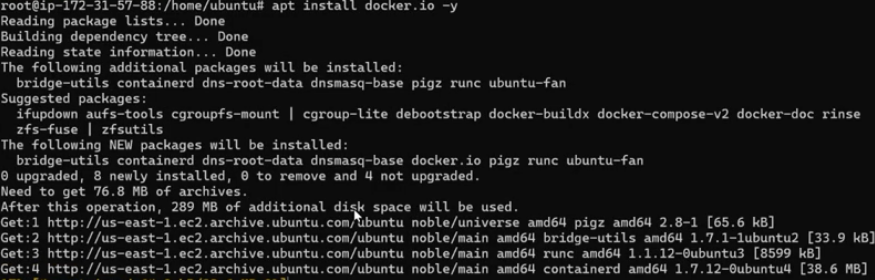
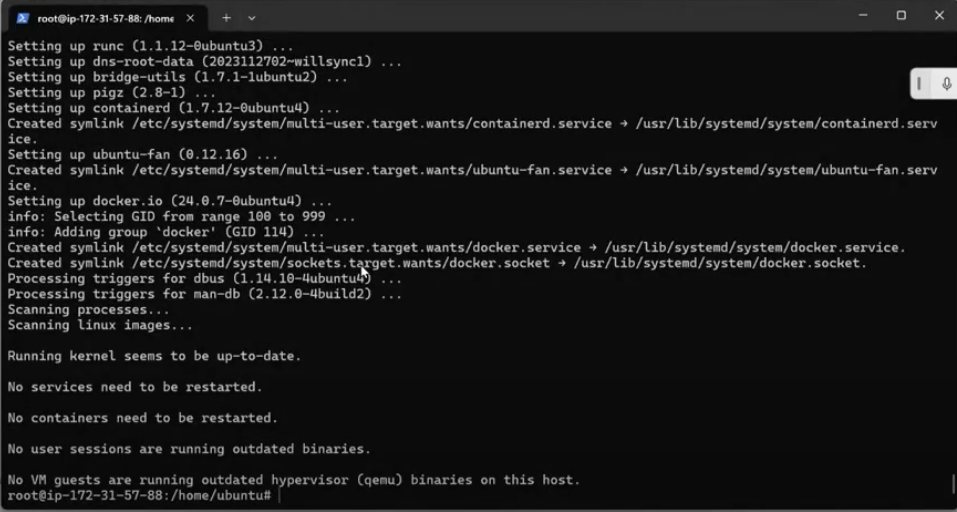
**Docker Notes**

1. Docker Installation.

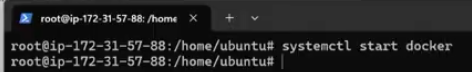
apt install docker.io -y (Command for docker installation on Ubuntu)



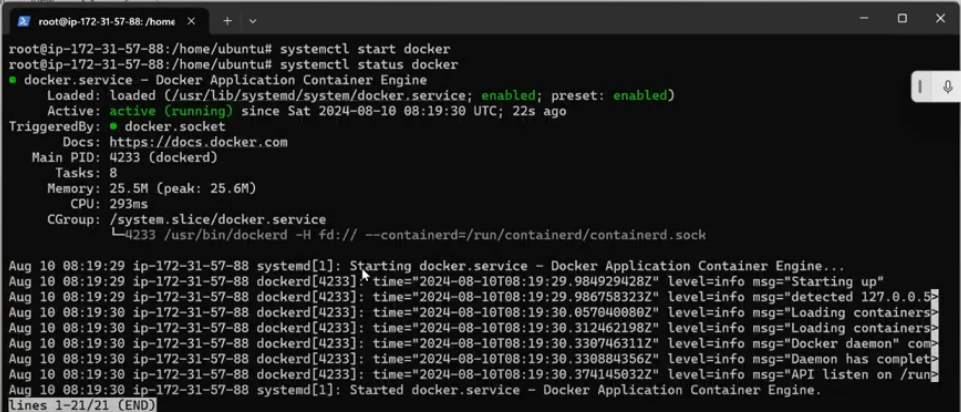




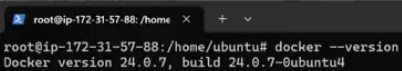
systemctl start docker (Starts docker service)



systemctl status docker (Gives the status of docker service)



docker –version (Give the Version of docker installed)



docker version

(If we give without – version, then it gives complete details of client and server version), By Default what is installed is client.

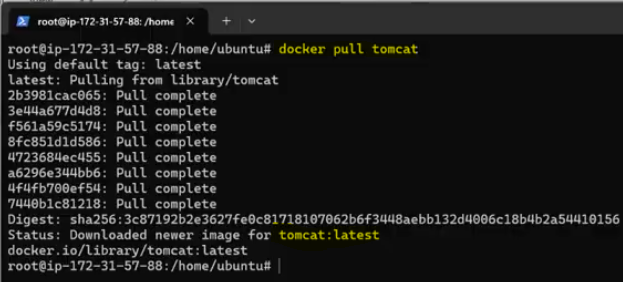


1. docker pull tomcat

This command helps only in pulling/downloading the docker image from docker Hub. It does not create container.

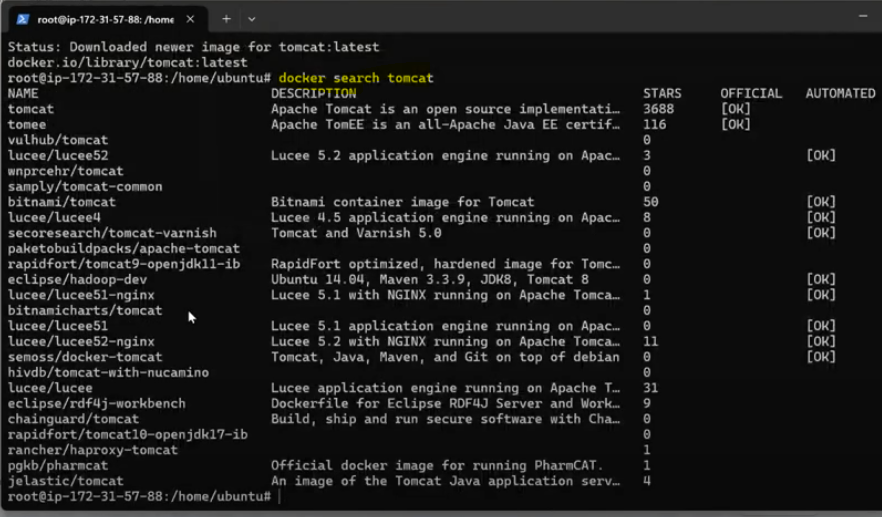
In our case we are pulling docker image for tomcat. (for example)

(Note: This is pulling the latest docker image only) Screenshot below shows the same



1. docker search tomcat

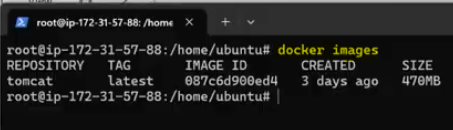
This command will give you all the docker images of a specific tool or product. In our case, the above command gives all the images of tomcat present on docker hub.



1. docker images

This command shows you the all the images that have been download.

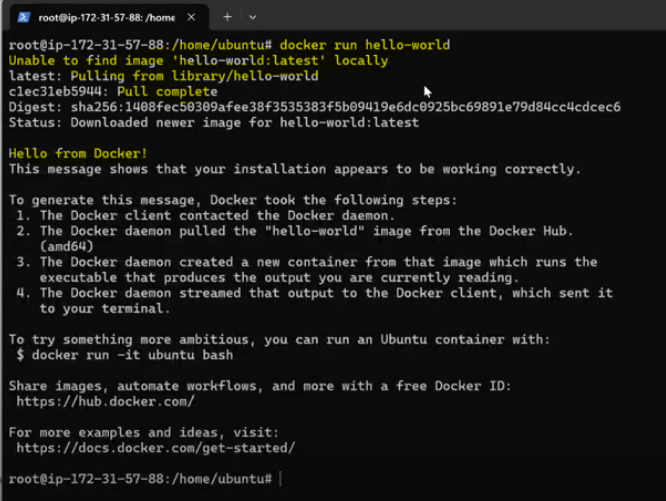
In our case, we have only downloaded tomcat, so it shows tomcat image only.



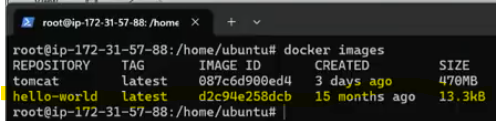
1. docker run hello-world

This command helps in pulling/downloading the docker image and creating the container from that image and runs the container also.

In our case it is pulling/downloading the hello-world docker image and creating a container from it and runs it also.

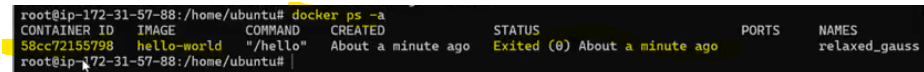


We can verify that hello-world image is downloaded by writing Docker images. Below is the output from the screen.

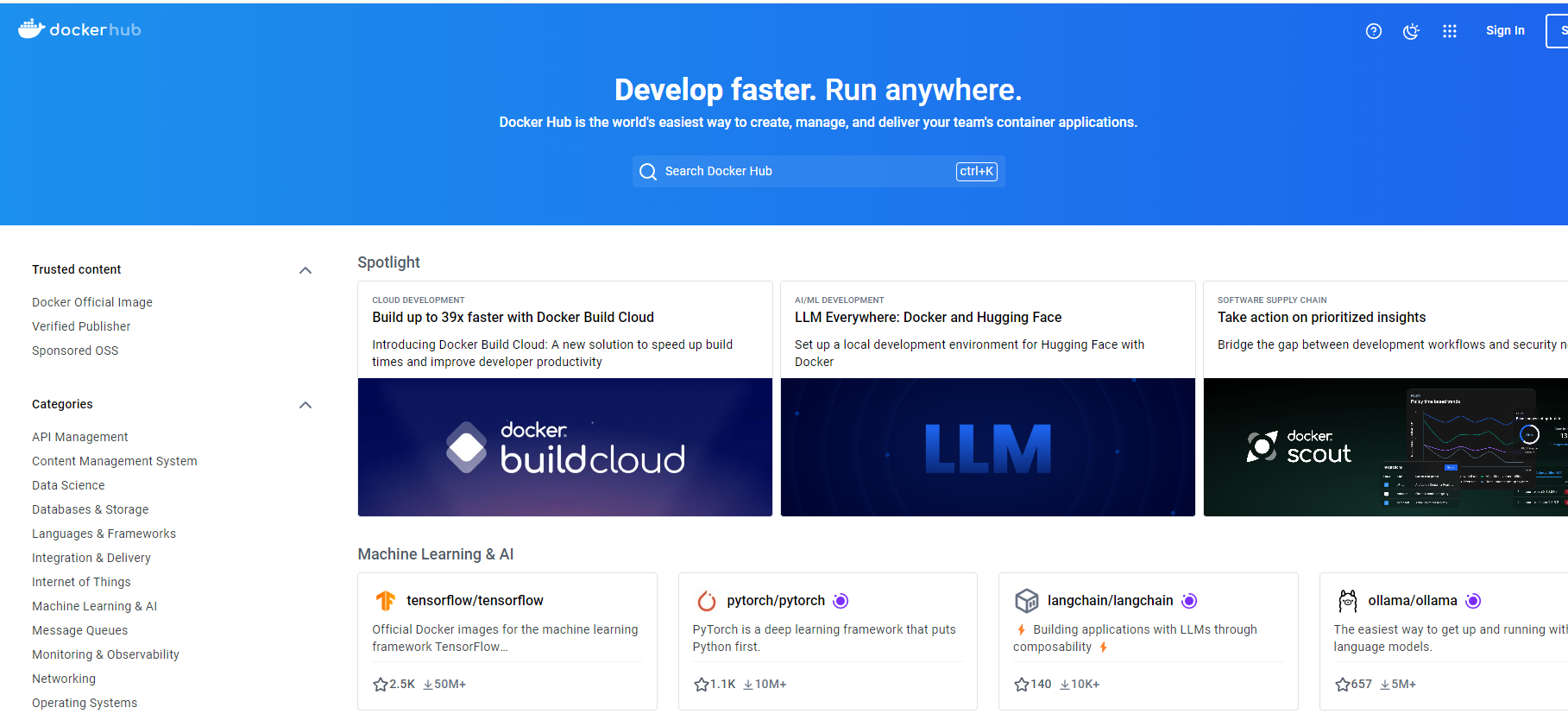


1. docker ps -a

This command gives all the containers that are created from the image and also its status. In our case hello-world container. Here container ran the command and then it dies (if Status in below screenshot shows Exited, then container has died after running, it’s not running continuously.)



1. Docker Repository (Docker Hub) – It is the place where we can store 1 or more versions of a specific docker image. An image can have 1 or more versions (tags).



1. Docker Registry (Docker Trusted Registry) – It’s an enterprise-grade storage solution for Docker Images. In other words, its an image storage service similar to GitHub but for Docker Images.
2. Docker Engine - Docker Engine is an open-source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

* A server with a long-running daemon process [dockerd](https://docs.docker.com/reference/cli/dockerd).
* APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
* A command line interface (CLI) client [docker](https://docs.docker.com/reference/cli/docker/).

1. Master Server - If the container is created on the docker server, then it’s called master server.
2. Worker Server - If the container is created on the different server, then it’s called worker server.
3. DockerFile - Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
4. Docker Container Lifecycle

Below are the steps involved in Docker Container Lifecyle:

**Create container**  
$ docker create –name ubuntu-cont ubuntu

**Run docker container**  
$ docker run -itd ubuntu  
$ docker run -itd –name ubuntu-cont ubuntu

**Pause container**  
$ docker pause <container-id/name>

**Unpause container**  
$ docker unpause <container-id/name>

**Start container**  
$ docker start <container-id/name>

**Stop container**  
$ docker stop <container-id/name>

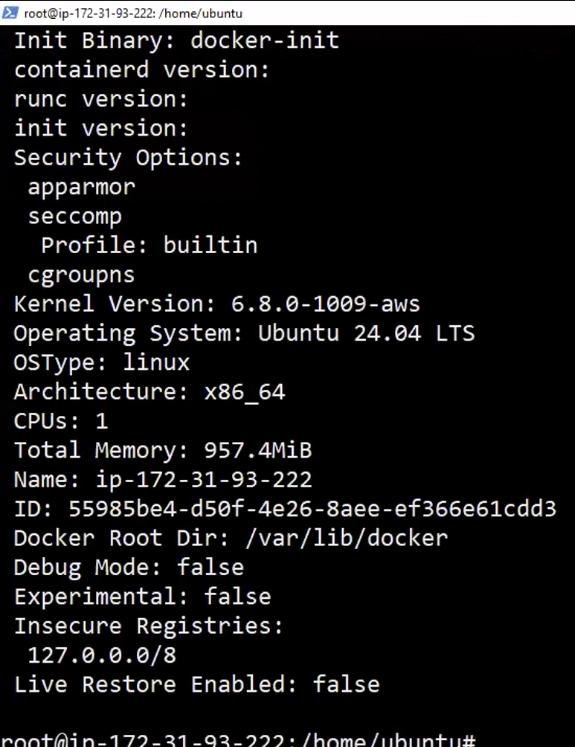
**Restart container**  
$ docker restart <container-id/name>

**Kill container**  
$ docker kill <container-id/name>

**Destroy container**  
$ docker rm <container-id/name>

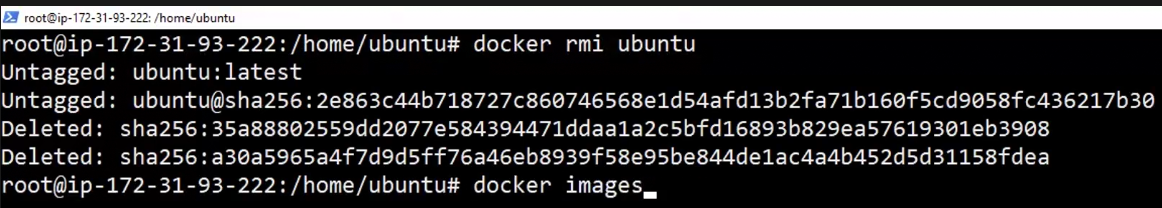
1. Docker Image Layer Caching - Docker Image layer caching is process where if have an image which has some contents in it like application or something and if we want to make a small change in the application, then I will not create the whole image again, it will only recreate that layer of change in the application since rest other things are already cached in the image.
2. docker Info

This command displays system wide information regarding the Docker installation. Information displayed includes the kernel version, number of containers and images. The number of images shown is the number of unique images. The same image tagged under different names is counted only once.

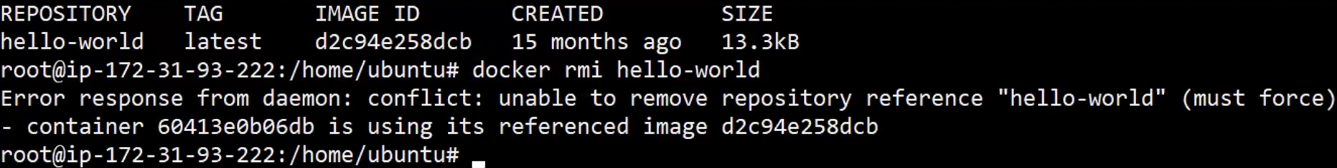


1. docker rmi (image name) – docker rmi ubuntu

This command is used to delete the downloaded images. In our case we are deleting ubuntu image.



Note: If there is a container running from the image downloaded and you try to delete that image, it will not delete, you will have to delete the container first in order to delete the image. Below screenshot shows the same.

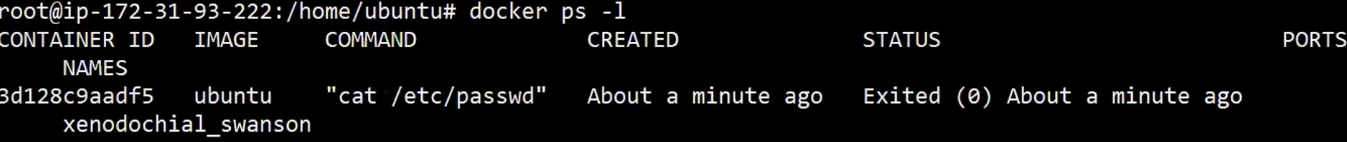


1. docker rm

This command is used to delete the containers created from an image.

1. docker ps -l

This command will only show the last executed container from an image.



1. docker start (container ID)

This command can start the container again with the command that was used to create it.

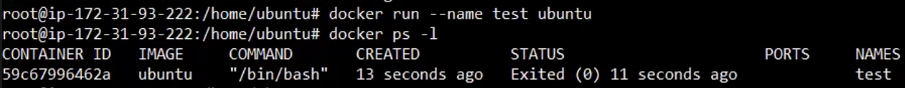


1. docker Stop (Name or Container ID)

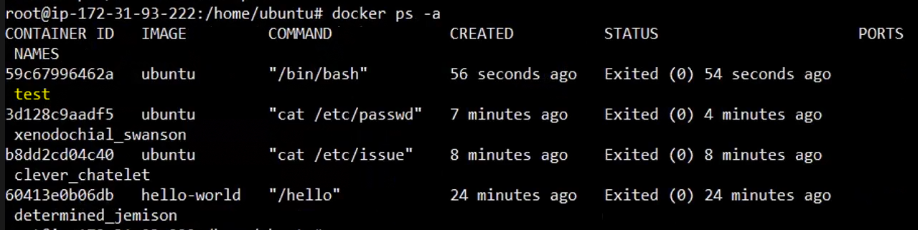
This command can stop the running container.

1. docker run --name myname ubuntu

This command will help to give every container a unique name so that you don’t have to remember the container ID.

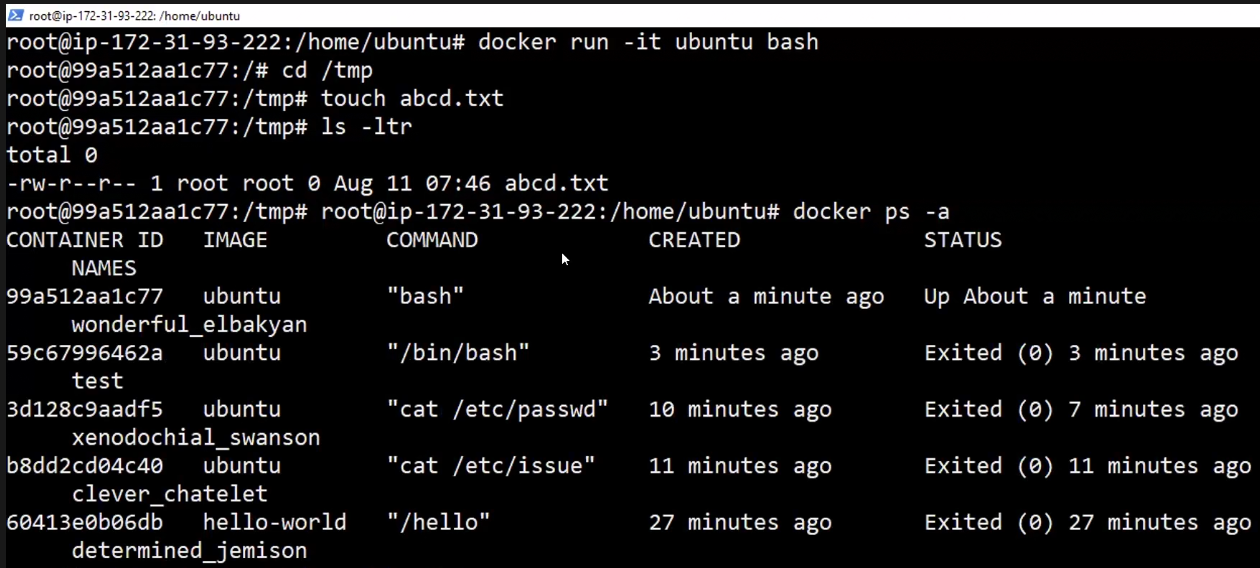


Below screenshot shows container created with our given name.



1. docker run -it ubuntu bash

This command is used to run the container in interactive mode, container will not die even after executing its task.



* **-i** is used to start an interactive session.
* -t allocates a tty and attaches stdin and stdout.
* **ubuntu** is the image that we used to create the container.
* **bash** (or /bin/bash) is the command that we are running inside the Ubuntu container.

To quit and return to host from the running container session you must type **exit** command. The exit command terminates all the container processes and stops it.

# exit

If you’re interactively logged on container terminal prompt and you need to keep the container in running state but exit from the interactive session, you can quit the console and return to host terminal by pressing **Ctrl p** and **q** keys.

1. docker attach <container id >

This command is used to reconnect to the running container, for that we need the container ID or name. We can check the container ID by running docker ps command and then run docker attach command with container ID to connect back to that container.

1. docker kill <container id>

This command is used to stop a running container from the host session.

1. Docker Swarm

* For creating Docker Swarm, we need two Ec2 instances which have **docker install** on both of them, one is Master (m) and other is worker (w)



* Define their hostname in ubuntu terminal so that there is no confusion.
* 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

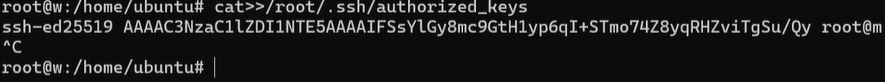
A computer screen with white text

Description automatically generated

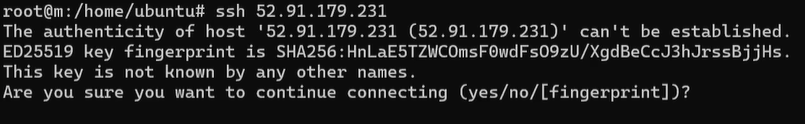
Now do **cat /root/.ssh/id\_ed25519.pub** and copy all the contents inside it.



On Worker: Do **cat >> authorized\_keys** (append authorized\_keys file and paste the contents copied from master server)

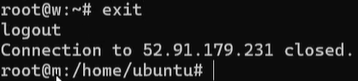


Now we will ssh to worker node from master using the Public IP address





If we exit the worker, we will return back to the master node.



* Now run the below command to master node to make it manager IP

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

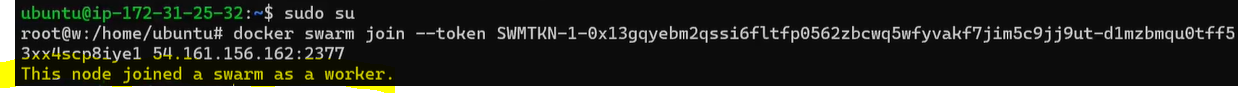
Below screenshot gives the output

A computer screen with white text

Description automatically generated

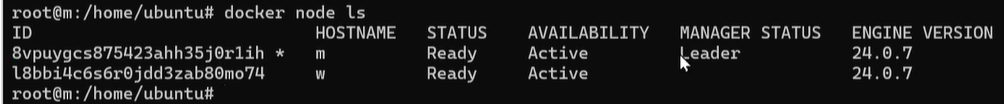
* Now run the next command highlighted the above screenshot on the worker node so that worker node joins swarm as worker.

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



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

docker node ls



The above output gives you both manager (leader) and worker details.

* Now we will create services (pod in Kubernetes) in docker swarm by running the below command on master node.

docker service create --replicas 3 -p 80:80 --name web nginx

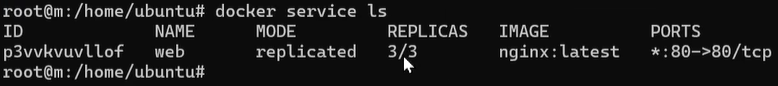
Above command will create 3 replicas (containers) on both manager and worker with the image name web and container is created for nginx with port mapping 80:80. (1st one is VM port and 2nd one after: is your container port)

A screenshot of a computer

Description automatically generated

* Next command will list the service (containers) is created or not.

docker service ls



In the above command, we can see under replicas that it is showing 3/3, which means all 3 containers are ready, it shows service name is web, image is nginx-latest, and port is 80:80

* We will test nginx is running on containers or not by checking the public IP on port 80 on web console. Below screenshot shows, it worked for master node public IP. Same we will verify for worker node Public IP also and that also worked.

A screenshot of a computer

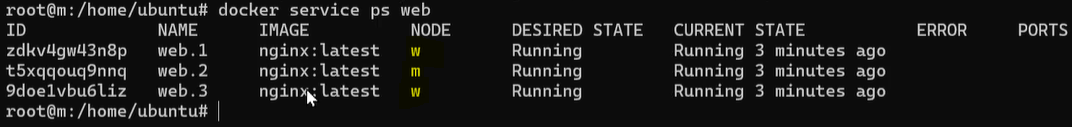
Description automatically generated

A screenshot of a computer

Description automatically generated

* Next command will check how many services are running on master and worker out of 3 services (containers) that we have created.

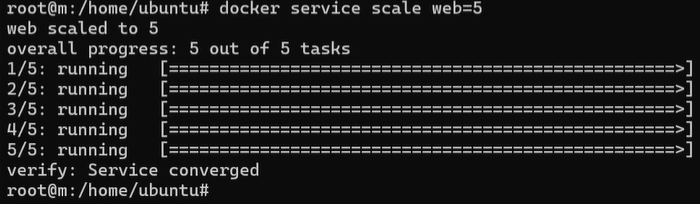
Docker service ps web (where web is the service name that we have created)



Above command shows that 2 service (containers) are running on worker node and 1 is running on master node. So, the general working model is that master also acts as worker sometimes, but it always has the least load. It also depends on the load on worker nodes, if it is too much on worker nodes, then also service (container) is created on master node.

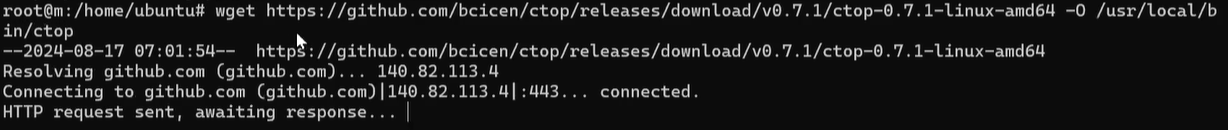
* Now if we want to scale the replicas up or down, then we will use the below command.

docker service scale web=5



* Now we will install ctop tool for docker monitoring.
* We will run the below command to install ctop for monitoring docker.

wget <https://github.com/bcicen/ctop/releases/download/v7.0.1/ctop-0.7.1-linux-amd64> -O /usr/local/bin/ctop



chmod +x /usr/local/bin/ctop



Now we will run ctop command and will see the below output

A black screen with white text

Description automatically generated

If we run the above command ctop in master, then it will show only containers that are created in master node, not the worker node. To see the containers on worker node, we need to install and run the ctop command on the worker node.

* One thing regarding leader election in docker swarm is that if the leader is not available, then a new leader will be elected from one of the worker nodes.

1. Set up a local registry

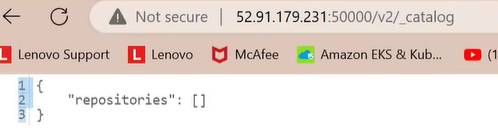
A registry can be considered private if pulling requires authentication.

docker run -d -p 50000:5000 --restart always --name my-registry registry: latest



* Above command downloads the registry image which is tagged latest. This tag references the latest version of the registry at the time of this writing.
* Exposes 5000 port to the host, under the same port
* Gives the container the name registry instead of assigning it a random name.
* -d states that the docker container creation from image will run in the background.
* -- restart always states that container will remain running.
* --name, you will give the name of your registry.
* Registry: latest, it pull and run the latest image of registry.
* Now we need to access the registry from the URL. For that we need to access it using the below link and verify whether the registry is up and running and that you have no repositories pushed to it. If below screenshot is there then registry is up and running.

http://localhost (public IP):50000/v2/\_catalog



* Now we will create a tag on nginx or any image and push the image to our registry.

docker tag [Image] [Public IP]:5000 [port of container]/ [tag name]

Below screenshot gives you the example.

A computer screen with white text

Description automatically generated

* Now we will push the image that is tagged into our registry.

docker push [Public IP]:5000/nginx

1. Docker File and Docker Image creation from Docker File

* We will now create a docker file by doing the below.

Vi Dockerfile

FROM ubuntu

MAINTAINER your\_name Sourabh

RUN apt-get -y install apache2

RUN echo “Hello Apache Server on Ubuntu Docker” > /var/www/html/index.html

EXPOSE 80

CMD /usr/sbin/apache2ctl -D FOREGROUND

A black screen with red and white text

Description automatically generated

A screen shot of a computer

Description automatically generated

* Once the dockerfile is created we will create the image.

docker build -t my-simple-web-app/webapplication:1.0 . (This command will not be supported in the latest version of docker) as shown in below screenshot also.

A screen shot of a computer program

Description automatically generated