)

```
root@DW2:/home/ubuntu# passwd test
New password:
Retype new password:
passwd: password updated successfully
root@DW2:/home/ubuntu#
```

Now we will also verify home directory is created for the test user or not by using the below command.

cd /home/test (Command to change directory)

```
root@DW2:/home/ubuntu# cd /home/test/
root@DW2:/home/test#
root@DW2:/home/test#
```

Home directory created and verified

NOTE: After creating test user, we need to perform all the tasks from home directory of test

- **Providing "test" user root level access with a script which we need to run on all 3 EC2 instances.**

Master Node (DM)

Create a shell script with any name, in my case I created "rootlevel.sh". Edit the file and enter the below contents in that file.

vi rootlevel.sh (Edit the file)

```
root@DM:/home/test# vi rootlevel.sh
root@DM:/home/test#
```

Contents of the script

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart

systemctl restart ssh
```

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart

systemctl restart ssh
~
```

Run the script as bash and test user will have same level of access as root

bash rootlevel.sh

```
root@DM:/home/test# bash rootlevel.sh
root@DM:/home/test#
```

Same steps needed to done on other worker nodes.

Worker Node 1 (DW1)

Create a shell script with any name, in my case I created “rootlevel.sh”. Edit the file and enter the below contents in that file.

vi rootlevel.sh (Edit the file)

```
root@DW1:/home/test# vi roolevel.sh
root@DW1:/home/test#
```

Contents of the script

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart
```

systemctl restart ssh

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart

systemctl restart ssh
```

Run the script as bash and test user will have same level of access as root

bash rootlevel.sh

```
root@DW1:/home/test# bash roolevel.sh
root@DW1:/home/test#
```

Worker Node 2 (DW2)

Create a shell script with any name, in my case I created “rootlevel.sh”. Edit the file and enter the below contents in that file.

vi rootlevel.sh (Edit the file)

```
root@DW2:/home/test# vi rootlevel.sh
root@DW2:/home/test#
root@DW2:/home/test#
```

Contents of the script

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart

systemctl restart ssh
```

```
#!/bin/bash

# Uncomment the file to allow password authentication

sed -i '/PasswordAuthentication yes/s/^#//g' /etc/ssh/sshd_config

# Add below paramaters in the cloud sshd file

echo "PasswordAuthentication yes" > /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

echo -e "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/60-cloudimg-settings.conf

# Giving same permissions as root to test

echo "test    ALL=(ALL:ALL) ALL" > a.txt

sed -i '47r a.txt' /etc/sudoers

# SSH Service Restart

systemctl restart ssh
~
```

Run the script as bash and test user will have same level of access as root

bash rootlevel.sh

```
root@DW2:/home/test# bash rootlevel.sh
root@DW2:/home/test#
```

- **Creating a temporary file “tempUserName.txt” for passing “test” user details on master EC2 instance.**

echo "test" > tempUserName.txt (Command creates tempUserName.txt and puts test as text inside)

```
root@DM:/home/test# echo "test" > tempUserName.txt
root@DM:/home/test#
```

- **Creating a file by name “ipaddr” which will contain the Public ip address of worker nodes on master EC2 instance.**

echo "34.201.144.186" > ipaddr (Command creates ipaddr file and puts the ip address as text inside)

```
root@DM:/home/test# echo "34.201.144.186" > ipaddr
```

echo "54.226.167.198" >> ipaddr (Command appends existing ipaddr file and puts the ip address as text inside)

```
root@DM:/home/test# echo "54.226.167.198" >> ipaddr
```

- **Setting up password less authentication between master and worker nodes from the master instance using script.**

Master Node (DM)

Create a shell script with any name, in my case I created "passwordless_ssh.sh". Edit the file and enter the below contents in that file.

vi passwordless_ssh.sh (Edit the file)

```
root@DM:/home/test# vi passwordless_ssh.sh
```

Contents of the script

#!/bin/bash

Defining variable for ipaddr file

```
ipAddFile="./ipaddr"

# Command runs SSH-Keygen without prompting anything

echo -e "\n" | ssh-keygen -N "" &> /dev/null

# Now the Script checks the IP address file and concat the
username and IP address to copy the ssh pub key

echo "$ipAddFile"

for IP in `cat $ipAddFile`; do
        if [[ $IP == *"["* ]]; then
                echo "$IP"|cut -d "[" -f2 | cut -d "]" -f1>tempUserName.txt
                 else
                        user=$(cat tempUserName.txt)
                        ssh-copy-id $user@$IP
                        echo "Key copied to $IP"
        fi
done

# Remove tempUserName.txt file

rm -rf tempUserName.txt
```

```
#!/bin/bash

# Defining variable for ipaddr file

ipAddFile="./ipaddr"

# Command runs SSH-Keygen without prompting anything

echo -e "\n" | ssh-keygen -N "" &> /dev/null

# Now the Script checks the IP address file and concat the username and IP address to copy the ssh pub key

echo "$ipAddFile"

for IP in `cat $ipAddFile`; do
          if [[ $IP == *"["* ]]; then
                    echo "$IP"|cut -d "[" -f2 | cut -d "]" -f1>tempUserName.txt
          else
                    user=$(cat tempUserName.txt)
                    ssh-copy-id $user@$IP
                    echo "Key copied to $IP"
                  fi
         done

# Remove tempUserName.txt file

rm -rf tempUserName.txt
```

Run the script as bash and passwordless SSH will be setup between master and worker nodes

bash passwordless_ssh.sh

```
root@DM:/home/test#
root@DM:/home/test# bash passwordless_ssh.sh
./ipaddr
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_ed25519.pub"
The authenticity of host '34.201.144.186 (34.201.144.186)' can't be established.
ED25519 key fingerprint is SHA256:pQHK8ZdJKIaaIpTob9Sf+m9kyz0ivzdXSrSBybmO4A0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
test@34.201.144.186's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'test@34.201.144.186'"
and check to make sure that only the key(s) you wanted were added.

Key copied to 34.201.144.186
```

We need to supply password for test user to get the key copied (Above screenshot shows key copied to first IP address in ipaddr file)

Continuing the script, pasting the screenshot for second IP address

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_ed25519.pub"
The authenticity of host '54.226.167.198 (54.226.167.198)' can't be established.
ED25519 key fingerprint is SHA256:SVxXmkEAse9xtygZT1RIt0fo0YStvuYfLTGqgjOCESo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
test@54.226.167.198's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'test@54.226.167.198'"
and check to make sure that only the key(s) you wanted were added.

Key copied to 54.226.167.198
root@DM:/home/test#
```

We need to supply password for test user to get the key copied (Above screenshot shows key copied to second IP address in ipaddr file)

Passwordless ssh has been setup correctly.

- Add the “test” user to docker group so that, our main script gets executed all 3 EC2 instances.

  usermod -a -G docker test (command to add test user to the docker group as is not a member of the docker group)

  Master Node (DM)

```
root@DM:/home/test# usermod -a -G docker test
root@DM:/home/test#
```

Worker Node 1 (DW1)

```
root@DW1:/home/test# usermod -a -G docker test
root@DW1:/home/test#
```

Worker Node 2 (DW2)

```
root@DW2:/home/test# usermod -a -G docker test
root@DW2:/home/test#
root@DW2:/home/test#
```

All the Pre-requisites have been completed.

- Now we will run the final script to initialize docker swarm and join worker node to docker swarm and verifying the same.

MAIN SCRIPT on Master Node (DM)

Create a shell script with any name, in my case I created "docker_swarm.sh". Edit the file and enter the below contents in that file.

vi docker_swarm.sh (Edit the file)

```
root@DM:/home/test# vi docker_swarm.sh
root@DM:/home/test#
```

Contents of the script

```
#!/bin/bash

# Initialize Docker Swarm and copy the output to text file
docker swarm init > /home/test/output.txt

# Extract the token line required to join worker node
sed -n -e 5p /home/test/output.txt > /home/test/out.txt

# Add the bin bash to the line so that it acts as a shell script
sed '1 i\ #!/bin/bash' /home/test/out.txt > /home/test/final.sh

# Now execute the final shell script on the worker nodes
for server in $(cat /home/test/ipaddr)
do
        ssh test@${server} 'bash -s' < /home/test/final.sh
 done
```

```
#!/bin/bash

# Initialize Docker Swarm and copy the output to text file

docker swarm init > /home/test/output.txt

# Extract the token line required to join worker node

sed -n -e 5p /home/test/output.txt > /home/test/out.txt

# Add the bin bash to the line so that it acts as a shell script

sed '1 i\ #!/bin/bash' /home/test/out.txt > /home/test/final.sh

# Now execute the final shell script on the worker nodes

for server in $(cat /home/test/ipaddr)
do
                ssh test@${server} 'bash -s' < /home/test/final.sh
        done
```

Run the script as bash and docker swarm script on the master node.

bash docker_swarm.sh

```
root@DM:/home/test# bash docker_swarm.sh
This node joined a swarm as a worker.
This node joined a swarm as a worker.
root@DM:/home/test#
```

Docker swarm initialized successfully and worker node also joined to docker swarm

We will verify the above by running the below command on master node.

docker node ls (Command to list the nodes of docker swarm)

```
root@DM:/home/test# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
huf2asqyl4eczehe7xhog28e5 *   DM         Ready     Active         Leader           24.0.7
w58u0cerd46ghz6eok66x8o14     DW1        Ready     Active                          24.0.7
ftgojfpzm8bn79h59j3wng5t7     DW2        Ready     Active                          24.0.7
root@DM:/home/test#
```

Successfully verified that docker swarm initialized and worker node also joined to docker swarm.