3rd June

Dockers is a containerization tool.

Virtualization -- Fixed hardware allocation.

Containerization - No Fixed Hardware

Process isolation (Dependency in os is removed)

++

In comparison to the traditional virtualization functionalities of hypervisors,

Docker containers eliminate the need for a separate guest operating system for every new virtual machine.

Docker implements a high-level API to provide lightweight containers that run processes in isolation.

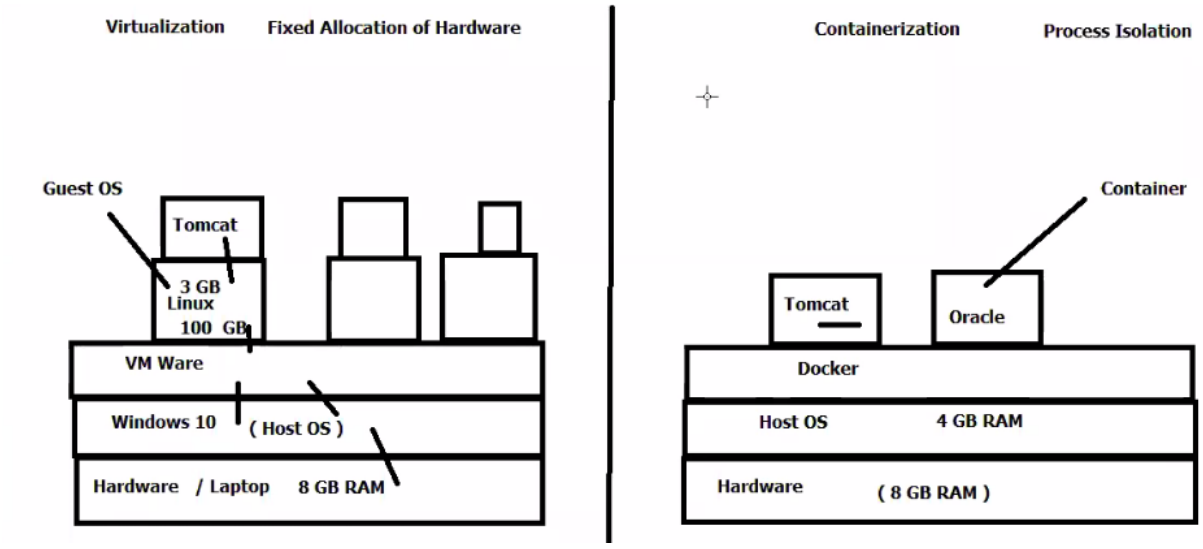
A Docker container enables rapid deployment with minimum run-time requirements. It also ensures better management and simplified portability.

This helps developers and operations team in rapid deployment of an application.

Any application which is up and running in docker is called containers

Docker and container doesn't need any os, the docker which runs on host os.

Even the o.s also possible as form of a conatiners like tomcat.



++++++++++++++++++++++++

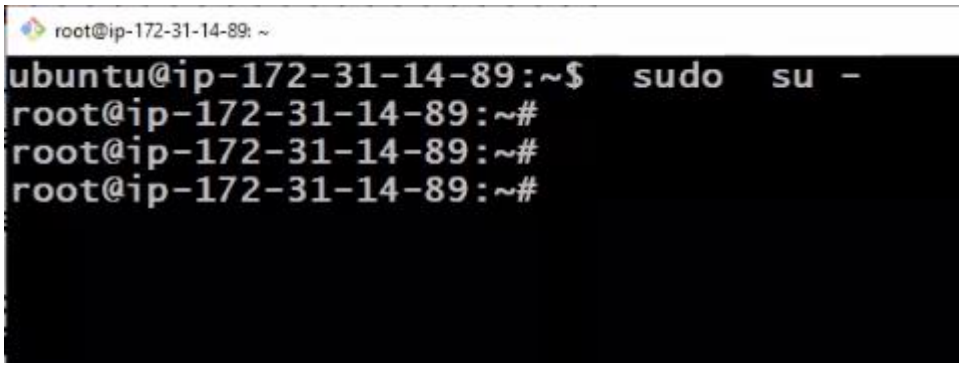
Create Ubuntu Machine on AWS—Host os

All Traffic - anywhere

Connect using git bash

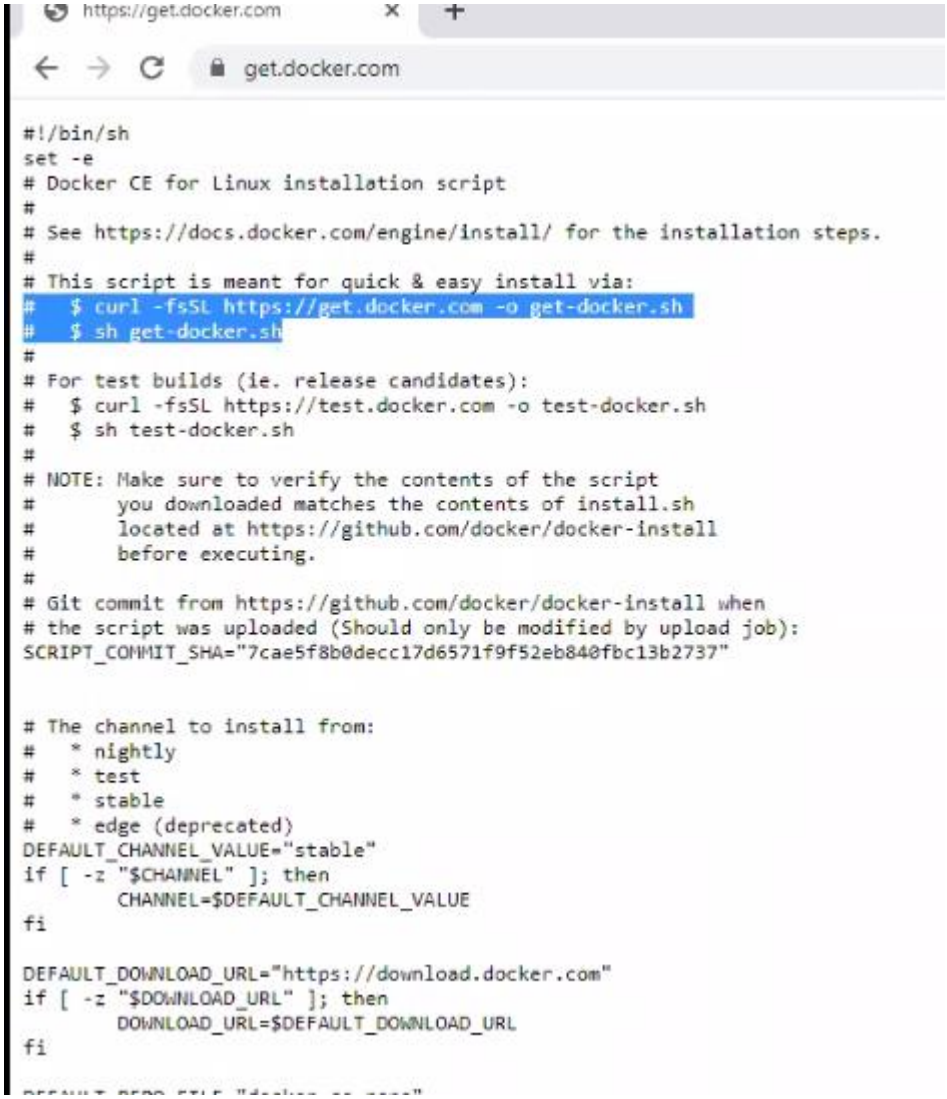
Go to Root Account

$ sudo su –

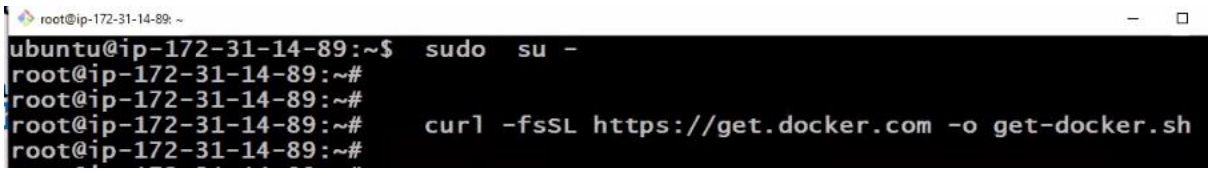


https://get.docker.com/

go to above site and copy first 2 commands to install docker.



 # curl -fsSL https://get.docker.com -o get-docker.sh (this will download shell script in the machine)



 # sh get-docker.sh (This will execute the shell script, which will install docker)

```
root@ip-172-31-14-89:~#  sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2
737
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-http
s ca-certificates curl >/dev/null
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | apt-key add
-qq - >/dev/null
Warning: apt-key output should not be parsed (stdout is not a terminal)
+ sh -c echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic s
table" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ [ -n  ]
+ sh -c apt-get install -y -qq --no-install-recommends docker-ce >/dev/null
```

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

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

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

root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

How to check the docker is installed or not

 # docker –version

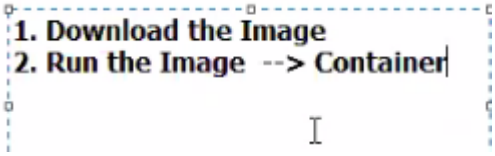
```
================================================================

root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker --version
Docker version 20.10.7, build f0df350
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

We should be comfortable with four terms

1) Docker Images

Combinations of binaries / libraries which are necessary for one software application.

2 step process



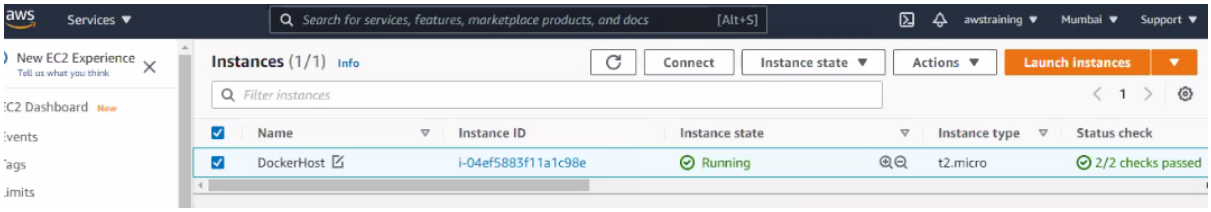
Once we download and run the image the app up and running is called container.

2) Docker Containers

When image is installed and in comes into running condition, it is called container.

3) Docker Host

Machine on which docker is installed, is called as Docker host. Ex: docker installed on Ubuntu so it is docker host.



4) Docker Client

Terminal used to run docker run commands (Git bash)

On linux machine, git bash will work like docker client.

++++++++++++++++++++++++++++-----

4th June

Docker Commands

Working on Images

---

1 To download a docker image

ducker pull image_name

Upload—

ducker Push image_name

2 To see the list of docker images (ex: if u downloaded images , want to see)

docker image ls

(or)

docker images

To see the list of containers

Container ls

3 To delete a docker image from docker host

docker rmi image_name/image_id

4) To upload a docker image into docker hub

docker push image_name

5) To tag an image

docker tag image_name ipaddress_of_local_registry:5000/image_name

6) To build an image from a customised container

docker commit container_name/container_id new_image_name

7) To create an image from docker file

docker build -t new_image_name

8) To search for a docker image

docker search image_name

9) To delete all images that are not attached to containers

docker system prune -a

++

Working on containers

----

10) To see the list of all running continers

docker container ls

11) To see the list of running and stopped containers

docker ps -a

12) To start a container

docker start container_name/container_id

13) To stop a running container

docker stop container_name/container_id

14) To restart a running container

docker restart container_name/container_id

To restart after 10 seconds

docker restart -t 10 container_name/container_id

15) To delete a stopped container

docker rm container_name/container_id

16) To delete a running container

docker rm -f container_name/container id

17) To stop all running containers

```
docker stop $(docker ps -aq)
```

18) To restart all containers

```
docker restart $(docker ps -aq)
```

19) To remove all stopped containers

```
docker rm $(docker ps -aq)
```

20) To remove all contianers(running and stopped)

```
docker rm -f  $(docker ps -aq)
```

21) To see the logs generated by a container

```
docker logs container_name/container_id
```

22) To see the ports used by a container

```
docker port container_name/container_id
```

23) To get detailed info about a container

```
docker inspect container_name/container_id
```

24) To go into the shell of a running container which is moved into background

```
docker attach container_name/container id
```

25) To execute anycommand inside a container

docker exec -it container_name/container_id command

Eg: To launch the bash shell in a contianer

docker exec -it container_name/container_id bash

26) To create a container from a docker image (imp)

docker run image_name

++

Run command options

-it **for opening an interactive terminal in a container**

--name **Used for giving a name to a container**

-d **Used for running the container in detached mode as a background process**

-e **Used for passing environment varaibles to the container**

-p **Used for port mapping between port of container with the dockerhost port.**

-P **Used for automatic port mapping ie, it will map the internal port of the container with some port on host machine.**

This host port will be some number greater than 30000

-v Used for attaching a volume to the container

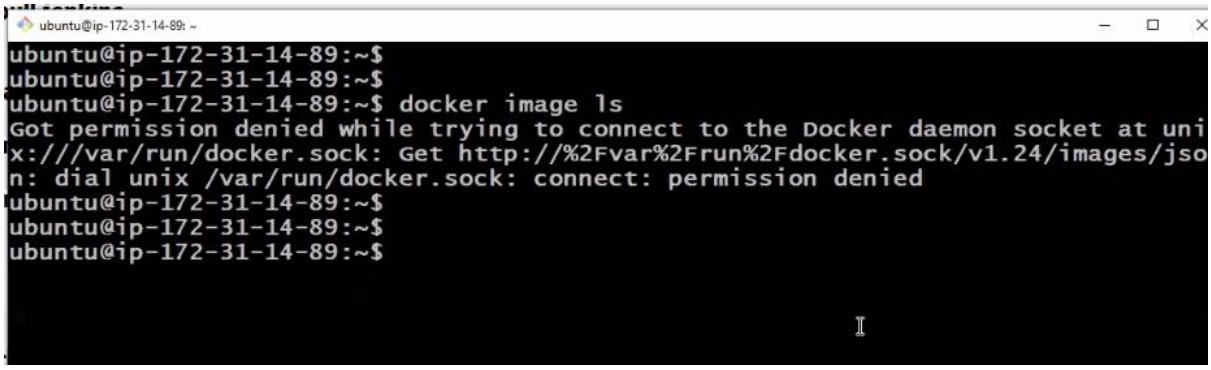
--volume -from Used for sharing volume between containers

--network Used to run the contianer on a specific network

--link Used for linking the container for creating a multi container architecture

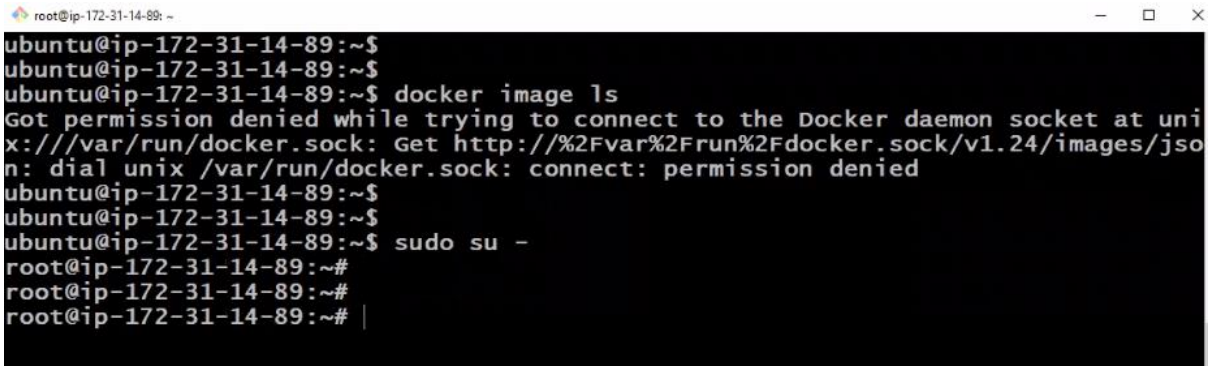
--memory Used to specify the maximum amount of ram that the container can use

++=



Access denied because we should work with Root user

Sudo su-



 # docker images (There are no images)

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~# docker image ls
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
root@ip-172-31-14-89:~#
```

No containers also.

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS   NAMES
root@ip-172-31-14-89:~#
```

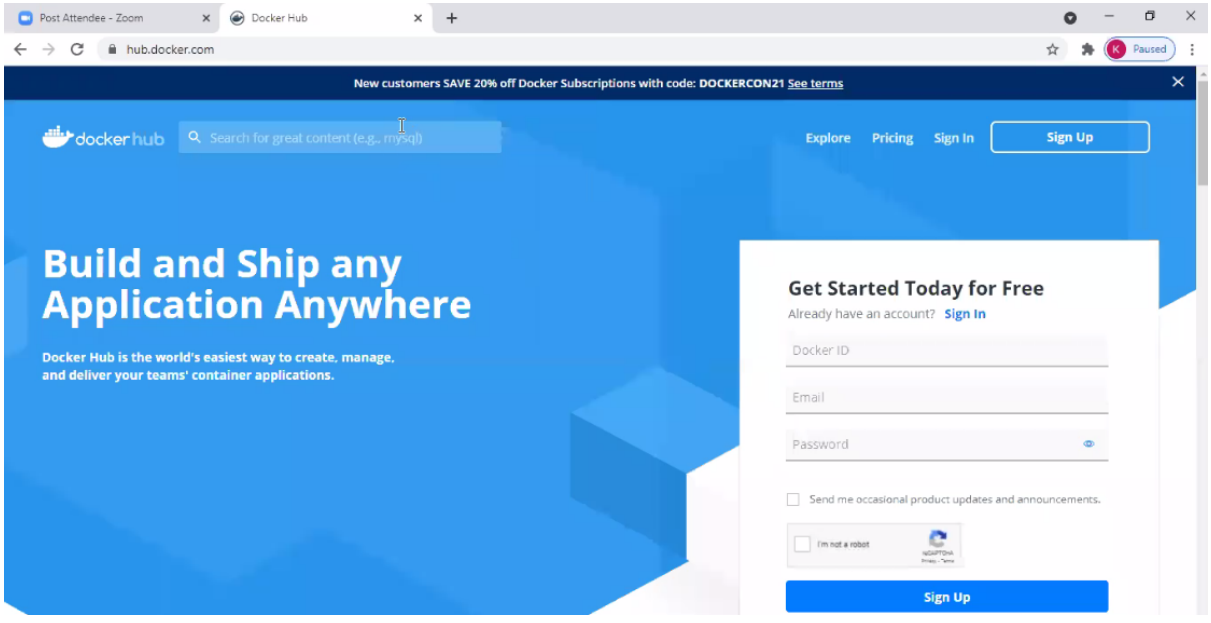
Whenever need container 1st down load image

2nd—run the image

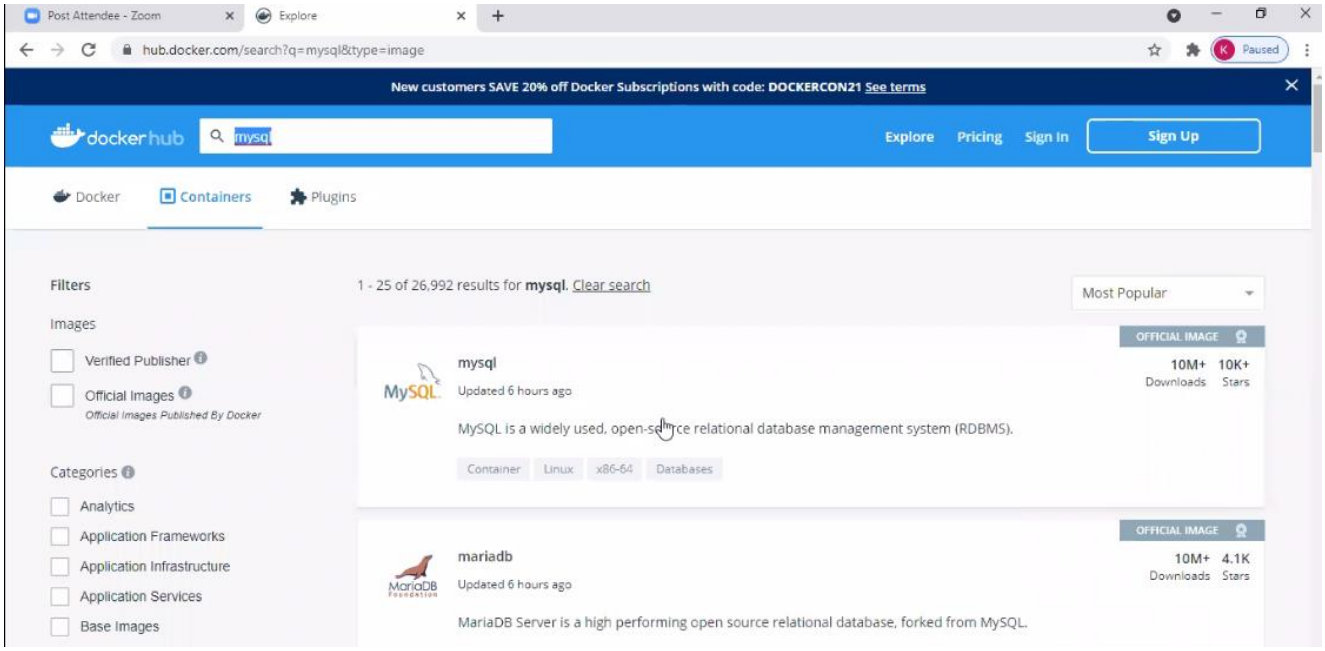
To download tomcat image

Downloaded tomee from docker hub and the connection was established automatically though internet = daemon process

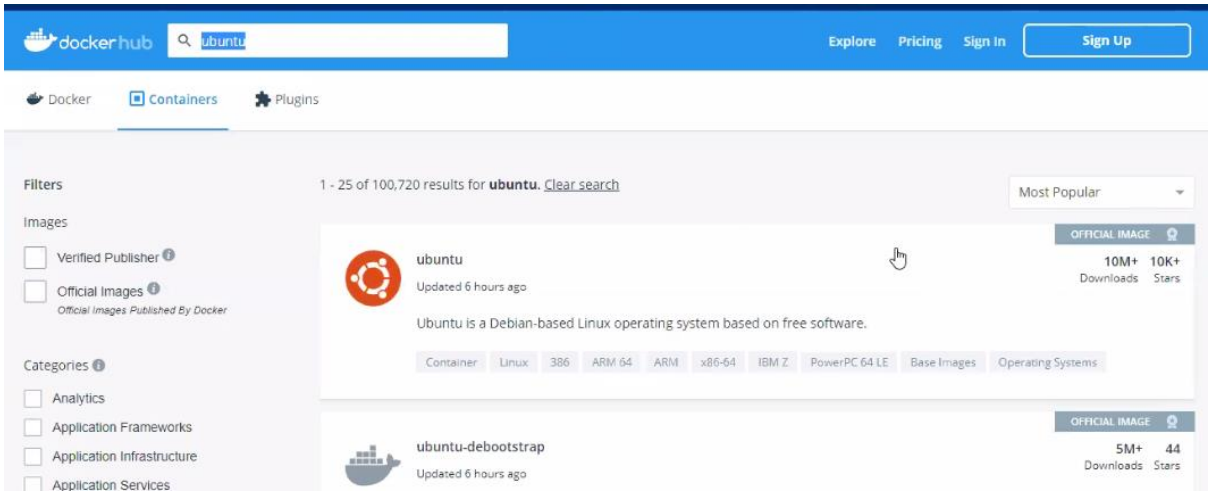
All the images are downloaded from docker hub hub.docker.com to docker host



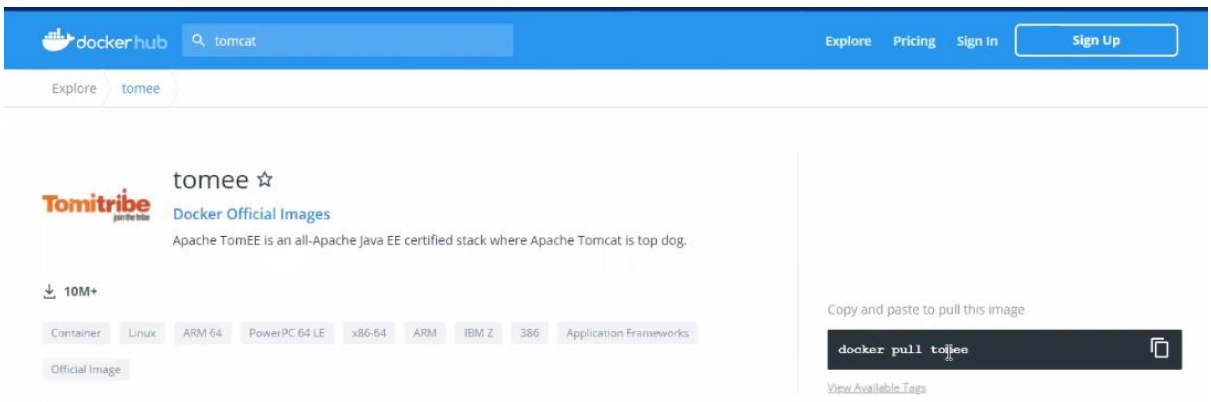
You want mysql search it



For Ubuntu search



Similarly for tomcat, we get tome and command also available



 # docker pull tomee

```
root@ip-172-31-14-89:~#  docker pull tomee
Using default tag: latest
latest: Pulling from library/tomee
b9a857cbf04d: Pull complete
d557ee20540b: Pull complete
3b9ca4f00c2e: Pull complete
6cee913589ff: Pull complete
c81f410639ad: Pull complete
3f2724b4ff1a: Pull complete
8dd9a1823c4d: Pull complete
a8f153359e1c: Pull complete
8a47b693ffcb: Pull complete
Digest: sha256:cb309d062c0d7081cb16597207b15fd03f45b45598a5f9f94b3506eca8f53c4f
Status: Downloaded newer image for tomee:latest
docker.io/library/tomee:latest
root@ip-172-31-14-89:~#
```

 # docker images

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~# docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
tomee        latest    bc73f72b54d0   4 months ago    340MB
root@ip-172-31-14-89:~#
```

 # docker pull Ubuntu

```
root@ip-172-31-14-89:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
345e3491a907: Pull complete
57671312ef6f: Pull complete
5e9250ddb7d0: Pull complete
Digest: sha256:adf73ca014822ad8237623d388cedf4d5346aa72c270c5acc01431cc
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
root@ip-172-31-14-89:~#
```

```
root@ip-172-31-14-89:~# docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
ubuntu       latest    7e0aa2d69a15   5 weeks ago     72.7MB
tomee        latest    bc73f72b54d0   4 months ago    340MB
root@ip-172-31-14-89:~#
```

If you do not specify the version, by default, we get latest version

I want to download jenkins

 # docker pull Jenkins/Jenkins

```
root@ip-172-31-14-89:~#  docker pull jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
d960726af2be: Pull complete
971efeb01290: Pull complete
63355dfa68bf: Pull complete
7e4efd84be57: Pull complete
5f4dc6ce5f68: Pull complete
1845b40ff179: Pull complete
63560e572de5: Pull complete
2778c0fb363c: Pull complete
b1bfe419e51c: Pull complete
a89ef4e70367: Pull complete
1ba4a3f12fb0: Pull complete
5d997a158691: Pull complete
e6817fe55377: Pull complete
e3f0fcf07941: Pull complete
d4c19a039946: Pull complete
21bafef74bc6: Pull complete
Digest: sha256:dff60a7e1f6e476be850c0abdaa9c1ad783c68e9abc0e3907de2a24e1
Status: Downloaded newer image for jenkins/jenkins:latest
docker.io/jenkins/jenkins:latest
root@ip-172-31-14-89:~#
```

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
jenkins/jenkins     latest    8959ce44055e   2 days ago      570MB
ubuntu              latest    7e0aa2d69a15   5 weeks ago     72.7MB
tomee               latest    bc73f72b54d0   4 months ago    340MB
root@ip-172-31-14-89:~#
```

The memory is low and light weight for all images, seen above.

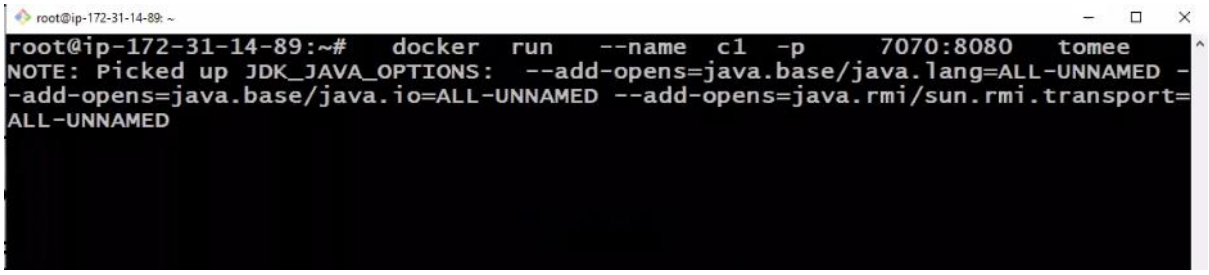
How to create container:

We know if run the image container is created. TO create a container from an image

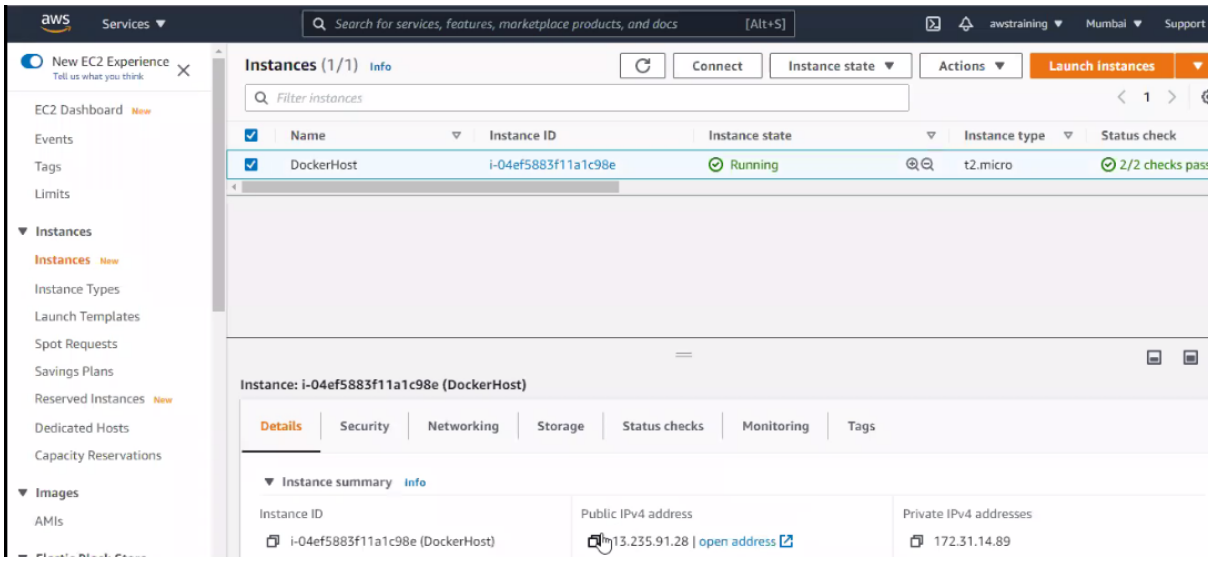
 # docker run --name mytomcat -p   7070:8080   tomee

-p= port maping default to 7070: tome=image name; --name=c1

docker run   --name c1   -p   7070:8080 tomee

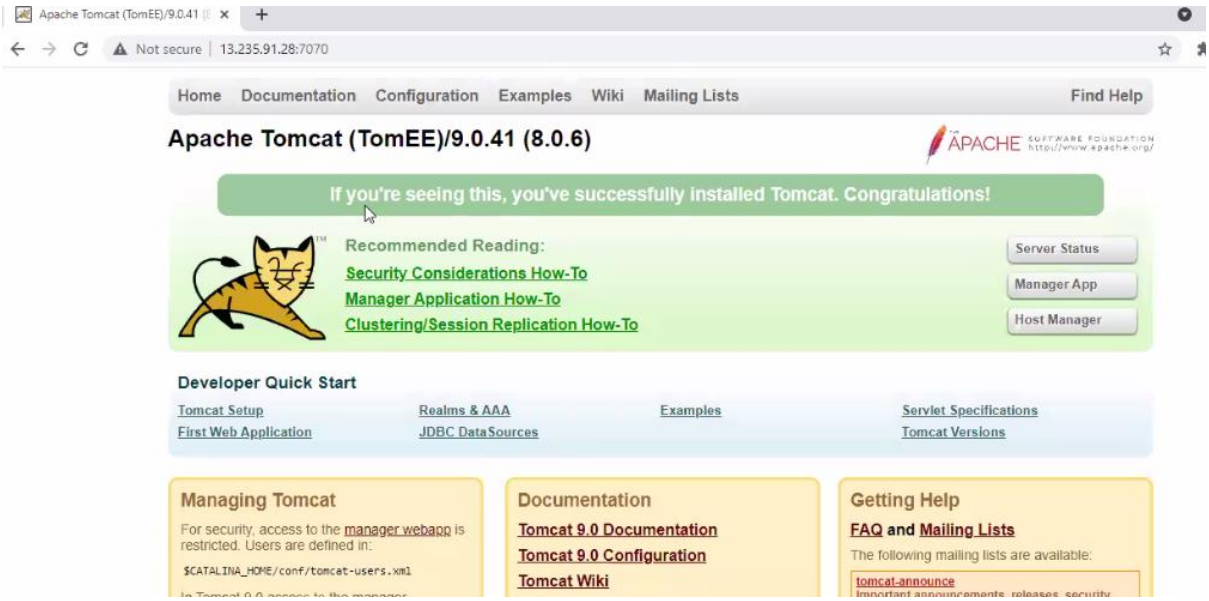


TO check the tomcat is running or not—take public ip of docker host

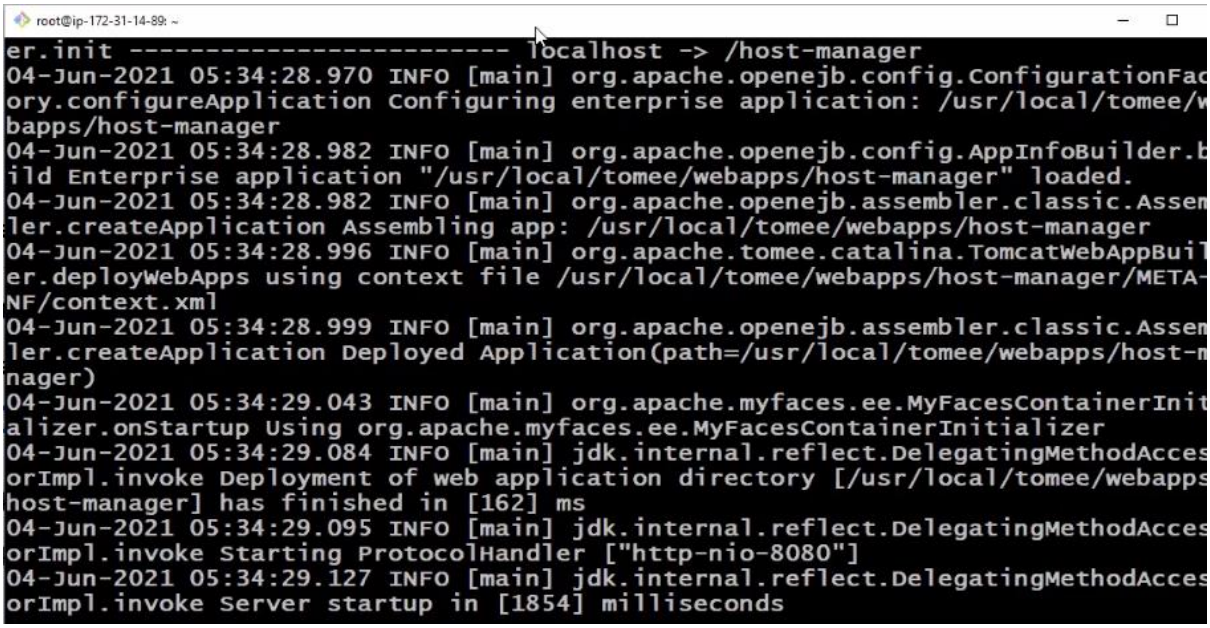


http://13.250.47.90:7070

(7070 is port number mapped in docker host)



Observe the docker is still busy with logs when start installation.

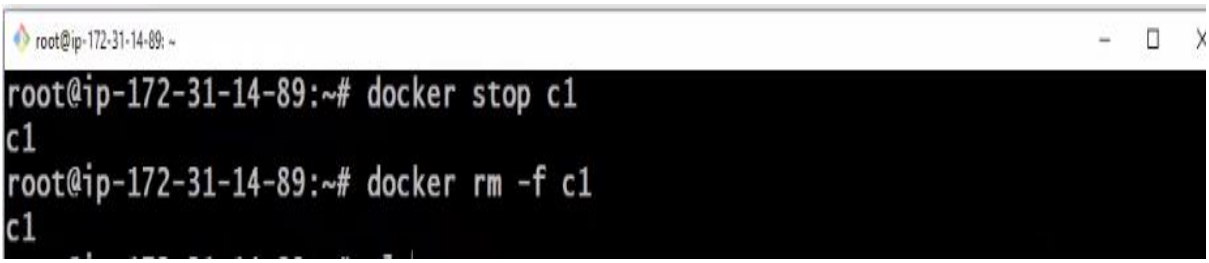


If you don’t give the port mapping for containers there is conflict because default port is same and docker host public ip is same for Jenkins.

Lets remove the container (Open another gitbash terminal)

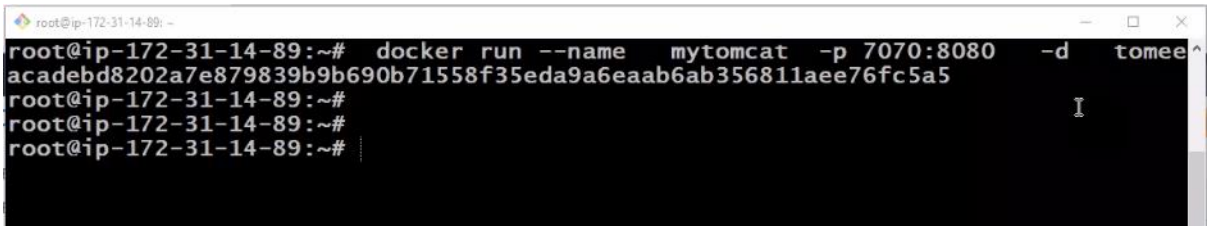
 # docker stop c1

 # docker rm -f c1



 # docker run --name mytomcat -p 7070:8080 -d tomee

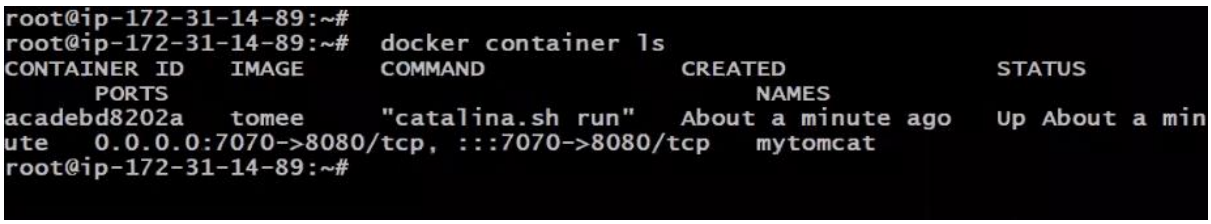
-d= detached mode.



Obsrve we are not getting log messages when installation start as compare to previous installation

(The above command runs tomcat in detached mode , so we get out # prompt back)

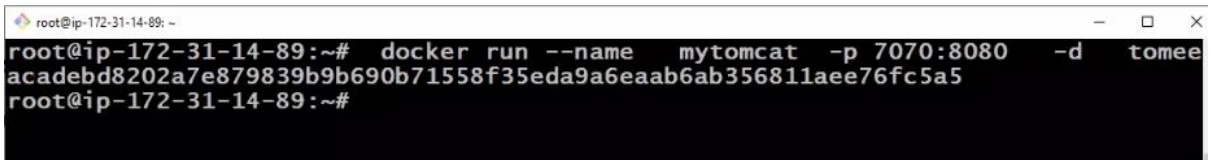
 # docker container ls



Jenkins

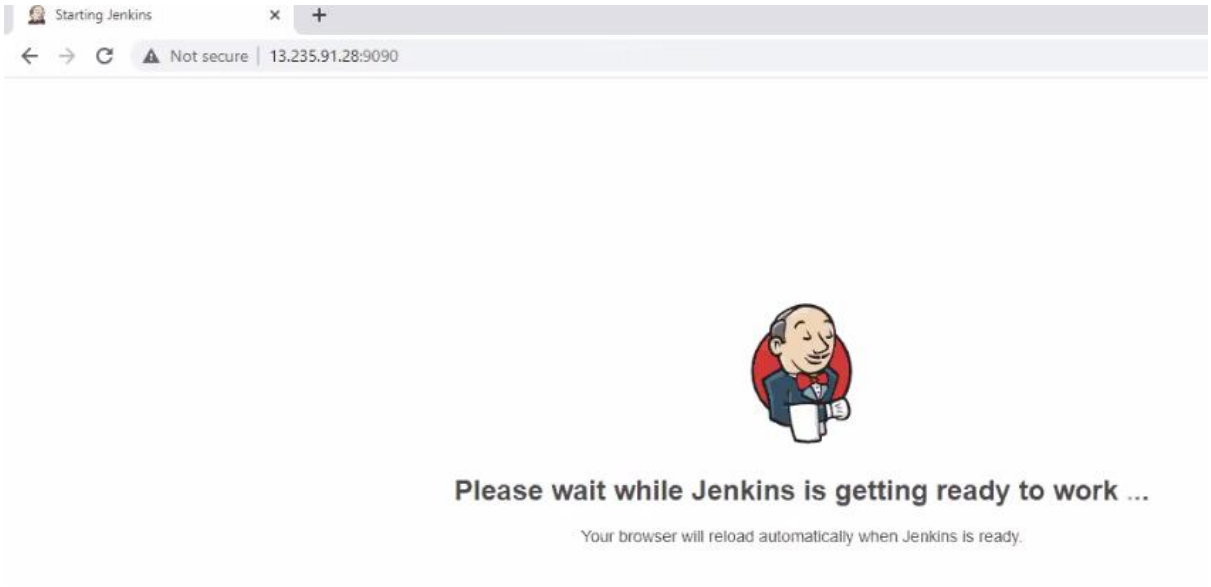
TO start jenkins

 # docker run --name myjenkins -p 9090:8080 -d Jenkins/jenkins



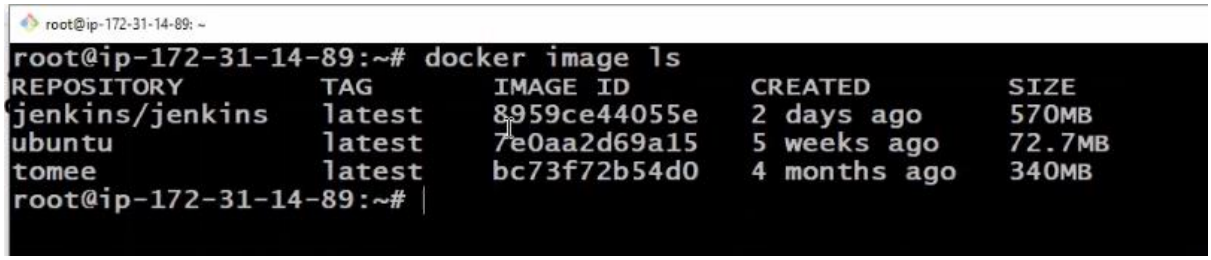
To check for jenkins (Open browser)

http://13.250.47.90:9090



To create ubuntu container—os as a container

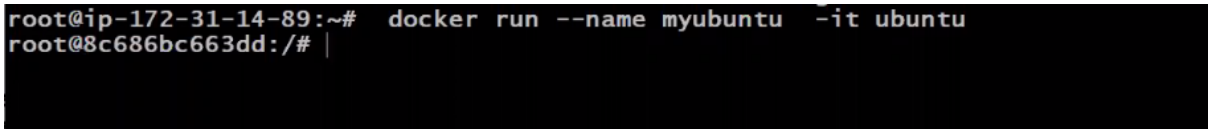
Os has no port mapping and detached(-d)

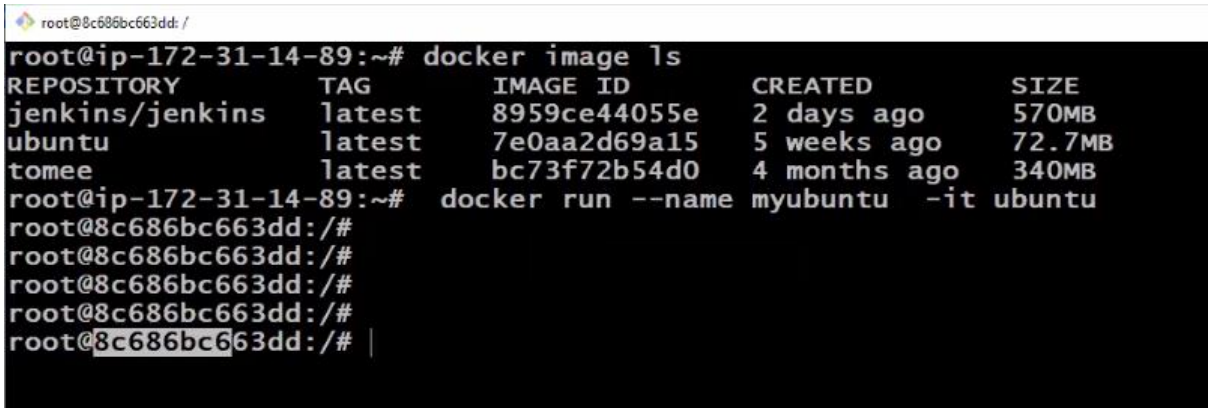


Ubuntu image is already there.

 # docker run --name myubuntu -it ubuntu

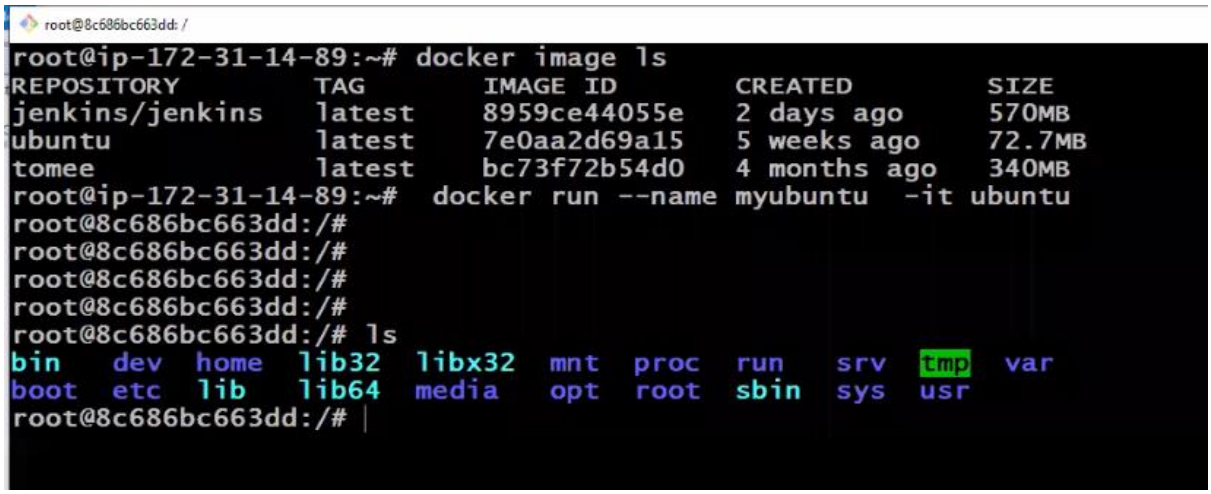
-it= interactive terminal.



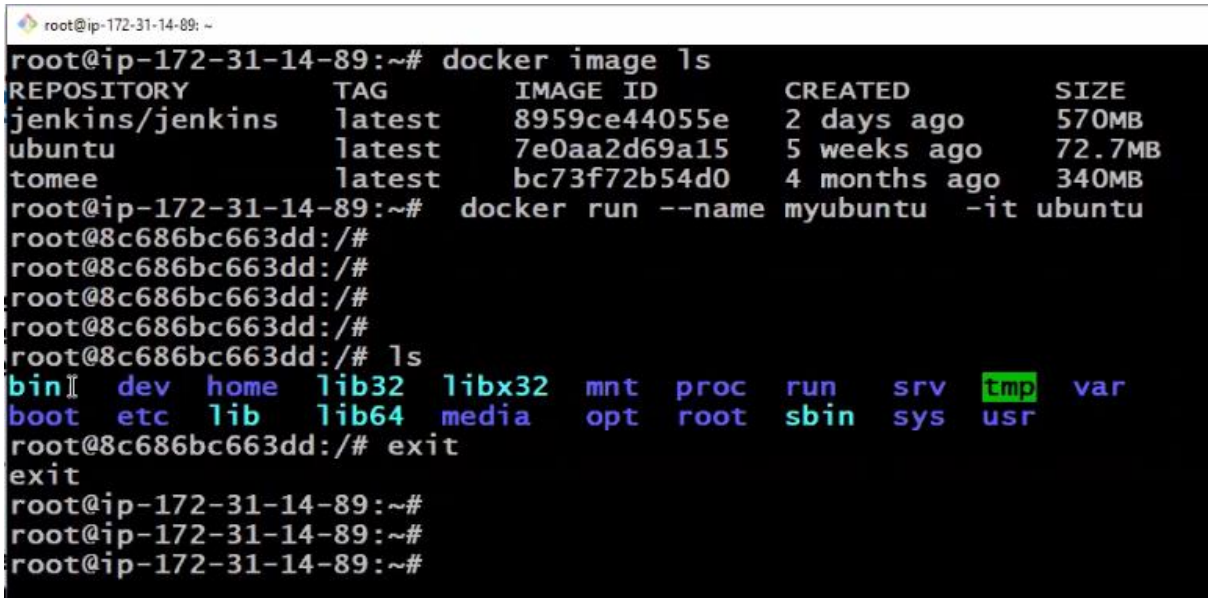


Observation: You have automatically entered into Ubuntu and didn't get any ip address

 # ls (To see the list of files in ubuntu)



 # exit (To comeout of container back to host)



Docker attach is the command that ubuntu is used to go into docker host

++
++

Scenario 1:

Start tomcat as a container and name it as "webserver". Perform port mapping and run this container in detached mode

 # docker run --name webserver -p 7070:8080 -d tomee

To access homepage of the tomcat container

Launch any browser

public_ip_of_dockerhost:7070

++++++++++++++++++++++++++++++++

Scenario 2:

Start jenkins as a container in detached mode , name is as "devserver", perform port mapping

 # docker run -d --name devserver -p 9090:8080 jenkins

To access home page of jenkins (In browser)

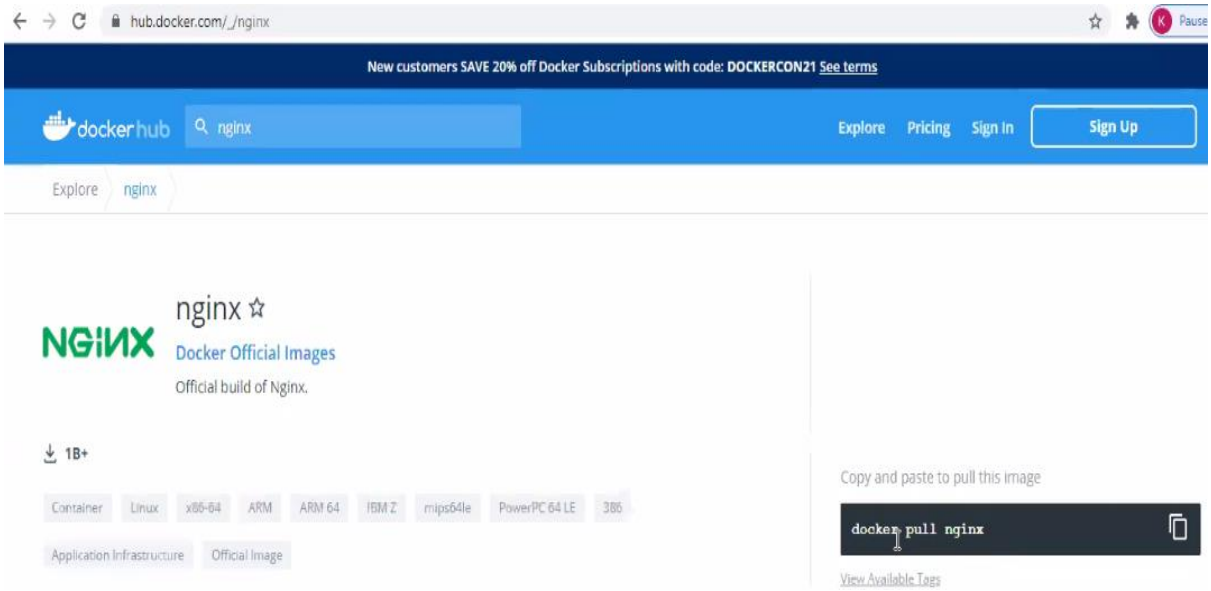
public_ip_of_dockerhost:9090

++

Scenario 3:

Start nginx as a container and name as "appserver", run this in detached mode , perform automatic port mapping

Generally we pull the image and run the image

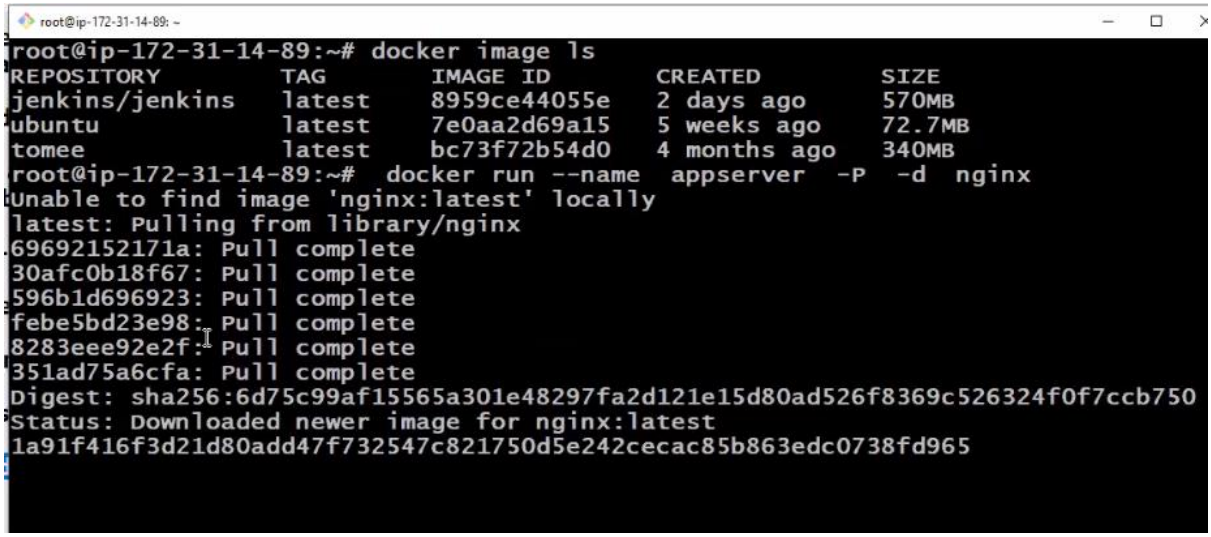


Instead of pulling, we directly run the docker to get container in one command.

 # docker run --name appserver -P -d nginx

(if image is not available, it perform pull operation automatically and its not mandatory to download image)

(Capital P , will perform automatic port mapping)



```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
jenkins/jenkins     latest    8959ce44055e   2 days ago      570MB
nginx               latest    d1a364dc548d   9 days ago      133MB
ubuntu              latest    7e0aa2d69a15   5 weeks ago     72.7MB
tomee               latest    bc73f72b54d0   4 months ago    340MB
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

How to check nginx is running or not? (we do not know the port number)

To know the port that is reserved for nginx)

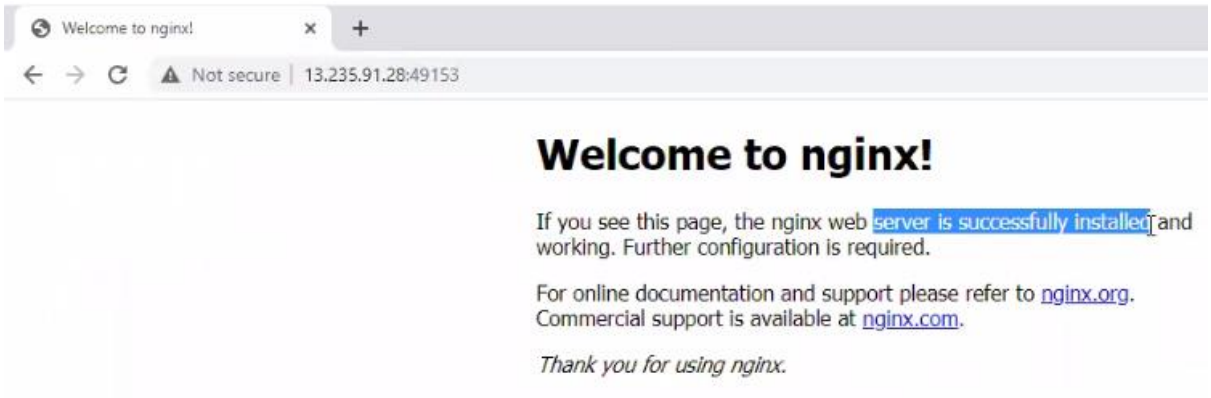
 # docker port appserver

80/tcp -> 0.0.0.0:32768

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker port appserver
80/tcp -> 0.0.0.0:49153
80/tcp -> :::49153
root@ip-172-31-14-89:~#
```

80 is nginx port

49153 is dockerhost port



or

 # docker container ls (to see the port of nginz and docker host)

```
root@ip-172-31-14-89:~# docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
1a91f416f3d2   nginx             "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    0.0.0.0:49153->80/tcp, :::49153->80/tcp                appserver
fefab04d9fb9   jenkins/jenkins   "/sbin/tini -- /usr/…"   9 minutes ago    Up 9 minutes    50000/tcp, 0.0.0.0:9090->8080/tcp, :::9090->8080/tcp   myjenkins
acadebd8202a   tomee             "catalina.sh run"        11 minutes ago   Up 11 minutes   0.0.0.0:7070->8080/tcp, :::7070->8080/tcp              mytomcat
root@ip-172-31-14-89:~#
```

To check nginx on browser

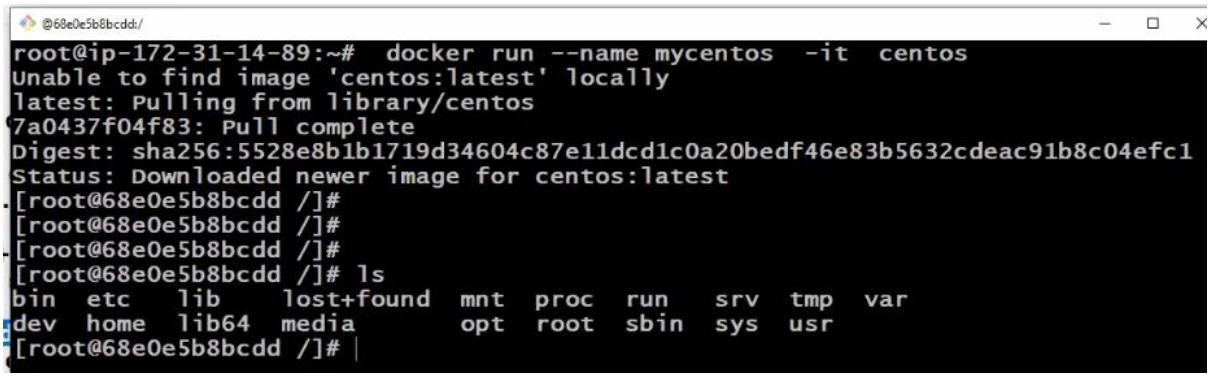
52.221.192.237:32768

++++++++++++++++++++++++++

To start centos as container

 # docker run --name mycentos -it centos

 # exit (To come back to dockerhost)



+++++++++++++++

Scenario 3:

Start nginx as a container and name as "appserver", run this in detached mode , perform automatic port mapping

Generally we pull the image and run the image

Instead of pulling, i directly

 # docker run --name appserver -P -d nginx

(if image is not available, it perform pull operation automatically)

(Capital P , will perform automatic port mapping)

How to check nginx is running or not? (we do not know the port number)

To know the port that is reserved for nginx)

 # docker port appserver

80/tcp -> 0.0.0.0:32768

80 is nginx port

32768 is dockerhost port

or

 # docker container ls (to see the port of nginz and docker host)

To check nginx on browser

52.221.192.237:32768

++++++++++++++++++++++++++

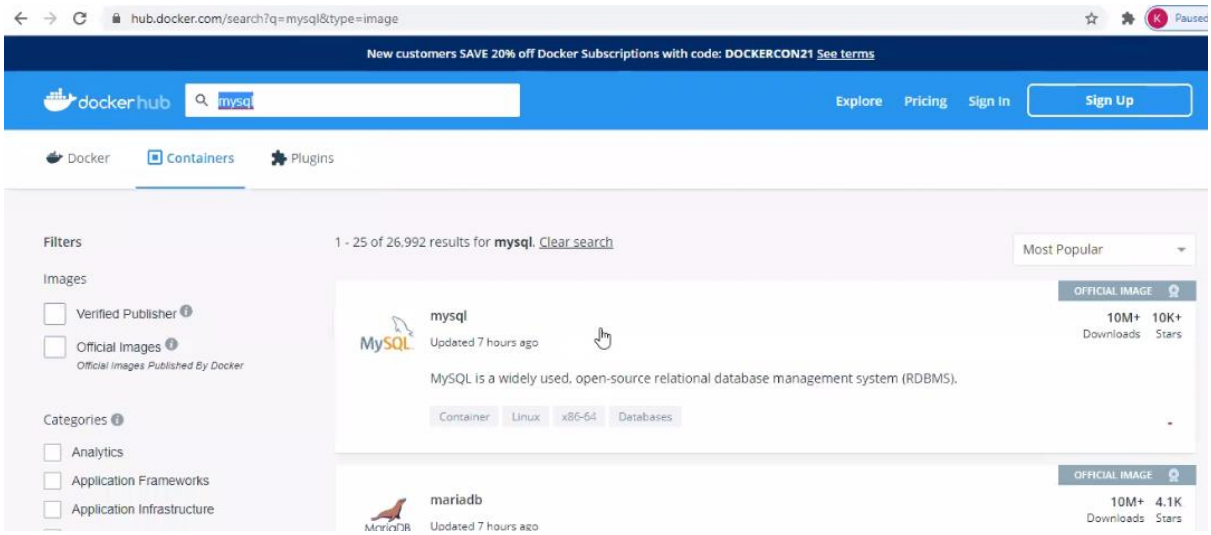
To start centos as container

 # docker run --name mycentos -it centos

 # exit (To come back to dockerhost)

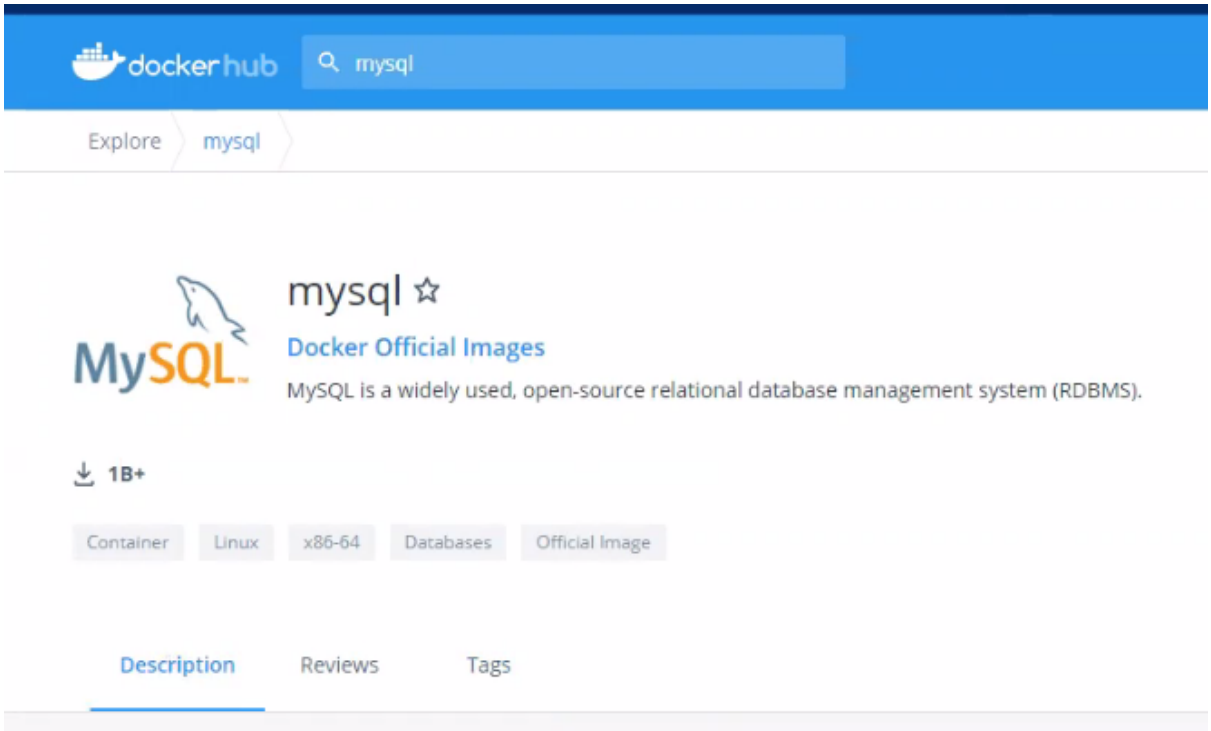
+++++++++++++++

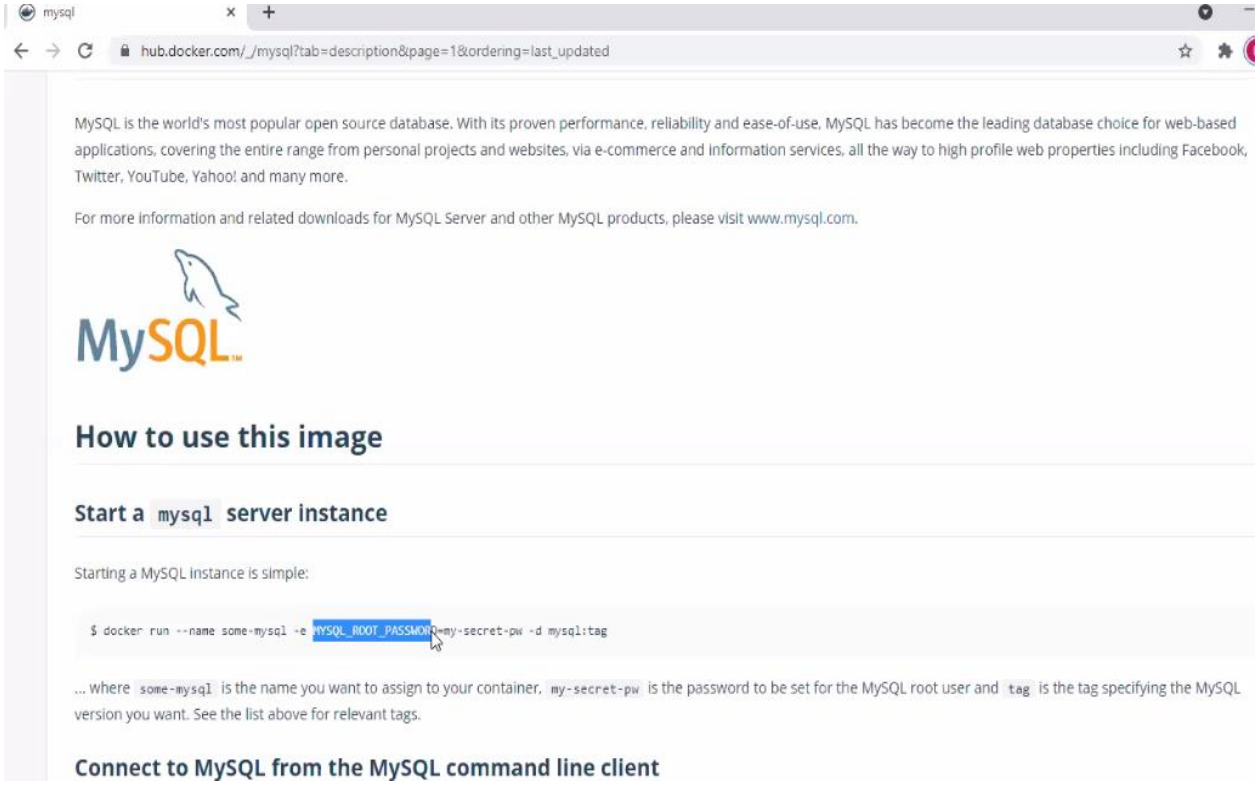
To start MySQL as container, open interactive terminal in it, create a sample table.



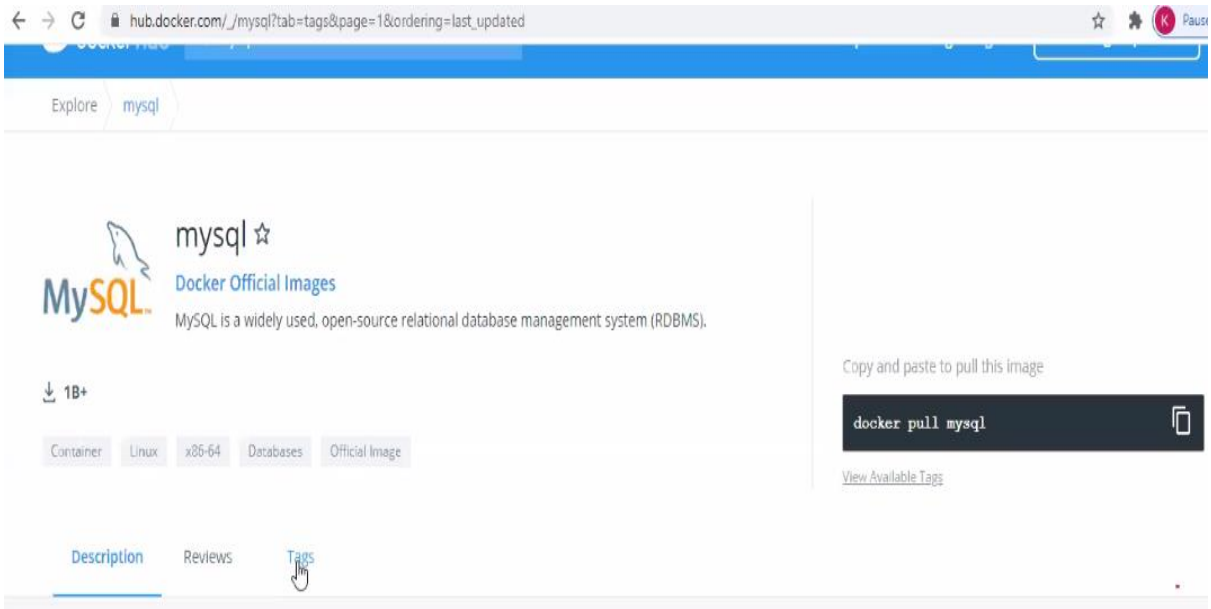
 # docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5

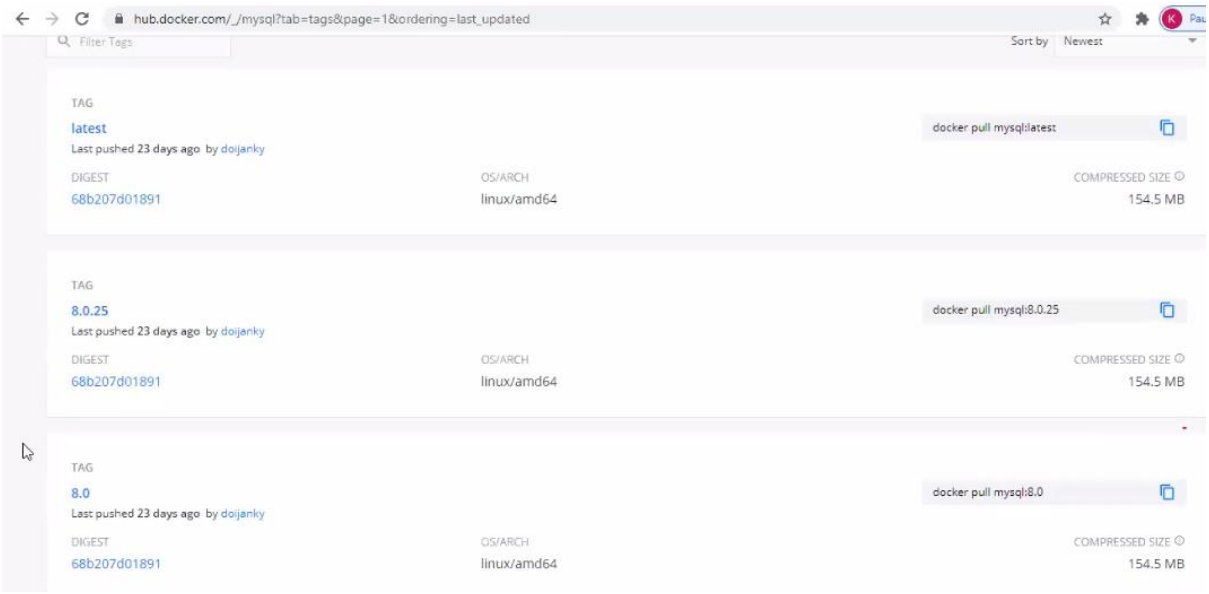
-e= environment variable (how we know that MYSQL_ROOT_PASSWORD is environment variable for tht we have to read doc in mysql git hub) clck on description

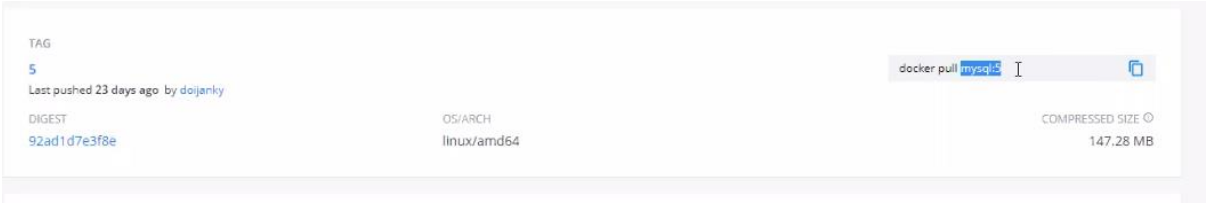


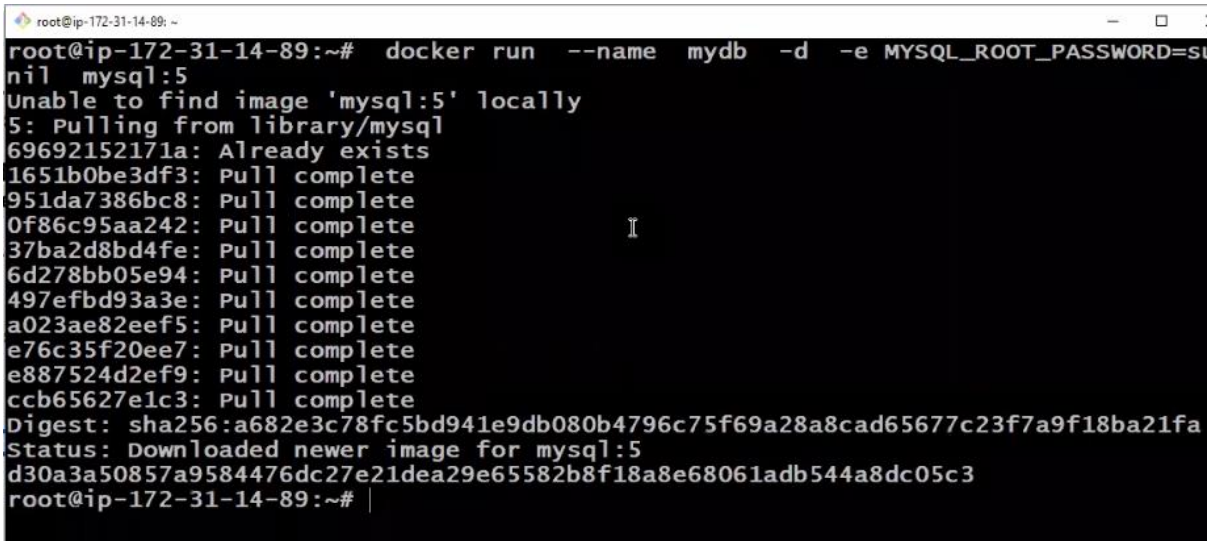


Mysql:5 5= Tags means versions, see in docker hub website




hub.docker.com/_/mysql?tab=tags&page=1&ordering=last_updated
Filter Tags
Sort by Newest
TAG
latest
Last pushed 23 days ago by doijanky
DIGEST
68b207d01891
OS/ARCH
linux/amd64
COMPRESSED SIZE
154.5 MB
docker pull mysql:latest
TAG
8.0.25
Last pushed 23 days ago by doijanky
DIGEST
68b207d01891
OS/ARCH
linux/amd64
COMPRESSED SIZE
154.5 MB
docker pull mysql:8.0.25
TAG
8.0
Last pushed 23 days ago by doijanky
DIGEST
68b207d01891
OS/ARCH
linux/amd64
COMPRESSED SIZE
154.5 MB
docker pull mysql:8.0


TAG
5
Last pushed 23 days ago by doijanky
DIGEST
92ad1d7e3f8e
OS/ARCH
linux/amd64
COMPRESSED SIZE
147.28 MB
docker pull mysql:5


root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~# docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=su
nil mysql:5
Unable to find image 'mysql:5' locally
5: Pulling from library/mysql
69692152171a: Already exists
1651b0be3df3: Pull complete
951da7386bc8: Pull complete
0f86c95aa242: Pull complete
37ba2d8bd4fe: Pull complete
6d278bb05e94: Pull complete
497efbd93a3e: Pull complete
a023ae82eef5: Pull complete
e76c35f20ee7: Pull complete
e887524d2ef9: Pull complete
ccb65627e1c3: Pull complete
Digest: sha256:a682e3c78fc5bd941e9db080b4796c75f69a28a8cad65677c23f7a9f18ba21fa
Status: Downloaded newer image for mysql:5
d30a3a50857a9584476dc27e21dea29e65582b8f18a8e68061adb544a8dc05c3
root@ip-172-31-14-89:~#

 # docker container ls

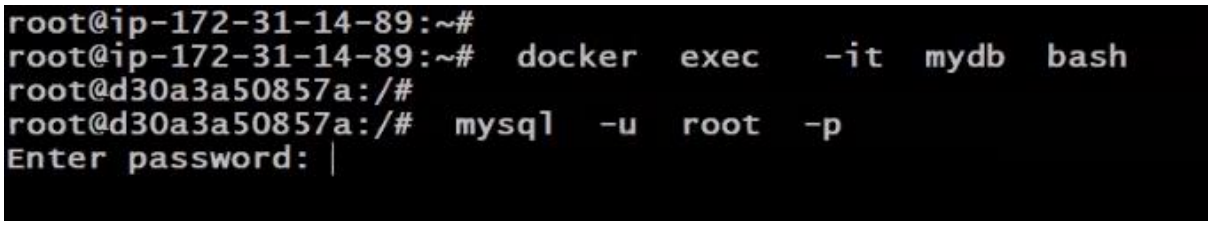
I want to open bash terminal of MySQL

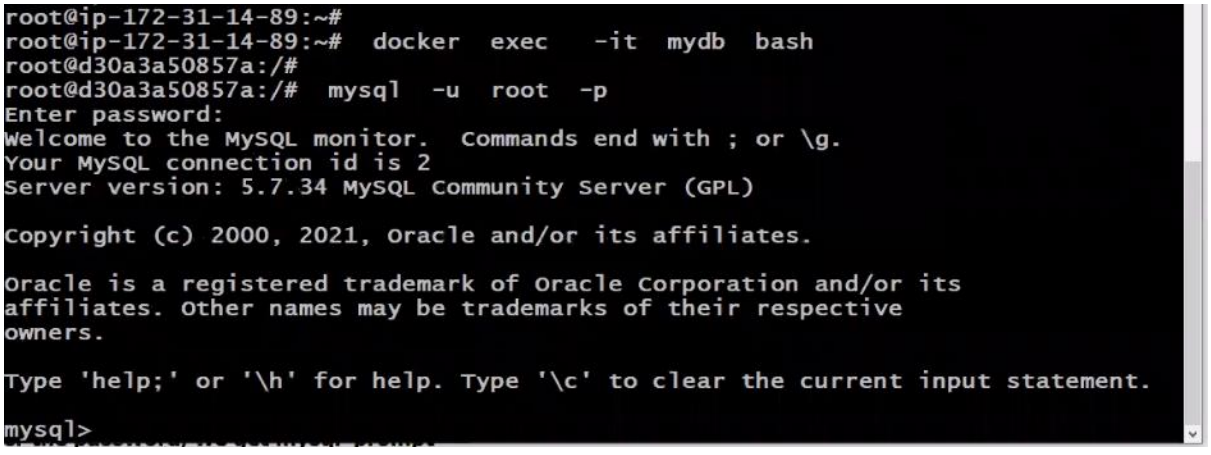
 # docker exec -it mydb bash

To connect to mysql database

 # mysql -u root -p

enter the password, we get mysql prompt





TO see list of databases

> show databases;

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql>
mysql>  show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.03 sec)

mysql>
```

TO switch to a databse

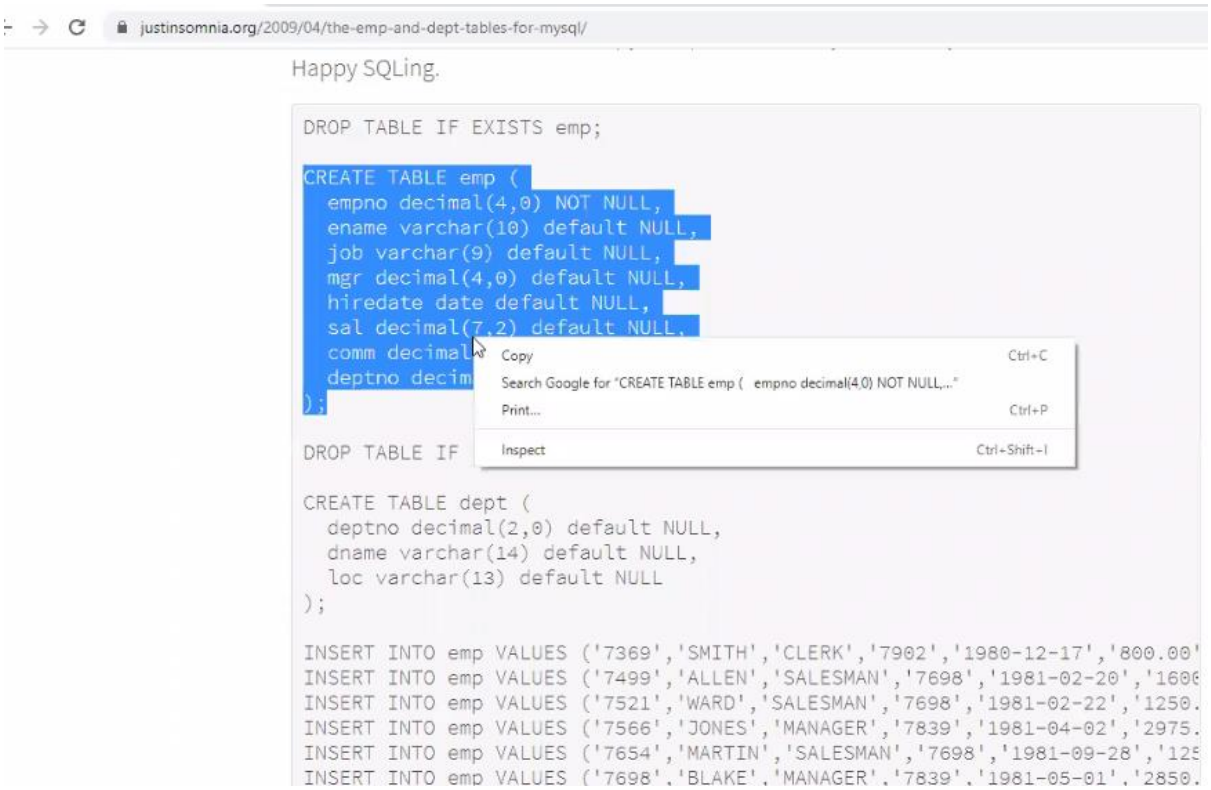
> use db_name

> use mysql

```
mysql>
mysql>  show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.03 sec)

mysql>  use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql>
mysql>
```

TO create emp tables and dept tables

https://justinsomnia.org/2009/04/the-emp-and-dept-tables-for-mysql/



Copy the code and paste

```
mysql>
mysql>
mysql> CREATE TABLE emp (
    ->   empno decimal(4,0) NOT NULL,
    ->   ename varchar(10) default NULL,
    ->   job varchar(9) default NULL,
    ->   mgr decimal(4,0) default NULL,
    ->   hiredate date default NULL,
    ->   sal decimal(7,2) default NULL,
    ->   comm decimal(7,2) default NULL,
    ->   deptno decimal(2,0) default NULL
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql>
```

> exit—come out of db

 # exit-- come out of container

 # exit

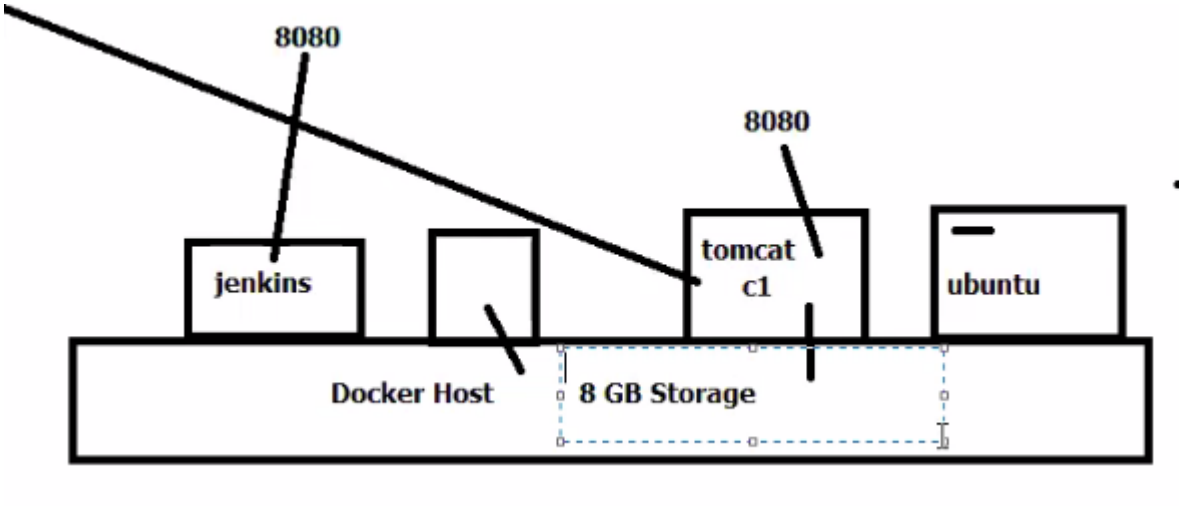
```
mysql> CREATE TABLE emp (
    ->   empno decimal(4,0) NOT NULL,
    ->   ename varchar(10) default NULL,
    ->   job varchar(9) default NULL,
    ->   mgr decimal(4,0) default NULL,
    ->   hiredate date default NULL,
    ->   sal decimal(7,2) default NULL,
    ->   comm decimal(7,2) default NULL,
    ->   deptno decimal(2,0) default NULL
    -> );
Query OK, 0 rows affected (0.10 sec)

mysql> exit
Bye
root@d30a3a50857a:/# exit
exit
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

Every container has unique ip address

```
root@ip-172-31-14-89:~# docker inspect  mydb
```

root@ip-172-31-14-89: ~


```
            "MacAddress": "02:42:ac:11:00:05",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "435623a722de5b2cb274dab2ef9c8c7e46da82f5685761
92aa85adb206d03d95",
                    "EndpointID": "c9f10f91f10060ee0bce7afc9fe9498883f0d13284d8a
fde034e49b32fceb622",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.5",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:05",
                    "DriverOpts": null
                }
            }
        }
    }
]
root@ip-172-31-14-89:~#
```



We can increase the storage of docker host when the containers are taking more storage.

Ctrl+p+q= exit from os to docker.

++
++

7th june

++++++++++++++++++

Multi container architecture using docker

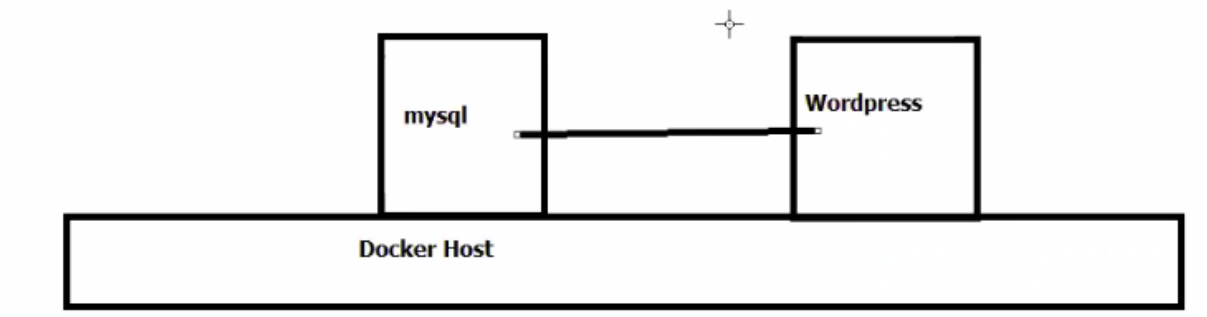
Ex: mysql, wordpress

Dev team may/may not need individual containers.

Dev team want 2 containers which are linked/connected each other.

Wordpress-front end

Mysql- backend



This can be done in 2 ways

1) --link

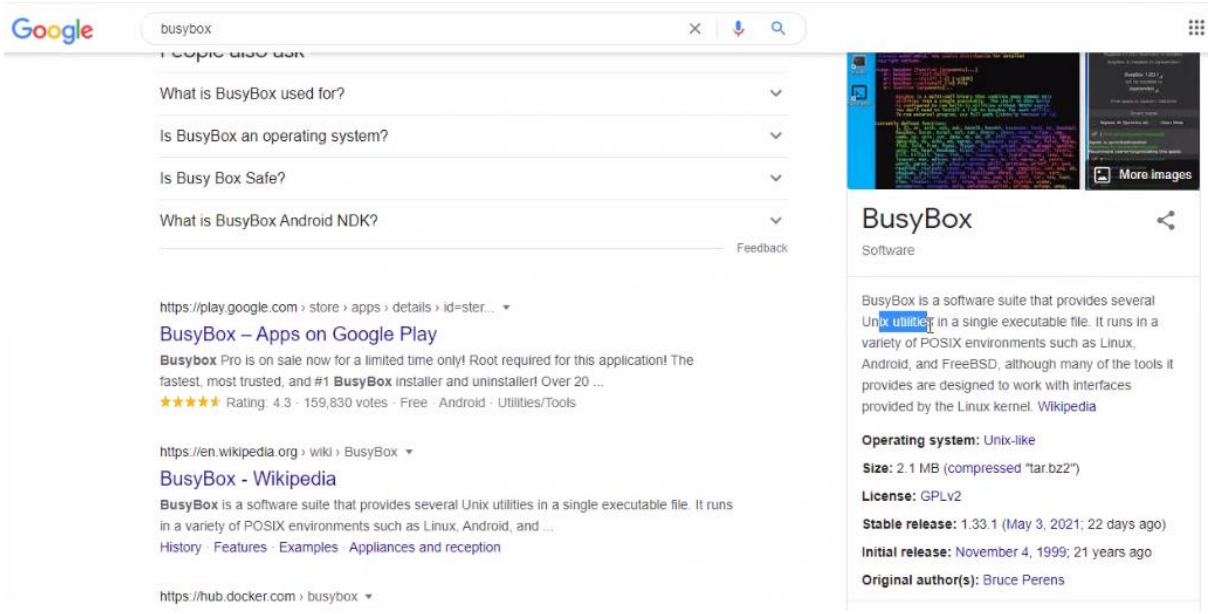
2) docker-compose

1) --link option

Use case:

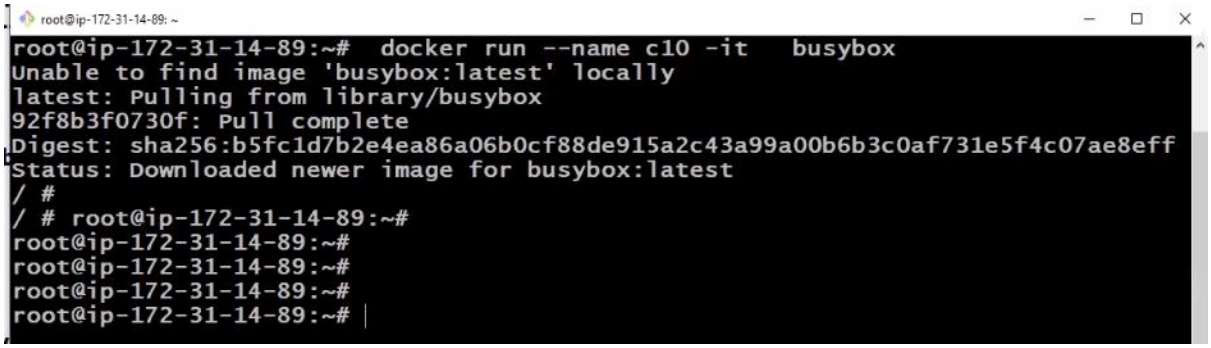
Start two busybox containers and create link between them

What is busy box



Create 1st busy box container

 # docker run --name c10 -it busybox



/ #

How to come out of the container without exit

(ctrl + p + q)

Create 2nd busy box container and establish link to c1 container

 # docker run --name c20 --link c10:c10-alias -it busybox (c10-alias is alias name)

--link= 2nd container linked to 1st container

--link c10:c10-alias(link name)

/ #

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker run   --name  c20 --link c10:c10-alias  -it
busybox
/ #
/ #
/ #
/ #
```

How to check link is established for not?

/ # ping c10

Ctrl +c (to come out from ping)

```
root@ip-172-31-14-89:~#  docker run   --name  c20 --link c10:c10-alias  -it
busybox
/ #
/ #
/ #
/ # ping c10
PING c10 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.093 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.074 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.072 ms
64 bytes from 172.17.0.2: seq=3 ttl=64 time=0.071 ms
64 bytes from 172.17.0.2: seq=4 ttl=64 time=0.071 ms
64 bytes from 172.17.0.2: seq=5 ttl=64 time=0.072 ms
64 bytes from 172.17.0.2: seq=6 ttl=64 time=0.075 ms
64 bytes from 172.17.0.2: seq=7 ttl=64 time=0.076 ms
64 bytes from 172.17.0.2: seq=8 ttl=64 time=0.074 ms
^C
--- c10 ping statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max = 0.071/0.075/0.093 ms
/ #
```

(ctrl + p + q)

++++++++++++++++++++++++++++++++

Ex 2: Creating development environment using docker

Start mysql as container and link it with wordpress container.

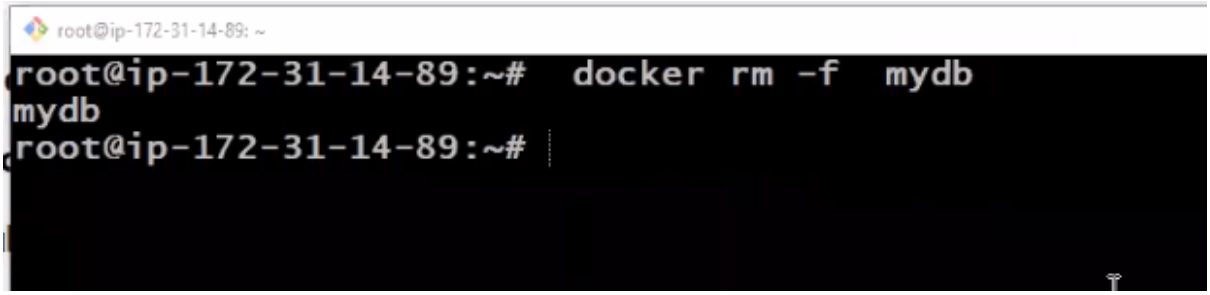
Developer should be able to create wordpress website

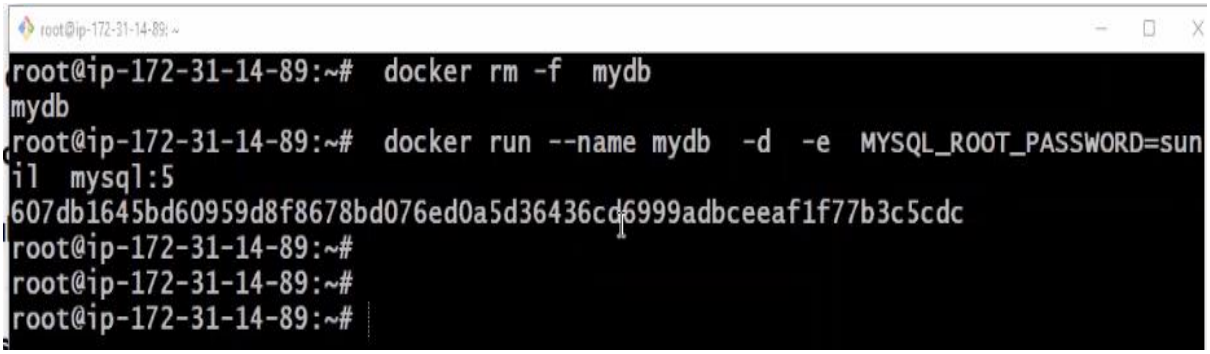
1) TO start mysql as container

 # docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5

(if container is already in use , remove it

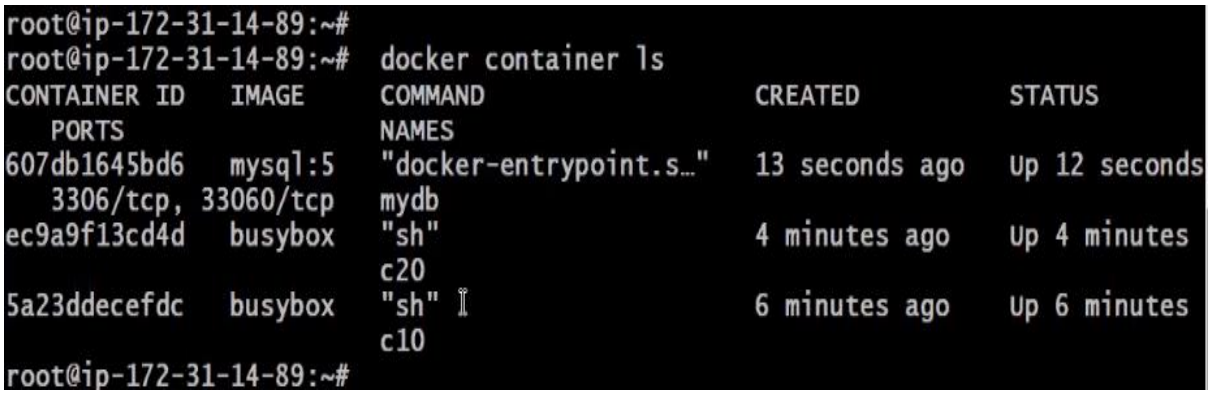
 # docker rm -f mydb)



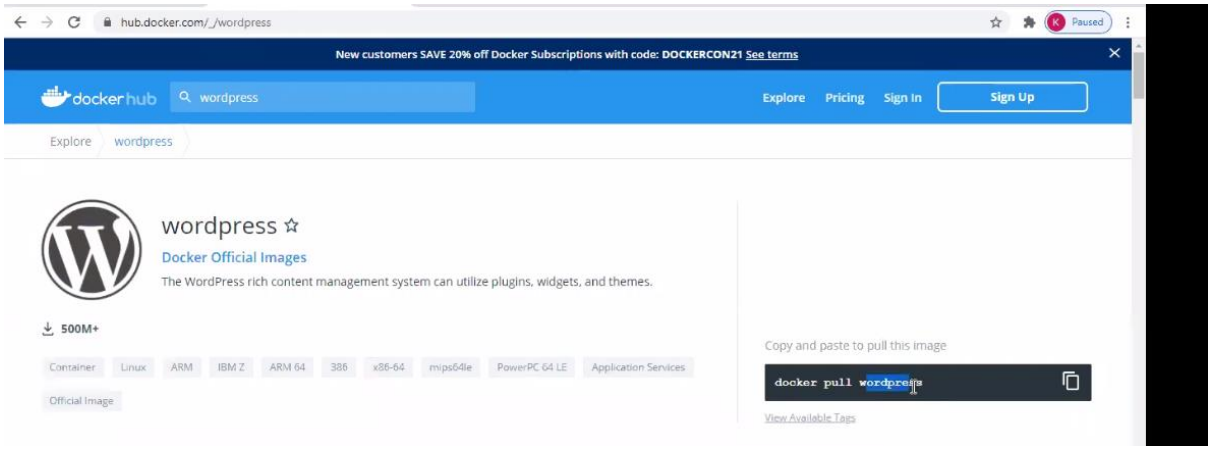


Check whether the container is running or not

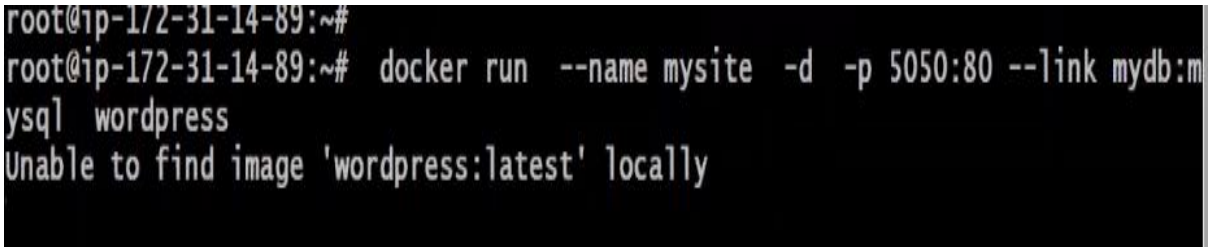
 # docker container ls



2) TO start wordpress container



 # docker run --name mysite -d -p 5050:80 --link mydb:mysql wordpress



Check wordpress installed or not

Open browser

public_ip:5050

18.138.58.3:5050

Note: if we want to link existing container to be done by the docker containers.

++

Ex 3: Create LAMP Architecture using docker

Lamp- all are open source technologies

Wmap- w=windows

L -- linux

A -- apache tomcat

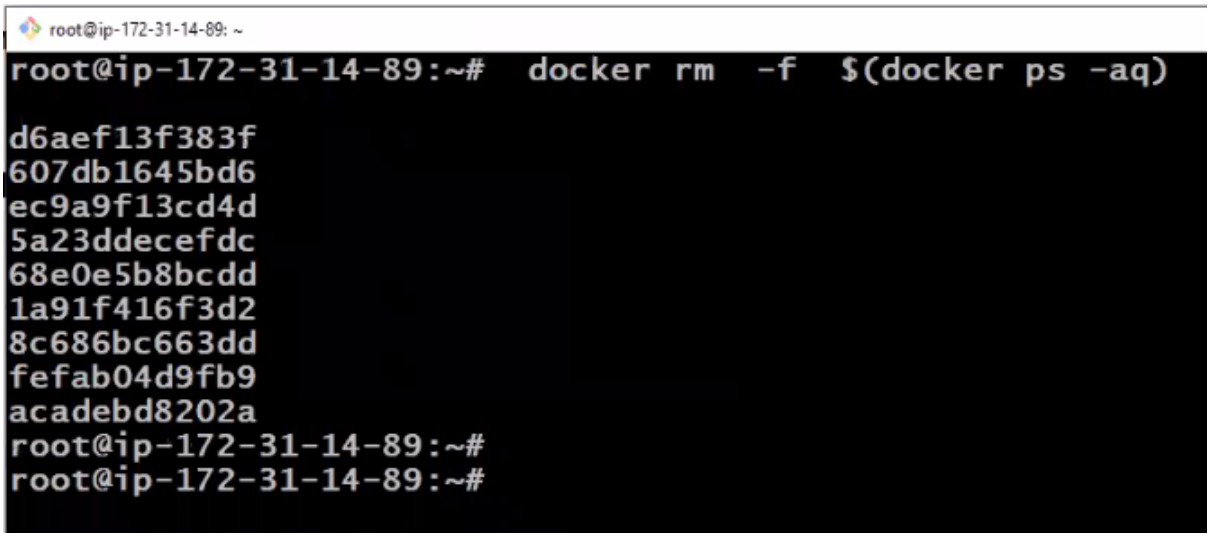
M -- mysql

P -- php

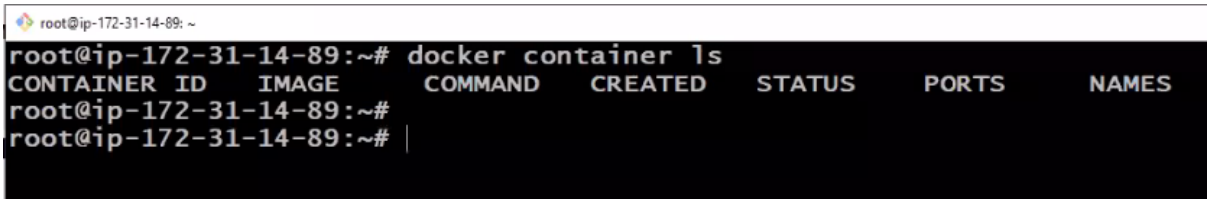
(Linux os we already have –docker host itself is a Linux)

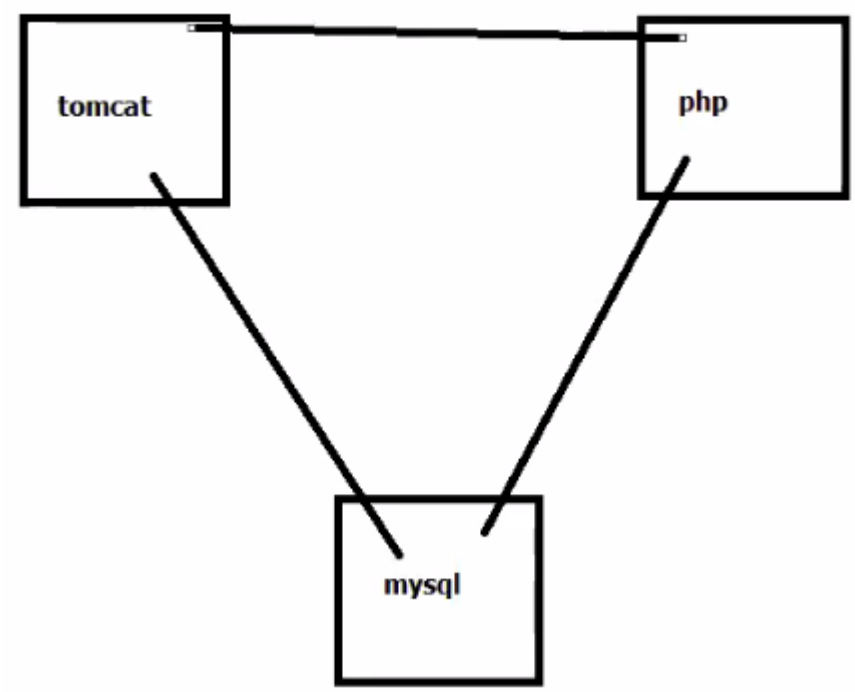
Lets remove all the docker containers

 # docker rm -f $(docker ps -aq)



 # docker container ls (we have no containers now)





1) TO start mysql as container

 # docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker run --name mydb  -d  -e  MYSQL_ROOT_PASSWORD=sun
il  mysql:5
02c289fd91855a5b23c662df601ffb5f9b65072ceee2d58ce315aa9493234b16
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

2) TO start tomcat as container

 # docker run --name apache -d -p 6060:8080 --link mydb:mysql tomcat

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker run  --name  apache  -d  -p 6060:8080  --link my
db:mysql  tomcat
Unable to find image 'tomcat:latest' locally
latest: Pulling from library/tomcat
d960726af2be: Already exists
e8d62473a22d: Pull complete
8962bc0fad55: Pull complete
65d943ee54c1: Pull complete
da20b77f10ac: Pull complete
8669a096f083: Pull complete
e0c0a5e9ce88: Pull complete
f7f46169d747: Pull complete
42d8171e56e6: Pull complete
774078a3f8bb: Pull complete
Digest: sha256:71703331e3e7f8581f2a8206a612dbeedfbc7bb8caeee972eadca1cc4a72e6b1
Status: Downloaded newer image for tomcat:latest
cc205edea9b493eceda6e12e7111f520802886e6901ff450a37c134733b6ccf1
```

TO see the list of containers

 # docker container ls

```
root@ip-172-31-14-89:~# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS
          PORTS                                      NAMES
cc205edea9b4   tomcat    "catalina.sh run"        15 seconds ago       Up 14 sec
onds       0.0.0.0:6060->8080/tcp, :::6060->8080/tcp   apache
02c289fd9185   mysql:5   "docker-entrypoint.s…"   About a minute ago   Up About
a minute   3306/tcp, 33060/tcp                         mydb
root@ip-172-31-14-89:~#
```

To check if tomcat is linked with mysql

 # docker inspect apache (apache is the name of the container)

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker inspect apache
```

```
root@ip-172-31-14-89: ~
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "private",
            "Cgroup": "",
            "Links": [
                "/mydb:/apache/mysql"
            ],
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
```

3) TO start php as container

 # docker run --name php -d --link apache:tomcat --link mydb:mysql php

We can attached n numbe of conatines by using --link

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker  run --name php  -d --link apache:tomcat  --link
 mydb:mysql    php
Unable to find image 'php:latest' locally
latest: Pulling from library/php
69692152171a: Already exists
2040822db325: Already exists
9b4ca5ae9dfa: Already exists
ac1fe7c6d966: Already exists
7994240c01ee: Downloading  114.4kB/11.09MB
17d30fab02f4: Download complete
0e0d506afaf7: Downloading    335kB/33.04MB
dd2edbf3c029: Waiting
195ee400b641: Waiting
```

PHP is not visible It is process.

Check with inspect command about linking

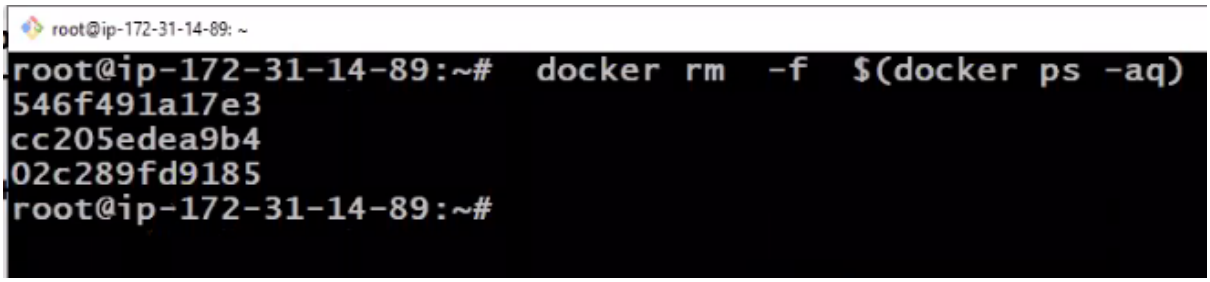
++++++++++++++++++++

ex 4:

Create CI-CD environment, where Jenkins container is linked with two tomcat containers.

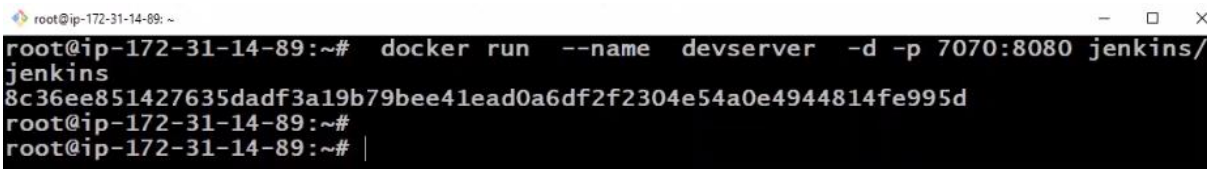
Lets delete all the container

 # docker rm -f $(docker ps -aq)



To start jenkins as a container

 # docker run --name devserver -d -p 7070:8080 Jenkins/Jenkins



to check jenkins is running or not?

Open browser

public_ip:7070

http://18.138.58.3:7070

We need two tomcat containers (qa server and prod server)

 # docker run --name qaserver -d -p 8080:8080 --link devserver:jenkins tomee

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#  docker run --name  qaserver  -d  -p 8080:8080 --link de
vserver:jenkins tomee
16dd63973f2a75cedf4127e20722a053fbaa4792c896c19f173d39bcc471fcc8
root@ip-172-31-14-89:~#
```

to check the tomcat use public_ip but port number will be 8080

http://18.138.58.3:8080

 # docker run --name prodserver -d -p 9090:8080 --link devserver:jenkins tomcat

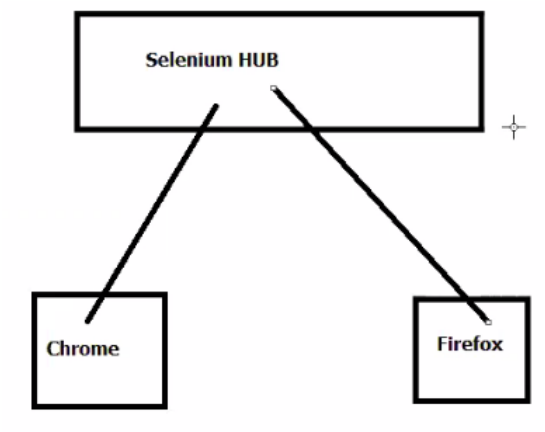
to check the tomcat of prodserver

```
root@ip-172-31-14-89:~#  docker run --name  prodserver  -d  -p 9090:8080 --link
devserver:jenkins tomee
a2ef0f2109c87921e2903ab1731adc11fe94b4878f02ac5a418436e8c351715a
root@ip-172-31-14-89:~#
```

http://18.138.58.3:9090

++++++++++++++++++++++++++++

Creating testing environment using docker



Create selenium hub container, and link it with two node containers.

One node with firefox installed, another node with chrome installed.

Tester should be able to run selenuim automation programs for testing the application on multiple browsers.

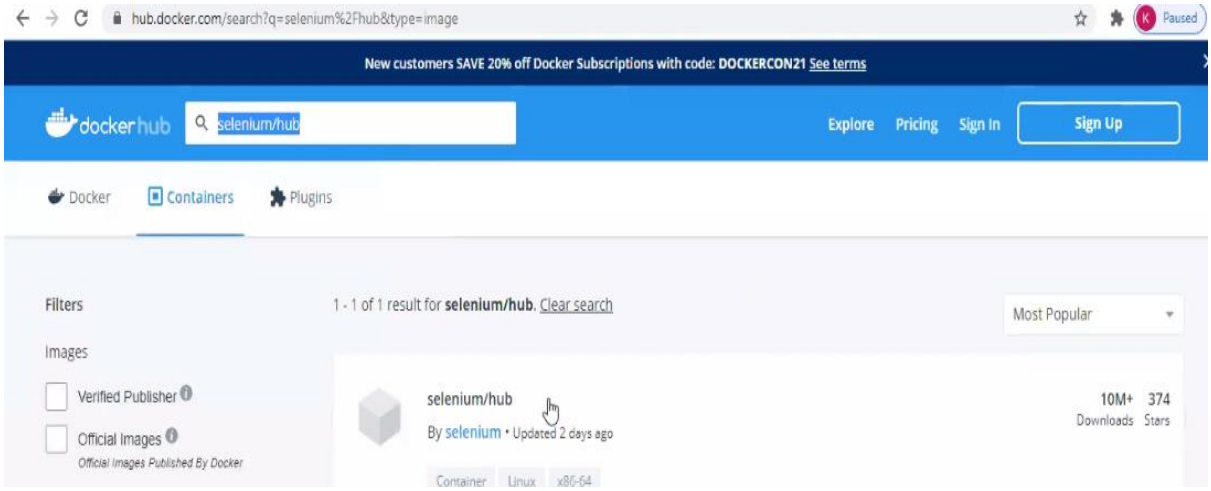
To delete all the running containers

#

In Browser -- open - hub.docker.com

Search for selenium

We have a image - selenium/hub



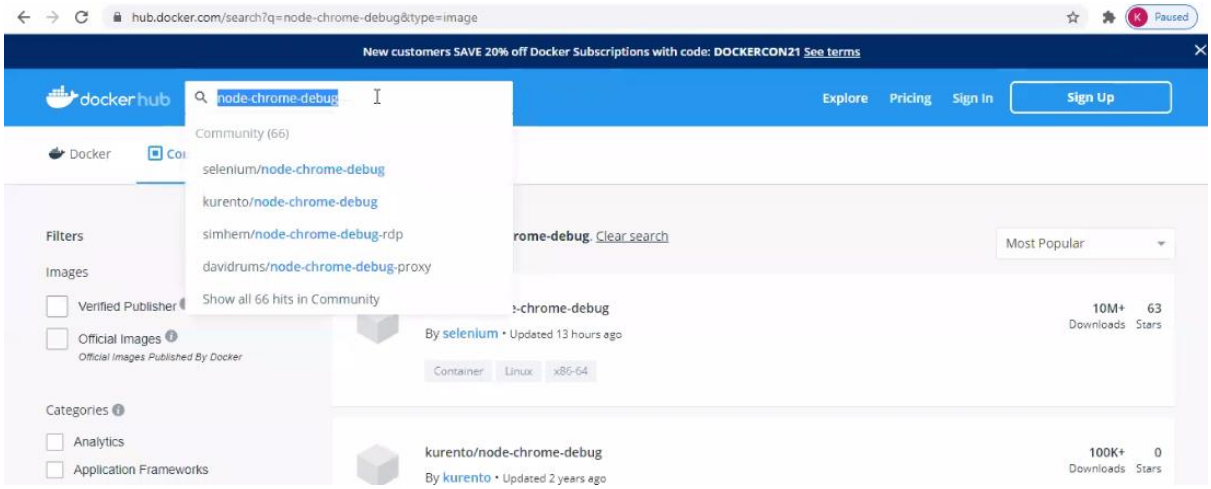
To start selenium/hub as container

 # docker run --name hub -d -p 4444:4444 selenium/hub

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~#  docker run --name  hub  -d -p 4444:4444
Unable to find image 'selenium/hub:latest' locally
latest: Pulling from selenium/hub
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
0219dfafe1a8: Pull complete
60f1c8a7de18: Pull complete
67ef97602b3c: Pull complete
05e72f362e68: Pull complete
aa56724423c5: Pull complete
8393388b6c0b: Pull complete
453c1f5950f6: Pull complete
6de7134fc6dc: Pull complete
56c4316364fd: Pull complete
5cc65f8e59d4: Pull complete
Digest: sha256:ebcbc572f52820546a0e6094e7a0d8c4d9ffc4ec9c5557cb7af6
Status: Downloaded newer image for selenium/hub:latest
0a18e71dc2413bccacc7a8ef6fa0129558afbd15055c69e206c58968f85f9b2a
```

In hub.docker.com

we also have- selenium/node-chrome-debug (It is ubuntu container with chrome)



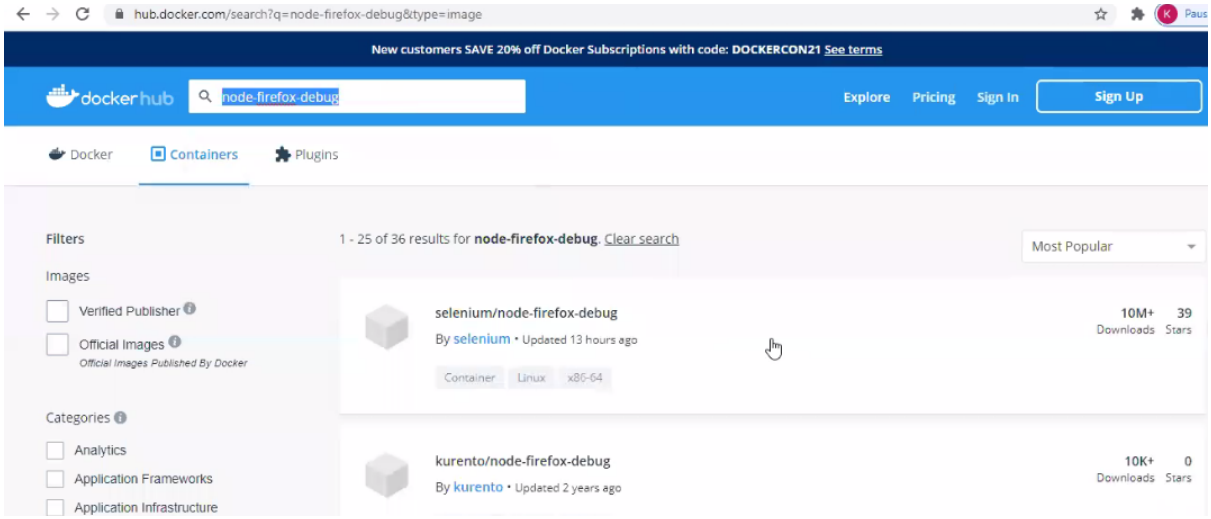
To start it as a container and link to hub (previous container)

 # docker run --name chrome -d -p 5901:5900 --link hub:selenium selenium/node-chrome-debug

```
root@ip-172-31-11-3:~#  docker run --name chrome  -d -p 5901:5900  --link hub:se
lenium   selenium/node-chrome-debug
Unable to find image 'selenium/node-chrome-debug:latest' locally
```

In hub.docker.com

we also have- selenium/node-firefox-debug



To start it as a container and link to hub (It is ubuntu container with firefox)

 # docker run --name firefox -d -p 5902:5900 --link hub:selenium selenium/node-firefox-debug

```
root@ip-172-31-11-3:~# docker run --name firefox -d -p 5902:5900 --link hub:s
elenium selenium/node-firefox-debug
```

To see the list of container

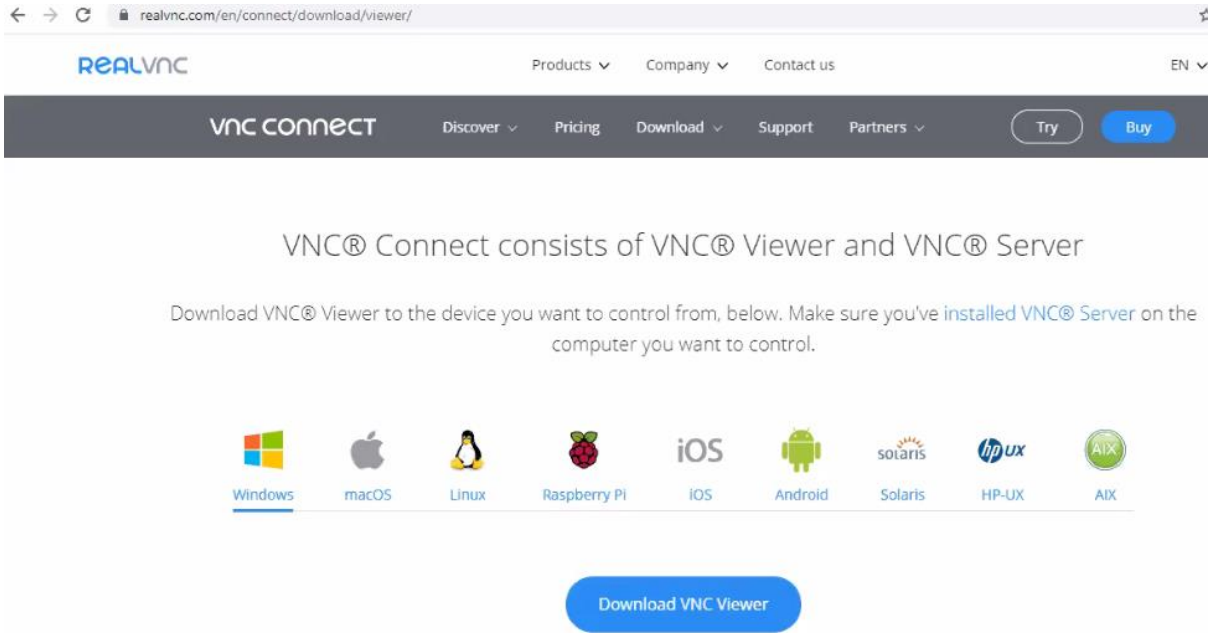
 # docker container ls

```
root@ip-172-31-11-3:~# docker container ls
CONTAINER ID   IMAGE                          COMMAND                  CREATED
          STATUS                  PORTS                                       NAMES
3eda066a9531   selenium/node-firefox-debug    "/opt/bin/entry_poin…"   13 seconds
 ago       Up 12 seconds           0.0.0.0:5902->5900/tcp, :::5902->5900/tcp   firef
ox
21fb177a1467   selenium/node-chrome-debug     "/opt/bin/entry_poin…"   About a mi
nute ago   Up About a minute       0.0.0.0:5901->5900/tcp, :::5901->5900/tcp   chrom
e
233459fae8b3   selenium/hub                   "/opt/bin/entry_poin…"   2 minutes
ago        Up 2 minutes            0.0.0.0:4444->4444/tcp, :::4444->4444/tcp   hub
root@ip-172-31-11-3:~#
```

Note: firefox and chrome containers are GUI containers.

To see the GUI interface to chrome / firefox container for these we connect using VNC viewer.

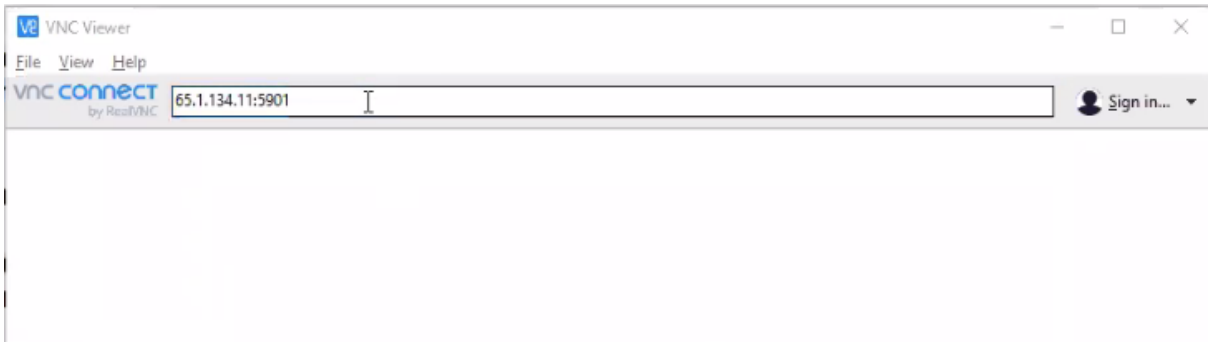
Download and install vnc viewer

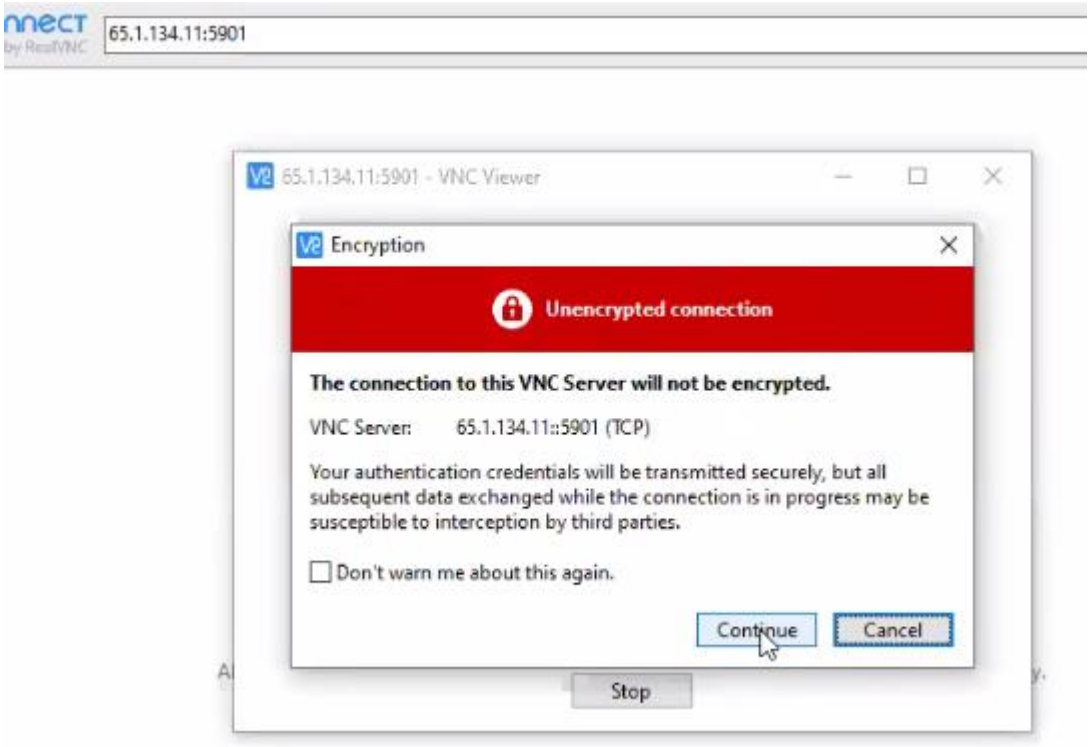


In VNC viewer search bar

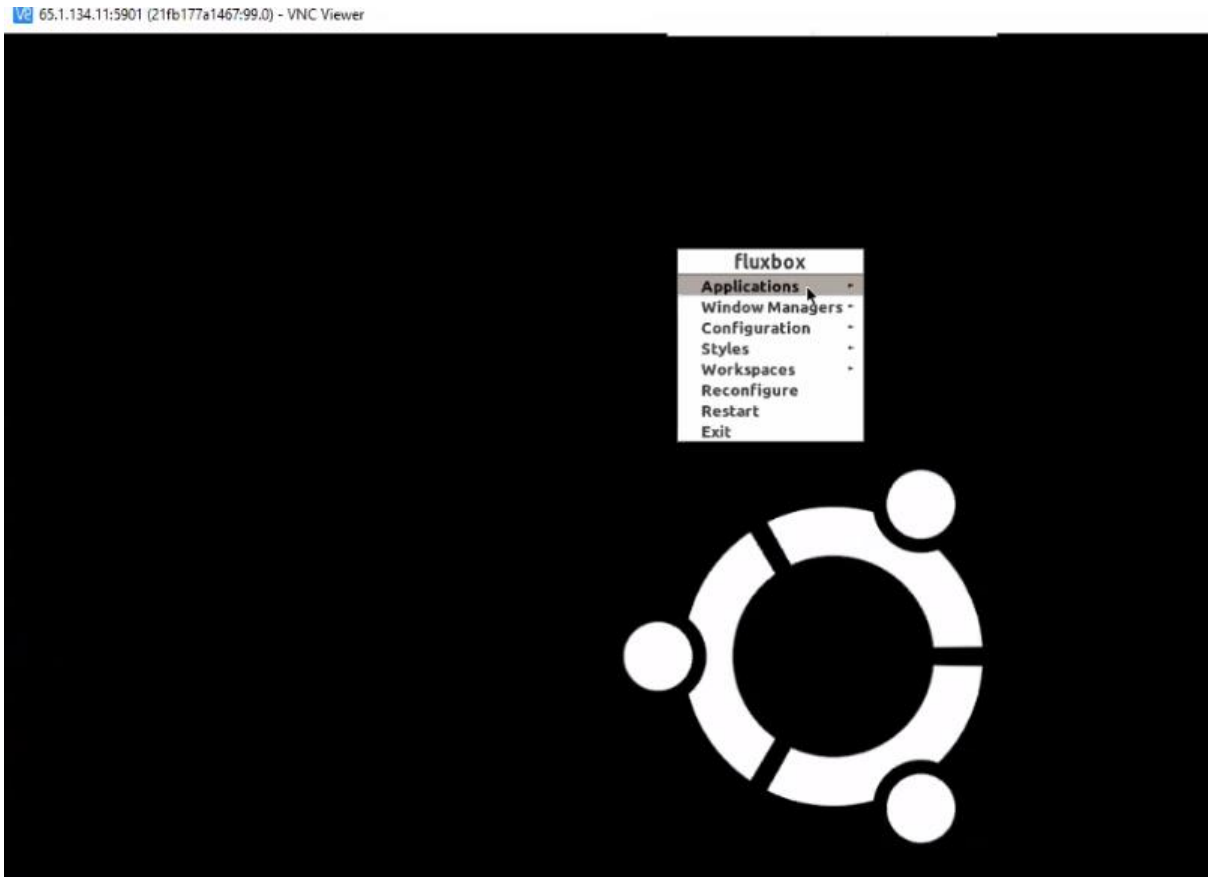
public_ip_dockerhost:5901

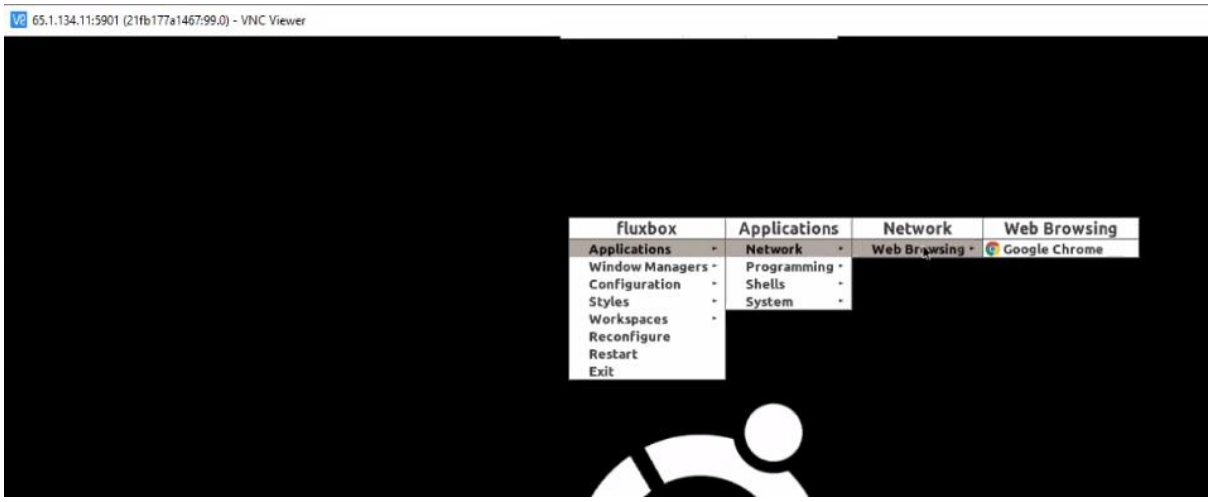
18.136.211.65:5901





Password – secret (which is default)





++

All the commands we learnt till date are adhoc commands.

In the previous use case we have installed two containers (chrome and firefox)

Lets say you need 80 containers?

Do we need to run 80 commands?

Instead of 80 commands, we can use docker compose

++++++++++++++++++++++++

Docker compose

This is a feature of docker using which we can create multicontainer architecture using yaml files. This yaml file contains information about the containers that we want to launch and how they have to be linked with each other.Yaml is a file format. It is not a scripting language.

Yaml will store the data in key value pairs

Lefthand side - Key

Righthand side - Value

Yaml file is space indented.

Sample Yaml file—start with and with …

…

durgasoft: root

Trainers: child

Sunil: Devops

raj: Python

Coordinators: child

lakshmi: Devops

rani: AWS

…

++++++++++++++++++++++

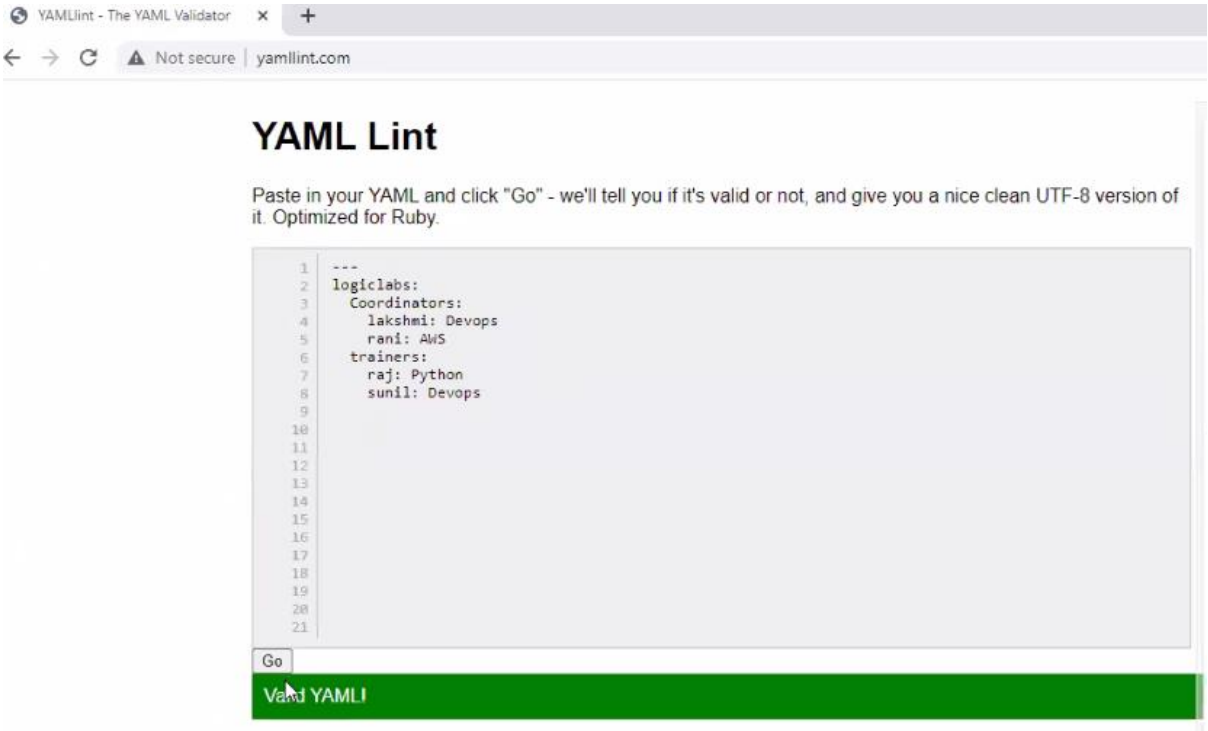
durgasoft -- root element

+++++++++++++++++++++++++

To validate the abvove Yaml file

Open http://www.yamllint.com/

Paste the above code -- Go button



++

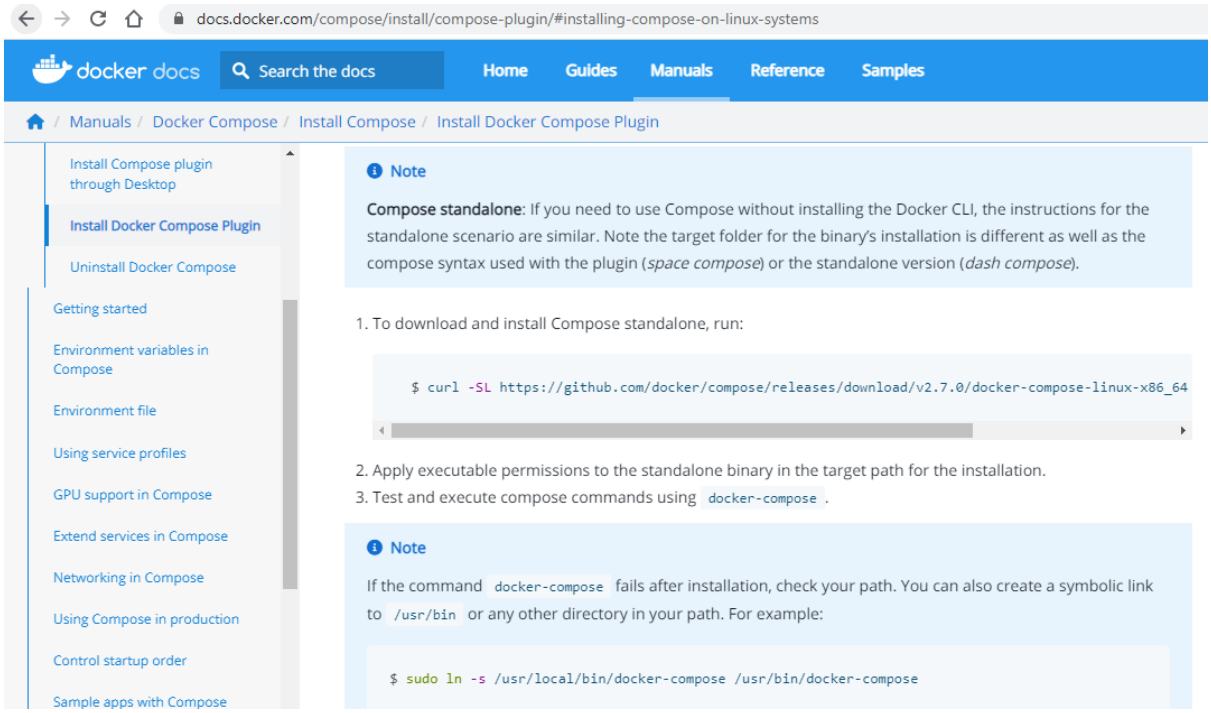
Exclusively install Docker compose on docker

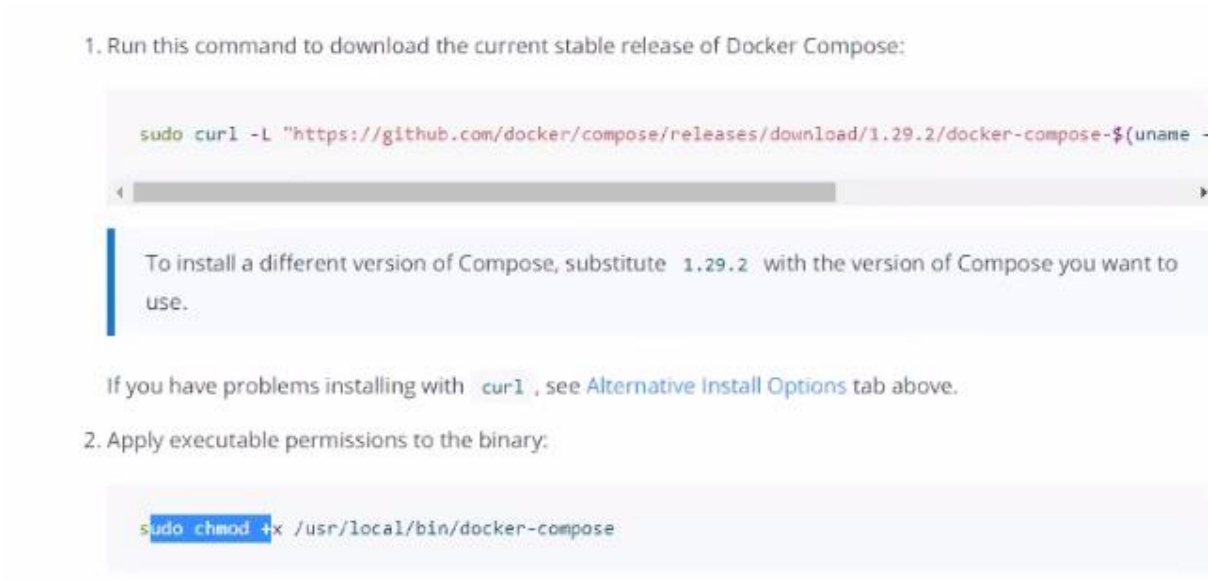
Installing Docker compose

----

1) Open https://docs.docker.com/compose/install/

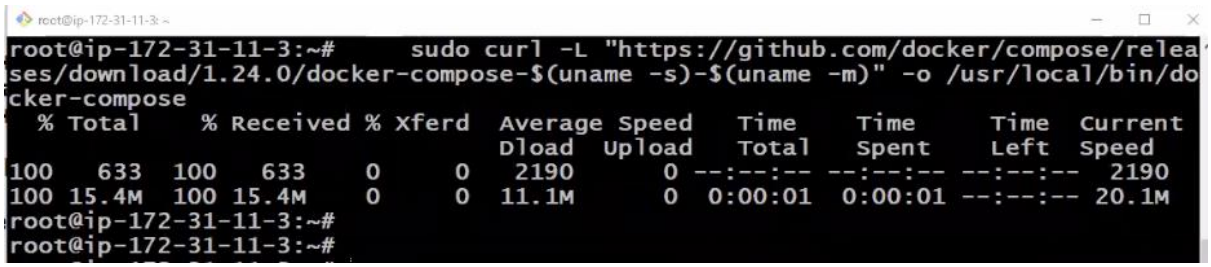
2) Go to linux section



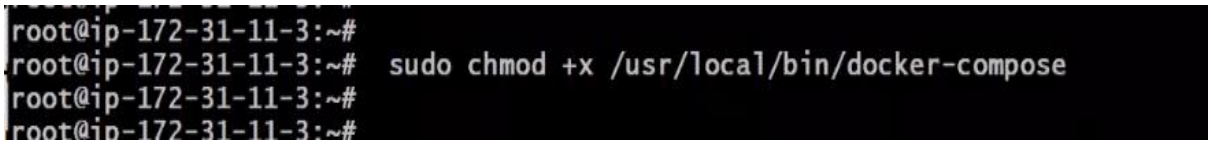


Copy and pase the below two commands

 # sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose



 # sudo chmod +x /usr/local/bin/docker-compose



How to check docker compose is installed or not?

 # Docker-compose --version

++++++++++++++++++++++++++

8th june

++++++++++++++++++++++++++

Installing Docker compose

----

1) Open https://docs.docker.com/compose/install/

2) Go to linux section

Copy and pase the below two commands

 # sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

 # sudo chmod +x /usr/local/bin/docker-compose

How to check docker compose is installed or not?

 # Docker-compose --version

++++++++++++++++++++++++

Create a docker compose file for setting up dev environment.

mysql container is linked with wordpress container.

 # vim docker-compose.yml (Name of the file should be docker-compose.yml)

```
---
version: '3'

services:  root
 mydb: child
 image: mysql:5   image is keyword
 environment:
  MYSQL_ROOT_PASSWORD: sunilsunil
(Container1)

(Container2)
 mysite:
  image: wordpress
 ports:
  - 5050:80
 links:
  - mydb:mysql
...

:wq
```

```
root@ip-172-31-11-3: ~
root@ip-172-31-11-3:~#  vim docker-compose.yml
```

```
root@ip-172-31-11-3: ~
---
version: '3'

services:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: sunilsunil

 mysite:
  image: wordpress
  ports:
   - 5050:80
  links:
   - mydb:mysql
...
~
~
~
~
~
~
~
-- INSERT --
```

Lets remove all the running containers to avoid conflicts

 # docker rm -f $(docker ps -aq)

```
root@ip-172-31-11-3:~#  docker rm -f $(docker ps -aq)
3eda066a9531
21fb177a1467
233459fae8b3
root@ip-172-31-11-3:~#
```

How to start the above services from dockerfile

 # Docker-compose up

```
root@ip-172-31-11-3:~#  docker-compose  up
Creating network "root_default" with the default driver
Pulling mydb (mysql:5)...
5: Pulling from library/mysql
69692152171a: Pulling fs layer
69692152171a: Pull complete
1651b0be3df3: Pull complete
951da7386bc8: Pull complete
0f86c95aa242: Pull complete
37ba2d8bd4fe: Pull complete
6d278bb05e94: Pull complete
497efbd93a3e: Pull complete
a023ae82eef5: Pull complete
e76c35f20ee7: Pull complete
e887524d2ef9: Pull complete
ccb65627e1c3: Pull complete
Digest: sha256:a682e3c78fc5bd941e9db080b4796c75f69a28a8cad65677c23f7a9f18ba21fa
Status: Downloaded newer image for mysql:5
Pulling mysite (wordpress:)...
```

We got lot of logs coming on the screen. to avoid it we use -d option

Ctrl+c

 # docker-compose stop

```
root@ip-172-31-11-3: ~
mydb_1     | 2021-06-08T05:32:02.914391Z 0 [Note] IPv6 is available.
mydb_1     | 2021-06-08T05:32:02.914990Z 0 [Note]   - '::' resolves to '::';
mydb_1     | 2021-06-08T05:32:02.915049Z 0 [Note] Server socket created on IP: '
:'.
mydb_1     | 2021-06-08T05:32:02.916620Z 0 [Note] InnoDB: Buffer pool(s) load com
pleted at 210608  5:32:02
mydb_1     | 2021-06-08T05:32:02.918497Z 0 [Warning] Insecure configuration for -
-pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users.
 Consider choosing a different directory.
mydb_1     | 2021-06-08T05:32:02.928069Z 0 [Note] Event Scheduler: Loaded 0 event
s
mydb_1     | 2021-06-08T05:32:02.928741Z 0 [Note] mysqld: ready for connections.
mydb_1     | Version: '5.7.34'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306
  MySQL Community Server (GPL)
^CGracefully stopping... (press Ctrl+C again to force)
Killing root_mysite_1  ... done
Killing root_mydb_1    ... done
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker-compose stop
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

Will careate compose without detachment by we use -d option

So Remove the container whch just installed

 # docker rm -f $(docker ps -aq)

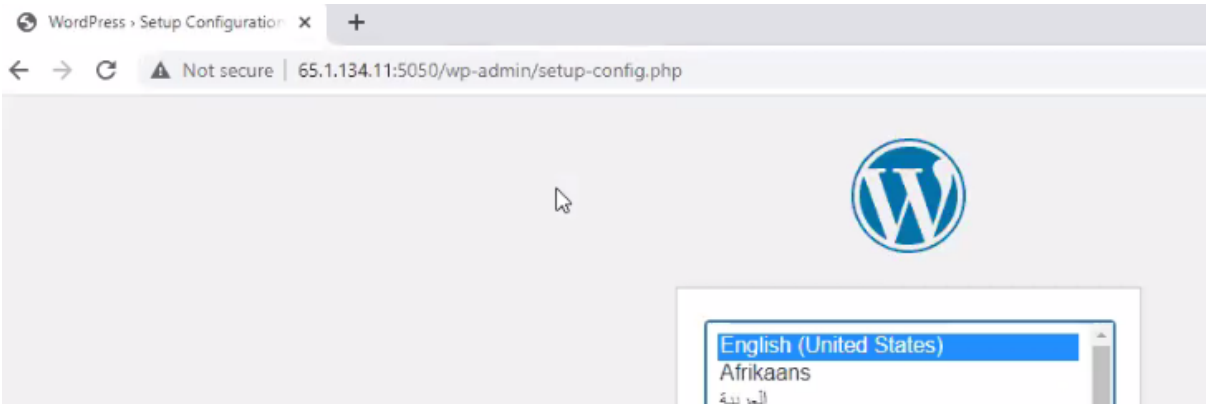
```
root@ip-172-31-11-3:~#  docker rm -f $(docker ps -aq)
27b3caa46cab
86613c15d2b3
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

 # docker-compose up -d

```
root@ip-172-31-11-3:~#  docker-compose up  -d
Creating root_mydb_1 ... done
Creating root_mysite_1 ... done
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

To check wordpress

public_ip:5050



++++++++++++++++++++++++++++

To stop both the containers

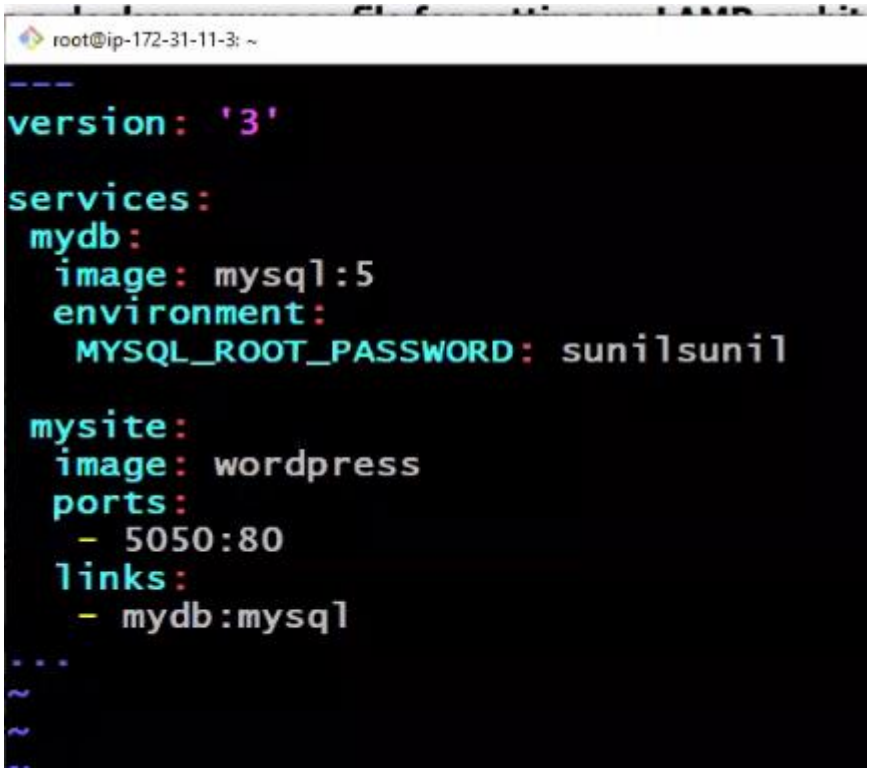
 # docker-compose   stop

```
root@ip-172-31-11-3:~#  docker-compose   stop

Stopping root_mysite_1 ... done
Stopping root_mydb_1   ... done
root@ip-172-31-11-3:~#
```

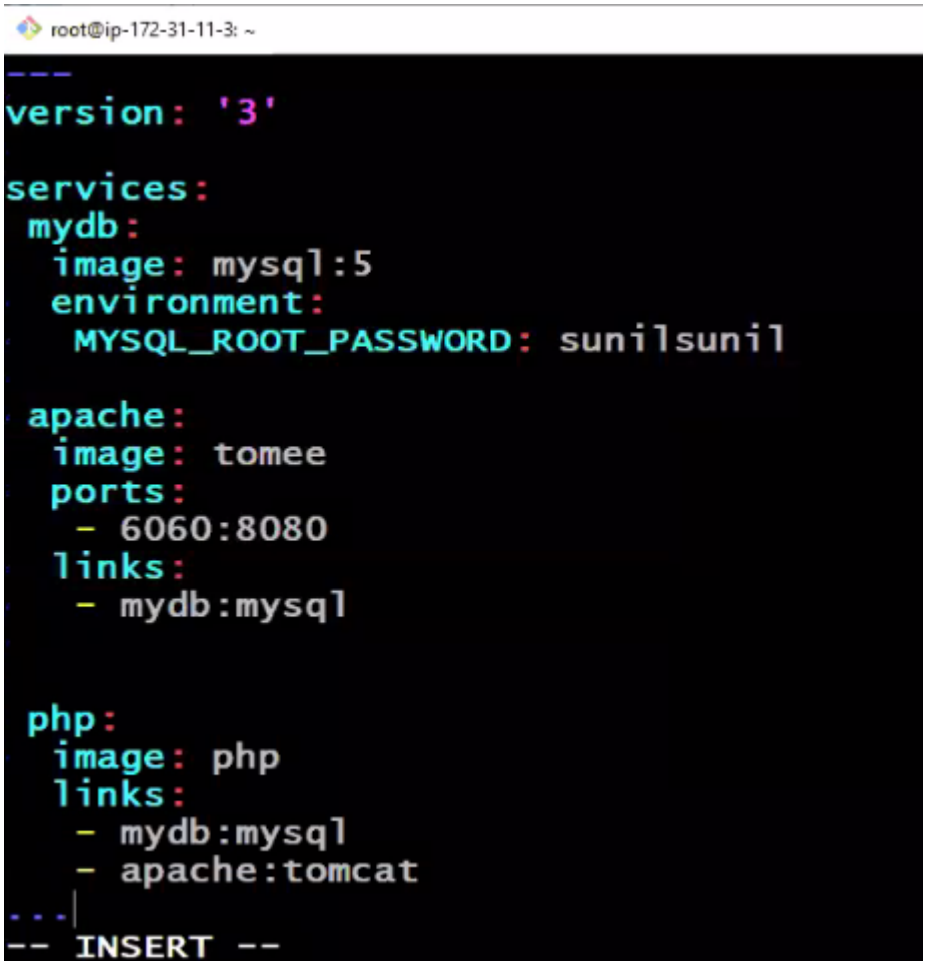
++++++++++++++++++++++++++++++

Create a docker compose file for setting up LAMP architecture

 # vim docker-compose.yml



Delete old version n paste the below yaml file



version: '3'

services:

mydb:containe name

image: mysql:5

environment:

MYSQL_ROOT_PASSWORD: sunilsunil

apache:

image: tomee

ports:

- 6060:8080

links:

- mydb:mysql

php:

image: php

links:

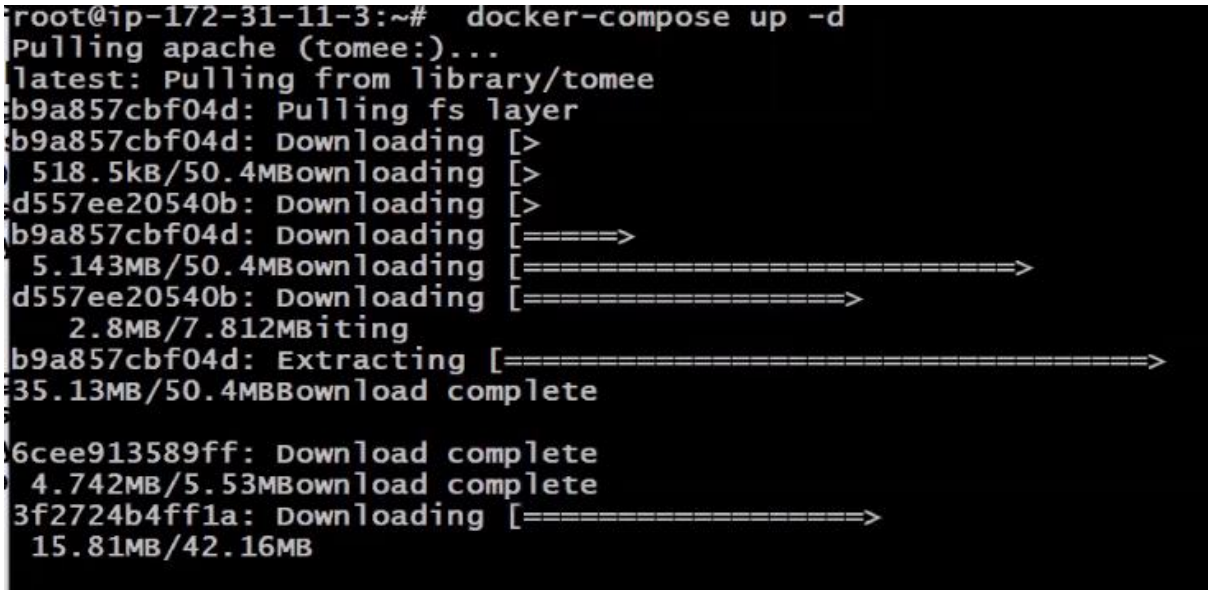
- mydb:mysql

- apache:tomcat

…

:wq

 # docker-compose up -d



To see the list of the containers

 # docker container ls

(Observation - we are unable to see the php container)

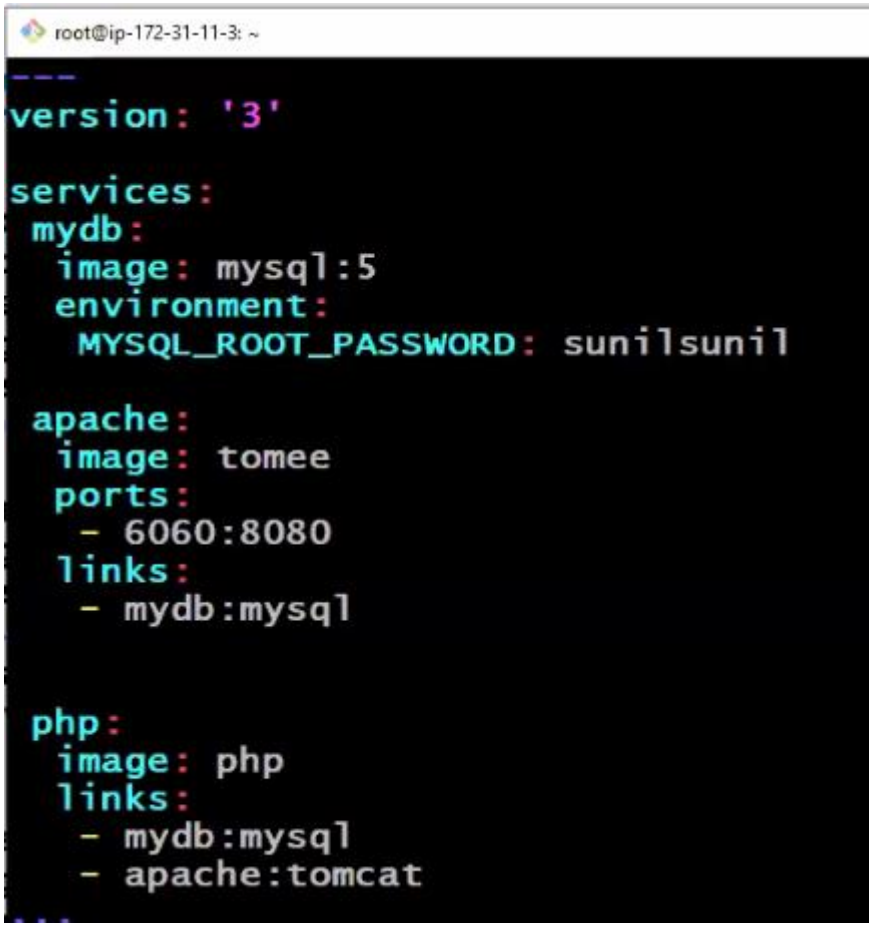
 # docker ps -a

++++++++++++++++++++++++++++++++++

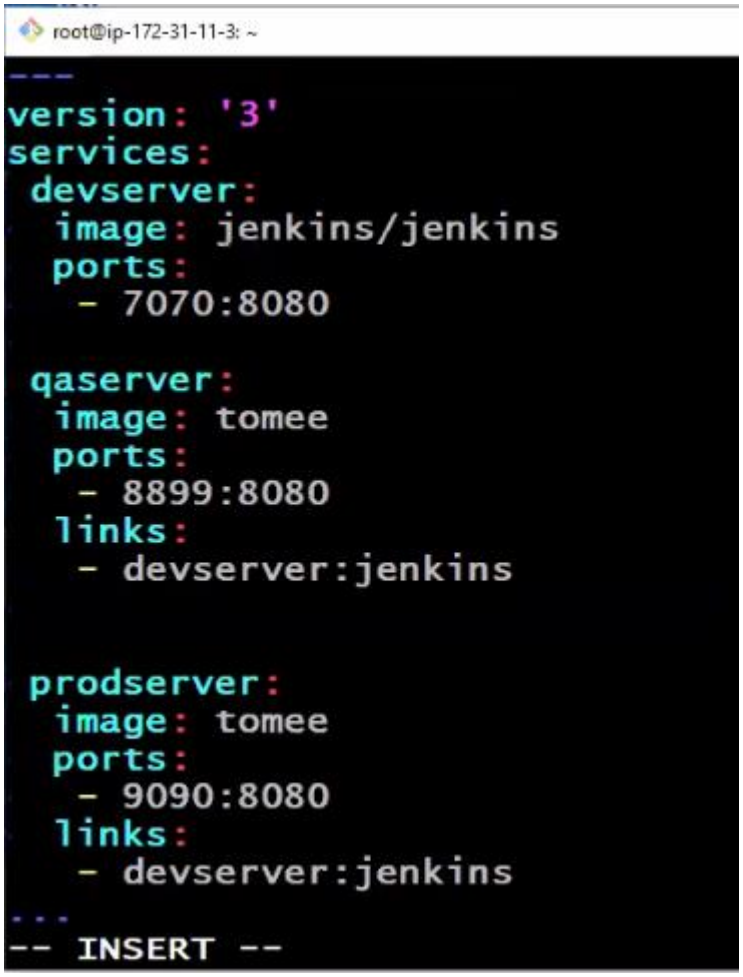
Ex: Docker-compose file for setting up CI-CD Environment.

Jenkins container is linked with two tomcat containers

 # vim docker-compose.yml



Remove old and pate yaml file



Please remove old containers bcz our docke host has low mem

---

version: '3'

services:

devserver:

image: jenkins/jenkins

ports:

- 7070:8080

qaserver:

image: tomee

ports:

- 8899:8080

links:

- devserver:jenkins

prodserver:

image: tomee

ports:

- 9090:8080

links:

- devserver:jenkins

…

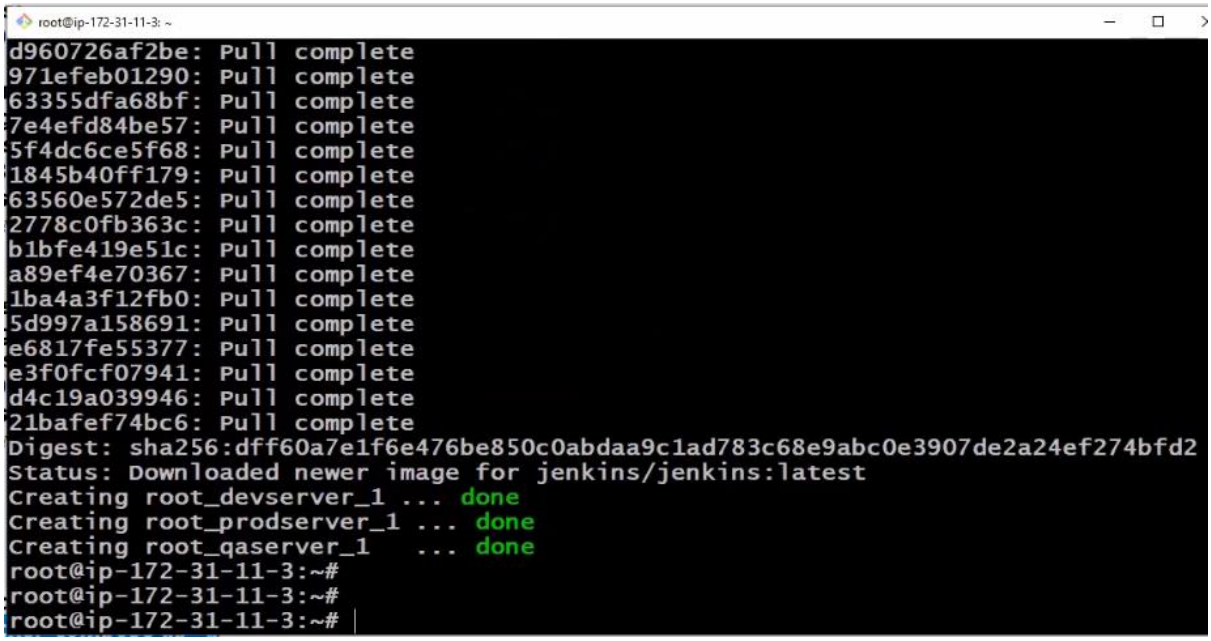
:wq

 # docker rm -f $(docker ps -aq)

 # docker-compose up –d

```
root@ip-172-31-11-3:~#  docker-compose up -d
Pulling devserver (jenkins/jenkins:)...
latest: Pulling from jenkins/jenkins
d960726af2be: Pull complete
971efeb01290: Pull complete
63355dfa68bf: Extracting [==================================>
75.76MB/103.6MBownload complete
 50.85MB/54.53MBwnload complete
1845b40ff179: Download complete
63560e572de5: Download complete
2778c0fb363c: Download complete
b1bfe419e51c: Download complete
 70.77MB/71.38MBwnload complete
1ba4a3f12fb0: Download complete
5d997a158691: Download complete
 1.924kB/1.924kBwnload complete
e3f0fcf07941: Download complete
    380B/380B Waiting
21bafef74bc6: Waiting
```

Jenkins image not available so it is pulling



 # docker container ls

To check

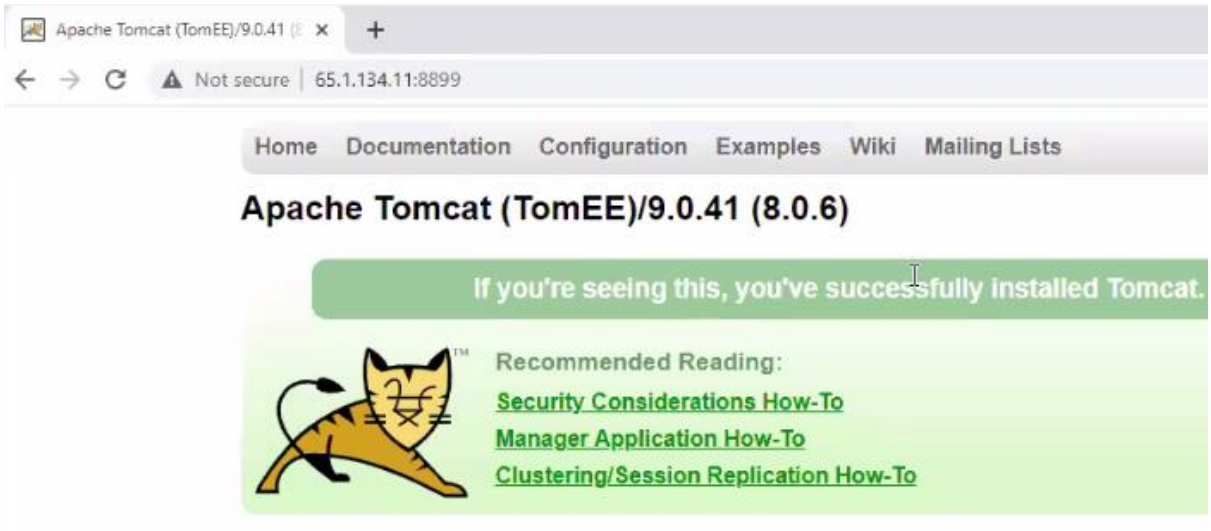
public_ip:7070 (To check jenkins)

public_ip:8899 (Tomcat qa server)

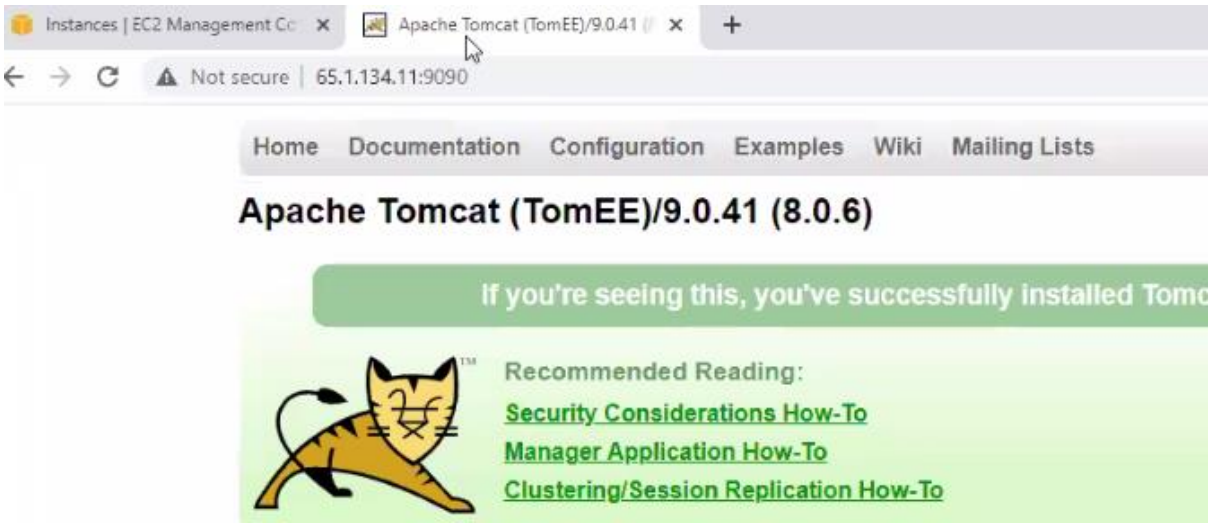
public_ip:9090 (Tomcat prod server)

13.126.58.183:7070

13.126.58.183:8899



13.126.58.183:9090



++++++++++++++++++++

Docker-compose file to set up testing environment.

selenium hub container is linked with two node containers.

 # vim docker-compose.yml

```
---
version: '3'
services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444

 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5901:5900
  links:
   - hub:selenium

 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5902:5900
  links:
   - hub:selenium
...
```

:wq

Lets delete all the running containers

```
# docker rm -f $(docker ps -aq)
# docker-compose up -d
```

 # docker container ls

As it is GUI container,

we can access using VNC viewer

Open VNC viewer

52.77.219.115:5901

password: secret

++

Docker volumes

Docker containers are ephemeral (temporary)

Whereas the data processed by the container should be permanent.

Generally, when a container is deleted all its data will be lost.

To preserve (save) the data, even after deleting the container, we use volumes.

Volumes are of two types

1) Simple docker volumes

2) Docker volume containers (Sharable volume)

Simple docker volumes

These volumes are used only when we want to access the data,

Even after the container is deleted.

But this data cannot be shared with other containers.

usecase

1) Create a directory called /data ,

Start centos as container and mount /data as volume.

Create files in mounted volume in centos container,

exit from the container and delete the container. Check if the files are still available.

Lets create a folder with the name

 # mkdir /data

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  mkdir  /data
root@ip-172-31-11-3:~#
```

 # docker run --name c1 -it -v /data centos (v option is used to attach volume)

-v= attaching volume to data particular container in docker host

Once volume is attached to centos and downloaded centos image(if not available pull the image) the control will enter into cent os as we knw.

```
@6ade908417d4:/
root@ip-172-31-11-3:~#  docker run --name c1   -it -v /data    centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
7a0437f04f83: Pull complete
Digest: sha256:5528e8b1b1719d34604c87e11dcd1c0a20bedf46e83b5632cdeac91b8c04efc1
Status: Downloaded newer image for centos:latest
[root@6ade908417d4 /]#
[root@6ade908417d4 /]#
[root@6ade908417d4 /]#
```

 # ls (Now, we can see the data folder also in the container)

```
[root@6ade908417d4 /]# ls
bin   dev  home  lib64       media  opt   root  sbin  sys  usr
data  etc  lib   lost+found  mnt    proc  run   srv   tmp  var
[root@6ade908417d4 /]#
```

 # cd data

 # touch file1 file2

```
[root@6ade908417d4 /]#  cd data
[root@6ade908417d4 data]#
[root@6ade908417d4 data]#  touch file1   file2
[root@6ade908417d4 data]#
```

 # ls

```
[root@6ade908417d4 data]# ls
file1  file2
[root@6ade908417d4 data]#
```

 # exit (To come out of the container)

```
[root@6ade908417d4 data]# exit
exit
root@ip-172-31-11-3:~#
```

 # docker inspect c1 -- to mount path----------can see data volume under mount

```
root@ip-172-31-11-3: ~
            },
            "Name": "overlay2"
        },
        "Mounts": [
            {
                "Type": "volume",
                "Name": "8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209
f81759508",
                "Source": "/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f651
38d6750495519d3c1665b71209f81759508/_data",
                "Destination": "/data",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
        "Config": {
            "Hostname": "6ade908417d4",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
            "AttachStdout": true,
            "AttachStderr": true,
```

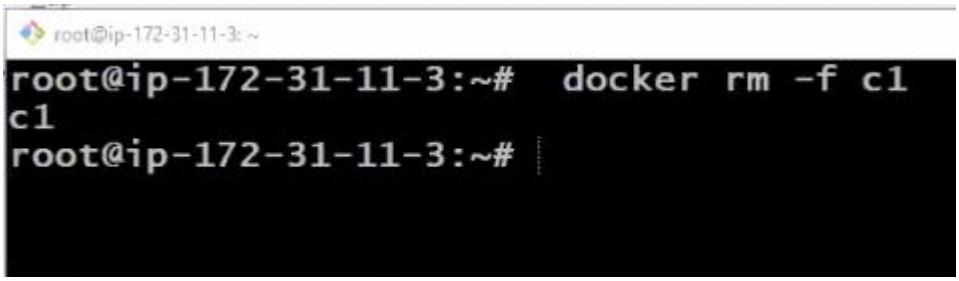
We can see under mounts "data" folder it located in the host machine.

Copy the path

/var/lib/docker/volumes/c5c85f87fdc3b46b57bb15f2473786fe7d49250227d1e9dc537bc594db001fc6/_data

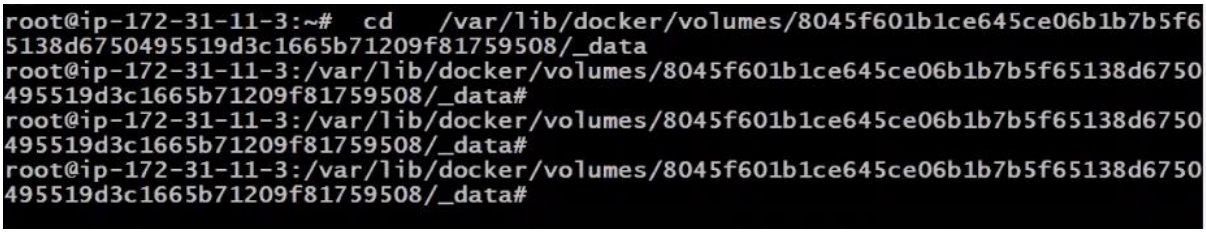
Now, lets delete the container

 # docker rm -f c1



After deleting the container, lets go to the location of the data folder

 # cd /var/lib/docker/volumes/d867766f70722eaf8cba651bc1d64c60e9f49c5b1f1ebb9e781260f777f3c7e8/_data



 # ls (we can see file1 file2)

(Observe, the container is deleted but still the data is persistent)

The moment the volume is created then mount path is generated. Whenever we creates files that files are available in mount path it is not volume

So we can access mount path. to see which files were created.

/var/lib/docker/volumes/d867766f70722eaf8cba651bc1d64c60e9f49c5b1f1ebb9e781260f777f3c7e8/_data

F1 and f2 saved in above mount path

++++++++++++

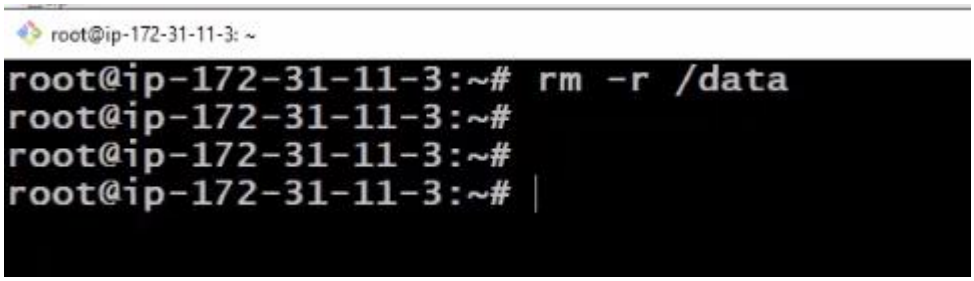
docker volume containers

These are also known as reusable volume.

The volume used by one container can be shared with other containers.

Even if all the containers are deleted, data will still be available on the docker host.

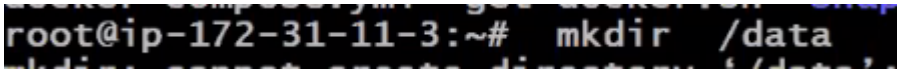
Ex:



 # sudo su -

Lets create a directory /data---- creates in docker host

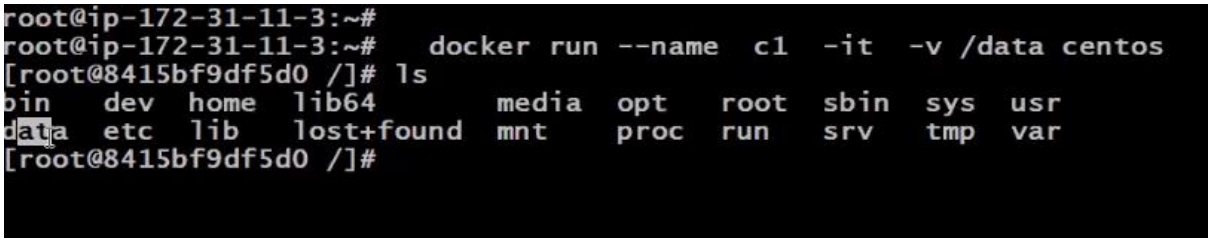
 # mkdir /data



Lets Start centos as container

 # docker run --name c1 -it -v /data centos

 # ls (we can see the list of files and dir in centos)



 # cd data

 # ls (currently we have no files)

Lets create some files

 # touch file1 file2 (These two files are available in c1 container)

```
[root@8415bf9df5d0 /]#  cd data
[root@8415bf9df5d0 data]#
[root@8415bf9df5d0 data]# ls
[root@8415bf9df5d0 data]#  touch file1  file2
[root@8415bf9df5d0 data]#
[root@8415bf9df5d0 data]#
```

Comeout of the container without exit(container stopped)

 # Ctrl +p Ctrl +q (container will still runs in background)

```
[root@8415bf9df5d0 /]#  cd data
[root@8415bf9df5d0 data]#
[root@8415bf9df5d0 data]# ls
[root@8415bf9df5d0 data]#  touch file1  file2
[root@8415bf9df5d0 data]#
[root@8415bf9df5d0 data]# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

Lets Start another centos as container (c2 container should use the same volume as c1)

docker run --name c2 -it *--volumes-from c1* centos

--volumes-from c1 = using volume from c1 conatiner inside c2

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#   docker run --name  c2  -it  --volumes-from c1  centos
[root@99279837edbd /]#
[root@99279837edbd /]#
```

 # cd data

```
[root@99279837edbd /]#
[root@99279837edbd /]# cd data
[root@99279837edbd data]#
[root@99279837edbd data]#
```

 # ls (we can see the files created by c1)

```
[root@99279837edbd data]#
[root@99279837edbd data]# ls
file1  file2
[root@99279837edbd data]#
```

Lets create some more files

 # touch file3 file4

 # ls (we see 4 files)

```
[root@99279837edbd data]#  touch file3  file4
[root@99279837edbd data]#
[root@99279837edbd data]# ls
file1  file2  file3  file4
[root@99279837edbd data]#
```

Comeout of the container without exit

 # Ctrl +p Ctrl +q (container will still runs in background)

```
[root@99279837edbd data]# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

Lets Start another centos as container

 # docker run --name c3 -it --volumes-from c2 centos

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#   docker run --name  c3  -it  --volumes-from c2 centos
[root@0dcf92ac9c1a /]#
[root@0dcf92ac9c1a /]#
[root@0dcf92ac9c1a /]# cd data
[root@0dcf92ac9c1a data]#
[root@0dcf92ac9c1a data]#
[root@0dcf92ac9c1a data]# ls
file1  file2  file3  file4
[root@0dcf92ac9c1a data]#
```

 # cd data

 # ls (we can see 4 files)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#   docker run --name  c3  -it  --volumes-from c2 centos
[root@0dcf92ac9c1a /]#
[root@0dcf92ac9c1a /]#
[root@0dcf92ac9c1a /]# cd data
[root@0dcf92ac9c1a data]#
[root@0dcf92ac9c1a data]#
[root@0dcf92ac9c1a data]# ls
file1  file2  file3  file4
[root@0dcf92ac9c1a data]#
```

 # touch file5 file6

 # ls

```
[root@0dcf92ac9c1a data]# touch  file5   file6
[root@0dcf92ac9c1a data]#
[root@0dcf92ac9c1a data]# ls
file1  file2  file3  file4  file5  file6
[root@0dcf92ac9c1a data]#
```

```
[root@0dcf92ac9c1a data]# ls
file1  file2  file3  file4  file5  file6
```

Comeout of the container without exit

 # Ctrl +p Ctrl +q (container will still runs in background)

```
[root@0dcf92ac9c1a data]# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

Now, lets connect to any container which is running in the background

docker attach c1

 # ls (you can see all the files)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker attach  c1
[root@8415bf9df5d0 data]#
[root@8415bf9df5d0 data]# ls
file1  file2  file3  file4  file5  file6
[root@8415bf9df5d0 data]#
```

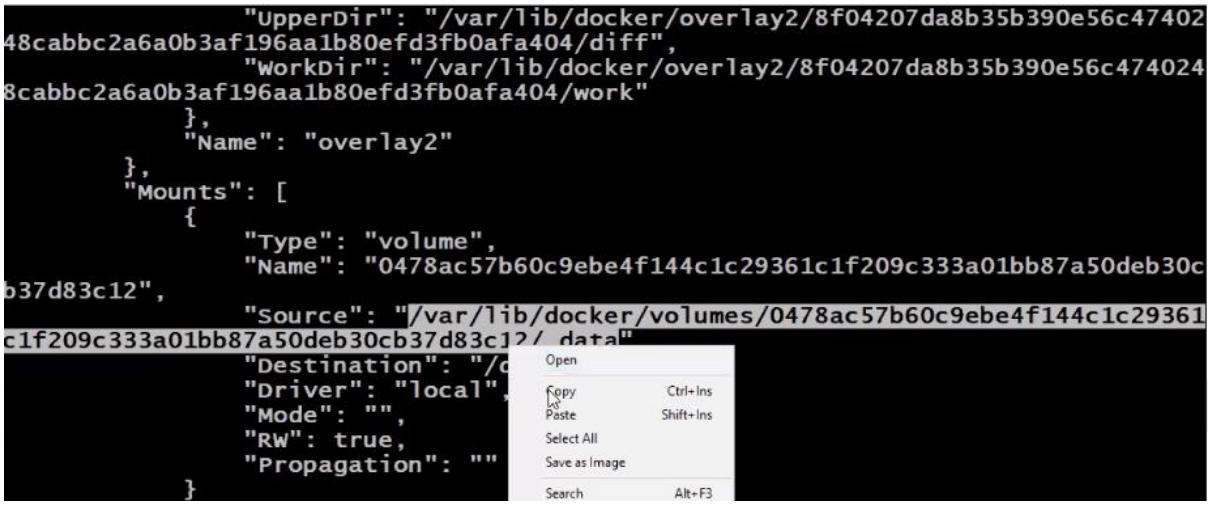
 # exit

Identify the mount location

$ docker inspect c1

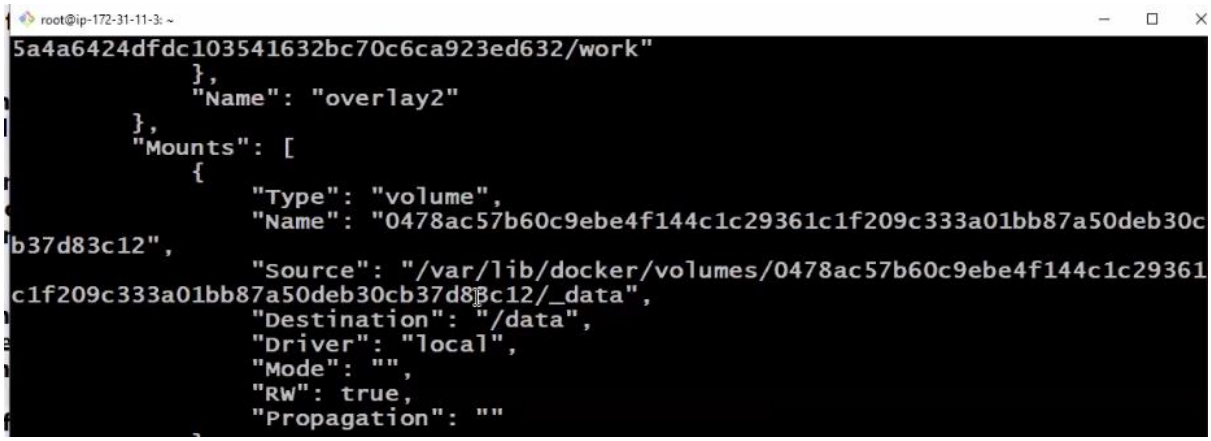
(Search for the mount section)

Take a note of the source path



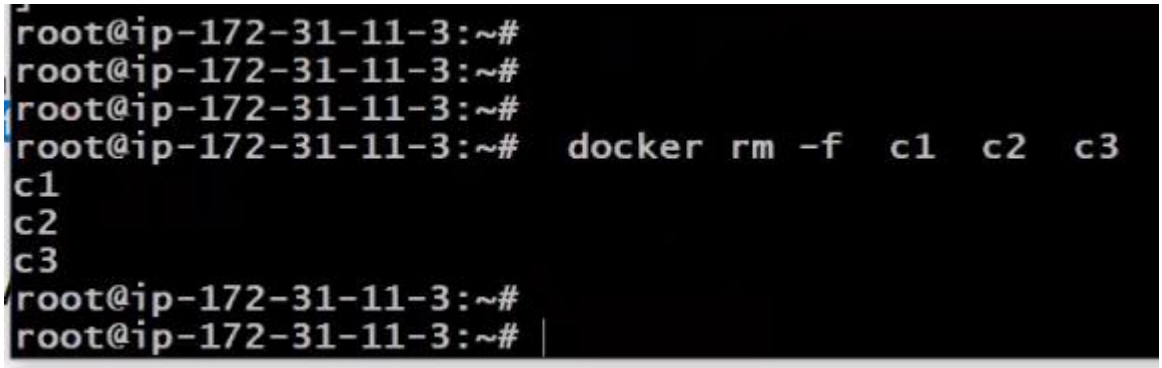
/var/lib/docker/volumes/e22a9b39372615727b964151b6c8108d6c02b13114a3fcce255df0cee7609e15/_data

Docker inspect c2—same path



Let's remove all the containers

 # docker rm -f c1 c2 c3



Lets go to the source path

 # cd
/var/lib/docker/volumes/e22a9b39372615727b964151b6c8108d6c02b13114a3fcce255df0cee7609e15/_data

```
root@ip-172-31-11-3:~#  cd  /var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c2936
1c1f209c333a01bb87a50deb30cb37d83c12/_data
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c
333a01bb87a50deb30cb37d83c12/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c
333a01bb87a50deb30cb37d83c12/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c
333a01bb87a50deb30cb37d83c12/_data#
```

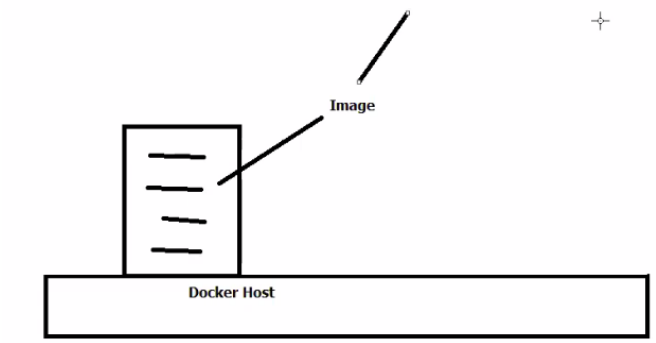
 # ls (we can see all the files)

```
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c
333a01bb87a50deb30cb37d83c12/_data# ls
file1  file2  file3  file4  file5  file6
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c
333a01bb87a50deb30cb37d83c12/_data#
```

++

9th June

Creating customized docker images



Whenever docker container is deleted,

all the soft wares,volumes etc whatever in that, that we have installed within the container will also be deleted.

If we can save the container as an image, then we can preserve the softwares. And future or later we will use whener we/developers need.

(Here we convert the container into image.) and saved in docker hub

This creation of customized docker images can be done in two ways.

1) using docker commit command

2) using docker file

Using docker commit

 # docker run --name c1 -it ubuntu

```
root@ip-172-31-11-3:~#  docker run  --name c1 -it  ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
345e3491a907: Pull complete
57671312ef6f: Pull complete
5e9250ddb7d0: Pull complete
Digest: sha256:adf73ca014822ad8238623d388cedf4d5346aa72c270c5acc01431cc93e18e2d
Status: Downloaded newer image for ubuntu:latest
root@24adc066a694:/#
root@24adc066a694:/#
```

Update apt repository—inside container

 # apt-get update

```
root@24adc066a694:/#  apt-get update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
0% [2 InRelease 69.2 kB/114 kB 61%]
```

 # apt-get install git

```
root@24adc066a694:/#  apt-get install git
Reading package lists... 7%
```

TO check the git

 # git --version

```
root@24adc066a694:/#  git  --version
git version 2.25.1
root@24adc066a694:/#
```

 # exit—to come out of container and back to docker host

```
root@24adc066a694:/# exit
exit
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

TO save the container as image (snapshot)

 # docker commit c1 myubuntu

Myubuntu= name of the image

c1 = container name

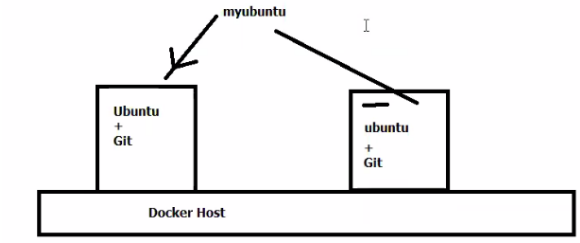
To see the list of images

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker commit   c1   myubuntu
sha256:dd3827ec8fcac3b56c637ff25e007973142796d5f2678ce51885cdda1b12625d
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

 # docker images (you can see the image which you have created)

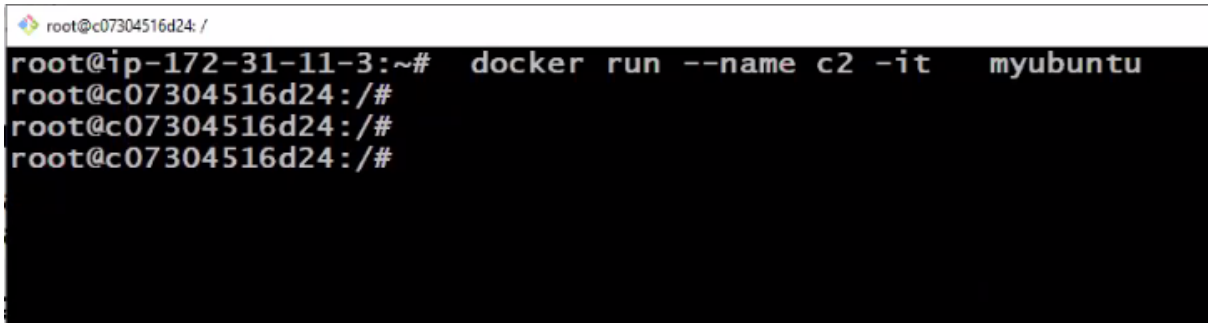
```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
myubuntu                      latest    dd3827ec8fca   6 seconds ago    203MB
selenium/node-firefox-debug   latest    5a02336ff62a   39 hours ago     988MB
selenium/node-chrome-debug    latest    2df3498bcff1   39 hours ago     1.07GB
selenium/hub                  latest    705be32777f0   39 hours ago     283MB
php                           latest    7a9e4bdd6171   4 days ago       423MB
wordpress                     latest    c2dd1984ad5b   5 days ago       551MB
jenkins/jenkins               latest    8959ce44055e   7 days ago       570MB
mysql                         5         2c9028880e58   3 weeks ago      447MB
ubuntu                        latest    7e0aa2d69a15   6 weeks ago      72.7MB
tomee                         latest    bc73f72b54d0   4 months ago     340MB
centos                        latest    300e315adb2f   6 months ago     209MB
root@ip-172-31-11-3:~#
```

++++++++++++++++++++++

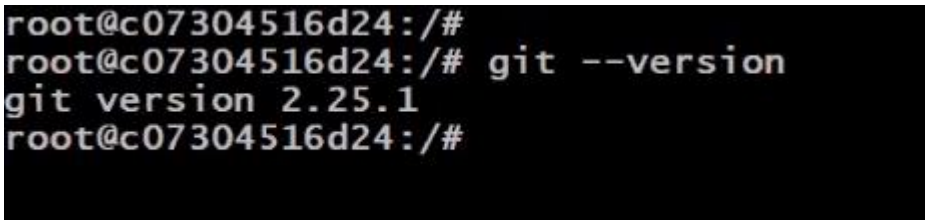


Now lets run the image which we have created

 # docker run --name c2 -it myubuntu



 # git --version (git is pre installed)



Note:

Exit(stop) the container then create image for infra Never do take image from running container

1.stop

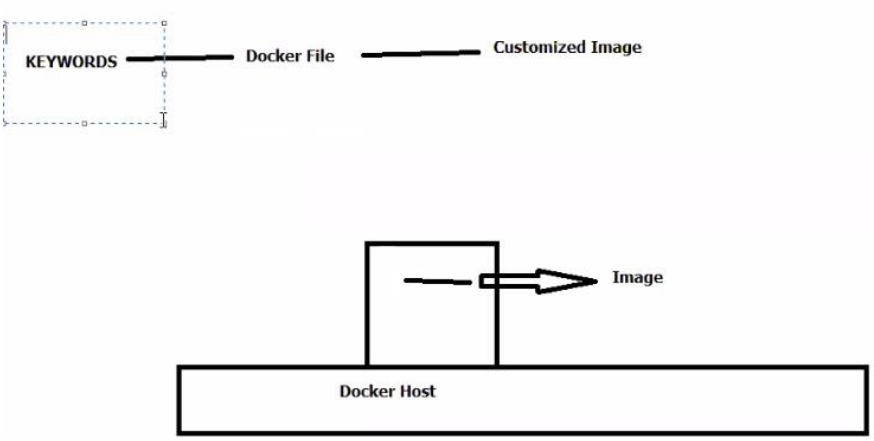
2.create image

3.creating images from running conatiners not recommended.

++

14th June

- **Understanding Keywords of Docker file**
- **Working with Docker file**
- **Version controlling on Docker file**
- **Cache Busting**
- **Working with Registry**
    - **Public Registry**
    - **Private Registry**



Using docker file

This is a simple text file, which uses predefined keywords for creating docker file for customized docker images.

Key words used in docker file (case sensitive)

1) FROM -- used to specify the base image from which the docker file has to be created.

2) MAINTAINER -- This represents name of the organization or the

author who created this docker file.

3) CMD -- This is used to specify the initial command that should be executed when the container starts.

4) ENTRYPOINT - used to specify the default process that should be executed when container starts.

It can also be used for accepting arguments from the CMD instruction.

5) RUN -- Used for running linux commands within the container. It is generally helpful for installing the software in the container.

6) USER -- used to specify the default user who should login into the container.

7) WORKDIR --

Used to specify default working directory in the container

8) COPY -- Copying the files from the host machine to the container.

9) ADD -- Used for copying files from host to container, it can also be used for downloading files from remote servers.

10) ENV -- used for specifying the environment variables that should be passed to the container.

EXPOSE -- Used to specify the internal port of the container

VOLUME -- used to specify the default volume that should be attached to the container.

LABEL -- used for giving label to the container

STOPSIGNAL -- Used to specify the key sequences that have to be passed in order to stop the container.

++++++++++++++++++++

++++++++++++++++++++++++++++++++

Create a dockerfile by taking nginx as the base image

and specify the maintainer as logiclabs. Construct an image from the dockerfile.

Creating customized docker images by using docker file.

$ sudo su –

```
ubuntu@ip-172-31-11-3:~$ sudo su -
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

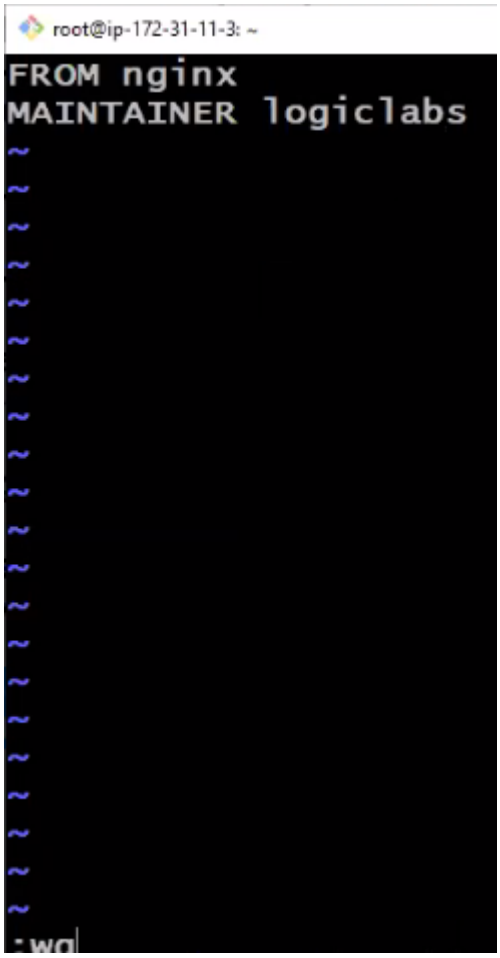
 # vim dockerfile (dockerfile= name of predefined file)

```
root@ip-172-31-11-3:~#  vim dockerfile
```

FROM nginx

MAINTAINER logiclabs

:wq



TO build an image from the dockerfile

 # docker build -t mynginx .

-t= tagging to image; mynginx=name of image

(t stands for tag,

. stands for current working dir

mynginx is the new image name)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker build -t  mynginx .
Sending build context to Docker daemon  32.77kB
Step 1/2 : FROM nginx
latest: Pulling from library/nginx
69692152171a: Already exists
30afc0b18f67: Pull complete
596b1d696923: Pull complete
febe5bd23e98: Pull complete
8283eee92e2f: Pull complete
351ad75a6cfa: Pull complete
Digest: sha256:6d75c99af15565a301e48297fa2d121e15d80ad526f8369c526324f0f7ccb750
Status: Downloaded newer image for nginx:latest
 ---> d1a364dc548d
Step 2/2 : MAINTAINER logiclabs
 ---> Running in 076a45698a37
Removing intermediate container 076a45698a37
 ---> d7681be4fec1
Successfully built d7681be4fec1
Successfully tagged mynginx:latest
root@ip-172-31-11-3:~#
```

TO see the image

 # docker images

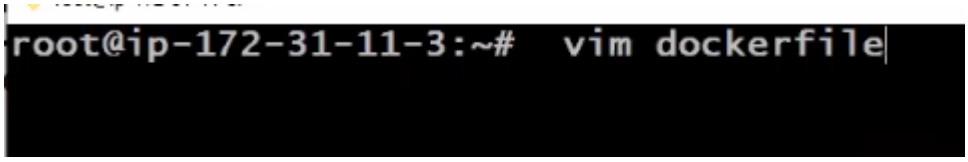
```
root@ip-172-31-11-3:~# docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
mynginx                       latest    d7681be4fec1   7 seconds ago    133MB
myubuntu                      latest    dd3827ec8fca   4 days ago       203MB
selenium/node-firefox-debug   latest    5a02336ff62a   6 days ago       988MB
selenium/node-chrome-debug    latest    2df3498bcff1   6 days ago       1.07GB
selenium/hub                  latest    705be32777f0   6 days ago       283MB
php                           latest    7a9e4bdd6171   9 days ago       423MB
wordpress                     latest    c2dd1984ad5b   10 days ago      551MB
jenkins/jenkins               latest    8959ce44055e   12 days ago      570MB
nginx                         latest    d1a364dc548d   2 weeks ago      133MB
mysql                         5         2c9028880e58   4 weeks ago      447MB
ubuntu                        latest    7e0aa2d69a15   7 weeks ago      72.7MB
tomee                         latest    bc73f72b54d0   4 months ago     340MB
centos                        latest    300e315adb2f   6 months ago     209MB
root@ip-172-31-11-3:~#
```

++

Whenever i start my container, i want a program to get executed automatically.

3) CMD -- This is used to specify the initial command that should be executed when the container starts

 # vim dockerfile

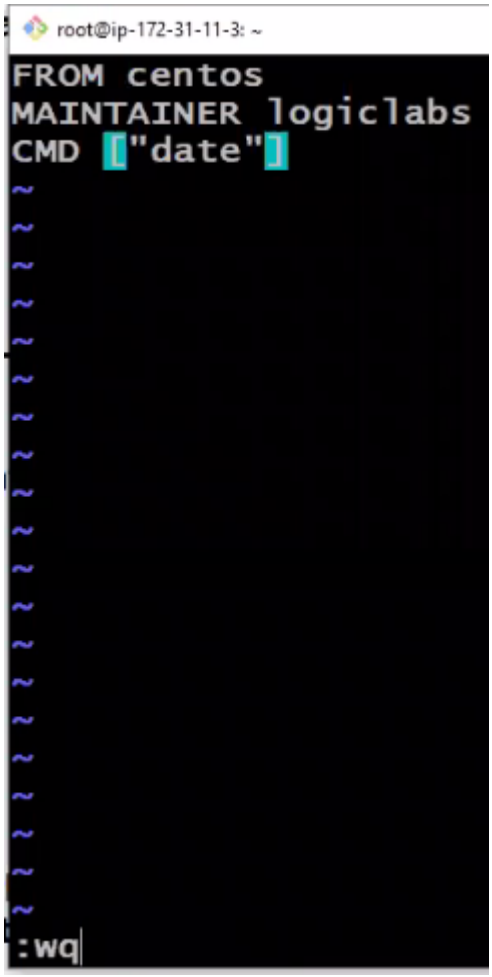


FROM centos

MAINTAINER logiclabs

CMD ["date"]

:wq



TO build an image from the dockerfile

 # docker build -t mycentos .

```
root@ip-172-31-11-3:~#  docker build -t  mycentos  .
Sending build context to Docker daemon  33.79kB
Step 1/3 : FROM centos
 ---> 300e315adb2f
Step 2/3 : MAINTAINER logiclabs
 ---> Running in de8c0e0fe856
Removing intermediate container de8c0e0fe856
 ---> 4016c0ea756e
Step 3/3 : CMD ["date"]
 ---> Running in 40df0abbcb60
Removing intermediate container 40df0abbcb60
 ---> 5b439db9689e
Successfully built 5b439db9689e
Successfully tagged mycentos:latest
root@ip-172-31-11-3:~#
```

TO see the image

 # docker images

```
root@ip-172-31-11-3: ~
 ---> 4016c0ea756e
Step 3/3 : CMD ["date"]
 ---> Running in 40df0abbcb60
Removing intermediate container 40df0abbcb60
 ---> 5b439db9689e
Successfully built 5b439db9689e
Successfully tagged mycentos:latest
root@ip-172-31-11-3:~# docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
mycentos                      latest    5b439db9689e   19 seconds ago   209MB
mynginx                       latest    d7681be4fec1   5 minutes ago    133MB
myubuntu                      latest    dd3827ec8fca   5 days ago       203MB
selenium/node-firefox-debug   latest    5a02336ff62a   6 days ago       988MB
selenium/node-chrome-debug    latest    2df3498bcff1   6 days ago       1.07GB
selenium/hub                  latest    705be32777f0   6 days ago       283MB
php                           latest    7a9e4bdd6171   9 days ago       423MB
wordpress                     latest    c2dd1984ad5b   10 days ago      551MB
jenkins/jenkins               latest    8959ce44055e   12 days ago      570MB
nginx                         latest    d1a364dc548d   2 weeks ago      133MB
mysql                         5         2c9028880e58   4 weeks ago      447MB
ubuntu                        latest    7e0aa2d69a15   7 weeks ago      72.7MB
tomee                         latest    bc73f72b54d0   4 months ago     340MB
centos                        latest    300e315adb2f   6 months ago     209MB
root@ip-172-31-11-3:~#
```

Running conainer from the image

 # docker run -it mycentos

```
root@ip-172-31-11-3: ~
root@ip-172-31-11-3:~#  docker run -it  mycentos
Mon Jun 14 05:23:56 UTC 2021
root@ip-172-31-11-3:~#
```

Date printed

++++++++++++++++++++++++++

In one docker file, we can have one CMD instruction.

If we give two CMD instruction, it executes the latest one

Lets try

 # vim dockerfile

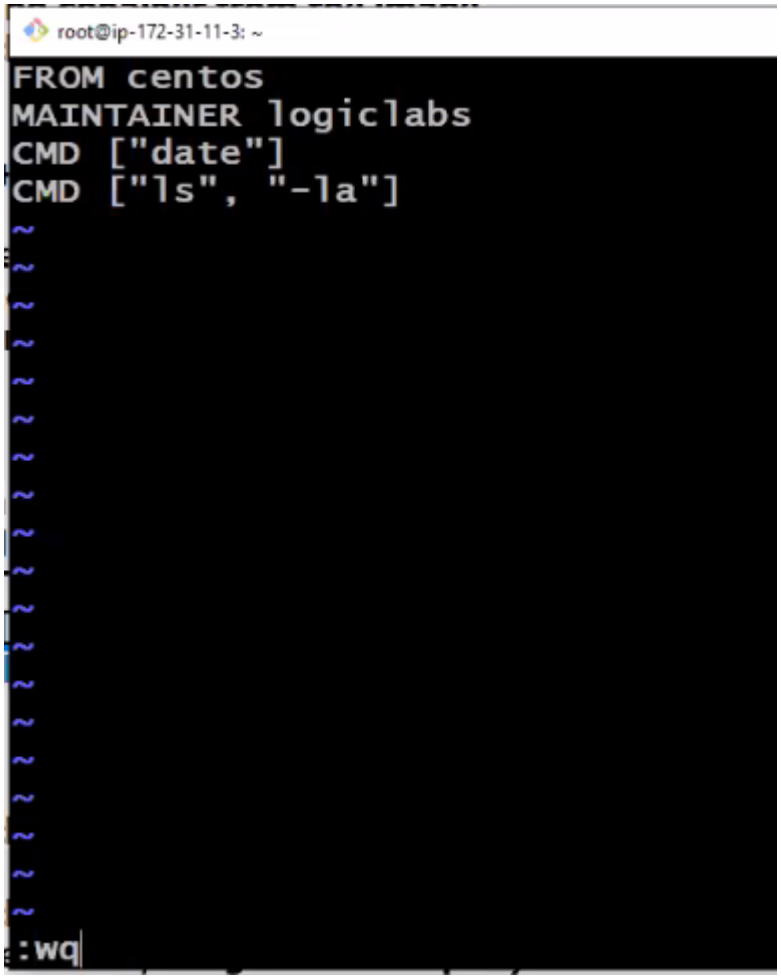
FROM centos

MAINTAINER logiclabs

CMD ["date"]

CMD ["ls", "-la"]

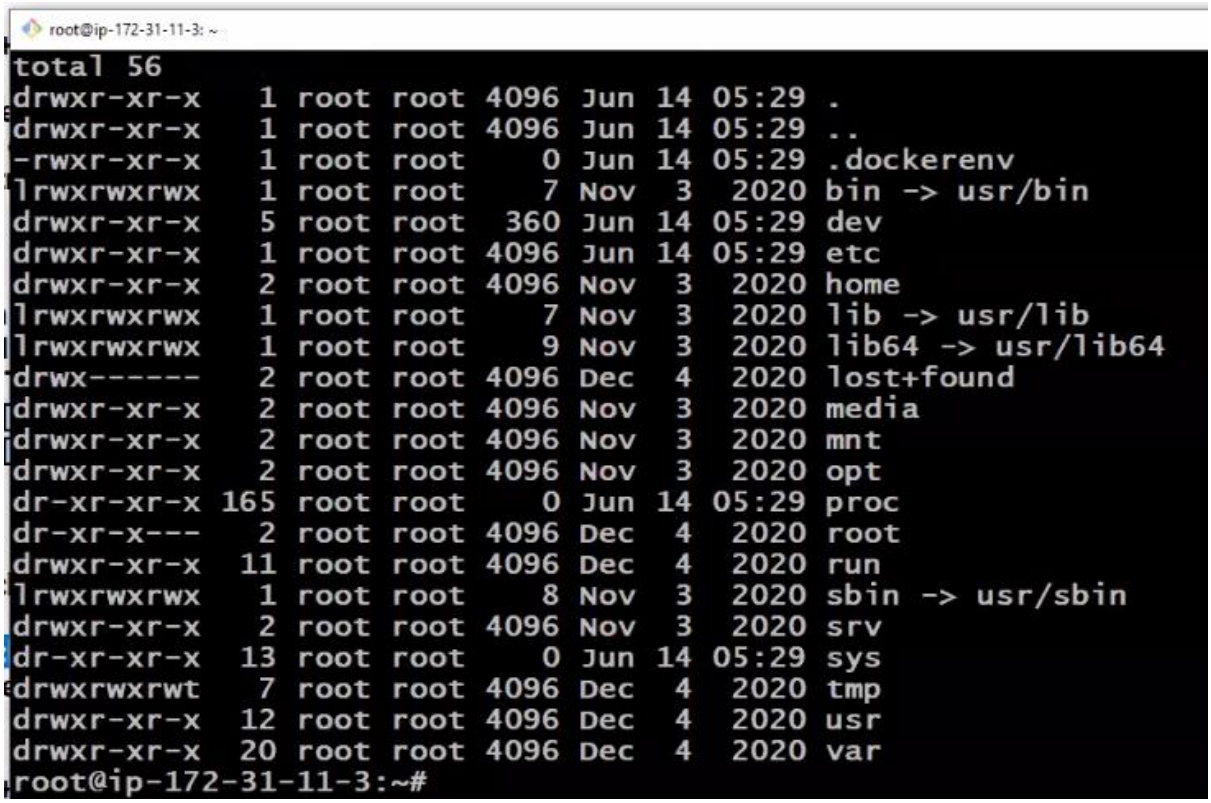
:wq



 # docker build -t mycentos .

```
root@ip-172-31-11-3:~#  docker build -t  mycentos .
Sending build context to Docker daemon  35.84kB
Step 1/4 : FROM centos
 ---> 300e315adb2f
Step 2/4 : MAINTAINER logiclabs
 ---> Using cache
 ---> 4016c0ea756e
Step 3/4 : CMD ["date"]
 ---> Using cache
 ---> 5b439db9689e
Step 4/4 : CMD ["ls", "-la"]
 ---> Running in 72ab28ef605d
Removing intermediate container 72ab28ef605d
 ---> 40b000dd85ed
Successfully built 40b000dd85ed
Successfully tagged mycentos:latest
root@ip-172-31-11-3:~#
```

 # docker run -it   mycentos



(Observation, we get ls -la output)

++

In ubuntu container, I want to install git in it.

Lets remove the docker file

 # rm dockerfile

 # vim dockerfile

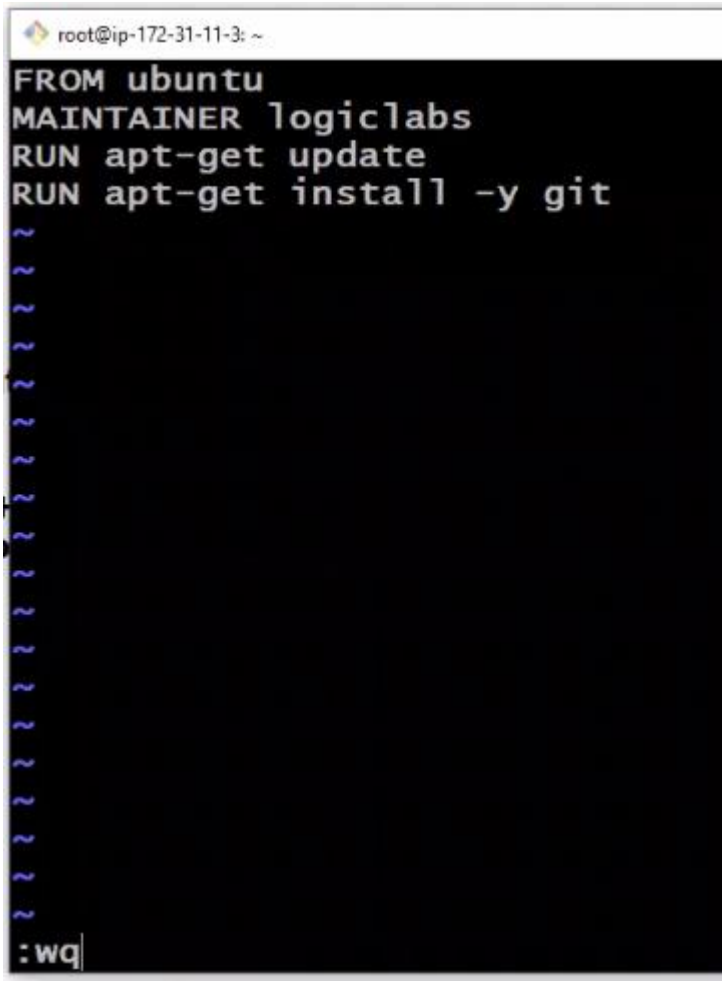
FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

:wq



Note: CMD -- will run when container starts.

RUN -- will executed when image is created.

 # docker build -t myubuntu .

```
root@ip-172-31-11-3:~#  docker build -t  myubuntu .
Sending build context to Docker daemon  39.42kB
Step 1/4 : FROM ubuntu
 ---> 7e0aa2d69a15
Step 2/4 : MAINTAINER logiclabs
 ---> Running in 8aa21ccf64b7
Removing intermediate container 8aa21ccf64b7
 ---> da4ffd4612eb
Step 3/4 : RUN apt-get update
 ---> Running in 7f836ed22fba
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [
114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [11
4 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [
101 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [
1275 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/multiverse
amd64 Packages [27.6 kB]
```

```
Setting up libxext6:amd64 (2:1.3.4-0ubuntu1) ...
Setting up perl (5.30.0-9ubuntu0.2) ...
Setting up xauth (1:1.1-0ubuntu1) ...
Setting up libkrb5-26-heimdal:amd64 (7.7.0+dfsg-1ubuntu1) ...
Setting up libheimntlm0-heimdal:amd64 (7.7.0+dfsg-1ubuntu1) ...
Setting up liberror-perl (0.17029-1) ...
Setting up libgssapi3-heimdal:amd64 (7.7.0+dfsg-1ubuntu1) ...
Setting up libldap-2.4-2:amd64 (2.4.49+dfsg-2ubuntu1.8) ...
Setting up libcurl3-gnutls:amd64 (7.68.0-1ubuntu2.5) ...
Setting up git (1:2.25.1-1ubuntu3.1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for ca-certificates (20210119~20.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Removing intermediate container 9751f9c439b2
 ---> 00bbac1c5e78
Successfully built 00bbac1c5e78
Successfully tagged myubuntu:latest
root@ip-172-31-11-3:~#
```

Observe updating apt repos

2.installing git

3. CMD -- will run when container starts.

so RUN -- will executed when image is created.

Lets see the images list and space consumed by our image

 # docker images

```
root@ip-172-31-11-3:~# docker image ls
REPOSITORY                      TAG       IMAGE ID       CREATED          SIZE
myubuntu                        latest    00bbac1c5e78   24 seconds ago   203MB
mycentos                        latest    40b000dd85ed   6 minutes ago    209MB
mynginx                         latest    d7681be4fec1   17 minutes ago   133MB
<none>                          <none>    dd3827ec8fca   5 days ago       203MB
selenium/node-firefox-debug     latest    5a02336ff62a   6 days ago       988MB
selenium/node-chrome-debug      latest    2df3498bcff1   6 days ago       1.07GB
selenium/hub                    latest    705be32777f0   6 days ago       283MB
php                             latest    7a9e4bdd6171   9 days ago       423MB
wordpress                       latest    c2dd1984ad5b   10 days ago      551MB
jenkins/jenkins                 latest    8959ce44055e   12 days ago      570MB
nginx                           latest    d1a364dc548d   2 weeks ago      133MB
mysql                           5         2c9028880e58   4 weeks ago      447MB
ubuntu                          latest    7e0aa2d69a15   7 weeks ago      72.7MB
tomee                           latest    bc73f72b54d0   4 months ago     340MB
centos                          latest    300e315adb2f   6 months ago     209MB
root@ip-172-31-11-3:~#
```

 # docker run -it myubuntu

 # git --version

```
root@d22e004a89be: /
root@ip-172-31-11-3:~#  docker run -it   myubuntu
root@d22e004a89be:/#
root@d22e004a89be:/#
root@d22e004a89be:/# git --version
git version 2.25.1
root@d22e004a89be:/#
root@d22e004a89be:/#
root@d22e004a89be:/#
```

 # exit

++++++++++++++++++++++++++++++++++

Lets perform version controlling in docker file

 # mkdir docker

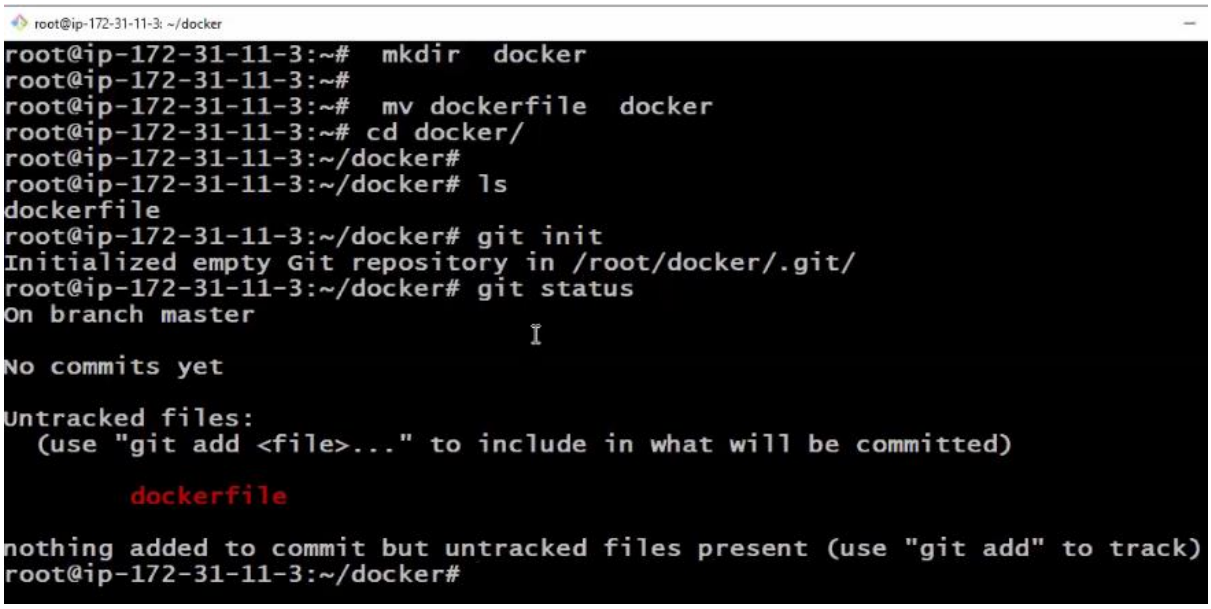
 # mv dockerfile docker

 # cd docker

 # ls

docker# git init

docker# git status



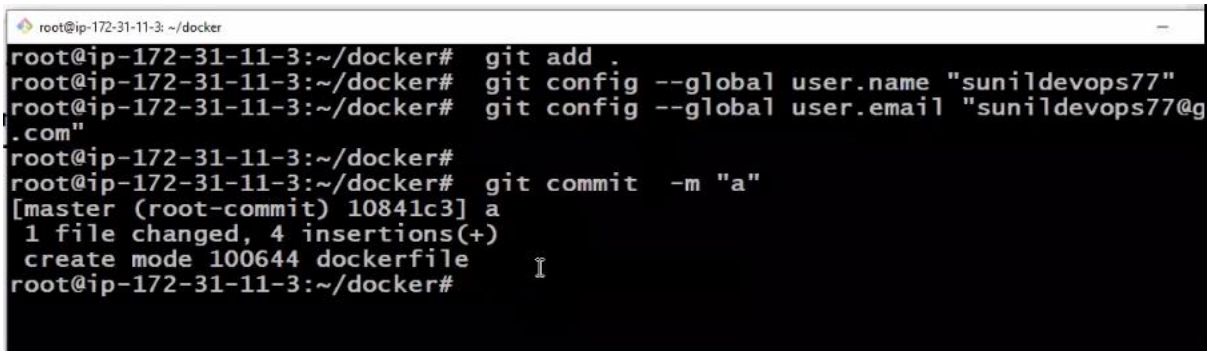
docker# git add .

docker# git commit -m "a"

(we get error, we need to config git)

docker# git config --global user.name "sunildevops77"

docker# git config --global user.email "sunildevops77@gmail.com"



Now, run the above commit command (git commit)

docker# vim dockerfile (lets make some changes add another RUN command)

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

RUN apt-get install -y default-jdk

:wq

```
root@ip-172-31-11-3:~/docker# git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   dockerfile

no changes added to commit (use "git add" and/or "git commit -a")
root@ip-172-31-11-3:~/docker#
```

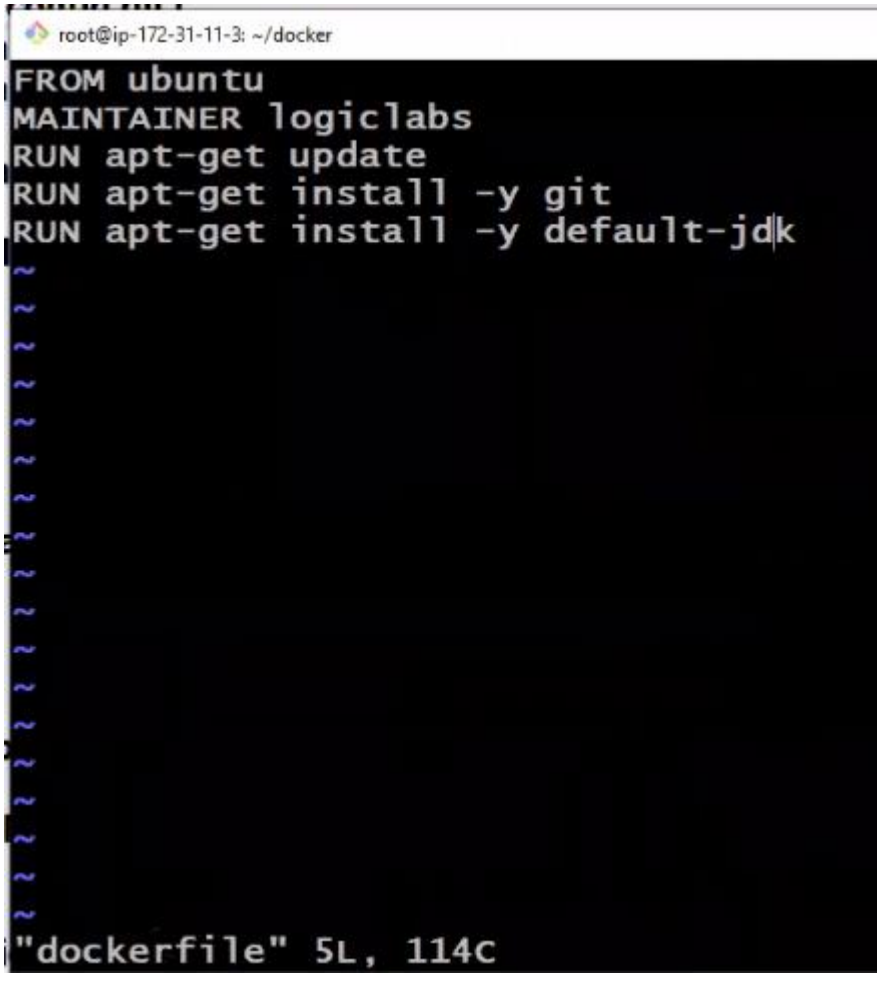
docker# git add .

docker# git commit -m "b"

```
root@ip-172-31-11-3:~/docker#  git add .
root@ip-172-31-11-3:~/docker#  git commit  -m "b"
[master f71122b] b
 1 file changed, 1 insertion(+)
root@ip-172-31-11-3:~/docker#
```

Now lets see the docker file

 # vim dockerfile (we see the latest one is added jdk)



Now, I want to have previous version

 # git log --oneline (to see the list of all the commits)

```
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker#  git log  --oneline
f71122b (HEAD -> master) b
10841c3 a
root@ip-172-31-11-3:~/docker#
```

We want to move to "a" commit (take note of commit id)-rolling back to a commit

```
root@ip-172-31-11-3:~/docker#  vim dockerfile
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker#  git log  --oneline
f71122b (HEAD -> master) b
10841c3 a
root@ip-172-31-11-3:~/docker# git reset --hard  10841c3
HEAD is now at 10841c3 a
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker#
```

 # git reset --hard 10841c3

Now lets see the docker file

 # vim dockerfile (we see the old one)

++++++++++++++++++++++++++++++++++

Cache busting

Whenever an image is build from a dockerfile, docker reads its memory and checks which instructions were already executed.

These steps will not be reexecuted.

It will execute only the latest instructions. This is a time saving mechanism provided by docker.

But, the disadvantage is, we can end up installing software packages from a repository which is updated long time back.

Ex:

```
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker# ls
dockerfile
```

 # cd docker

 # vim dockerfile

```
FROM ubuntu
MAINTAINER logiclabs
RUN apt-get update
RUN apt-get install -y git
~
~
```

Lets just add one more instruction

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

RUN apt-get install -y tree

:wq

Lets build an image

 # docker build -t myubuntu .

```
root@ip-172-31-11-3:~/docker#  docker build -t myubuntu  .
Sending build context to Docker daemon   55.3kB
Step 1/5 : FROM ubuntu
 ---> 7e0aa2d69a15
Step 2/5 : MAINTAINER logiclabs
 ---> Using cache
 ---> da4ffd4612eb
Step 3/5 : RUN apt-get update
 ---> Using cache
 ---> 5fafd3831bf4
Step 4/5 : RUN apt-get install -y git
 ---> Using cache
 ---> 00bbac1c5e78
Step 5/5 : RUN apt-get install -y tree
 ---> Running in a5ddbc8f502f
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 5 not upgraded.
Need to get 43.0 kB of archives.
After this operation, 115 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 tree amd64 1.8.0-1 [43.0 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 43.0 kB in 1s (73.3 kB/s)
Selecting previously unselected package tree.
(Reading database ... 7943 files and directories currently installed.)
```

(Observe the output, Step 2, 3, 4 is using cache. Only step 5 is executed freshly)

Advantage: time saving mechanism

Disadvantage : Lets say, you are running after 4 months, We are installing tree from apt which is updated long time back.)

TO avoid this disadvanatge we use cache busting

Busting =clearing cache

Note: cache busting is implemented using && symbol.

Which ever statement in the docker file has && will be re-executed.

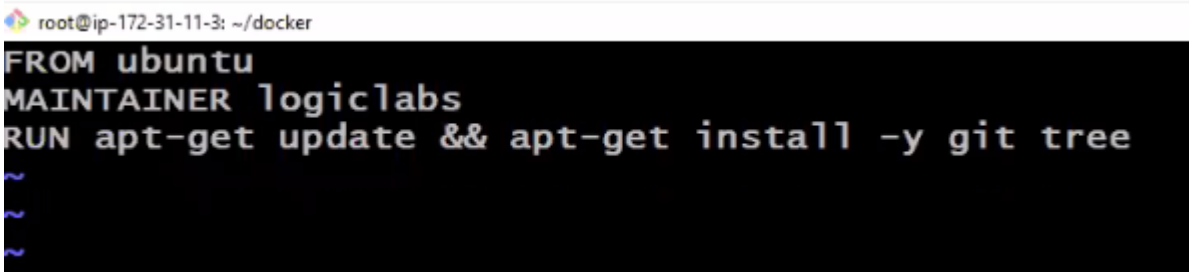
 # vim dockerfile

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update && apt-get install -y git tree

:wq



Lets build an image

 # docker build -t myubuntu .

```
root@ip-172-31-11-3:~/docker#  docker build -t myubuntu .
Sending build context to Docker daemon   55.3kB
Step 1/3 : FROM ubuntu
 ---> 7e0aa2d69a15
Step 2/3 : MAINTAINER logiclabs
 ---> Using cache
 ---> da4ffd4612eb
Step 3/3 : RUN apt-get update && apt-get install -y git tree
 ---> Running in e8c7f9ca14b1
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [27.6 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [879 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [328 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [729 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:13 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [355 kB]
Get:14 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [32.0 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [1299 kB]
Get:16 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [979 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages [4305 B]
```

(Observe the output, step 3 - It is not using cache)

If u dnt provide name to the container it uses its own name

15th June

Ex: Create a dockerfile, for using ubuntu as base image, and install java in it.

Download jenkins.war and make execution of "java -jar jenkins.war" as the default process.

Every docker image come with default process. Ex:Ubuntu, Jenkins comes with default process.

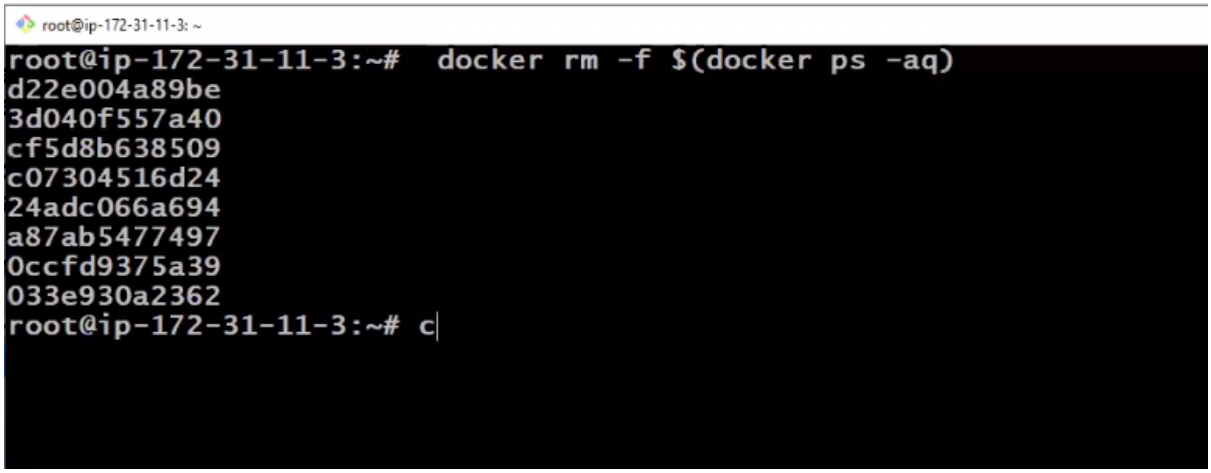
Whenever we run image the dafult process gets executed

We run the image the container gets created at the same time As long as default process is running, the container will be running condition.

The moment, the default process is closed, the container will be exited.(process stop→container stops)

Lets remove all the container

 # docker rm -f $(docker ps -aq)

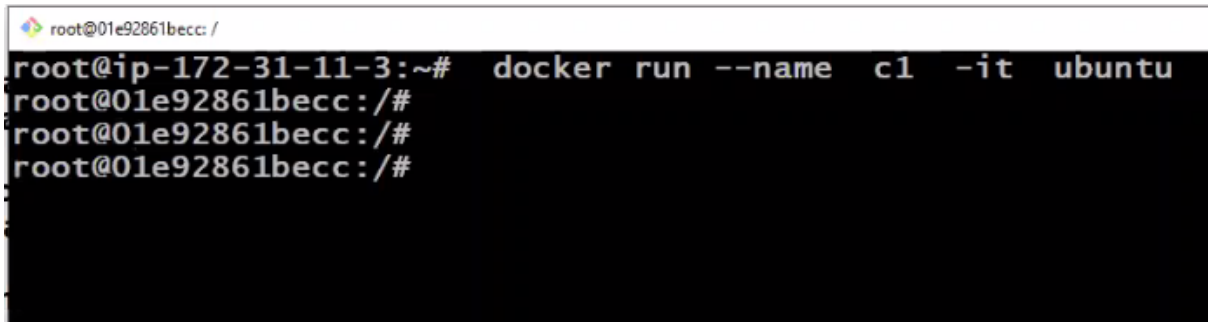


Observation 1:

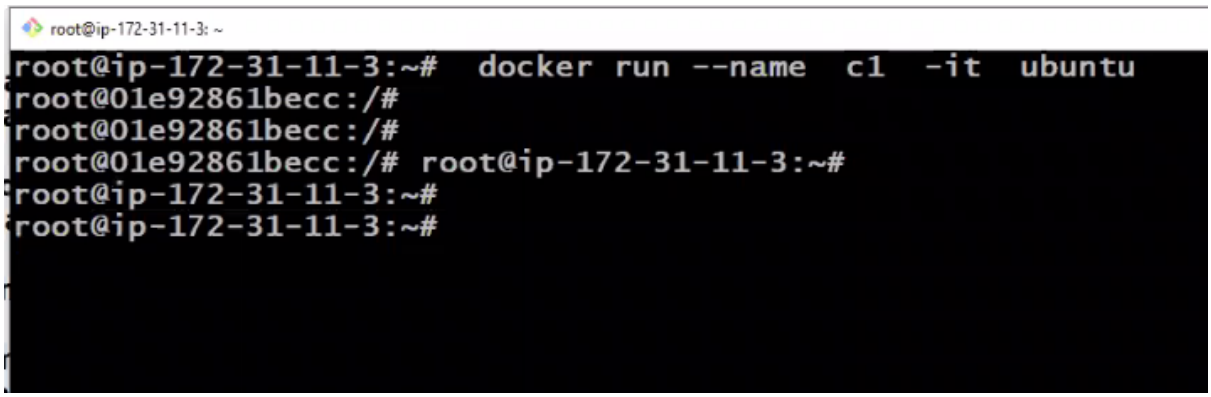
When we start ubuntu container, we use below command

 # docker run --name c1 -it ubuntu

/#

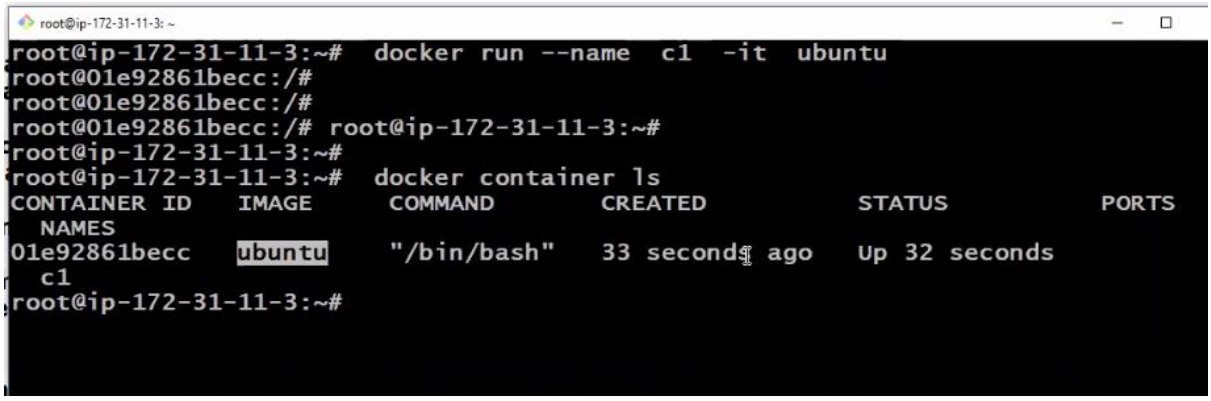


To comeout of the container we use Ctrl + p + q



 # docker container ls

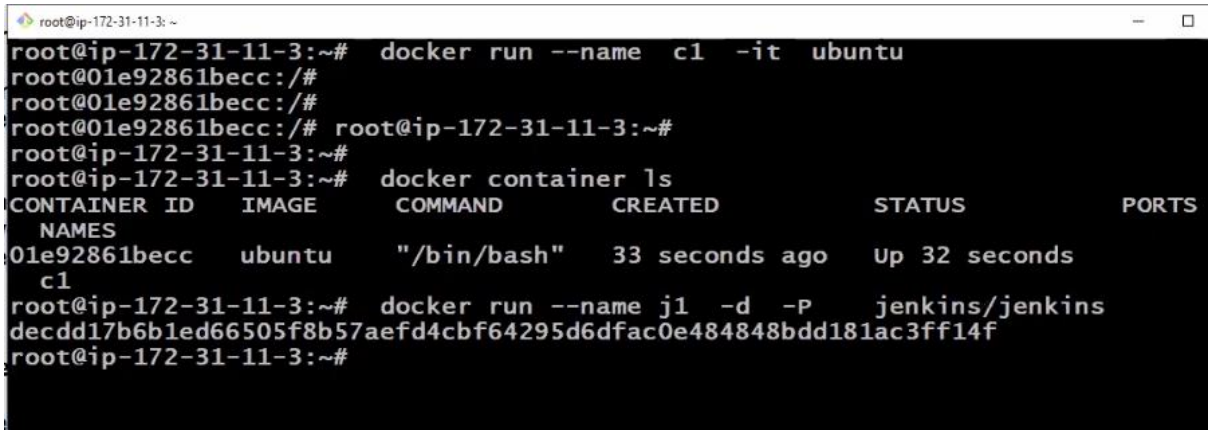
(our container c1 is running in the background)



Observation 2:

When we start jenkins container, we use below command

 # docker run --name j1 -d -P jenkins/jenkins



Now, I want to open interactive terminal to enter jenkins

 # docker exec -it j1 bash

(In ubuntu container, I can directly go into -it terminal,

where as in jenkins i am running an additional command exec ?)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker  exec -it  j1  bash
jenkins@decdd17b6b1e:/$
jenkins@decdd17b6b1e:/$
jenkins@decdd17b6b1e:/$ read escape sequence
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

ctrilpq

Lets try to go to interactive terminal in docker run command)

 # docker run --name j2 -it jenkins/jenkins

(we are not getting interactive terminal in Jenkins but Ubuntu we can enter into interactive terminal)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker run  --name  j2  -it  jenkins/jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
2021-06-15 05:13:08.050+0000 [id=1]     INFO    org.eclipse.jetty.util.log.Log#i
nitialized: Logging initialized @1134ms to org.eclipse.jetty.util.log.JavaUtilLo
g
2021-06-15 05:13:08.334+0000 [id=1]     INFO    winstone.Logger#logInternal: Beg
inning extraction from war file
2021-06-15 05:13:10.297+0000 [id=1]     WARNING o.e.j.s.handler.ContextHandler#s
etContextPath: Empty contextPath
2021-06-15 05:13:10.443+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#
doStart: jetty-9.4.41.v20210516; built: 2021-05-16T23:56:28.993Z; git: 98607f93c
7833e7dc59489b13f3cb0a114fb9f4c; jvm 1.8.0_292-b10
2021-06-15 05:13:10.982+0000 [id=1]     INFO    o.e.j.w.StandardDescriptorProces
sor#visitServlet: NO JSP Support for /, did not find org.eclipse.jetty.jsp.Jetty
JspServlet
2021-06-15 05:13:11.088+0000 [id=1]     INFO    o.e.j.s.s.DefaultSessionIdManage
r#doStart: DefaultSessionIdManager workerName=node0
2021-06-15 05:13:11.096+0000 [id=1]     INFO    o.e.j.s.s.DefaultSessionIdManage
r#doStart: No SessionScavenger set, using defaults
2021-06-15 05:13:11.099+0000 [id=1]     INFO    o.e.j.server.session.HouseKeeper
#startScavenging: node0 Scavenging every 660000ms
```

```
root@ip-172-31-11-3: ~
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password
 generated.
Please use the following password to proceed to installation:

e1074922a2c94e7fb0abae14c345994f

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************

^Croot@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

I want to run tomcat as container

 # docker run --name t1 -d -P tomee

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker run  --name   t1  -d   -P   tomee
8f7e038c01f409e7cdfb0ef04a853d8003696f7ee453ab0ea3d148a96154804b
root@ip-172-31-11-3:~#
```

Lets find the reason

----

docker container ls (to see the list of containers)

```
root@ip-172-31-11-3:~# docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATU
S          PORTS                                                                 NAMES
8f7e038c01f4   tomee             "catalina.sh run"        11 seconds ago   Up 10
 seconds   0.0.0.0:49155->8080/tcp, :::49155->8080/tcp                           t1
decdd17b6b1e   jenkins/jenkins   "/sbin/tini -- /usr/…"   3 minutes ago    Up 3
minutes   0.0.0.0:49154->8080/tcp, :::49154->8080/tcp, 0.0.0.0:49153->50000/tcp
, :::49153->50000/tcp   j1
01e92861becc   ubuntu            "/bin/bash"              4 minutes ago    Up 4
minutes                                                                         c1
root@ip-172-31-11-3:~#
```

Observer the command section.

It tells you the default process that gets executed, when we start the container.

Container	Default process
tomcat	catalina.sh
jenkins	/bin/tini
ubuntu	/bin/bash

bash -- is nothing but the terminal.

For linux based container, the default process is shell process wx:linux,ubuntu

(ex of shell process are bash shell, bourne shell etc)

Hence we are able to enter -it mode in ubuntu)

Docker attach is back to connect containers only for linux,Ubuntu which has bash.

We are trying to change the default process of the container.

Means bin/bash to java.war for ubuntu

 # vim dockerfile

FROM Ubuntu (base image)

MAINTAINER logiclabs (org /author)

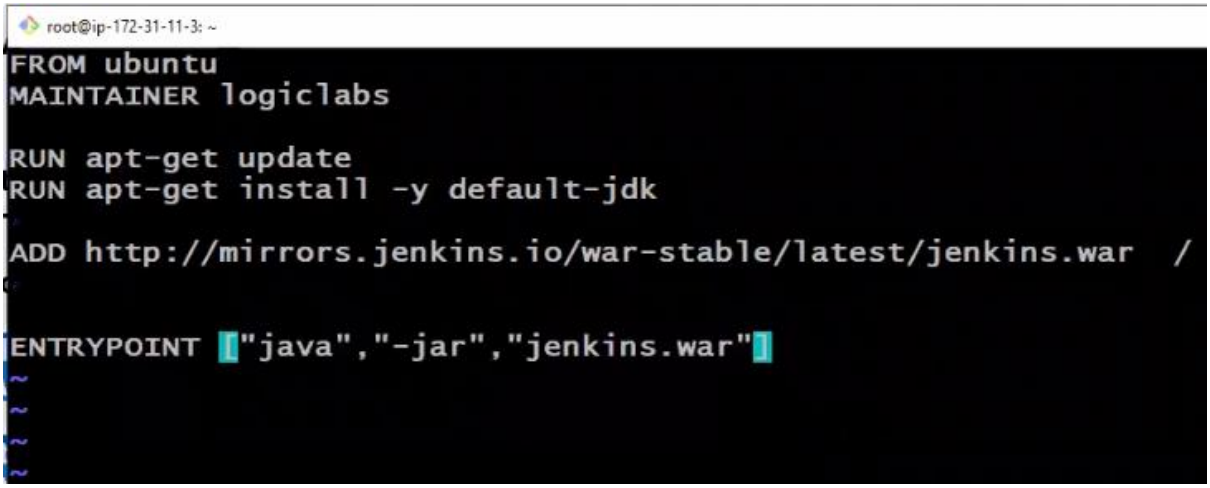
RUN apt-get update (installed softwares in creating image)

RUN apt-get install -y default-jdk

ADD http://mirrors.jenkins.io/war-stable/latest/jenkins.war /

ENTRYPOINT ["java","-jar","jenkins.war"] (add=download particular file)

:wq



Build an image from the dockerfile

 # docker build -t myubuntu . (. To be in current directory)

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker build -t  myubuntu  .
Sending build context to Docker daemon  94.21kB
Step 1/6 : FROM ubuntu
 ---> 7e0aa2d69a15
Step 2/6 : MAINTAINER logiclabs
 ---> Using cache
 ---> da4ffd4612eb
Step 3/6 : RUN apt-get update
 ---> Using cache
 ---> 5fafd3831bf4
Step 4/6 : RUN apt-get install -y default-jdk
 ---> Running in 2a41c63d8473
```

TO see the list of images (we can see our new image)

 # docker image ls

```
root@ip-172-31-11-3: ~

done.
done.
Removing intermediate container 66a928687055
 ---> f75f229664d3
Step 5/6 : ADD https://get.jenkins.io/war-stable/2.289.1/jenkins.war  /
Downloading  74.25MB/74.25MB

 ---> c6253f656679
Step 6/6 : ENTRYPOINT ["java","-jar","jenkins.war"]
 ---> Running in 735ef7355e12
Removing intermediate container 735ef7355e12
 ---> 0ef3175f4043
Successfully built 0ef3175f4043
Successfully tagged newubuntu:latest
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
newubuntu           latest    0ef3175f4043   13 seconds ago   831MB
jenkins/jenkins     latest    8959ce44055e   13 days ago      570MB
ubuntu              latest    7e0aa2d69a15   7 weeks ago      72.7MB
tomee               latest    bc73f72b54d0   4 months ago     340MB
root@ip-172-31-11-3:~#
```

Standard image size is 72mb and new Ubuntu size is 831 bcz it has Jenkins in ubuntu

TO start container from new image

When you run below image can see logs of Jenkins.

 # docker run newubuntu (Observe the logs generated on the

screen, we got logs related to jenkins , jenkins is fully up and running)

```
root@ip-172-31-11-3:~#  docker run  newubuntu
Running from: /jenkins.war
webroot: $user.home/.jenkins
2021-06-15 05:34:28.408+0000 [id=1]     INFO    org.eclipse.jetty.util.log.Log#i
nitialized: Logging initialized @1297ms to org.eclipse.jetty.util.log.JavaUtilLo
g
2021-06-15 05:34:28.613+0000 [id=1]     INFO    winstone.Logger#logInternal: Beg
inning extraction from war file
2021-06-15 05:34:30.361+0000 [id=1]     WARNING o.e.j.s.handler.ContextHandler#s
etContextPath: Empty contextPath
2021-06-15 05:34:30.532+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#
doStart: jetty-9.4.39.v20210325; built: 2021-03-25T14:42:11.471Z; git: 9fc7ca5a9
22f2a37b84ec9dbc26a5168cee7e667; jvm 11.0.11+9-Ubuntu-0ubuntu2.20.04
2021-06-15 05:34:31.283+0000 [id=1]     INFO    o.e.j.w.StandardDescriptorProces
sor#visitServlet: NO JSP Support for /, did not find org.eclipse.jetty.jsp.Jetty
JspServlet
2021-06-15 05:34:31.384+0000 [id=1]     INFO    o.e.j.s.s.DefaultSessionIdManage
r#doStart: DefaultSessionIdManager workerName=node0
2021-06-15 05:34:31.385+0000 [id=1]     INFO    o.e.j.s.s.DefaultSessionIdManage
r#doStart: No SessionScavenger set, using defaults
2021-06-15 05:34:31.388+0000 [id=1]     INFO    o.e.j.server.session.HouseKeeper
#startScavenging: node0 Scavenging every 660000ms
```

Its an ubuntu container, it is behaving as a jenkins container)

Ctrl +c

Observe newubuntu has java –jar jenkns.war

RUn the below command

 # docker ps -a

For myubuntu the command is java -jar jenkins.war

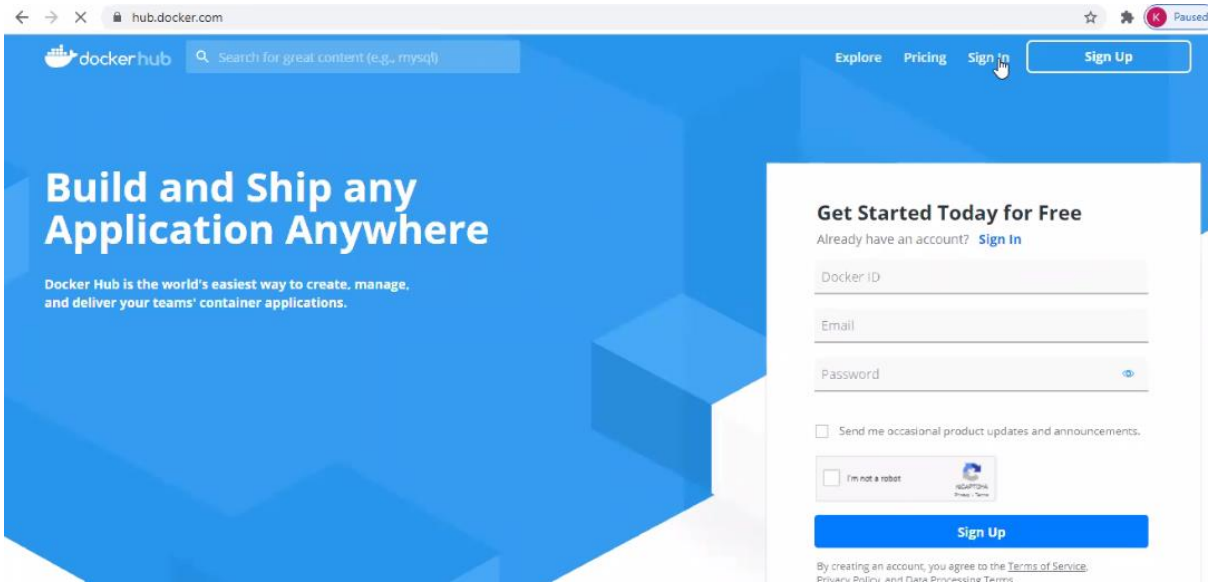
For ubuntu the commans is /bin/bash

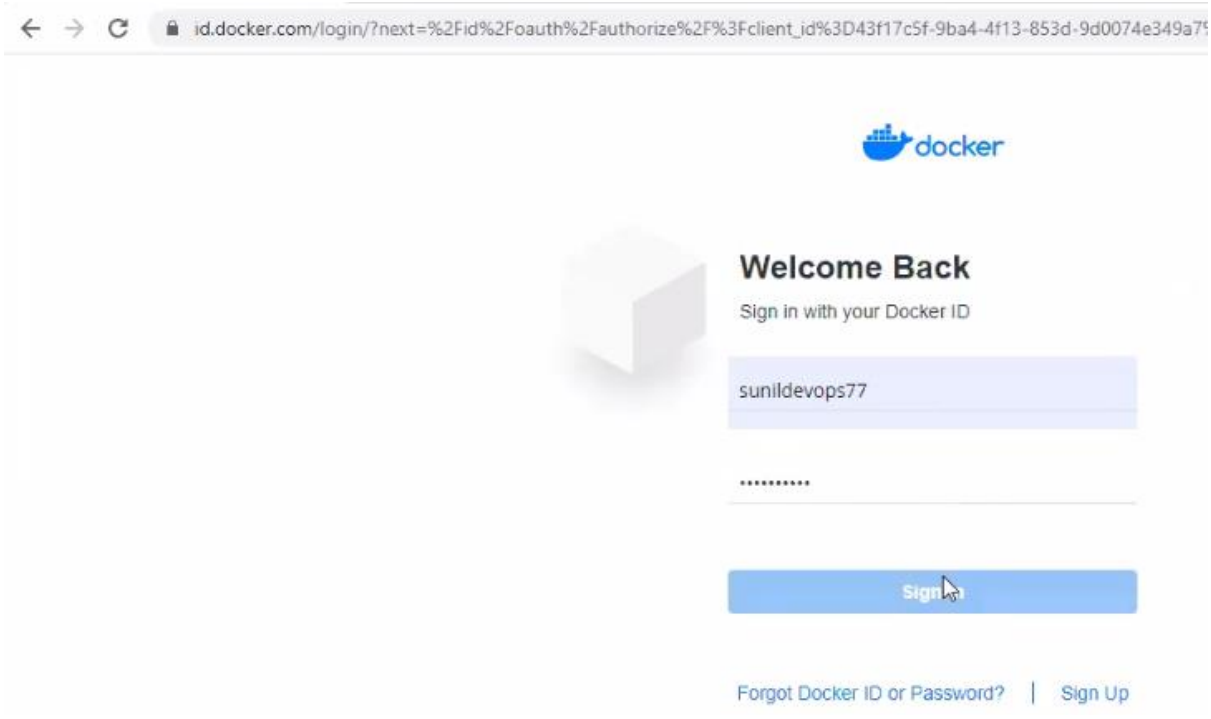
```
root@ip-172-31-11-3:~#  docker ps -a

CONTAINER ID   IMAGE             COMMAND                  CREATED          STATU
S                      PORTS                                                  
                                 NAMES
499724ccda8a   newubuntu         "java -jar jenkins.w…"   32 seconds ago   Exite
d (130) 11 seconds ago                                                         
                                 hopeful_babbage
8f7e038c01f4   tomee             "catalina.sh run"        21 minutes ago   Up 21
 minutes                0.0.0.0:49155->8080/tcp, :::49155->8080/tcp
                                 t1
decdd17b6b1e   jenkins/jenkins   "/sbin/tini -- /usr/…"   24 minutes ago   Up 24
 minutes                0.0.0.0:49154->8080/tcp, :::49154->8080/tcp, 0.0.0.0:49
153->50000/tcp, :::49153->50000/tcp   j1
01e92861becc   ubuntu            "/bin/bash"              25 minutes ago   Up 25
 minutes
                                 c1
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

++

Working on docker registry

Create an account inn docker hub





Registry is a location where docker images are saved.

Types of registry

1) public registry

2) private registry

public registry is hub.docker.com

Images uploaded here are available for everyone.

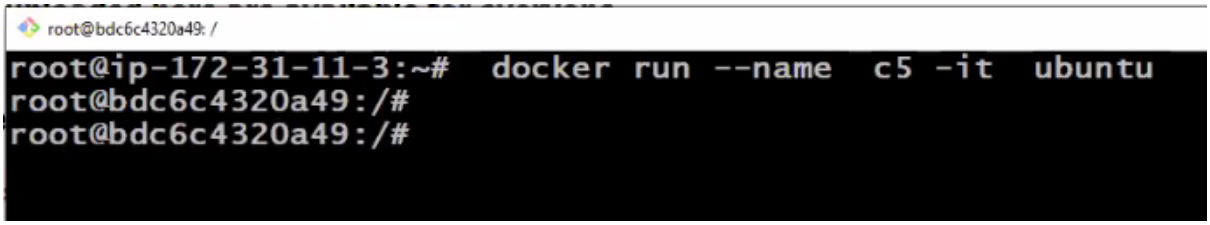
Usecase: Create a customized ubuntu image, by installing tree in it.

Save this container as an image, and upload this image in docker hub.

Step 1: Create a new account in hub.docker.com

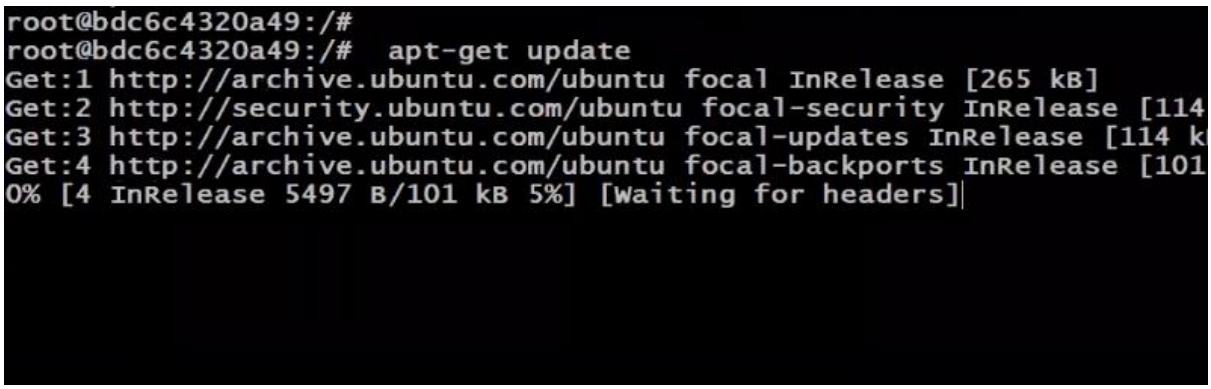
Step 2: Creating our own container

 # docker run --name c5 -it ubuntu



Lets install tree package in this container

/# apt-get update



/# apt-get install tree

```
root@bdc6c4320a49:/#
root@bdc6c4320a49:/#   apt-get  install tree
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 5 not upgraded.
Need to get 43.0 kB of archives.
After this operation, 115 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal/universe amd64 tree amd64 1.8.0-1
43.0 kB]
Fetched 43.0 kB in 0s (114 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package tree.
(Reading database ... 4121 files and directories currently installed.)
Preparing to unpack .../tree_1.8.0-1_amd64.deb ...
Unpacking tree (1.8.0-1) ...
Setting up tree (1.8.0-1) ...
root@bdc6c4320a49:/#
```

/# exit

Step 3: Save the above container as an image

 # docker commit c5 sunildevops77/ubuntu_img291

(sunildevops77/ubuntu_img291 -- is the image name followed by image name)

Note: Image name should start with docker_id/

To see the list of images

 # docker image ls (we can see the new image)

```
root@ip-172-31-11-3: ~
(Reading database ... 4121 files and directories currently installed.)
Preparing to unpack .../tree_1.8.0-1_amd64.deb ...
Unpacking tree (1.8.0-1) ...
Setting up tree (1.8.0-1) ...
root@bdc6c4320a49:/#
root@bdc6c4320a49:/#
root@bdc6c4320a49:/# exit
exit
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker commit  c5  sunildevops77/ubuntu_img12
sha256:8fb73048870c789e01539241a7e14e5316f9a78466cda3ff0d979b16fef424be
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
sunildevops77/ubuntu_img12    latest    8fb73048870c   5 seconds ago    102MB
newubuntu                     latest    0ef3175f4043   8 minutes ago    831MB
jenkins/jenkins               latest    8959ce44055e   13 days ago      570MB
ubuntu                        latest    7e0aa2d69a15   7 weeks ago      72.7MB
tomee                         latest    bc73f72b54d0   4 months ago     340MB
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

TO upload the image to hub.docker.com (docker login command is used)

 # docker login (provide docker_id and password)—docker hub login from terminal

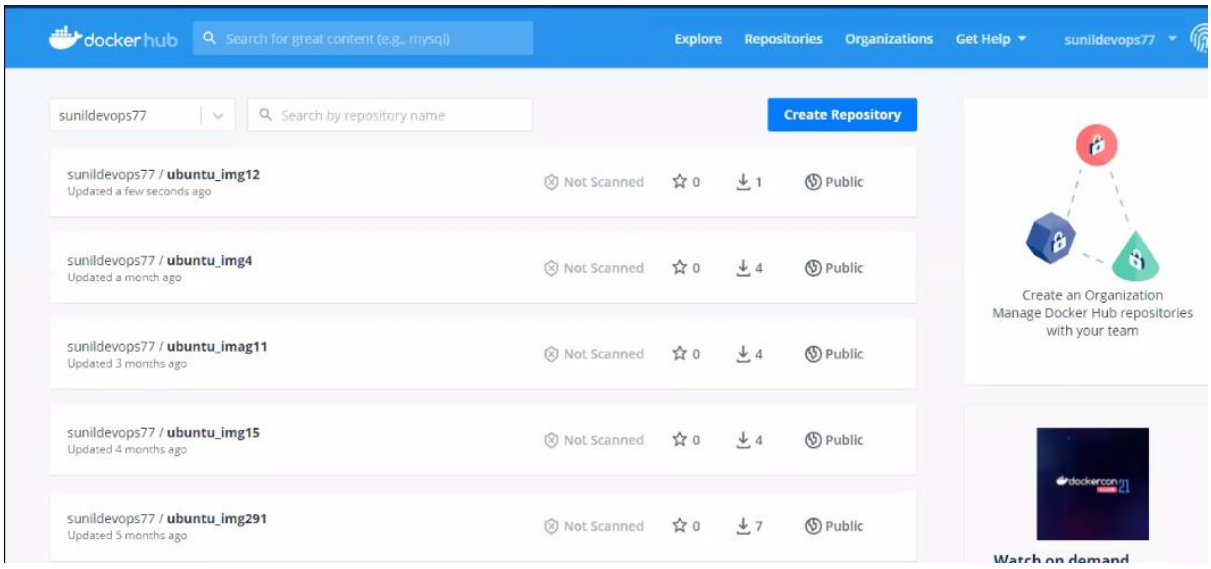
```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: sunildevops77
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@ip-172-31-11-3:~#
```

To upload the image into docker hub

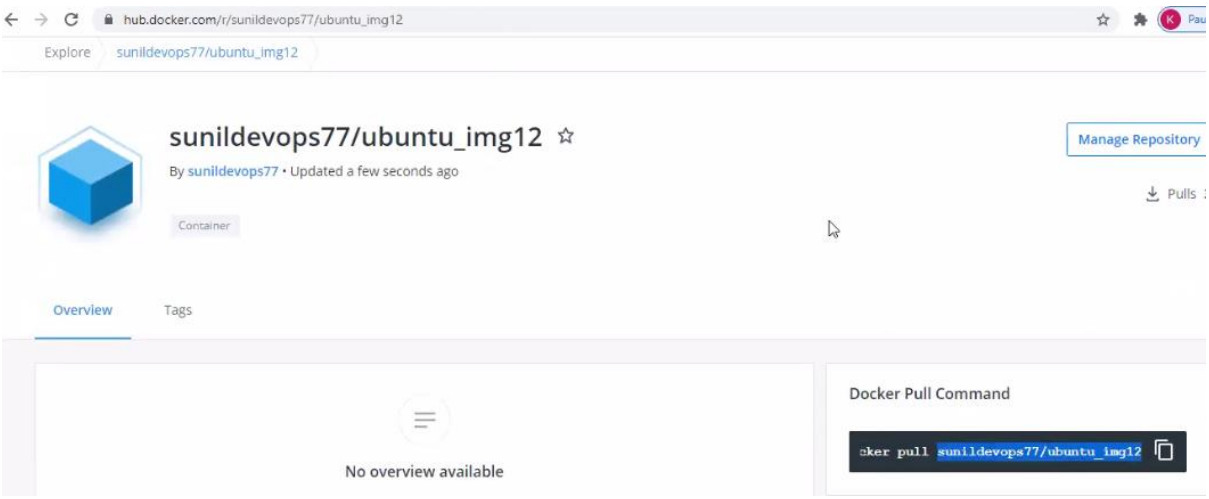
 # docker push <image_name>

 # docker push sunildevops77/ubuntu_img291

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#  docker push sunildevops77/ubuntu_img12
Using default tag: latest
The push refers to repository [docker.io/sunildevops77/ubuntu_img12]
78a3d5b563b5: Pushing  29.65MB
2f140462f3bc: Mounted from library/ubuntu
63c99163f472: Mounted from library/ubuntu
ccdbb80308cc: Mounted from library/ubuntu
```



login to docker hub to see your image



++

Container orchestration

This is the process of running docker containers in a distributed environment is orchestration, on multiple docker host machines.

(Why: Hardware failure – running all conatiners in isngle docker host then all conatineers stopped due to this.Use orchestration- any Hardware failure on docker hosts then the particular conatiners will move to another docker host)

All these containers can have a single service running on them and they share the resources between each other, even running on different host machines.

Docker swarm is the tool used for performing container orchestration

Advantages

----

1) Load balancing

2) scaling of containers—scale up /scale down without any down time

3) performing rolling updates

4) handling failover scenarios

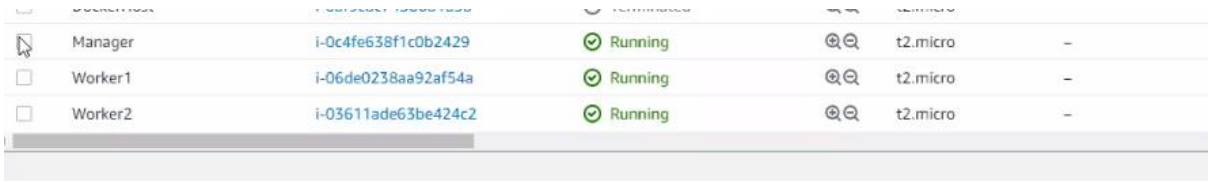
++++++++++++++++++++++++++++

Machine on which docker swarm is installed is called as manager.

Other machines are called as workers.

Lets create 3 machines

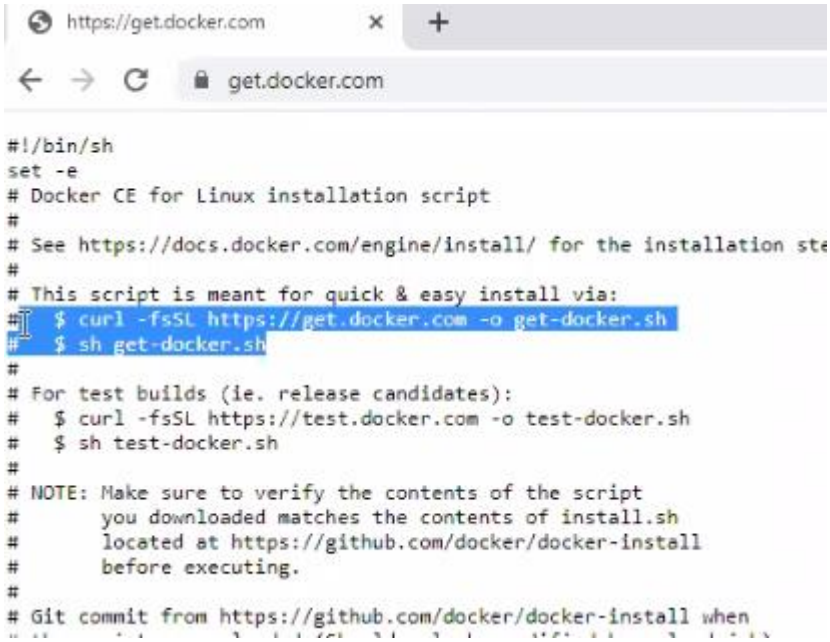
Name is as Manager, Worker1, Worker2



Connect mchines by git bash/putty

All the above machines should have docker installed in it.

Install docker using get.docker.com



Change toroot user if you dnt see

Sudo su -

```
root@ip-172-31-5-111: ~
root@ip-172-31-5-111:~#  curl -fsSL https://get.docker.com -o get-docker.sh
root@ip-172-31-5-111:~#    sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc
737
+ sh -c apt-get update -qq >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-
s ca-certificates curl >/dev/null
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" | apt-key
-qq - >/dev/null
Warning: apt-key output should not be parsed (stdout is not a terminal)
+ sh -c echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu bion
table" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get update -qq >/dev/null
+ [ -n  ]
+ sh -c apt-get install -y -qq --no-install-recommends docker-ce >/dev/null
```

```
docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

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

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface
         documentation for details: https://docs.docker.com/go/attack-surface/

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

root@ip-172-31-5-111:~#
```

```
root@ip-172-31-5-111:~#
root@ip-172-31-5-111:~# docker --version
Docker version 20.10.7, build f0df350
root@ip-172-31-5-111:~#
root@ip-172-31-5-111:~#
```

```
root@ip-172-31-5-111:~# docker --version
Docker version 20.10.7, build f0df350
root@ip-172-31-5-111:~#
root@ip-172-31-5-111:~#
root@ip-172-31-5-111:~# exit
logout
ubuntu@ip-172-31-5-111:~$
```

(Optional step to change the prompt)

Install docker in worker 1 and 2.

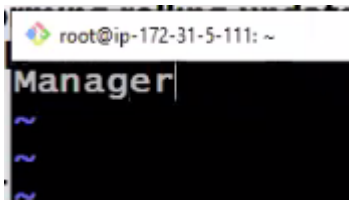
After installing docker in the 1st machine (Manager), Lets change the host name.

Host name will be available in the file hostname. We will change the hostname to manager.

 # vim /etc/hostname

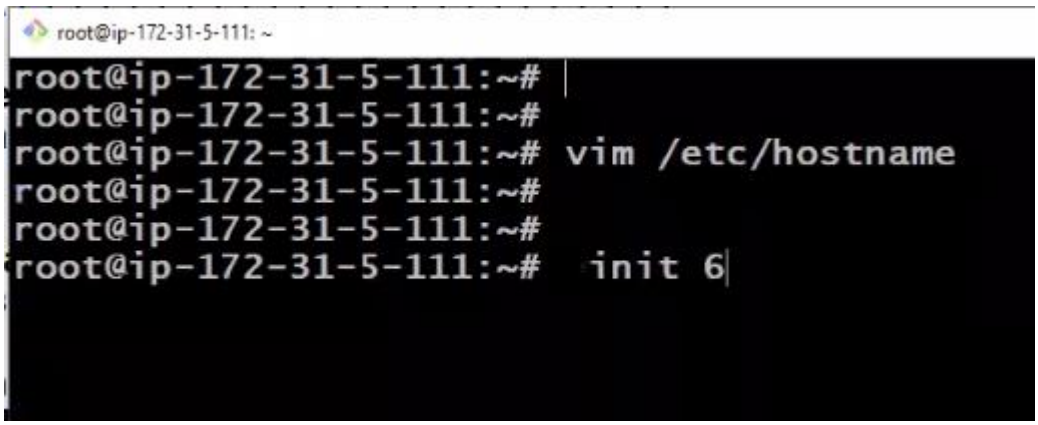
Manager

:wq



After changing the hostname, lets restart the machine

 # init 6



++++++++++++++++++++++++++

Similary repeat the same in worker1 and worker2

++++++++++++++++++++++++++++++++++++

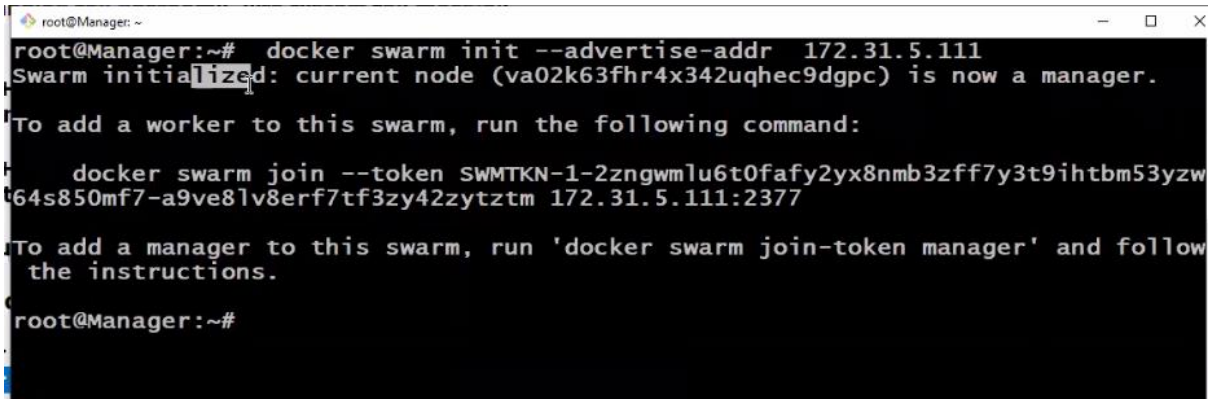
Connect to Manager, install docker swarm in it.

$ sudo su -

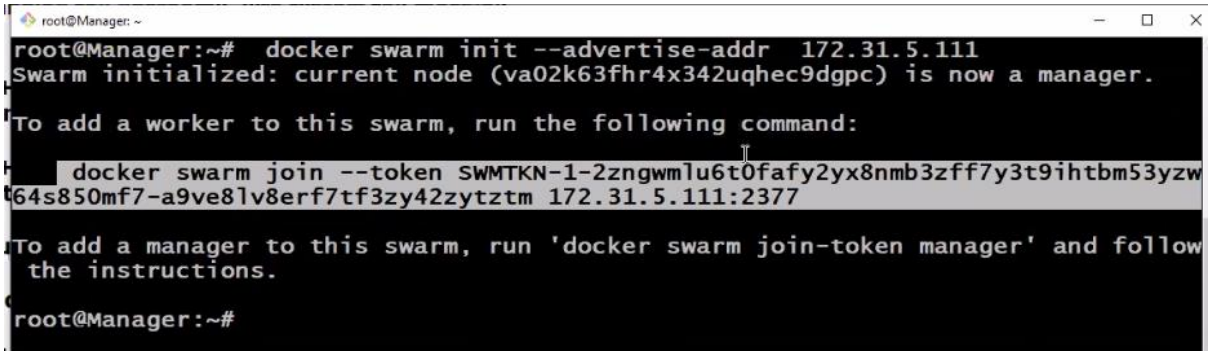
Command to install docker swarm in manager machine

 # docker swarm init --advertise-addr private_ip_of_manager

 # docker swarm init --advertise-addr 172.31.27.151



Please read the log messages and take note of the token



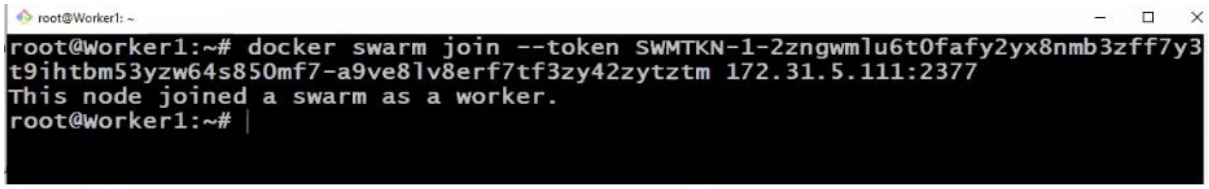
Now, we need to add workers to manager

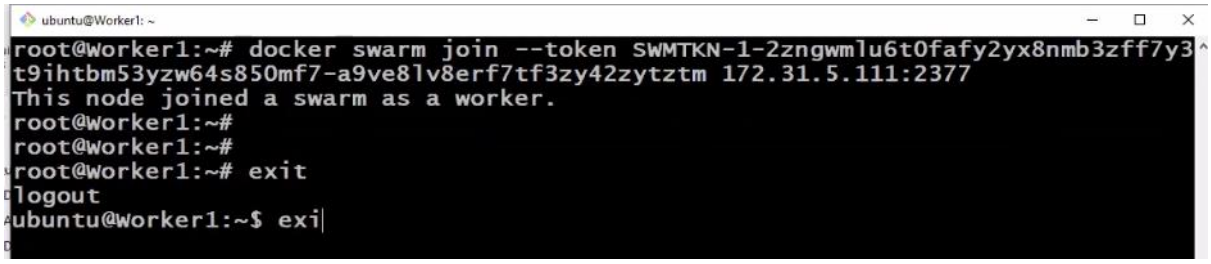
Copy the docker swarm token command to join command in the log and run in the worker1 and worker2

Open another gitbash terminal, connect to worker1

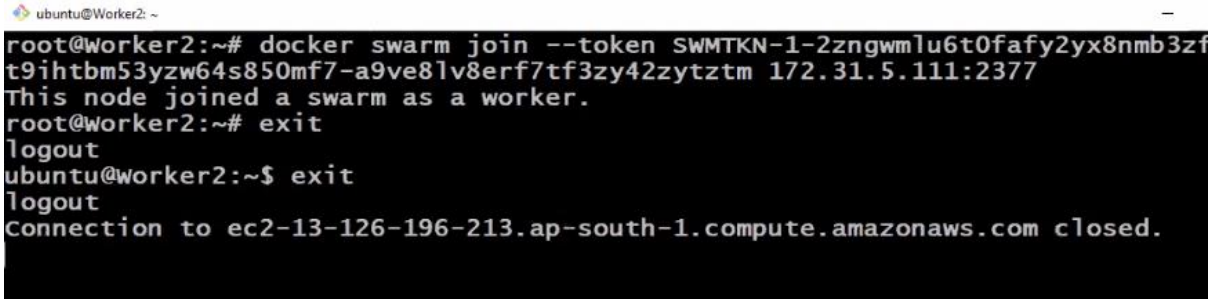
sudo su -

 # docker swarm join --token SWMTKN-1-0etsmfa26vreeytq278q8ohhi73il7j1lpnrzzlowuld1r8yex-9x04pjmiq85jxjzjayzlglh1c 172.31.27.151:2377





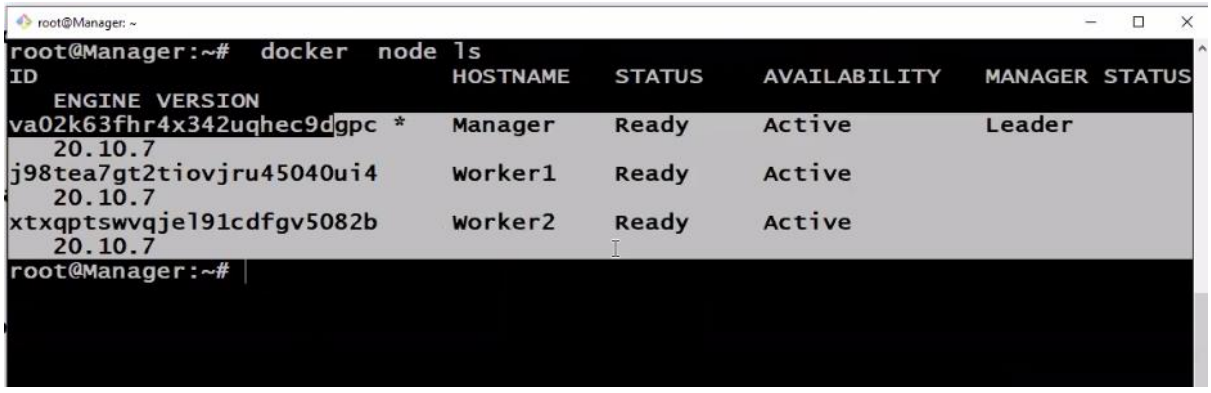
Repeat for worker2



++++++++++++++++++++++++++++++

TO see the no of nodes from the manager

Manager # docker node ls (we can see manager, worker1 and worker 2)

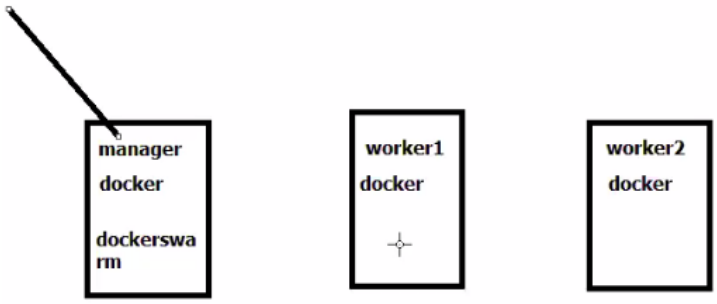


Cluster: group of distributed nodes/machines.

Manager—which controls everything and run evrythign will run on the manager and load will taken to workers.

It will take hardware configuration of all 3 machines.

Worker 1 and worker 2



16th June

Container Orchestration

- **Load Balancing**
- **Scaling of containers**
- **Performing rolling Updates**
- **Handling failover Scenarios**

Container orchestration

This is the process of running docker containers on multiple docker host machines.

in a distributed environment,

All these containers can have a single service running on them and they share the resources between eachother, even running on different host machines.

Docker swarm is the tool used for performing container orchestration

Advantages

----

1) Load balancing

2) scaling of containers

3) performing rolling updates

4) handling failover scenarios

++++++++++++++++++++++++++++

Machine on which docker swarm is installed is called as manager.

Other machines are called as workers.

Lets create 3 machines

Name is as Manager, Worker1, Worker2

All the above machines should have docker installed in it.

Install docker using get.docker.com

(Optional step to change the prompt)

After installing docker in the 1st machine (Manager), Lets change the host name.

Host name will be available in the file hostname. We will change the hostname to manager.

 # vim /etc/hostname

Manager

:wq

After changing the hostname, lets restart the machine

 # init 6

++++++++++++++++++++++++++

Similary repeat the same in worker1 and worker2

++++++++++++++++++++++++++++++++++

Connect to Manager, install docker swarm in it.

$ sudo su -

Command to install docker swarm in manager machine

 # docker swarm init --advertise-addr private_ip_of_manager

 # docker swarm init --advertise-addr 172.31.27.151

Please read the log messages

Now, we need to add workers to manager

Copy the docker swarm join command in the log and run in the worker1 and worker2

Open another gitbash terminal, connect to worker1

sudo su -

 # docker swarm join --token SWMTKN-1-0etsmfa26vreeytq278q8ohhi73il7j1lpnrzzlowuld1r8yex-9x04pjmiq85jxjzjayzlglh1c 172.31.27.151:2377

Repeat for worker2

++++++++++++++++++++++++++++

TO see the no of nodes from the manager

Manager # docker node ls (we can see manager, worker1 and worker 2)

++++++++++++++++

Load balancing: load is distributed to multiple host machines to avoid downtime.

Each docker container is designed to withstand a specific user load.

When the load increases, we can replica containers in docker swarm and distribute the load.

Ex: Start tomcat in docker swarm with 5 replicas and name it as webserver.

Service= bundle of containers

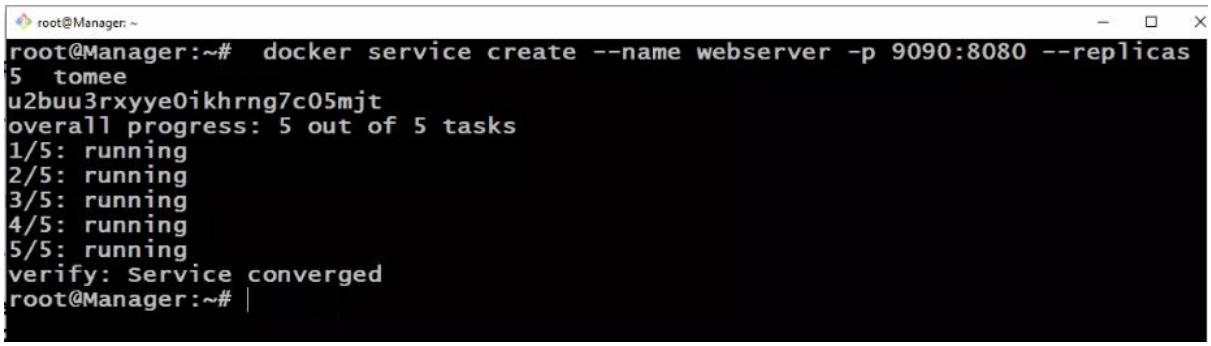
Sudo su -

Manager# docker service create --name webserver -p 9090:8080 --replicas 5 tomee

Only one service= 5 containers

--name of service

(5 conainers with the same service, distributed load in 3 ec2 machines)



How to see where thay are running?

Manager# docker service ps webserver

```
root@Manager:~#
root@Manager:~#  docker service  ps  webserver
ID             NAME          IMAGE          NODE      DESIRED STATE   CURRENT ST
ATE            ERROR     PORTS
6tex9opvvb3i   webserver.1   tomee:latest   Manager   Running         Running 48
 seconds ago
c9ctol7tmuhq   webserver.2   tomee:latest   Worker2   Running         Running 48
 seconds ago
ysuq37miosnj   webserver.3   tomee:latest   Worker1   Running         Running 49
 seconds ago
q8obqofu1gtg   webserver.4   tomee:latest   Manager   Running         Running 47
 seconds ago
mabgi6s46sjz   webserver.5   tomee:latest   Worker2   Running         Running 49
 seconds ago
root@Manager:~#
```

Lets take the note

Manager - 2 container

Worker1 - 1 container

Worker2 - 2 container

We cant decide how the number of conatiners are running on the which the ec2 machine , it is depends on docker swarm allocated automatically to distribute the load on docker host.

We cant distribute the load its automatic process, docker swram will distribute the load to the available docker hosts

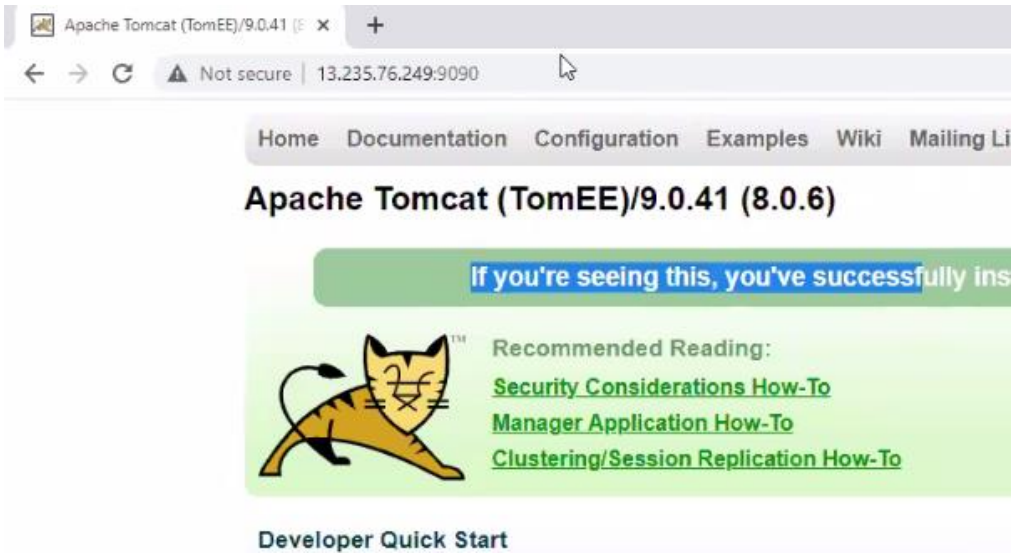
++

Note: Only one tomcat is running and load is shared to 3 machines

Service cant be distributed to multiple docker hosts so we create one srveive as conatiners distribute the load muntile ec2 machines

Lets check

public_ip_manager:9090 (Will show tomcat page)



public_ip_worker1:9090 (Will show tomcat page)

public_ip_worker2:9090 (Will show tomcat page)

13.235.76.249:9090

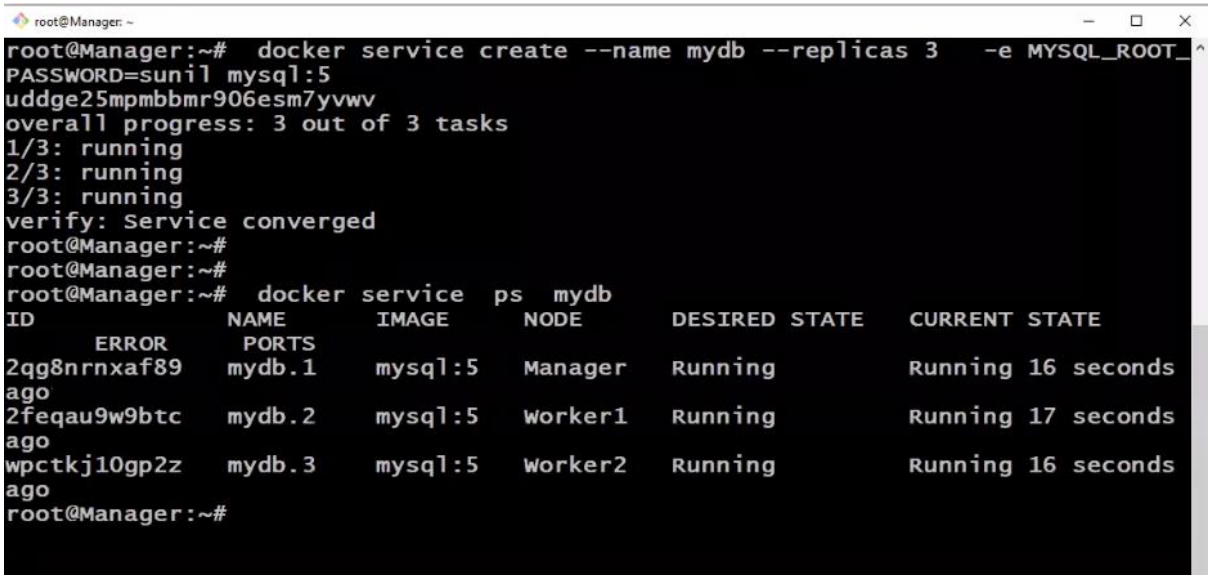
13.233.121.155:9090

13.126.47.212:9090

++++++++++++++++++++++++++++++++++++

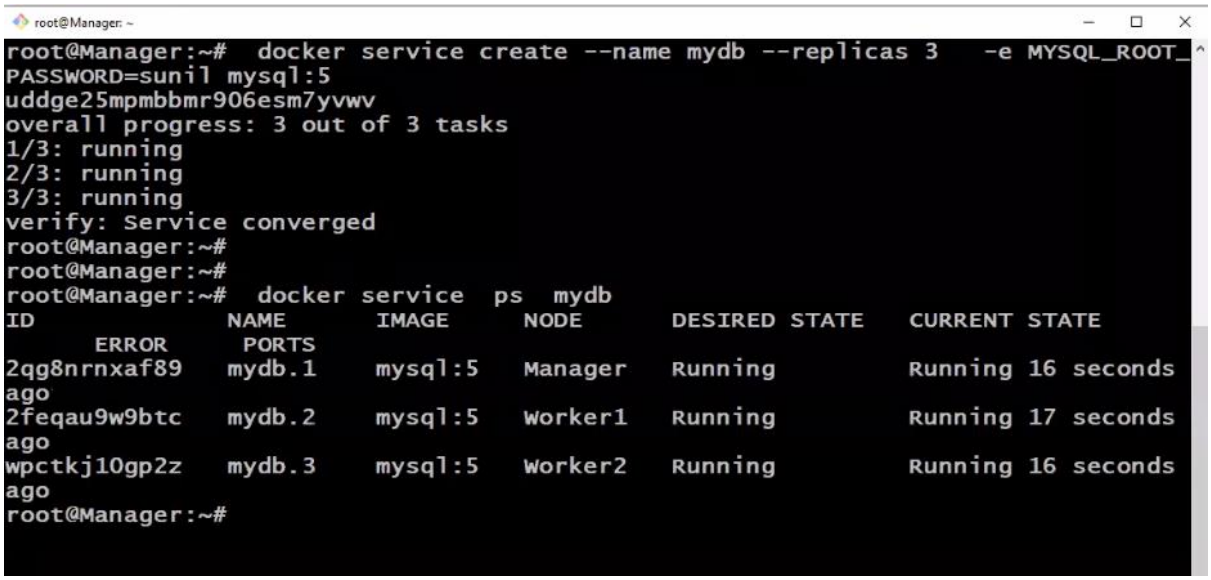
Ex 2: Start mysql in docker swarm with 3 replicas.

Manager# docker service create --name mydb --replicas 3 -e MYSQL_ROOT_PASSWORD=sunil mysql:5



How to see where thay are running?

Manager# docker service ps mydb

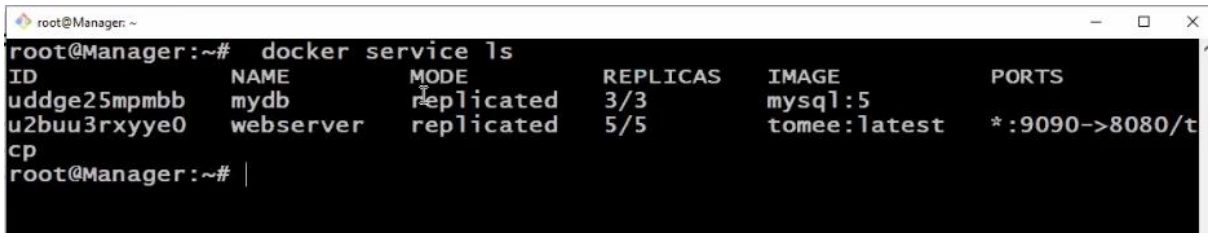


To know the total no of services running in docker swarm so far 2 services

1. tomee service

2. mydb

Manager# docker service ls



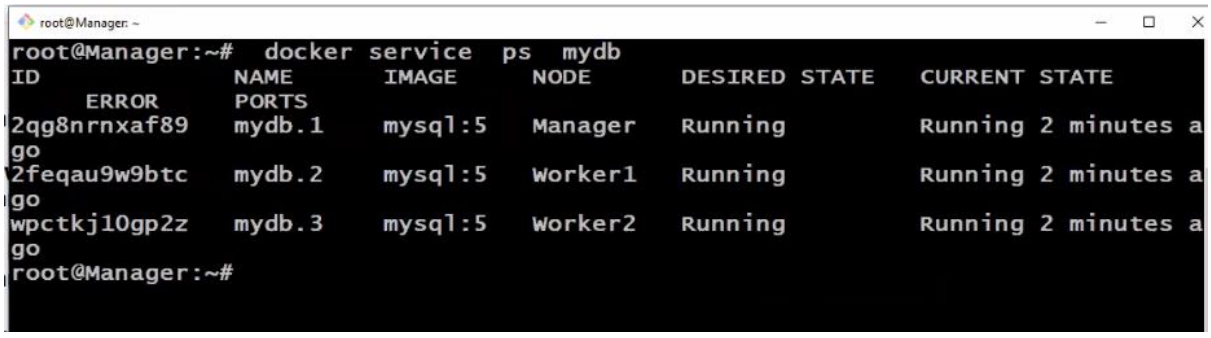
Docs.docker.com –ca se service related commands

++++++++++++++++++++++++++++++++++

If you delete a container, it will create another container.

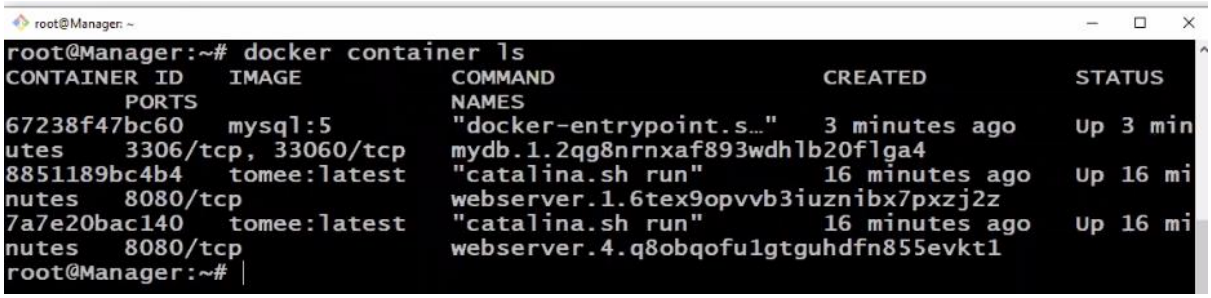
Now,

Manager# docker service ps mydb



We can see one container is running in Manager machine

Manager# docker container ls



(we can see 2 mysql container, 1 tomcat container)

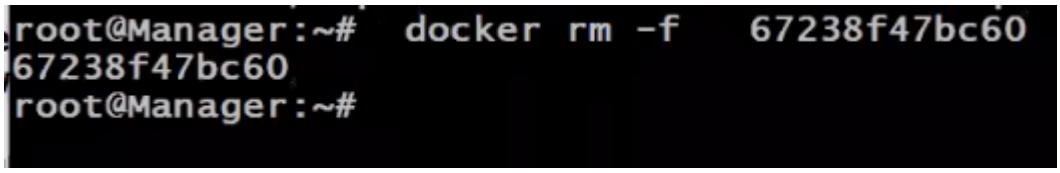
I want to delete the container which is running in manager

Take note of the container_id of mysql

67238f47bc60

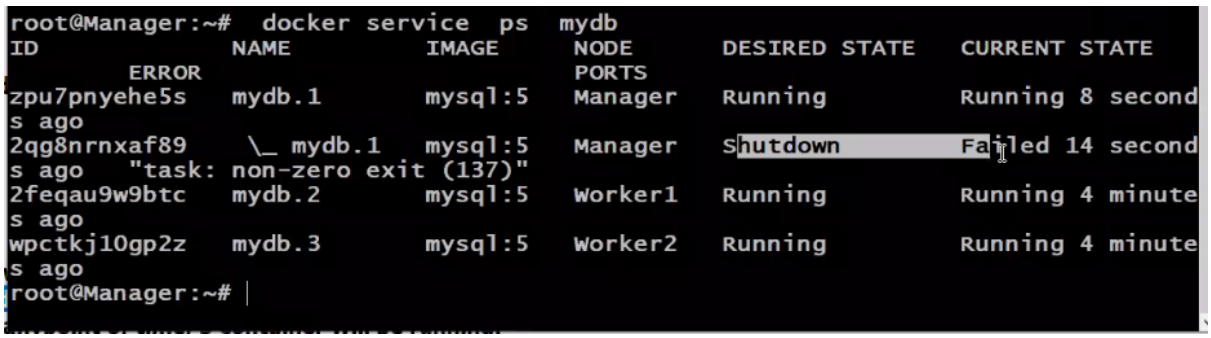
TO delete the container

 # docker rm -f 67238f47bc60



Now lets check the mydb service

 # docker service ps mydb (we can see one service is failed, automatically 2nd service is started)



At anypoint of time, 3 container will be running.

Eventhough deleted one service but see 3 are running because we mentioned replicas:3 in while creating the service mydb

++

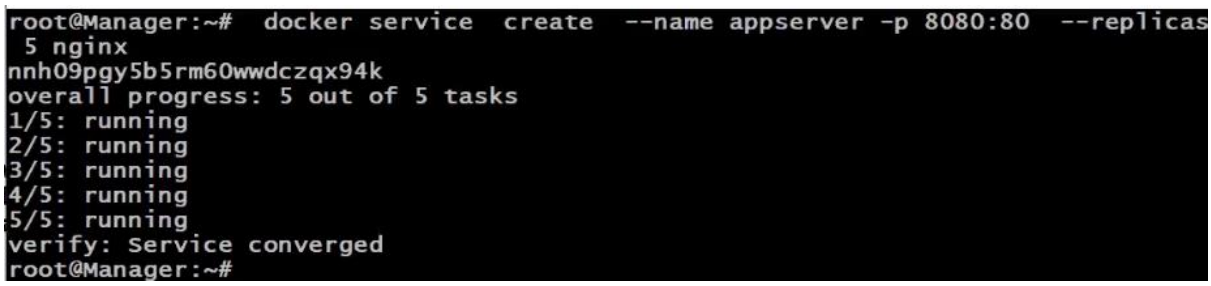
Scaling of containers

When business requirement increases, we should be able to increase the no of replicas.

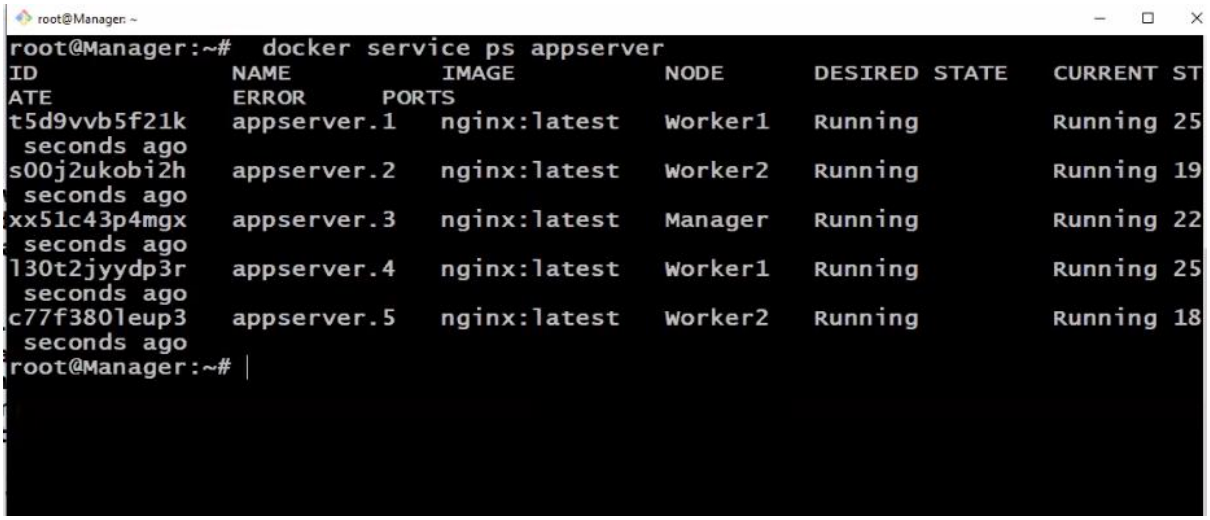
Similarly, we should also be able to decrease the replica count based on business requirement. This scaling should be done without any downtime.

Ex 3: Start nginx with 5 replicas, later scale the services to 10.

 # docker service create --name appserver -p 8080:80 --replicas 5 nginx



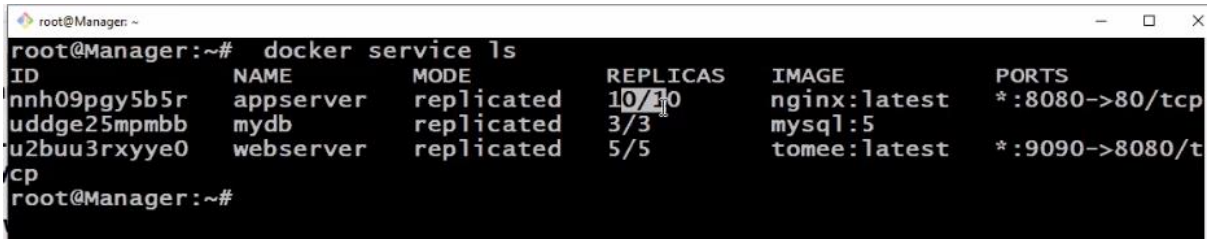
 # docker service ps appserver



Scale up

Command to scale

 # docker service scale appserver=10

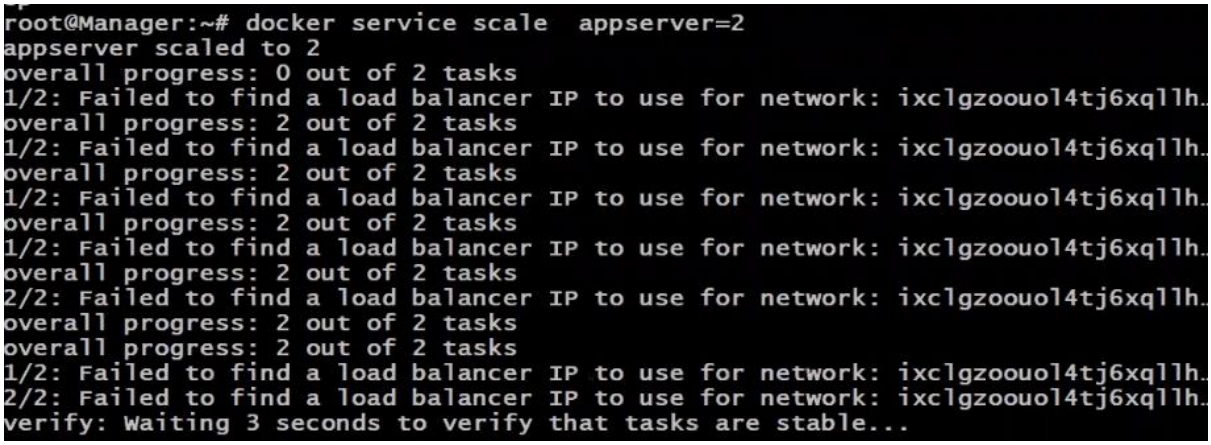


To check

 # docker service ps appserver

Scale down: **Now I want only two containers**

 # docker service scale appserver=2



To check

 # docker service ps appserver

```
root@Manager:~#
root@Manager:~# docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
nnh09pgy5b5r   appserver   replicated   2/2        nginx:latest   *:8080->80/tcp
uddge25mpmbb   mydb        replicated   3/3        mysql:5
u2buu3rxyye0   webserver   replicated   5/5        tomee:latest   *:9090->8080/t
cp
```

++

To remove a node from the docker swarm

Two ways

1) Manager can drain

2) Node can leave

To see the list of nodes

 # docker node ls

```
root@Manager: ~
root@Manager:~#  docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
   ENGINE VERSION
va02k63fhr4x342uqhec9dgpc *   Manager    Ready     Active         Leader
   20.10.7
j98tea7gt2tiovjru45040ui4     Worker1    Ready     Active
   20.10.7
xtxqptswvqjel91cdfgv5082b     Worker2    Ready     Active
   20.10.7
root@Manager:~#
```

 # docker node update --availability drain Worker1

```
root@Manager:~#  docker node update --availability drain  Worker1
Worker1
root@Manager:~#
```

All the container running in Worker1 , will be migrated to Worker2 or manager.

 # docker service ps mydb

 # docker node ls

```
root@Manager:~#  docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
   ENGINE VERSION
va02k63fhr4x342uqhec9dgpc *   Manager    Ready     Active         Leader
   20.10.7
j98tea7gt2tiovjru45O4Oui4     Worker1    Ready     Drain
   20.10.7
xtxqptswvqjel91cdfgv5082b     Worker2    Ready     Active
   20.10.7
root@Manager:~#
```

To add the node

 # docker node update --availability active Worker1

```
root@Manager:~#  docker node update --availability active  Worker1
Worker1
root@Manager:~#
```

 # docker node ls

```
root@Manager:~#  docker node update --availability active  Worker1
Worker1
root@Manager:~#
root@Manager:~#
root@Manager:~# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
   ENGINE VERSION
va02k63fhr4x342uqhec9dgpc *   Manager    Ready     Active         Leader
   20.10.7
j98tea7gt2tiovjru45O4Oui4     Worker1    Ready     Active
   20.10.7
xtxqptswvqjel91cdfgv5082b     Worker2    Ready     Active
   20.10.7
root@Manager:~#
```

2nd Way (Node can leave)

Lets Connect to worker2 from git bash

Worker2# docker swarm leave

```
root@Worker2: ~
root@worker2:~# docker swarm leave
Node left the swarm.
root@worker2:~#
```

++

TO see the list of services

 # docker service ls

```
root@Manager:~# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS
   ENGINE VERSION
va02k63fhr4x342uqhec9dgpc *   Manager    Ready     Active         Leader
   20.10.7
j98tea7gt2tiovjru45040ui4     Worker1    Ready     Active
   20.10.7
xtxqptswvqjel91cdfgv5082b     Worker2    Down      Active
   20.10.7
root@Manager:~#
```

```
root@Manager: ~
root@Manager:~#  docker service ls
ID             NAME         MODE          REPLICAS   IMAGE          PORTS
nnh09pgy5b5r   appserver    replicated    3/2        nginx:latest   *:8080->80/tcp
uddge25mpmbb   mydb         replicated    4/3        mysql:5
u2buu3rxyye0   webserver    replicated    8/5        tomee:latest   *:9090->8080/t
cp
root@Manager:~#
```

TO delete the services

Manager# docker service rm appserver mydb webserver

```
root@Manager: ~
root@Manager:~#  docker service rm appserver mydb webserver
appserver
mydb
webserver
root@Manager:~#
```

Rolling Updates

The services running in docker swarm, can be updated to any other version

without any downtime.

This is perfomed by docker swarm by updating one replica after another. This is called as rolling update.

Ex: Create redis 3 service with 6 replicas. Update from redis 3 to redis 4 version.

docker service create --name myredis --replicas 6 redis:3

To check the replicas

docker service ps myredis

```
root@Manager: ~
root@Manager:~# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE     PORTS
giqgtvjy3rsd   myredis   replicated   3/6        redis:3
root@Manager:~#
```

To update

 # docker service update --image redis:4 myredis

```
root@Manager:~#  docker service update --image redis:4 myredis
myredis
overall progress: 6 out of 6 tasks
1/6: running
2/6: running
3/6: running
4/6: running
5/6: running
6/6: running
verify: Waiting 5 seconds to verify that tasks are stable...
```

docker service ps myredis

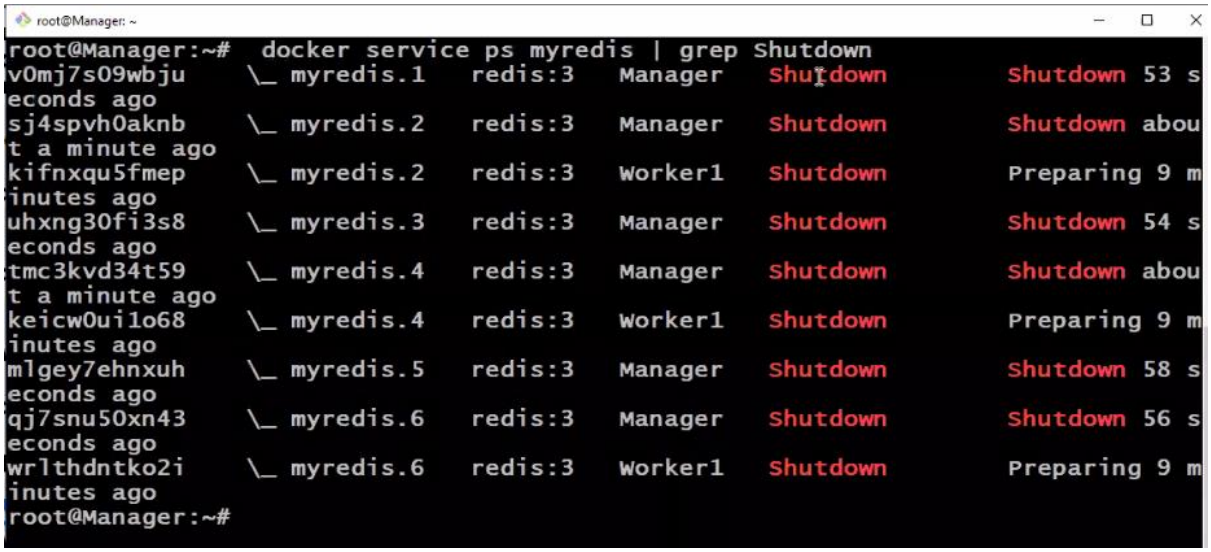
```
root@Manager:~#
root@Manager:~# docker service ps myredis
ID             NAME            IMAGE     NODE      DESIRED STATE
```

```
root@Manager: ~
econds ago
kifnxqu5fmep    \_ myredis.2   redis:3   Worker1   Shutdown        Preparing 8 m
inutes ago
yjjh9bzoaqbr   myredis.3       redis:4   Manager   Running         Running 16 se
conds ago
uhxng30fi3s8    \_ myredis.3   redis:3   Manager   Shutdown        Shutdown 16 s
econds ago
y5mkvak9s48n   myredis.4       redis:4   Manager   Running         Running 24 se
conds ago
tmc3kvd34t59    \_ myredis.4   redis:3   Manager   Shutdown        Shutdown 31 s
econds ago
keicwOui1o68    \_ myredis.4   redis:3   Worker1   Shutdown        Preparing 8 m
inutes ago
uviw8wx7gsi4   myredis.5       redis:4   Manager   Running         Running 20 se
conds ago
mlgey7ehnxuh    \_ myredis.5   redis:3   Manager   Shutdown        Shutdown 20 s
econds ago
l0pdbtcysiu4   myredis.6       redis:4   Manager   Running         Running 18 se
conds ago
qj7snu50xn43    \_ myredis.6   redis:3   Manager   Shutdown        Shutdown 18 s
econds ago
wrlthdntko2i    \_ myredis.6   redis:3   Worker1   Shutdown        Preparing 8 m
inutes ago
root@Manager:~#
```

Older versions shutdown and newer versions running

I want to display running containers not shutdown containers

 # docker service ps myredis | grep Shutdown (We get shutdown container)

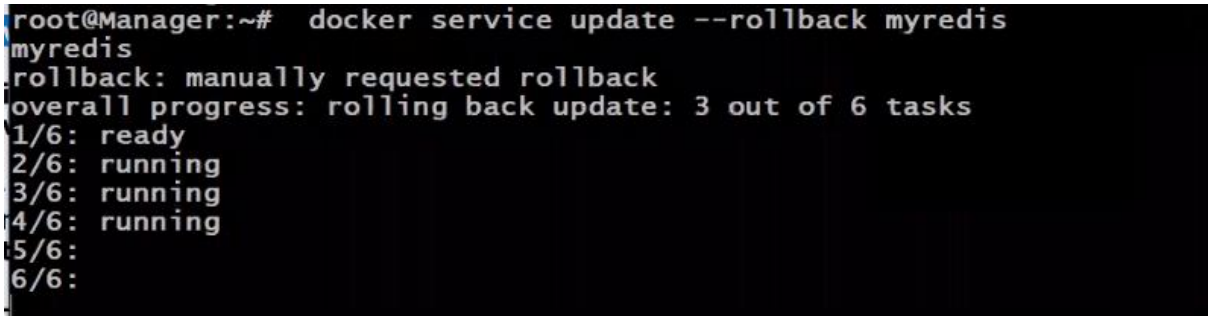


 # docker service ps myredis | grep -v Shutdown (-v used for inverse operation)

++++++++++++++++++++++++++++++++++

Performing rolling rollback , to downgrade to redis:3 version

 # docker service update --rollback myredis



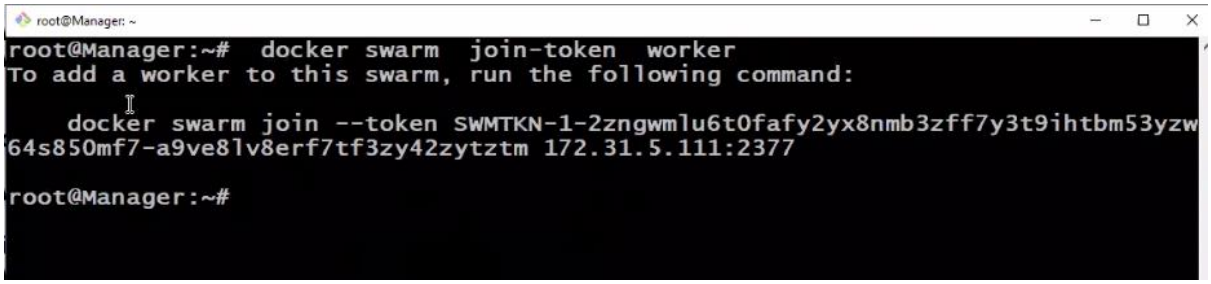
To check redis:3 is running with 6 replicas and other version are shutdown.

 # docker service ps myredis

++++++++++++++++++++++++++++

TO add new nodes, in future, we need to docker swarm join command.

To generate the command



 # docker swarm join-token worker (We will get the command)

docker swarm join --token SWMTKN-1-0etsmfa26vreeytq278q8ohhi73il7j1lpnrzzlowuld1r8yex-9x04pjmiq85jxjzjayzlglh1c 172.31.27.151:2377

++++++++++++++++++++++

To add a new machine as a manager

 # docker swarm join-token manager

docker swarm join --token SWMTKN-1-5wbamgr8x7gxabwtlm1j1i91bm5ilzotgna6bc0edubtwtjxi1-3jmzi67qdn5aawvielkcng2e4 172.31.34.112:2377

++++++++++++++++++++

If there are two managers, one will be leader

If haradware failed in one manager then 2nd manager will come online(high availablity)

 # docker node ls (we can see who is the leader)

Decision of which is machine should be leader is automatic.

If one manager goes down, other manager automatically become leader.

We cant decide which one is on priority

++++++++++++++++++++++++++++++++++++

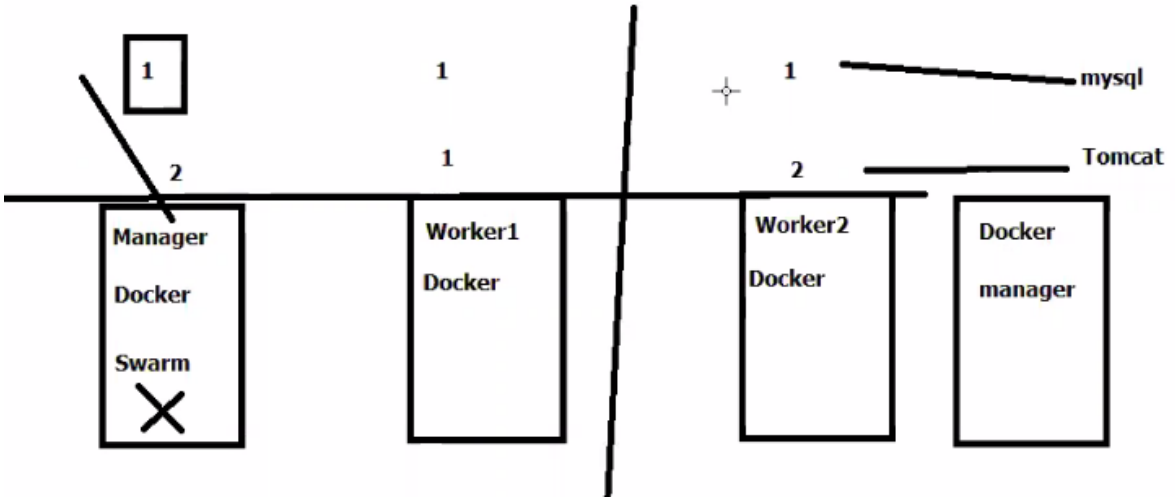
To promote worker1 as a manager node

 # docker node promote Worker1

```
root@Manager:~#  docker node promote Worker1
Node Worker1 promoted to a manager in the swarm.
root@Manager:~#
```

To demote Worker1 and make him back as a worker

 # docker node demote Worker1



++++++++++++----------

• Docker Networking

- **Understading Docker Networking**

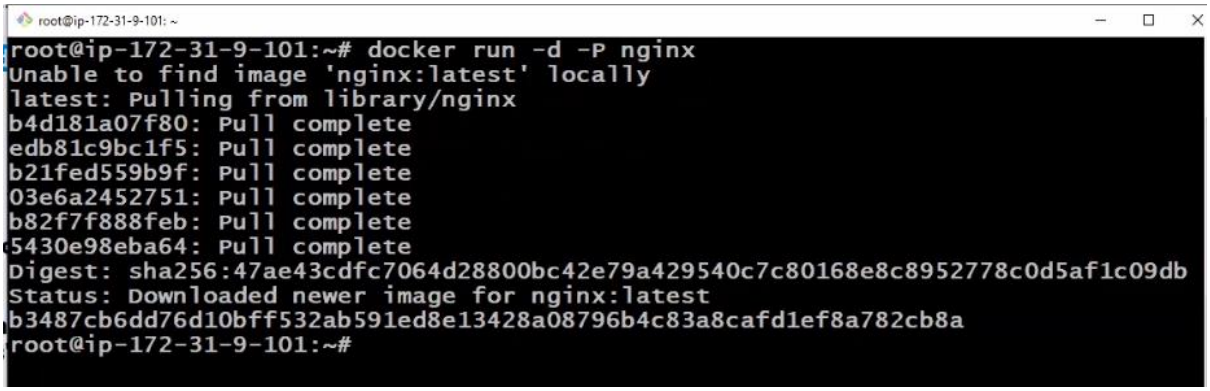
- **Drawback to Docker0 Network**
- **Creating our custom network**
- **Need for creating custom network**
- **Creating container in custom network**

 # curl -fsSL https://get.docker.com -o get-docker.sh

 # sh get-docker.sh

---

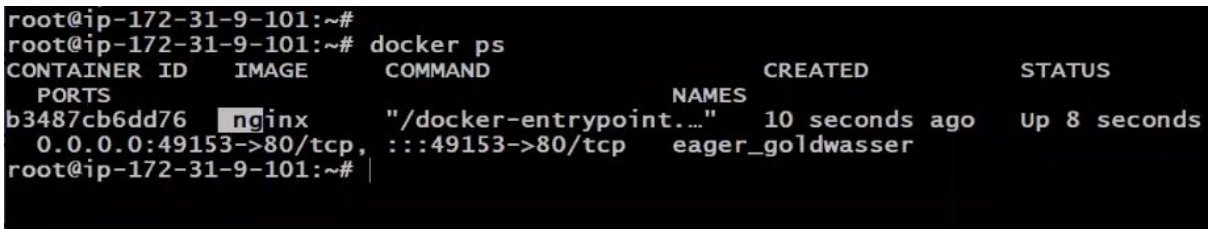
docker run -d -P nginx (--name no given sos creates its with own name)



docker ps

(we can see nginx container running)

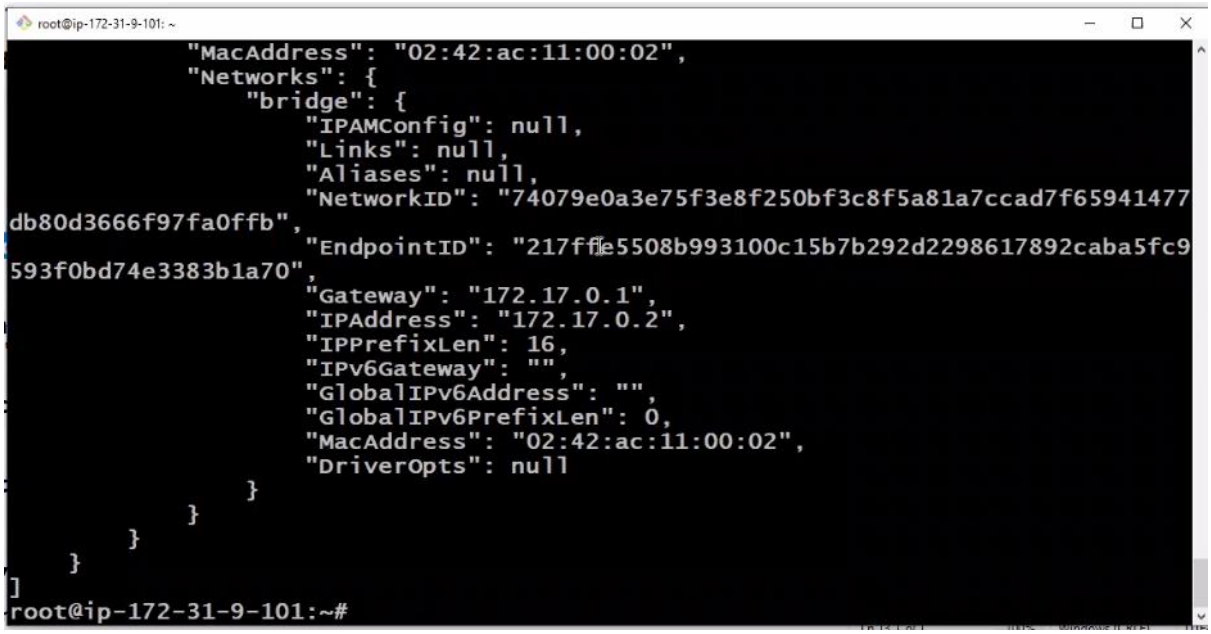
Take note of container id and inspect the container



docker inspect b3487cb6dd76

eager_goldwasser= name of conatiner

Note: we can see the IP address

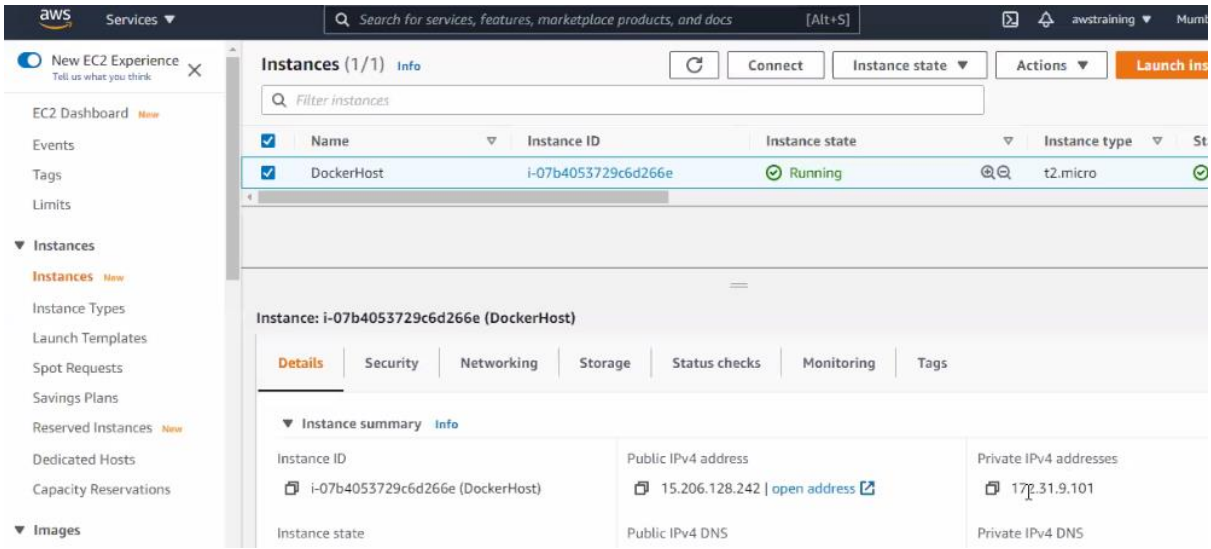


"IPAddress": "172.17.0.2",

Note: Every container will have IP address

Lets take note of docker host IP address - Internal IP - (from aws console)

172.31.7.87



Do you think, they both are the same?

NO

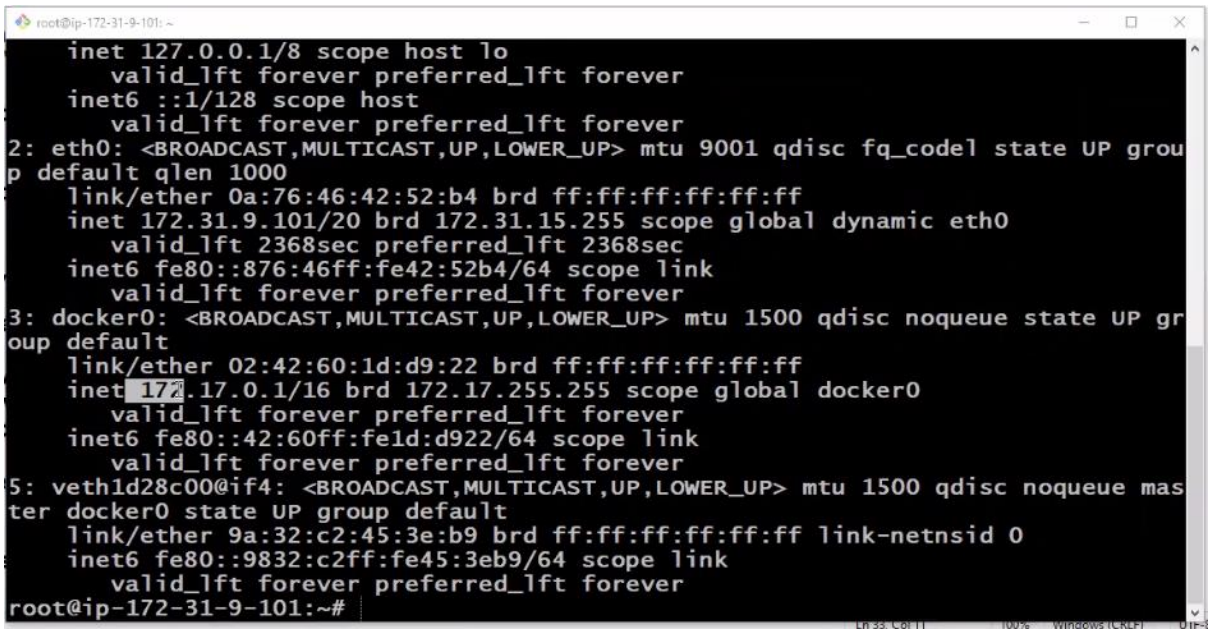
How conatiner get IP address?

Note: When we install docker, docker installs a network by default in the host machine called docker0.

Docker0 network comes with default series - 172.x.x.x

Lets check docker0 network.

 # ip a (we get all networks in the machine)



Observation: we get docker0 also.

Observe the series - 172.17.0.1

Container IP address are taken from docker0 network.

++++++++++++++++++++++++++++++

Lets create another container

 # docker run -d -P redis

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~# docker run -d -P redis
Unable to find image 'redis:latest' locally

latest: Pulling from library/redis
b4d181a07f80: Already exists
86e428f79bcb: Pull complete
ba0d0a025810: Pull complete
ba9292c6f77e: Pull complete
b96c0d1da602: Pull complete
5e4b46455da3: Pull complete
Digest: sha256:7c540ceff53f0522f6b1c264d8142df08316173d103586ddf51ed91ca49deec8
Status: Downloaded newer image for redis:latest
e069dbb52ddde064e15642590dd20f5d7431592e9b33c0d8f8577132889e81a7
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#
```

 # docker ps (we can see two containers are running)

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
   PORTS                                       NAMES
e069dbb52ddd   redis     "docker-entrypoint.s…"   18 seconds ago   Up 17 seconds
   0.0.0.0:49154->6379/tcp, :::49154->6379/tcp  sad_rosalind
b3487cb6dd76   nginx     "/docker-entrypoint.…"   5 minutes ago    Up 5 minutes
   0.0.0.0:49153->80/tcp, :::49153->80/tcp      eager_goldwasser
root@ip-172-31-9-101:~#
```

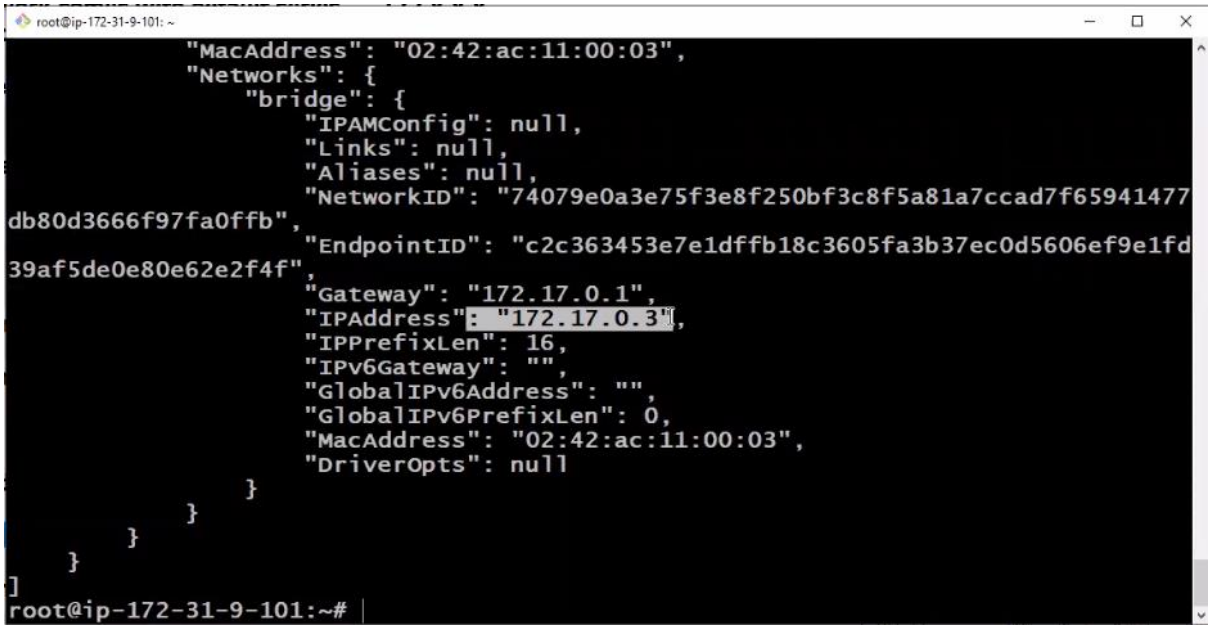
Take note of redis container id

e069dbb52ddd

 # docker inspect e069dbb52ddd

It takes the next ip address from same series

"IPAddress": "172.17.0.3",



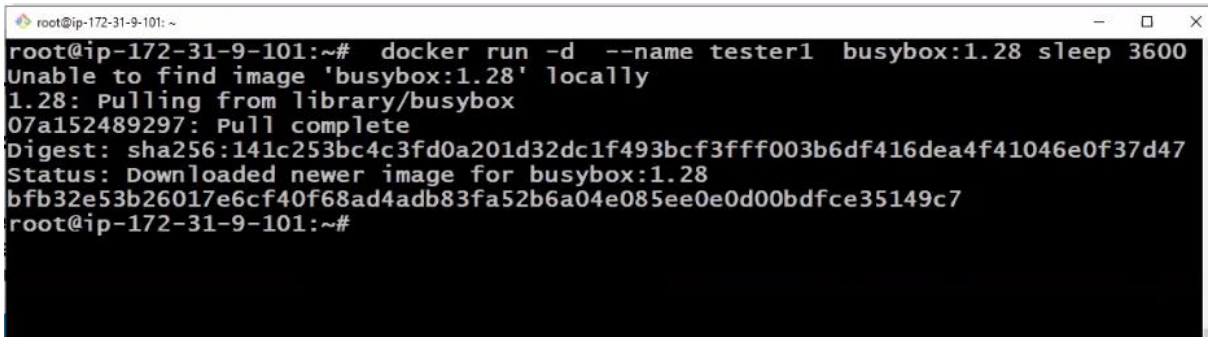
++++++++++++++++++++++++

Now, can these two container communicate with each other?

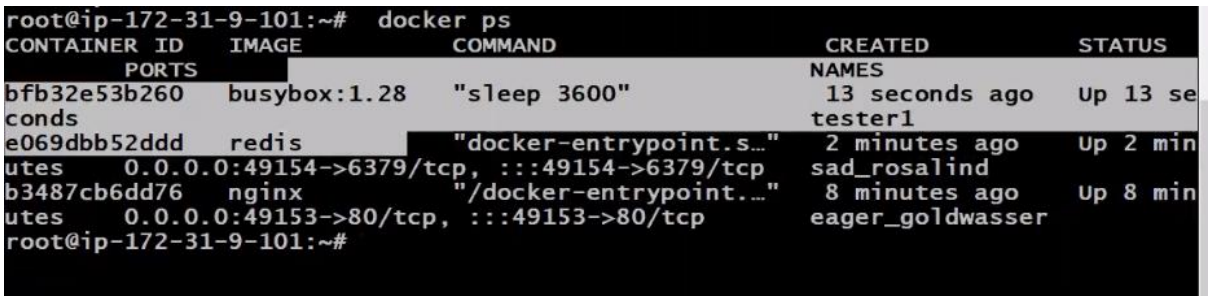
To check whether containers can communicate with each other or not

Lest create busybox container, as it containes ping utility.

 # docker run -d --name tester1 busybox:1.28 sleep 3600



 # docker ps (we can see 3 container running)



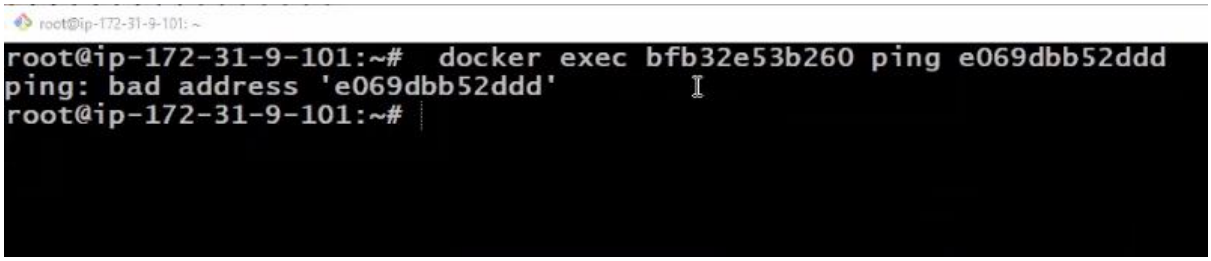
I want to check, if the containers are connected are not?

Take note of container id of busybox

bfb32e53b260 -- container id of busybox

e069dbb52ddd -- container id of redis

 # docker exec bfb32e53b260 ping e069dbb52ddd



Is it pinging?

No

So, with docker0 network we cannot ping with container ids

Lets try to ping with container name

Take note of container name of redis

```
root@ip-172-31-9-101:~#  docker exec bfb32e53b260 ping e069dbb52ddd
ping: bad address 'e069dbb52ddd'
root@ip-172-31-9-101:~# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
       PORTS                                       NAMES
bfb32e53b260   busybox:1.28   "sleep 3600"             2 minutes ago    Up 2 min
utes                                               tester1
e069dbb52ddd   redis          "docker-entrypoint.s…"   4 minutes ago    Up 4 min
utes    0.0.0.0:49154->6379/tcp, :::49154->6379/tcp   sad_rosalind
b3487cb6dd76   nginx          "/docker-entrypoint.…"   10 minutes ago   Up 10 mi
nutes   0.0.0.0:49153->80/tcp, :::49153->80/tcp       eager_goldwasser
root@ip-172-31-9-101:~#
```

Name of redis container--sad_rosalind

 # docker exec bfb32e53b260 ping sad_rosalind

```
root@ip-172-31-9-101:~#  docker exec bfb32e53b260 ping sad_rosalind
ping: bad address 'sad_rosalind'
root@ip-172-31-9-101:~#
```

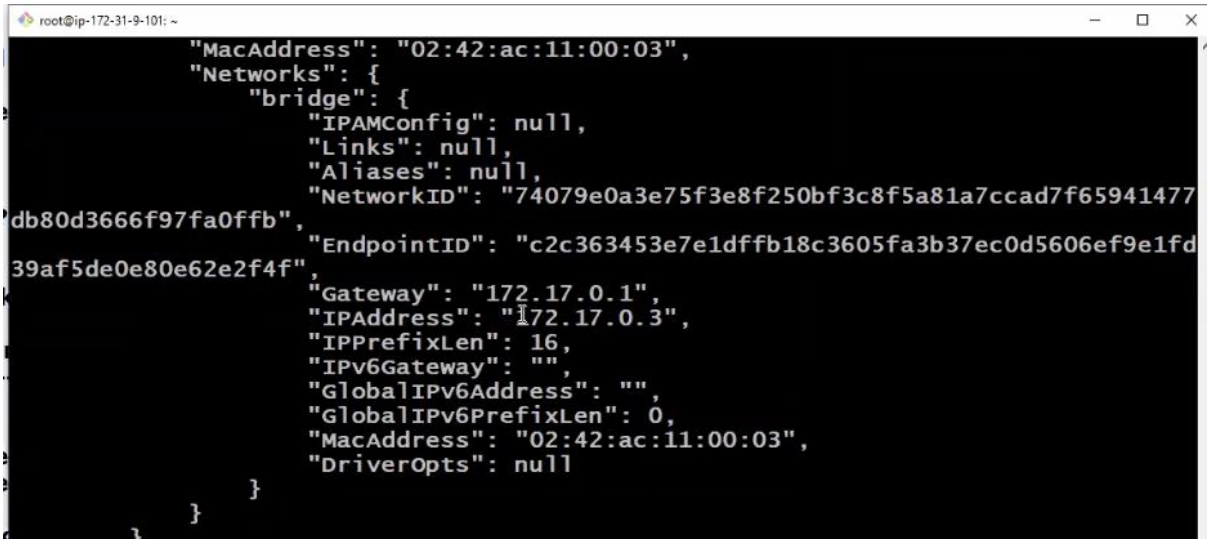
Is it pinging?

No

So, with docker0 network we cannot ping with container name

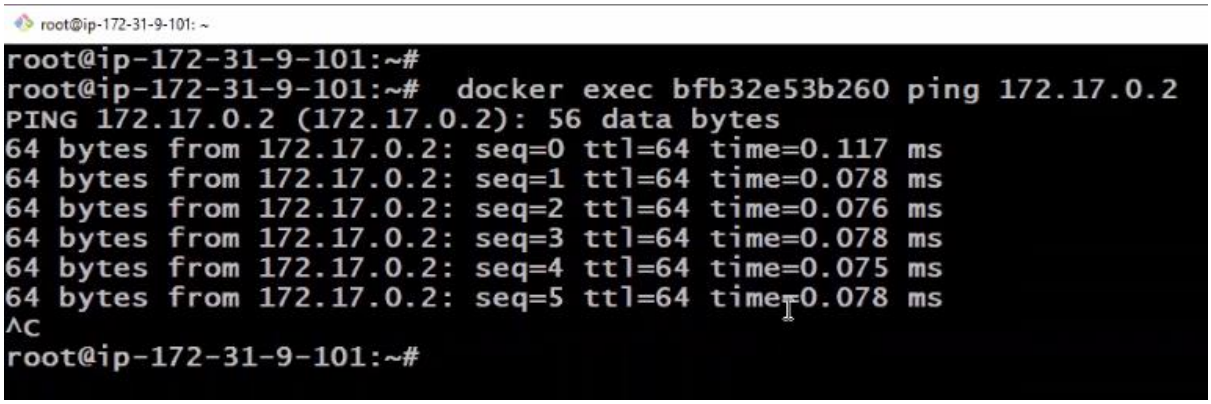
Lets try to ping with IP address

```
root@ip-172-31-9-101:~# docker inspect e069dbb52ddd
[
    {
        "Id": "e069dbb52ddde064e15642590dd20f5d7431592e9b33c0d8f8577132889e81a7"
,
        "Created": "2021-06-28T05:12:34.080770635Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server"
```

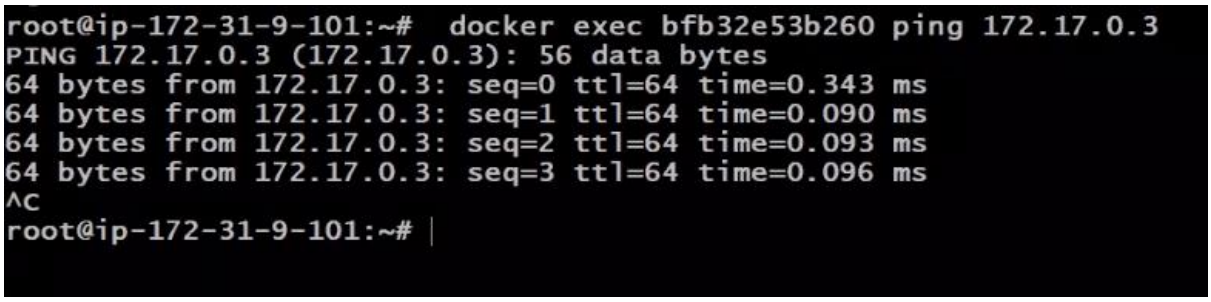


Take note of container IP of redis

 # docker exec bfb32e53b260 ping 172.17.0.2 (172.17.0.2 -- ip address of nginx)



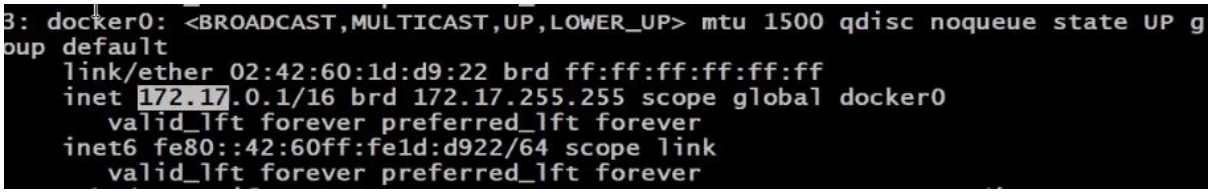
 # docker exec bfb32e53b260 ping 172.17.0.3 (172.17.0.3 -- ip address of redis)



Yes!!! it works.

So, By using docker0 network, one container can communicate to another container by using IP address.

Note: docker0 n/w starts with 172.17.0.1 to



Assume, you have two containers

1st one is application container

2nd one is db container

I want to communicate from application container with DB container.

Generally Application container will have the code to connect to DB

like jdbc:odbc:dbname:dbport/schema/

When both the containers are running on docker0 network.

Instead of dbname , we should mention IP address

like jdbc:odbc:IP:dbport/schema/

Now, Is IP address of container is permanent?

NO

When conainer acquires new IP, we have connection issues.

So,

jdbc:odbc:IP:dbport/schema/ -- is not valid

This is the drawback of docker0

We can use --link option for linking two containers.

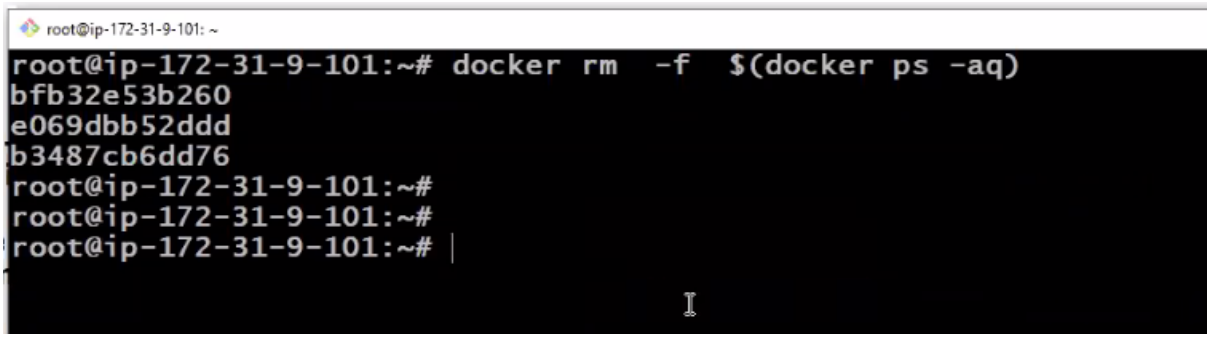
--link option is outdated.

To overcome the Docker0 n/w so At enterprise level We will create our own network.

using Custom networks, we can ping the containers using container name.

Lets remove all the containers

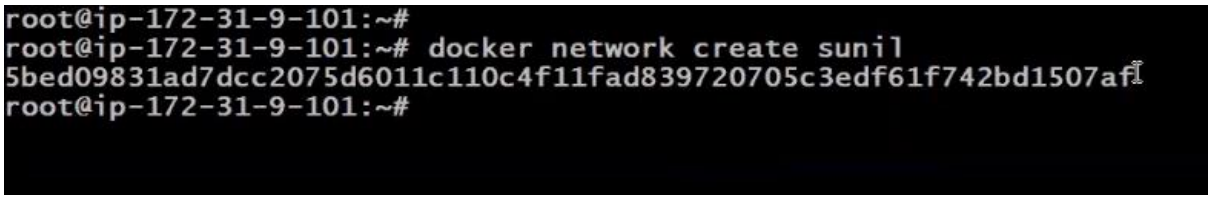
 # docker rm -f $(docker ps -aq)



To create a new network

 # docker network create sunil

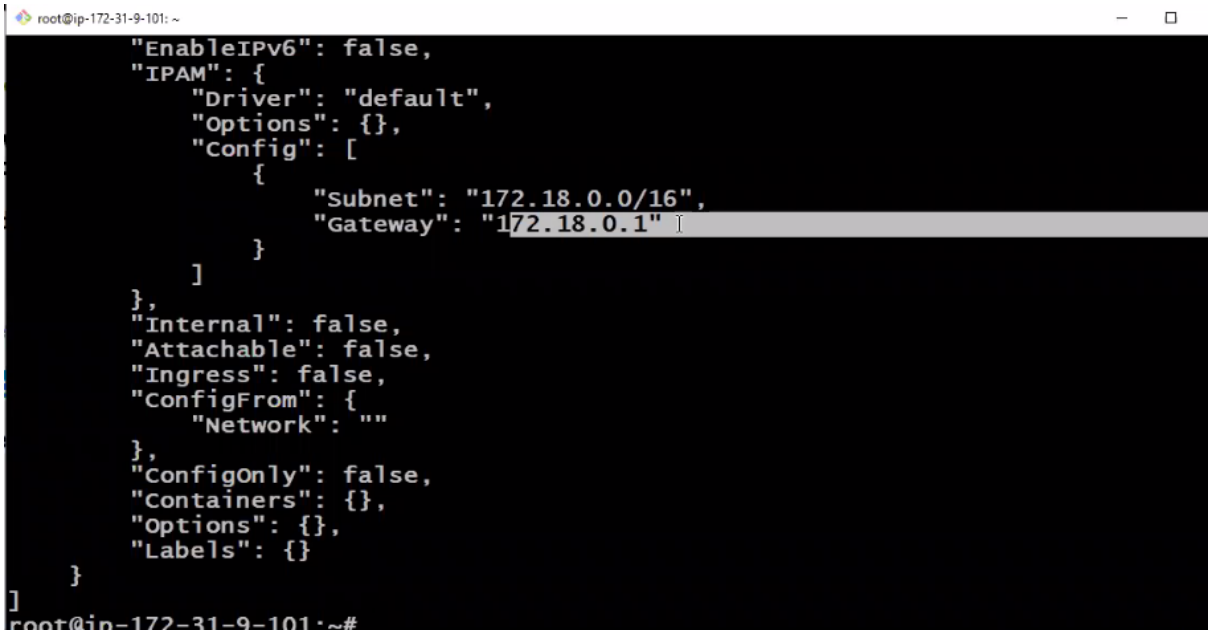
we get network id



5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af

To get the more network information

 # docker inspect sunil



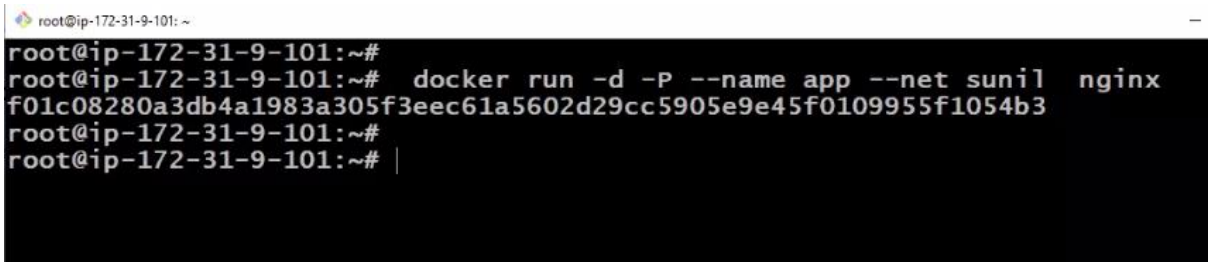
Observe gateway series

"Gateway": "172.18.0.1"

++++++++++++++++

Now, we can create containers in "sunil" network

 # docker run -d -P --name app --net sunil nginx



(--net is used to assign network to the container)

 # docker ps (we can see nginx container is running)

```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
   PORTS                                  NAMES
f01c08280a3d   nginx     "/docker-entrypoint.…"   21 seconds ago   Up 19 seconds
   0.0.0.0:49155->80/tcp, :::49155->80/tcp   app
root@ip-172-31-9-101:~#
```

To know the IP address of the container

 # docker inspect app

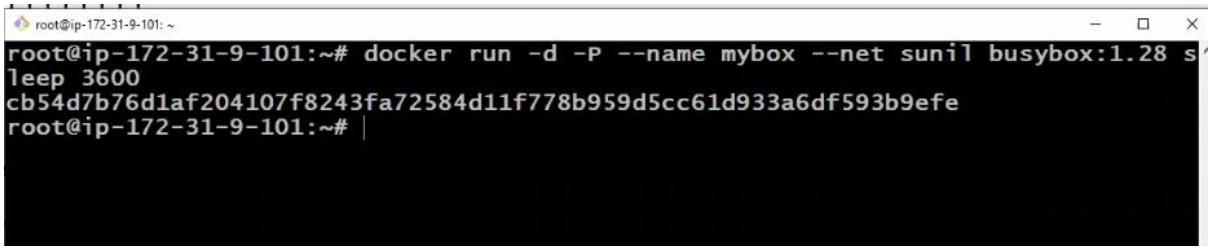
```
root@ip-172-31-9-101: ~
                "sunil": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": [
                        "f01c08280a3d"
                    ],
                    "NetworkID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c
3edf61f742bd1507af",
                    "EndpointID": "1fc716b2185d9f1985e1ff92aaf2b6f5693dea149bb1d
67c74d66678bf487845",
                    "Gateway": "172.18.0.1",
                    "IPAddress": "172.18.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:12:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
root@ip-172-31-9-101:~#
```

"IPAddress": "172.18.0.2",

We get custom network IP series

Lets create one more container in Sunil network

 # docker run -d -P --name mybox --net sunil busybox:1.28 sleep 3600



```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~# docker run -d -P --name mybox --net sunil busybox:1.28 s
leep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~#
```

 # docker ps (we can see two containers)

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~# docker run -d -P --name mybox --net sunil busybox:1.28 s
leep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
      PORTS                                  NAMES
cb54d7b76d1a   busybox:1.28   "sleep 3600"             8 seconds ago   Up 7 seco
nds                                          mybox
f01c08280a3d   nginx          "/docker-entrypoint.…"   2 minutes ago   Up 2 minu
tes   0.0.0.0:49155->80/tcp, :::49155->80/tcp   app
root@ip-172-31-9-101:~#
```

I want to check, if containers can be communicated using name

Lets enter into busybox

 # docker exec -it cb54d7b76d1a sh

```
root@ip-172-31-9-101:~#  docker exec -it cb54d7b76d1a  sh
/ #
/ #
/ # i
```

 # ip a (its a linux command, which will show list of network adapters)

We can see the ip

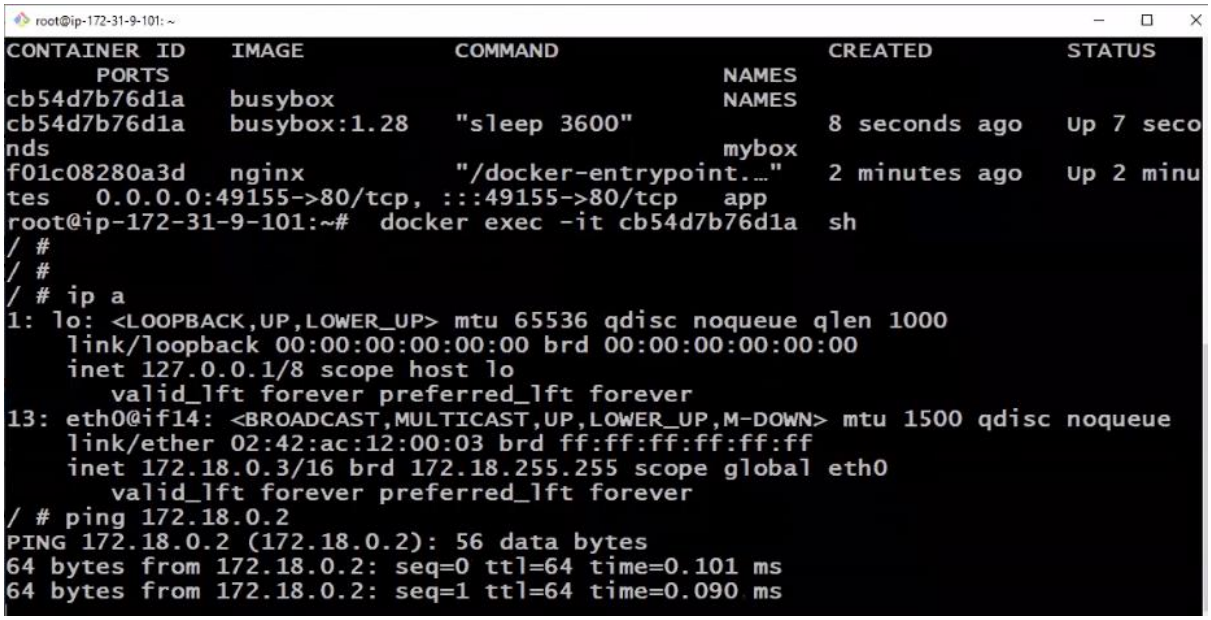
inet 172.18.0.3

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~# docker run -d -P --name mybox --net sunil busybox:1.28 s
leep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
      PORTS                                  NAMES
cb54d7b76d1a   busybox:1.28   "sleep 3600"             8 seconds ago   Up 7 seco
nds                                          mybox
f01c08280a3d   nginx          "/docker-entrypoint.…"   2 minutes ago   Up 2 minu
tes   0.0.0.0:49155->80/tcp, :::49155->80/tcp   app
root@ip-172-31-9-101:~#  docker exec -it cb54d7b76d1a  sh
/ #
/ #
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.3/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

++++++++++++

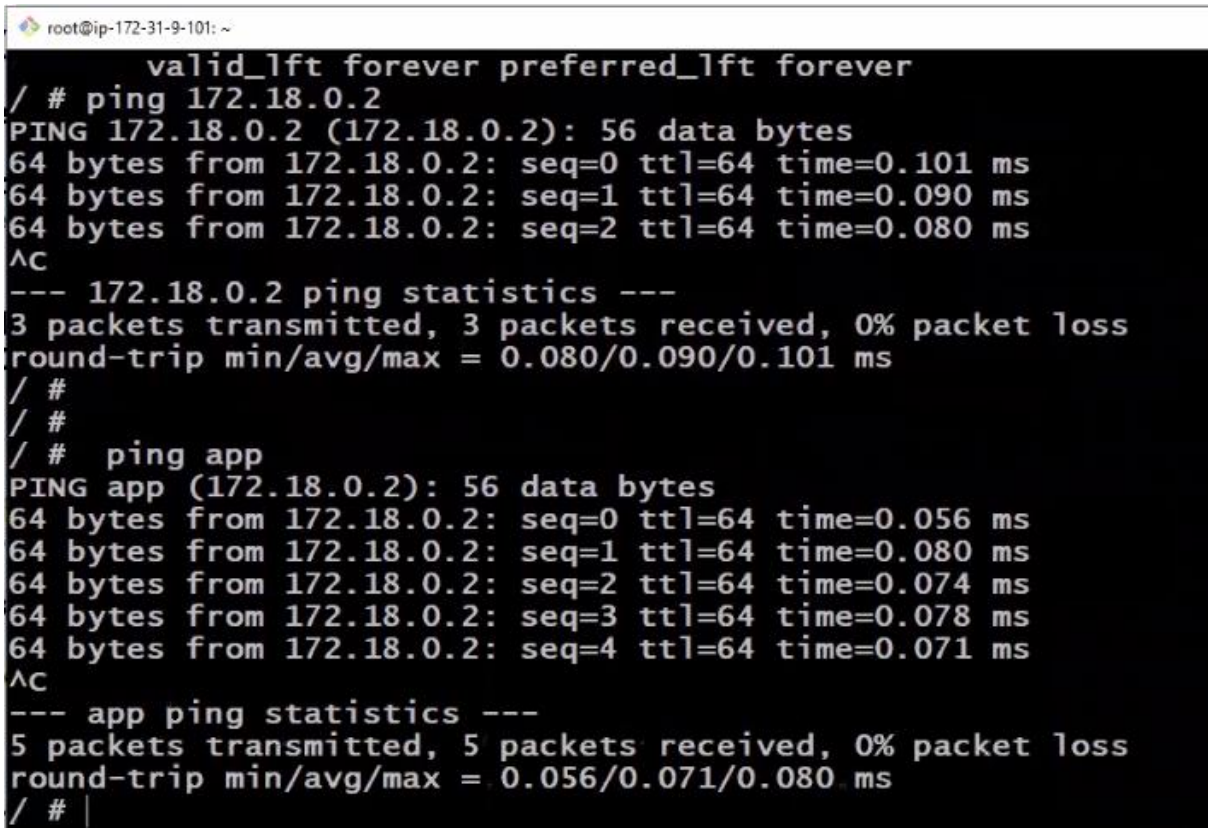
Now from busybox, I want to ping nginx container

/ # ping 172.18.0.2



Yes!! It pings

I want to ping with name



/# ping app

Yes!! It pings!!!!!!!!!!!!!

Similary we can also ping with container ID

We avoid links, we used Custom networks

/# exit (from conatinerr to the docker host)

To see the list of container in a network

 # docker inspect sunil

```
root@ip-172-31-9-101:~#   docker inspect  sunil
[
    {
        "Name": "sunil",
        "Id": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af"
,
        "Created": "2021-06-28T05:28:14.078873399Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
```

```
root@ip-172-31-9-101: ~
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe":
{
                "Name": "mybox",
                "EndpointID": "c5b2f487de4e1647fbab197cd5381c7a26fcc13795ed5a85f
f0f9e0c9771c2c8",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "f01c08280a3db4a1983a305f3eec61a5602d29cc5905e9e45f0109955f1054b3":
{
                "Name": "app",
                "EndpointID": "1fc716b2185d9f1985e1ff92aaf2b6f5693dea149bb1d67c7
4d66678bf487845",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
```

Lets create a redis container again

 # docker run -d -P redis

```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#  docker run -d -P redis
145c4da0603cbd12ff494c1f9ebfce435547666d5b586649490c572a538b1e4d
root@ip-172-31-9-101:~#
```

As we have not mentiond network information, by default it takes docker0 network.

 # docker ps (to see the list of container)

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
        PORTS                                       NAMES
145c4da0603c   redis          "docker-entrypoint.s…"   11 seconds ago   Up 10 se
conds   0.0.0.0:49156->6379/tcp, :::49156->6379/tcp   nervous_tereshkova
cb54d7b76d1a   busybox:1.28   "sleep 3600"             3 minutes ago    Up 3 min
utes                                                 mybox
f01c08280a3d   nginx          "/docker-entrypoint.…"   5 minutes ago    Up 5 min
utes    0.0.0.0:49155->80/tcp, :::49155->80/tcp      app
root@ip-172-31-9-101:~#
```

redis belongs to docker0 network

app belongs to sunil network.

do you think app container will communicate to redis?

They cannot communicate.

 # docker ps

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
        PORTS                                       NAMES
145c4da0603c   redis          "docker-entrypoint.s…"   11 seconds ago   Up 10 se
conds   0.0.0.0:49156->6379/tcp, :::49156->6379/tcp   nervous_tereshkova
cb54d7b76d1a   busybox:1.28   "sleep 3600"             3 minutes ago    Up 3 min
utes                                                 mybox
f01c08280a3d   nginx          "/docker-entrypoint.…"   5 minutes ago    Up 5 min
utes    0.0.0.0:49155->80/tcp, :::49155->80/tcp      app
root@ip-172-31-9-101:~#
```

Take note of redis container id

145c4da0603c

 # docker network connect sunil 145c4da0603c

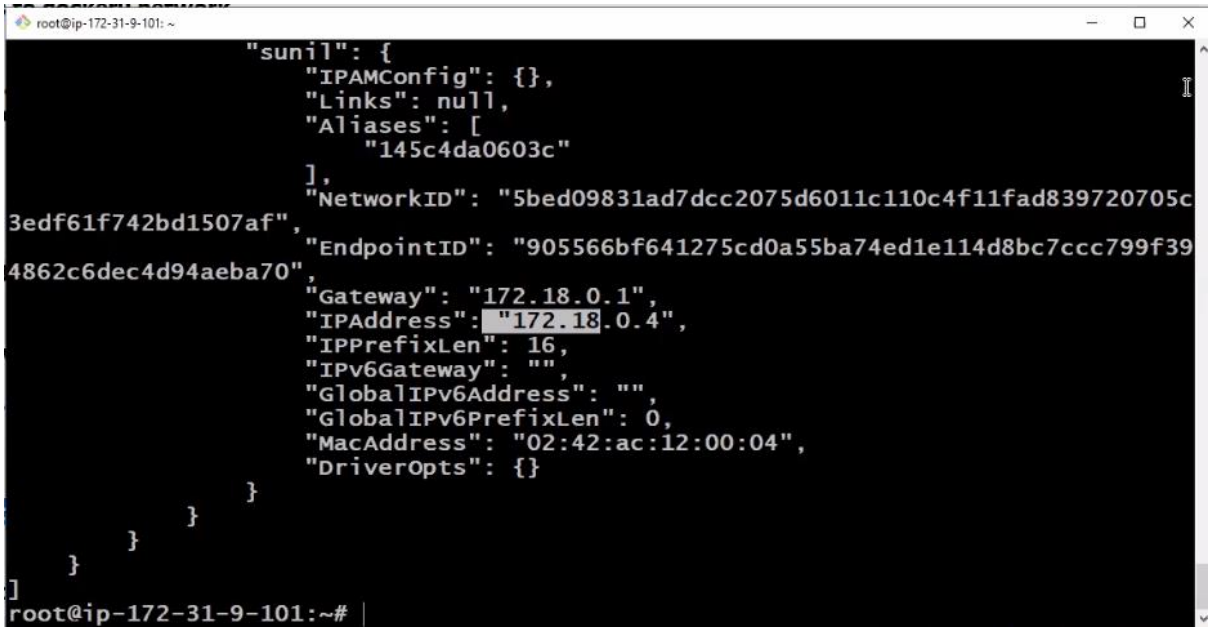
```
root@ip-172-31-9-101:~#  docker network connect sunil 145c4da0603c
```

The above command is used to connect redis container to sunil network

Use only container id is static to connect where ip not able to connect.

Now, inspect the redis container

 # docker inspect 145c4da0603c



Now, its part of two networks

1.sunil

2.docker0

Becz one container can be part of multiple n/ws.

Now, redis can be pinged from app container or busybox container

 # docker exec -it 4f110bebbfb3 sh

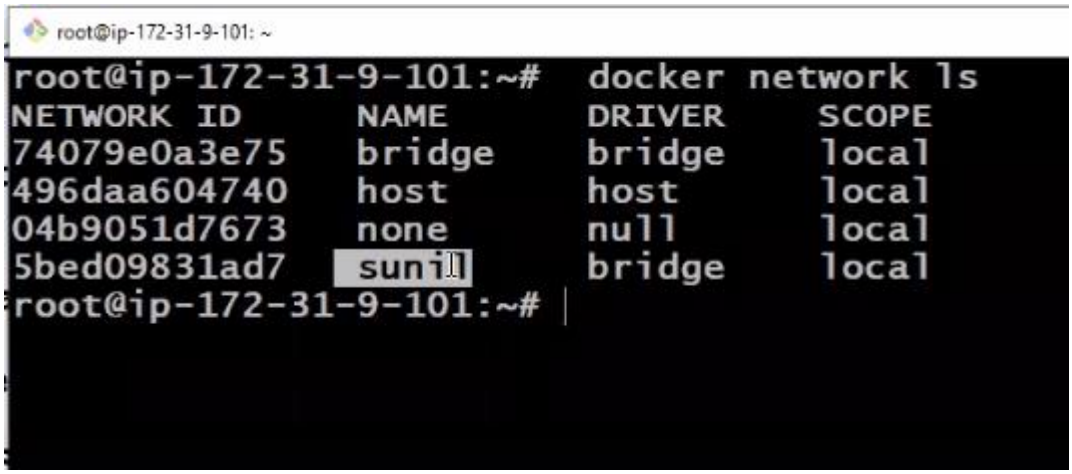
/ # ping 93c81bf83f1c

It works!!!

/ # exit

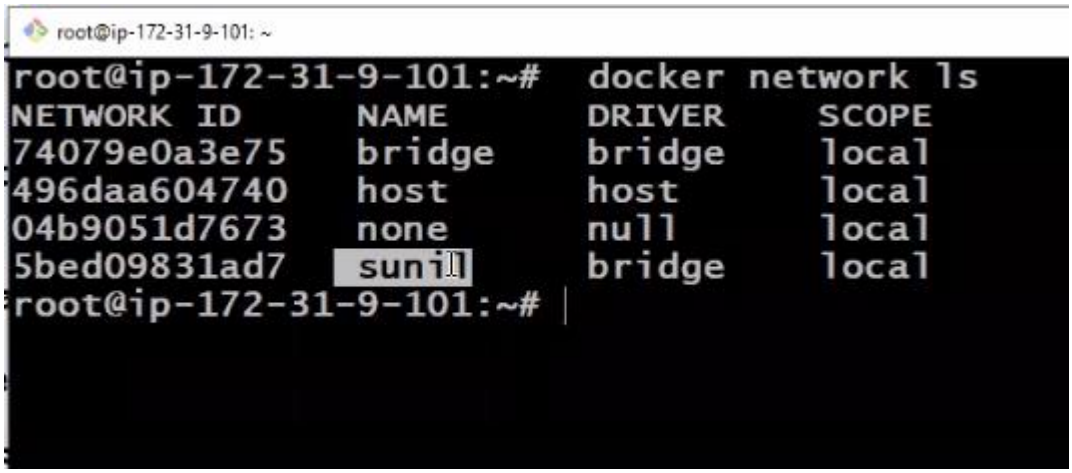
To see the list of networks

 # docker network ls



++++++++++++++++++++++++++++++++

Types of networks in Docker



Actually, they are not networks. They are network drivers.

Docker follows CNM (Container Network Model), By using which container capabilities come to docker by defualt.

There are Following network drivers in-built

----

1) Bridge

2) Host

3) None

4) Overlay

Last session, we have experienced two types of networks.

1. Docker0

2. sunil (Custom network)

By default docker0 is of type bridge network.

In bridge networks, container on one machine can communicate to another container, in the same machine (dockerhost)

By default type is bridge.

 # docker network ls

We can see "sunil" network is of type bridge.

Bridge network is confined to one machine only.

Host Network

Assume, I am running a database.

I want to use machine IP(ec2) as the container IP

It is possible by using Host network.

We already have host network available.

 # docker network ls

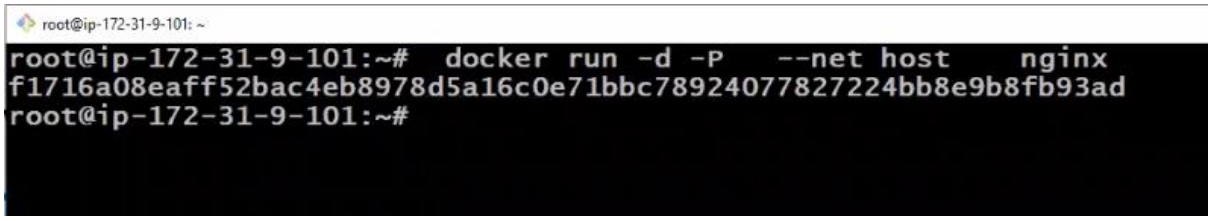
We can see one host network

Delete all conatiners.

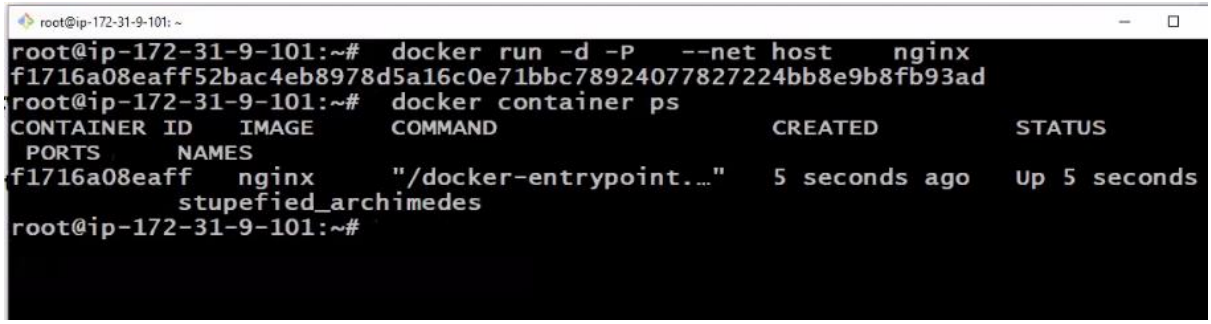
```
root@ip-172-31-9-101:~#  docker rm  -f  $(docker ps -aq)
145c4da0603c
cb54d7b76d1a
f01c08280a3d
root@ip-172-31-9-101:~#
```

Nginx Link to host n/w

 # docker run -d -P --net host nginx



 # docker container ps



Observe: There is no port no

Eventhough we have mentioned -P, is has not published port.

We can access directly using IP without port, to connect to the container.

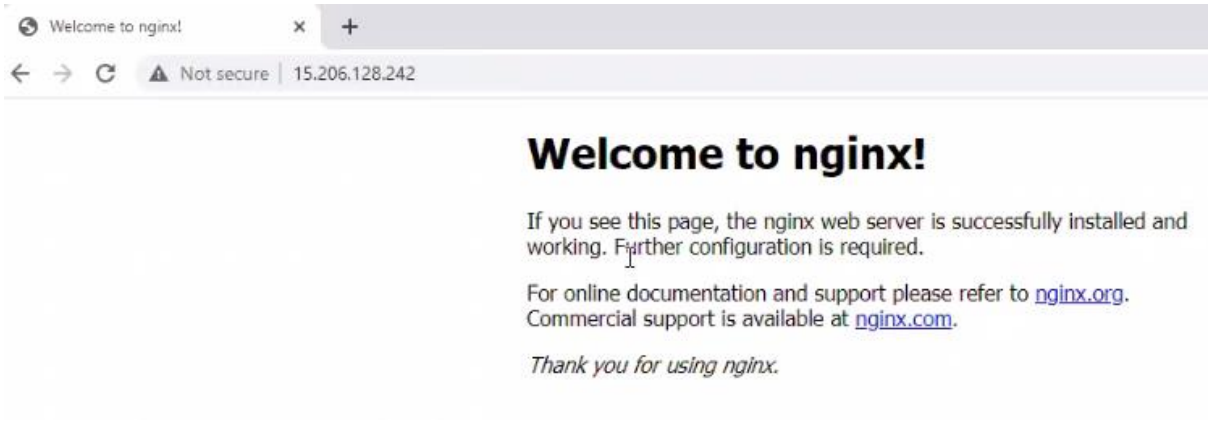
Take the public IP of the Dockerhost and access using default port.

13.233.156.196:80

13.233.149.155:80

We get the nginx page

Ok. fine.



But, can we create the same container again?

No!

 # docker run -d -P --net host nginx

```
root@ip-172-31-9-101:~#  docker run -d -P --net host  nginx
481345d3c7e60037176779447acf11f761a28d31cafa1b208169a1a5ce8ed36a
root@ip-172-31-9-101:~#
```

As, 80 port number is already published

When we create another container,It will not be in running.

 # docker ps

We can see only one container.

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED
          PORTS     NAMES
f1716a08eaff   nginx     "/docker-entrypoint.…"   About a minu
a minute             stupefied_archimedes
root@ip-172-31-9-101:~#
```

 # docker ps –a

```
root@ip-172-31-9-101:~#  docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
               PORTS     NAMES
481345d3c7e6   nginx     "/docker-entrypoint.…"   23 seconds ago   Exited (1) 2
 seconds ago             beautiful_mcnulty
f1716a08eaff   nginx     "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes
                         stupefied_archimedes
root@ip-172-31-9-101:~#
```

We can see the latest container is exited.

To know, Why the container is exited?

Take note of the container id, which is exited

f70267b40b4d

 # docker logs f70267b40b4d

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker logs 481345d3c7e6
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perf
rm configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-defa
lt.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.
/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/con
.d/default.conf
```

```
root@ip-172-31-9-101: ~
2021/06/28 05:49:43 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address alrea
y in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2021/06/28 05:49:43 [emerg] 1#1: bind() to [::]:80 failed (98: Address already
n use)
nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
2021/06/28 05:49:43 [notice] 1#1: try again to bind() after 500ms
2021/06/28 05:49:43 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address alrea
y in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2021/06/28 05:49:43 [emerg] 1#1: bind() to [::]:80 failed (98: Address already
n use)
nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
2021/06/28 05:49:43 [notice] 1#1: try again to bind() after 500ms
2021/06/28 05:49:43 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address alrea
y in use)
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
2021/06/28 05:49:43 [emerg] 1#1: bind() to [::]:80 failed (98: Address already
n use)
nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
2021/06/28 05:49:43 [notice] 1#1: try again to bind() after 500ms
2021/06/28 05:49:43 [emerg] 1#1: still could not bind()
nginx: [emerg] still could not bind()
root@ip-172-31-9-101:~#
```

We can see clearly as " Address already in use"

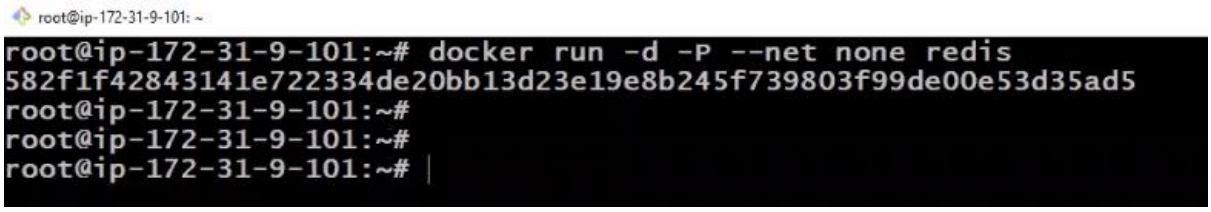
So, with host network, one type of image we can use only once.

None Network

Refers to no network.

If we do not want the container to get exposed to the world.

 # docker run -d -P --net none redis



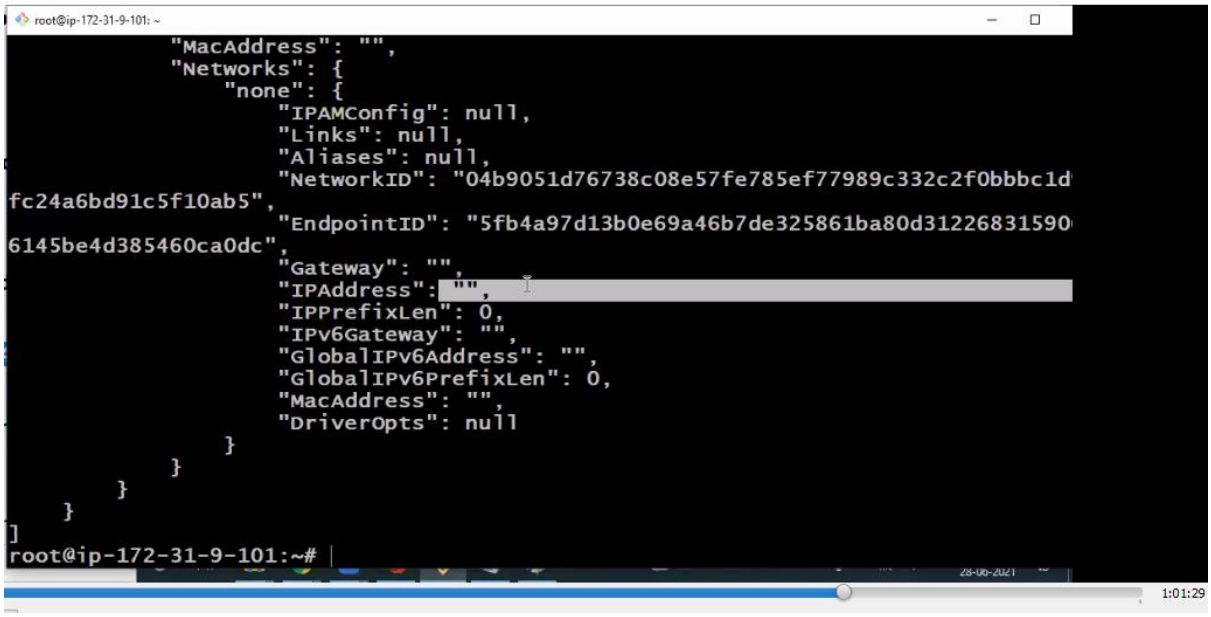
 # docker ps

Take note of container id

fadebe0ad3cf

 # docker inspect fadebe0ad3cf

We do not see any IP for the container



"IPAddress": ""

When none network is used?

When we use orchestration tools like Kubernetes or openshift.

If you want orchestration tools to take of networking, in such case we do not want docker networking.In such case we use none network of docker.

++++++++++++++++++++++++++

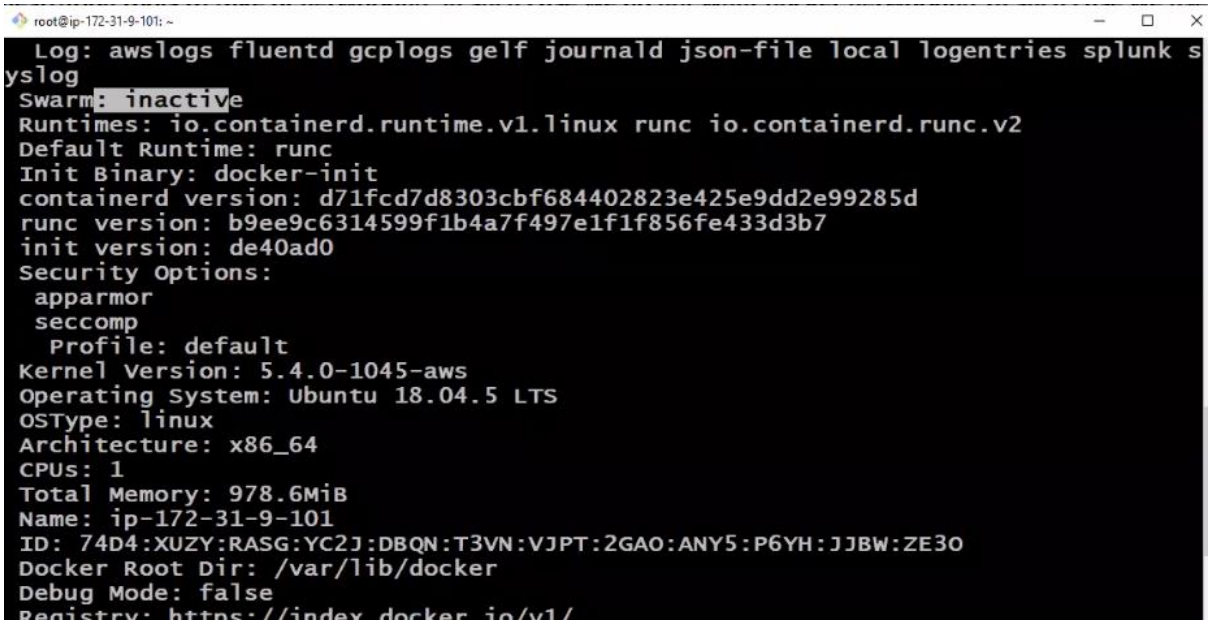
In realtime, we know, container will be running on multiple machines (docker swarm).

When container present in one machine, want to communicate to container in another machine

 # docker info

We can see - Swarm in inactive

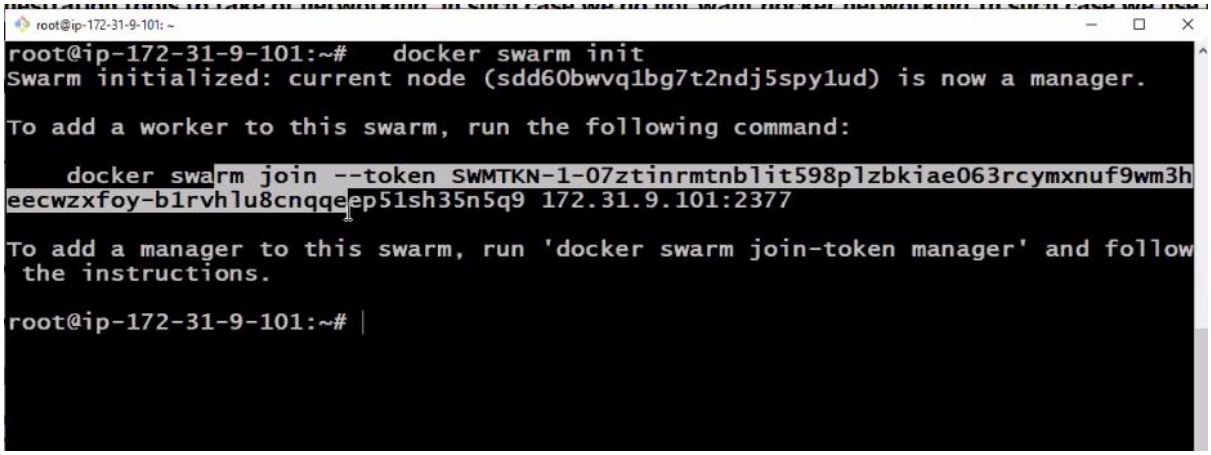
Swarm: inactive



I want to initialize swarm

 # docker swarm init

Now, lets see the list of networks



 # docker network ls

Observer, a new network is created of type "overlay"

Name of the network is ingress and scope swarm

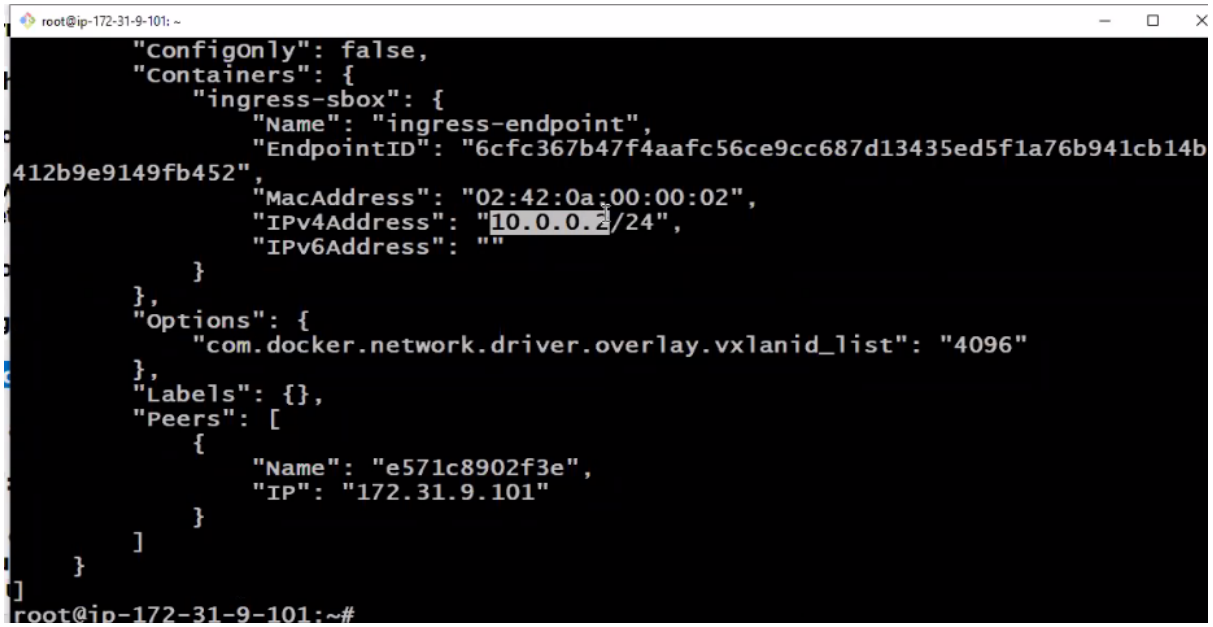
One machine container can communicate to another machine container using overlay network.

Lets inspect ingress network.

 # docker inspect ingress

Observe the IP address, Lets take a note of it.

"IPv4Address": "10.0.0.2

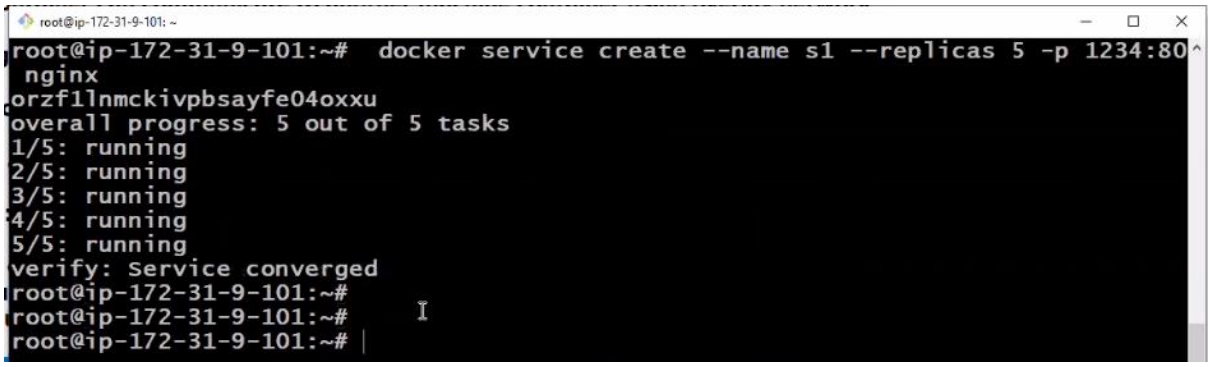


Now, when we create container using service concept,

The container uses overlay network.

Docker swarm uses overlay network.

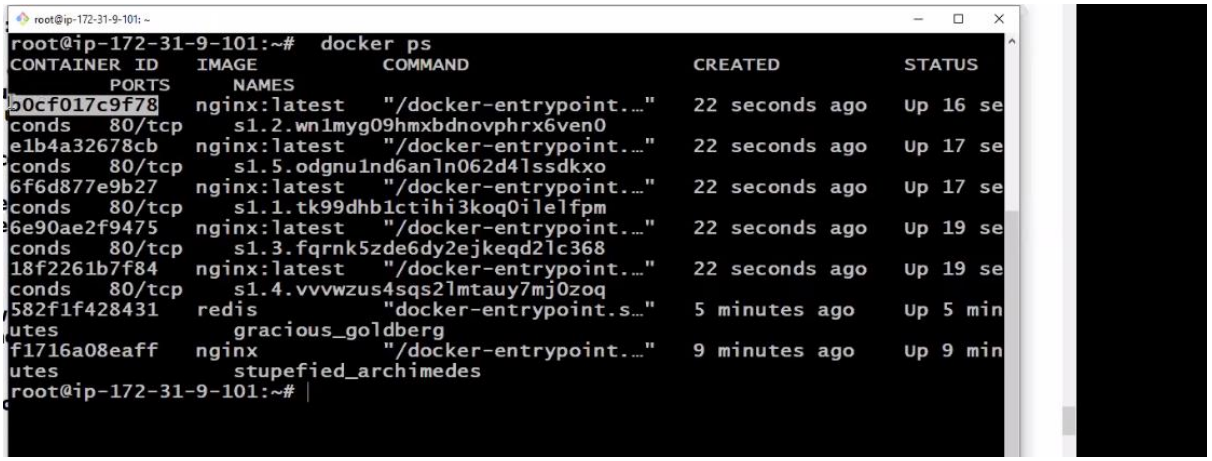
 # docker service create --name s1 --replicas 5 -p 1234:80 nginx



As we have one machine right now.

All the container will be running in the same machine (Manager)

 # docker ps



I want to know the IP addres of the 1st container.

Take note of the any (1st) container ID

eaa1e645ae48

 # docker inspect 96631cf17237

Observer the IP address is

"IPAddress": "10.0.0.5"

It has taken overlay network series. It belongs to ingress/overlay n/w

So, swarm uses existing overlay network for communication between containers on different machines.

++++++++++++++++++++++++++

Can we create our own overlay network?

Yes!!

Lets create

 # docker network create ol1 --driver overlay

```
root@ip-172-31-9-101:~#  docker network create ol1 --driver overlay
h9w8lbraafdnacpom6a71y7z9
root@ip-172-31-9-101:~#
```

 # docker network ls (to see the list of networks)

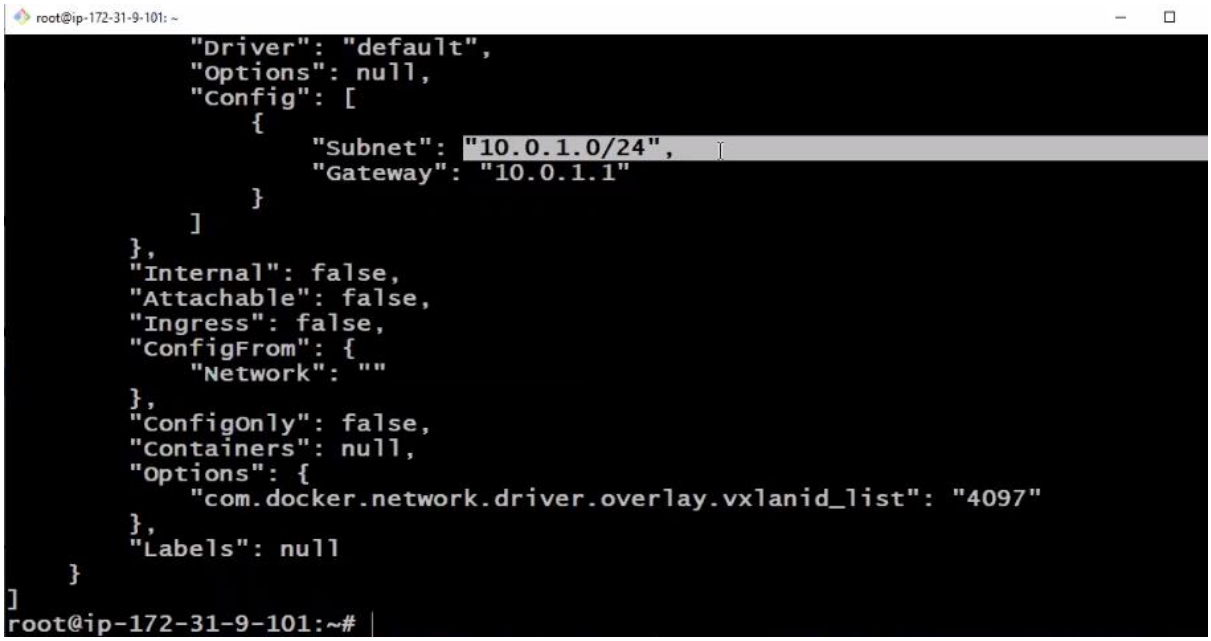
```
root@ip-172-31-9-101:~#  docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
74079e0a3e75   bridge            bridge    local
55536bc048aa   docker_gwbridge   bridge    local
496daa604740   host              host      local
145uvi3idu94   ingress           overlay   swarm
04b9051d7673   none              null      local
h9w8lbraafdn   ol1               overlay   swarm
5bed09831ad7   sunil             bridge    local
root@ip-172-31-9-101:~#
```

We can see ol1 network is created of type overlay

 # docker inspect ol1 (To know the series it is taking)

"Subnet": 10.0.1.0

It has taken 10 series



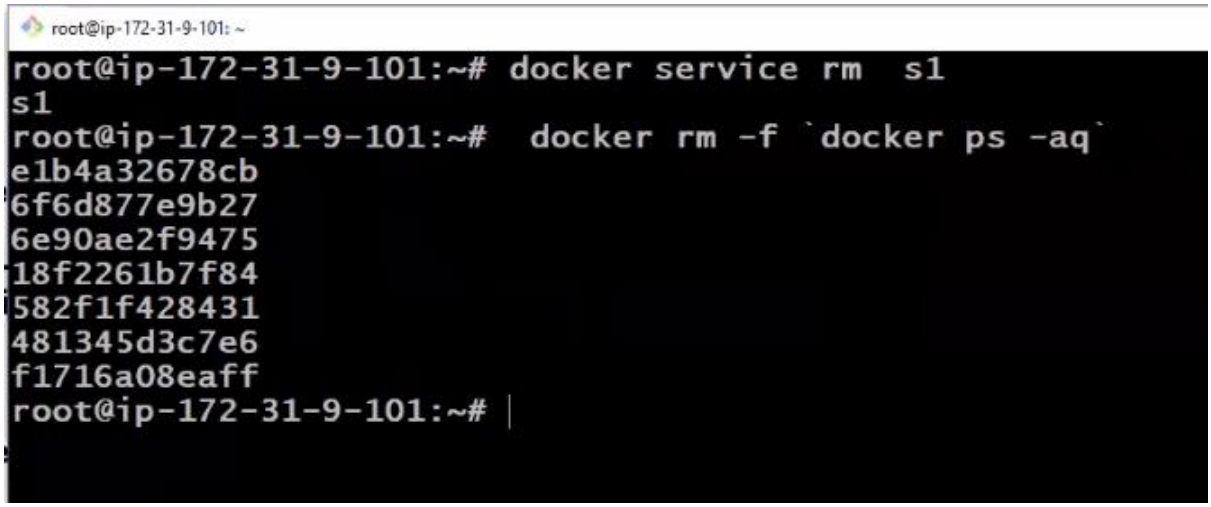
++++++++++++++++++++++++

Lets delete the existing service

 # docker service rm s1

Lets remove all the existing containers

 # docker rm -f `docker ps -aq`



Lets create our own service on ol1 network

 # docker service create --name mynginx --replicas 5 --network ol1 -p 1224:80   nginx

```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#  docker service create --name mynginx --replicas 5 --net
work ol1 -p 1224:80   nginx
ouu7c8j3qy2xOld4pmon18nwb
overall progress: 0 out of 5 tasks
1/5: starting
2/5: starting
3/5: starting
4/5: preparing
5/5: starting
```

 # docker ps

Take note of the 1st container id

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~#  docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS
        PORTS     NAMES
35b7d1a30038   nginx:latest   "/docker-entrypoint.…"   21 seconds ago   Up 18 se
conds   80/tcp    mynginx.3.tp8sgqh6gtfx5z8sqOhjlc1a1
8c7071da6e99   nginx:latest   "/docker-entrypoint.…"   22 seconds ago   Up 18 se
conds   80/tcp    mynginx.2.lz497izbp6kOopfys7ved48ej
2c7517937c5a   nginx:latest   "/docker-entrypoint.…"   22 seconds ago   Up 17 se
conds   80/tcp    mynginx.1.ydur1alaj4nkwiuq7sickrutx
7c674846840f   nginx:latest   "/docker-entrypoint.…"   22 seconds ago   Up 18 se
conds   80/tcp    mynginx.4.rt54sc5iohqr3qcyrf78e51wh
1c34b0d9526f   nginx:latest   "/docker-entrypoint.…"   22 seconds ago   Up 17 se
conds   80/tcp    mynginx.5.4o7yai5mkmhkidcy31sxmqb57
root@ip-172-31-9-101:~#
```

814d5db02f47

 # docker inspect 814d5db02f47

```
root@ip-172-31-9-101: ~
                    "IPAMConfig": {
                        "IPv4Address": "10.0.1.5"
                    },
                    "Links": null,
                    "Aliases": [
                        "35b7d1a30038"
                    ],
                    "NetworkID": "h9w8lbraafdnacpom6a71y7z9",
                    "EndpointID": "6f5a6e024316fcb41e8375a3c2d8a222b575721e1c767
0e1555f077f798f9aac",
                    "Gateway": "",
                    "IPAddress": "10.0.1.5",
                    "IPPrefixLen": 24,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:0a:00:01:05",
                    "DriverOpts": null
                }
            }
        }
    }
]
root@ip-172-31-9-101:~#
```

We can see the network as ol1. And the Ip address series.

"ol1": {

"IPAMConfig": {

"IPv4Address": "10.0.1.7"

We generally used bridge and overlay networks.

++++++++++++++++++++++++++++++++++++