

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.

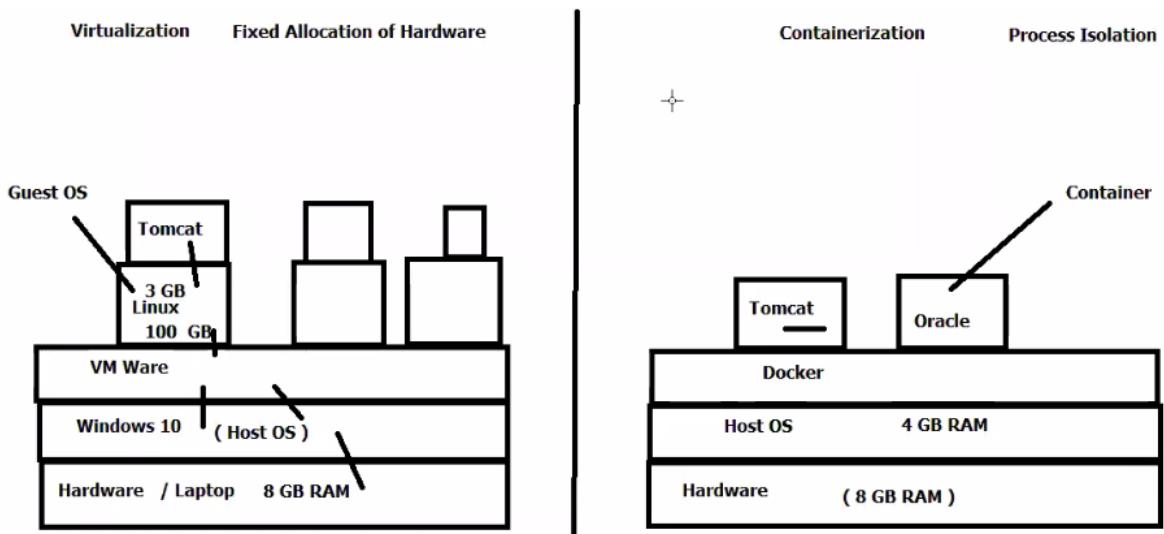
A Docker container enables rapid deployment with minimum run-time requirements. It also ensures better management and simplified portability.

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 containers like tomcat.



+++++

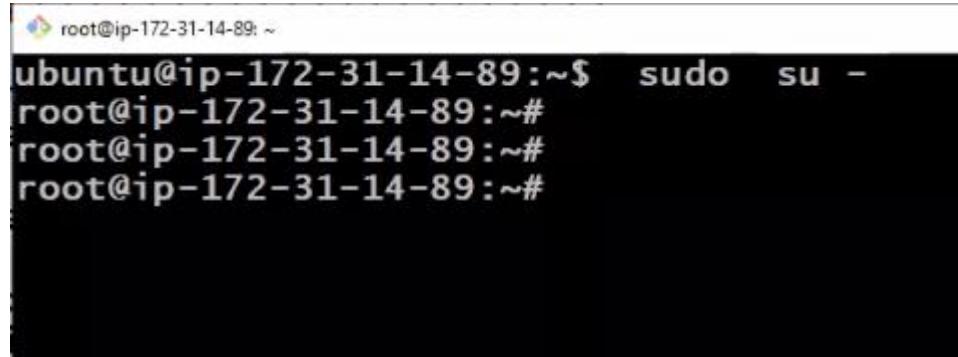
Create Ubuntu Machine on AWS—Host os

All Traffic - anywhere

Connect using git bash

Go to Root Account

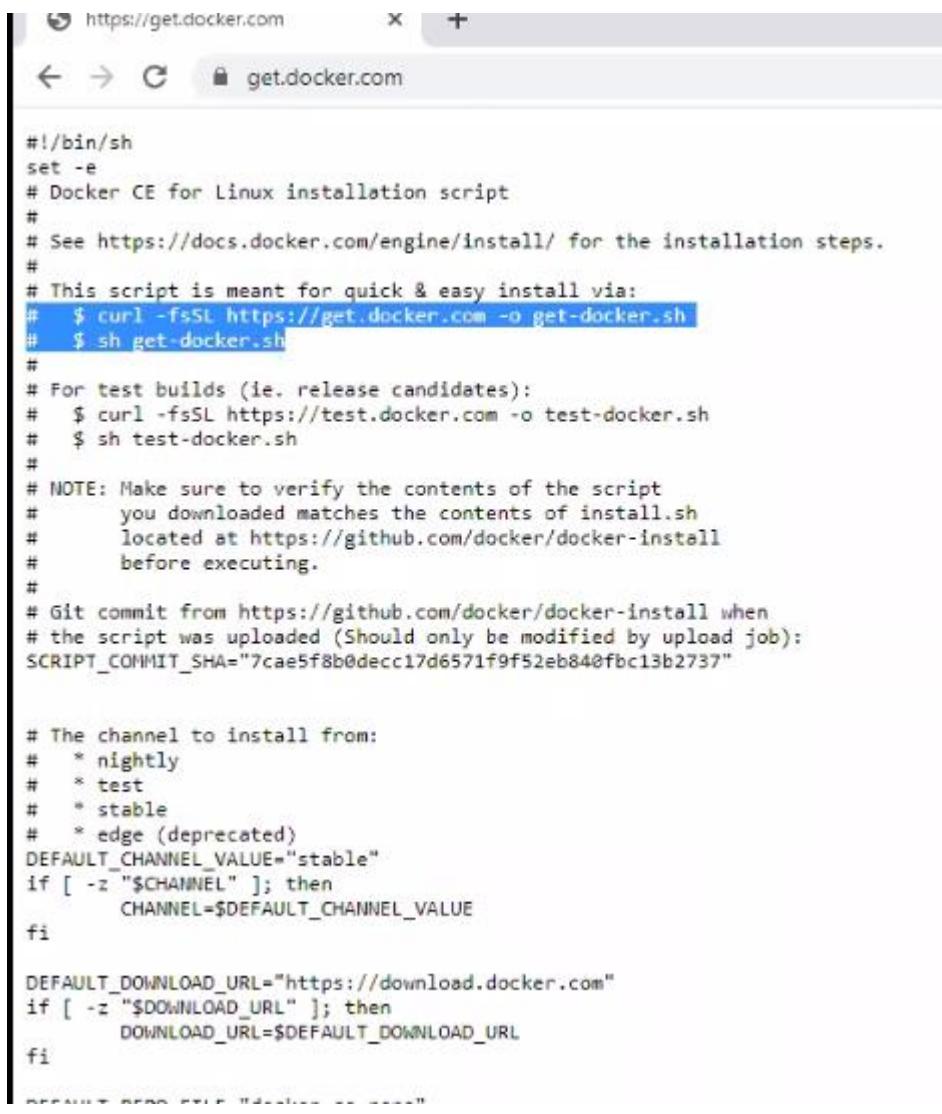
\$ sudo su –



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

<https://get.docker.com/>

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



A screenshot of a web browser window displaying the Docker CE for Linux installation script at <https://get.docker.com>. The script is a shell script with several comments explaining its purpose and usage. It includes instructions for quick & easy install via curl, test builds, and verifying the contents. It also defines variables like CHANNEL and DOWNLOAD_URL, and handles command-line arguments.

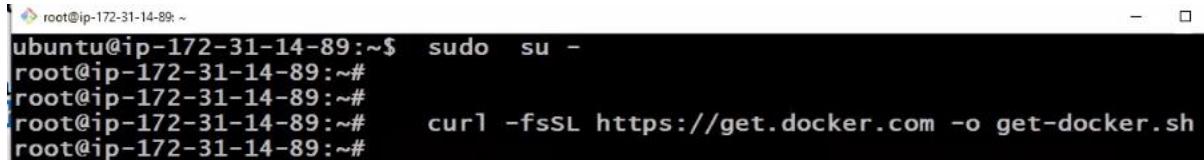
```
#!/bin/sh
set -e
# Docker CE for Linux installation script
#
# See https://docs.docker.com/engine/install/ for the installation steps.
#
# This script is meant for quick & easy install via:
#   $ curl -fsSL https://get.docker.com -o get-docker.sh
#   $ sh get-docker.sh
#
# For test builds (ie. release candidates):
#   $ curl -fsSL https://test.docker.com -o test-docker.sh
#   $ sh test-docker.sh
#
# NOTE: Make sure to verify the contents of the script
#       you downloaded matches the contents of install.sh
#       located at https://github.com/docker/docker-install
#       before executing.
#
# Git commit from https://github.com/docker/docker-install when
# the script was uploaded (Should only be modified by upload job):
SCRIPT_COMMIT_SHA="7cae5f8b0decc17d6571f9f52eb840fbc13b2737"

# The channel to install from:
# * nightly
# * test
# * stable
# * edge (deprecated)
DEFAULT_CHANNEL_VALUE="stable"
if [ -z "$CHANNEL" ]; then
    CHANNEL=$DEFAULT_CHANNEL_VALUE
fi

DEFAULT_DOWNLOAD_URL="https://download.docker.com"
if [ -z "$DOWNLOAD_URL" ]; then
    DOWNLOAD_URL=$DEFAULT_DOWNLOAD_URL
fi

#-----
```

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



A screenshot of a terminal window on an Ubuntu system. The user runs the curl command to download the Docker installation script. The terminal shows the command entered and the resulting output, which is the Docker installation script itself.

```
root@ip-172-31-14-89:~$ sudo su -
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# curl -fsSL https://get.docker.com -o get-docker.sh
root@ip-172-31-14-89:~#
```

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-https
  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:~#
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 it 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.

Name	Instance ID	Instance state	Instance type	Status check
DockerHost	i-04ef5883f11a1c98e	Running	t2.micro	2/2 checks passed

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

docker pull image_name

Upload—

docker 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 containers

```
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 containers(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 any command inside a container

docker exec -it container_name/container_id command

Eg: To launch the bash shell in a container

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 variables 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

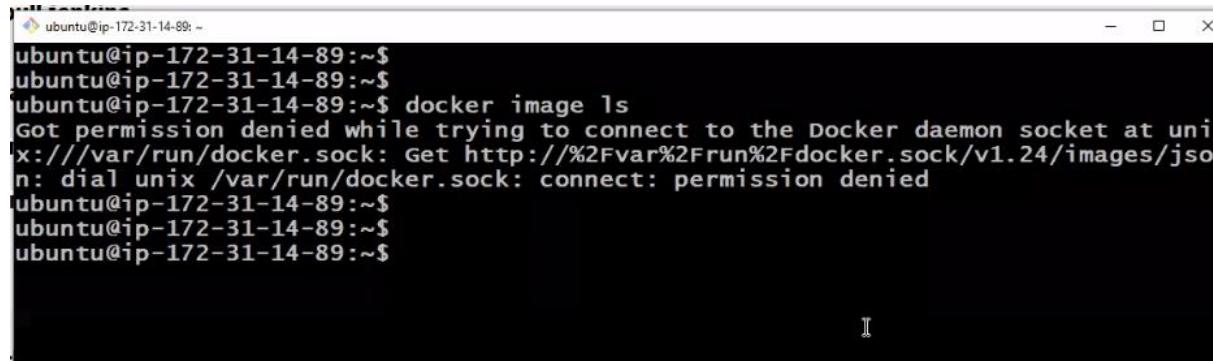
--volume -from Used for sharing volume between containers

--network Used to run the container 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

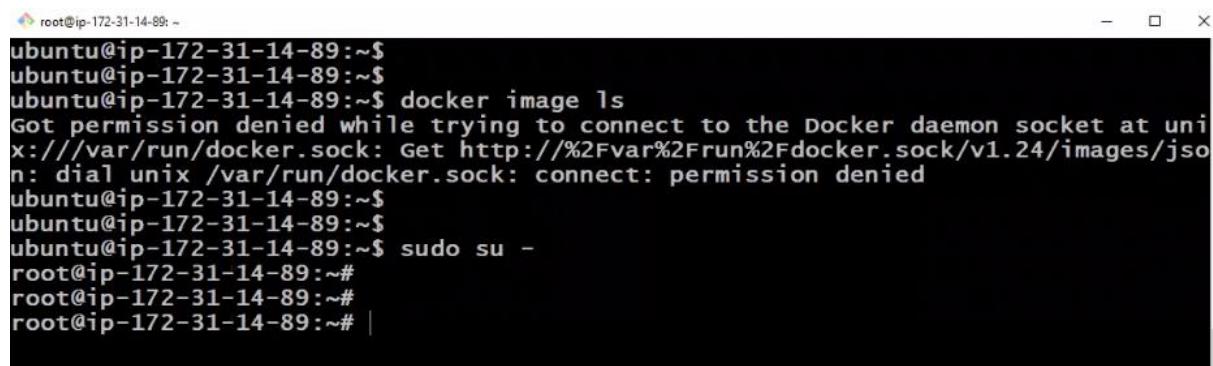
+++++=



```
ubuntu@ip-172-31-14-89:~$ docker image ls
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/json: dial unix /var/run/docker.sock: connect: permission denied
ubuntu@ip-172-31-14-89:~$
```

Access denied because we should work with Root user

Sudo su-



```
root@ip-172-31-14-89:~$ docker image ls
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.24/images/json: dial unix /var/run/docker.sock: connect: permission denied
root@ip-172-31-14-89:~$ sudo su -
root@ip-172-31-14-89:~#
```

docker images (There are no images)

```
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:~# |
```

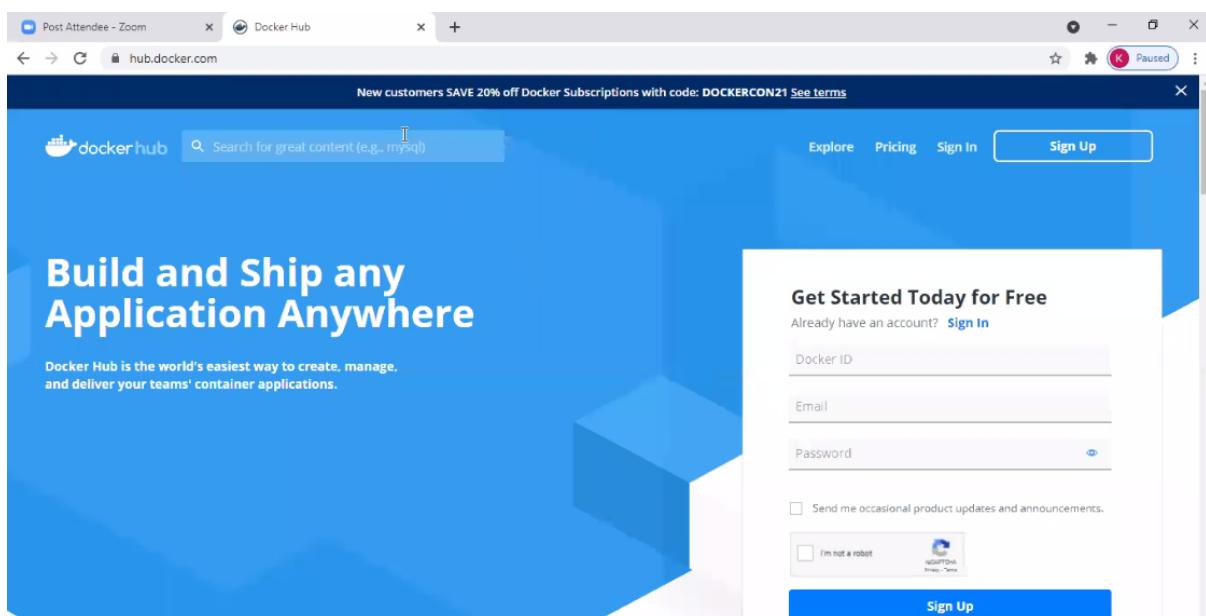
Whenever need container 1st down load image

2nd—run the image

To download tomcat image

Downloaded tomeec 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

Post Attendee - Zoom Explore hub.docker.com/search?q=mysql&type=image

New customers SAVE 20% off Docker Subscriptions with code: DOCKERCON21 See terms

Dockerhub mysql

Explore Pricing Sign In Sign Up

Docker Containers Plugins

Filters 1 - 25 of 26,992 results for mysql. Clear search

Images

- Verified Publisher
- Official Images Published By Docker

Categories

- Analytics
- Application Frameworks
- Application Infrastructure
- Application Services
- Base Images

mysql

MySQL Updated 6 hours ago

MySQL is a widely used, open-source relational database management system (RDBMS).

Container Linux x86-64 Databases

mariadb

MariaDB Updated 6 hours ago

MariaDB Server is a high performing open source relational database, forked from MySQL.

OFFICIAL IMAGE 10M+ 10K+ Downloads Stars

OFFICIAL IMAGE 10M+ 4.1K Downloads Stars

For Ubuntu search

Post Attendee - Zoom Explore hub.docker.com/search?q=ubuntu&type=image

New customers SAVE 20% off Docker Subscriptions with code: DOCKERCON21 See terms

Dockerhub ubuntu

Explore Pricing Sign In Sign Up

Docker Containers Plugins

Filters 1 - 25 of 100,720 results for ubuntu. Clear search

Images

- Verified Publisher
- Official Images Published By Docker

Categories

- Analytics
- Application Frameworks
- Application Infrastructure
- Application Services

ubuntu

Ubuntu Updated 6 hours ago

Ubuntu is a Debian-based Linux operating system based on free software.

Container Linux 386 ARM 64 ARM x86-64 IBM Z PowerPC 64 LE Base Images Operating Systems

ubuntu-debootstrap

Ubuntu Updated 6 hours ago

Ubuntu-debootstrap

OFFICIAL IMAGE 10M+ 10K+ Downloads Stars

OFFICIAL IMAGE 5M+ 44 Downloads Stars

Similarly for tomcat, we get tomee and command also available

Post Attendee - Zoom Explore hub.docker.com/search?q=tomcat&type=image

New customers SAVE 20% off Docker Subscriptions with code: DOCKERCON21 See terms

Dockerhub tomcat

Explore Sign In Sign Up

tomee ☆

Tomitribe join the tribe

Docker Official Images

Apache TomEE is an all-Apache Java EE certified stack where Apache Tomcat is top dog.

10M+

Container Linux ARM 64 PowerPC 64 LE x86-64 ARM IBM Z 386 Application Frameworks

Official Image

Copy and paste to pull this image

docker pull toomee

View Available Tags

```
# 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:adf73ca014822ad8237623d388cedf4d5346aa72c270c5acc01431cc9
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: tomee=image name; --name=c1
docker run --name c1 -p 7070:8080 tomee
```

```
root@ip-172-31-14-89:~# docker run --name c1 -p 7070:8080 tomee
NOTE: Picked up JDK_JAVA_OPTIONS: --add-opens=java.base/java.lang=ALL-UNNAMED --
--add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=
ALL-UNNAMED
```

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

The screenshot shows the AWS EC2 Instances page. On the left sidebar, under the 'Instances' section, 'Instances' is selected. In the main content area, the heading 'Instances (1/1) Info' is displayed above a table. The table has columns: Name, Instance ID, Instance state, Instance type, and Status check. One row is visible, showing 'DockerHost' as the name, 'i-04ef5883f11a1c98e' as the instance ID, 'Running' as the state, 't2.micro' as the instance type, and '2/2 checks pass' as the status. Below the table, a detailed view for 'Instance: i-04ef5883f11a1c98e (DockerHost)' is shown. The 'Details' tab is selected, displaying the instance ID, Public IPv4 address (13.235.91.28), and Private IPv4 addresses (172.31.14.89).

<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.

```
root@ip-172-31-14-89: ~
er.init ----- Localhost -> /host-manager
04-Jun-2021 05:34:28.970 INFO [main] org.apache.openejb.config.ConfigurationFactory.configureApplication Configuring enterprise application: /usr/local/tomee/webapps/host-manager
04-Jun-2021 05:34:28.982 INFO [main] org.apache.openejb.config.AppInfoBuilder.build Enterprise application "/usr/local/tomee/webapps/host-manager" loaded.
04-Jun-2021 05:34:28.982 INFO [main] org.apache.openejb assembler.classicAssembler.createApplication Assembling app: /usr/local/tomee/webapps/host-manager
04-Jun-2021 05:34:28.996 INFO [main] org.apache.tomee.catalina.TomcatWebAppBuilder.deployWebApps using context file /usr/local/tomee/webapps/host-manager/META-INF/context.xml
04-Jun-2021 05:34:28.999 INFO [main] org.apache.openejb assembler.classicAssembler.createApplication Deployed Application(path=/usr/local/tomee/webapps/host-manager)
04-Jun-2021 05:34:29.043 INFO [main] org.apache.myfaces.ee.MyFacesContainerInitializer.onStartUp Using org.apache.myfaces.ee.MyFacesContainerInitializer
04-Jun-2021 05:34:29.084 INFO [main] jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke Deployment of web application directory [/usr/local/tomee/webapps/host-manager] has finished in [162] ms
04-Jun-2021 05:34:29.095 INFO [main] jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke Starting ProtocolHandler ["http-nio-8080"]
04-Jun-2021 05:34:29.127 INFO [main] jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke Server startup in [1854] milliseconds
```

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.

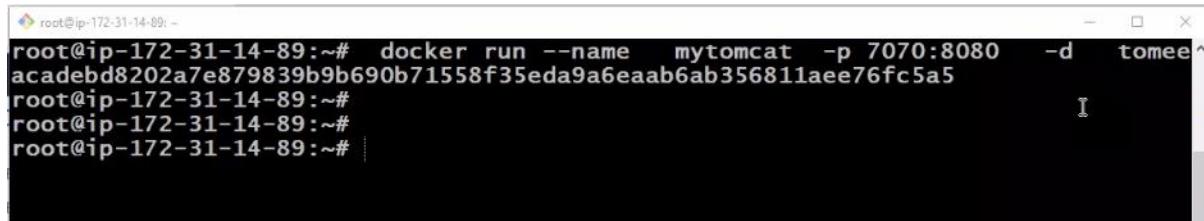
Lets remove the container (Open another gitbash terminal)

```
# docker stop c1
# docker rm -f c1
```

```
root@ip-172-31-14-89: ~
root@ip-172-31-14-89:~# docker stop c1
c1
root@ip-172-31-14-89:~# docker rm -f c1
c1
```

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

-d= detached mode.

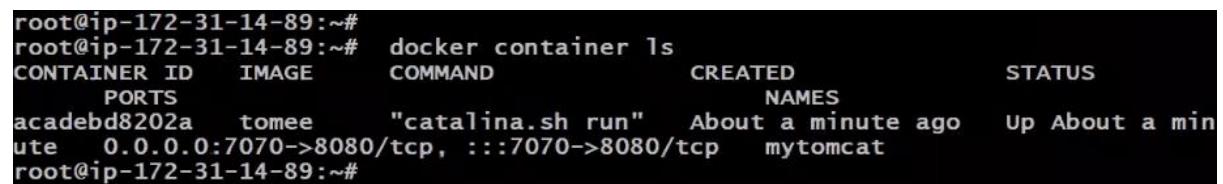


```
root@ip-172-31-14-89:~# docker run --name mytomcat -p 7070:8080 -d tomee
acadebd8202a7e879839b9b690b71558f35eda9a6eaab6ab356811aee76fc5a5
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

Observe 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
```

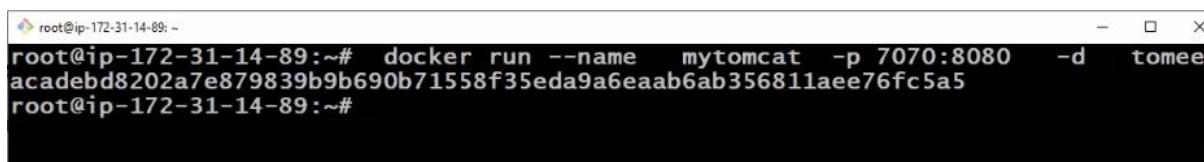


```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker container ls
CONTAINER ID   IMAGE      COMMAND       CREATED          STATUS          NAMES
          Ports
acadebd8202a   tomee     "catalina.sh run"   About a minute ago   Up  About a min
ute   0.0.0.0:7070->8080/tcp, :::7070->8080/tcp   mytomcat
root@ip-172-31-14-89:~#
```

Jenkins

TO start jenkins

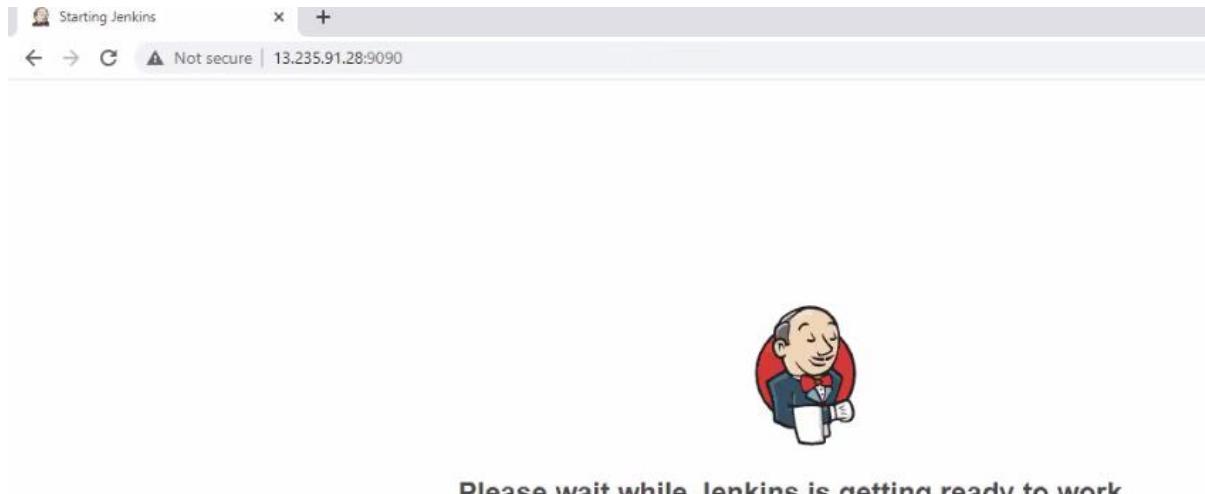
```
# docker run --name myjenkins -p 9090:8080 -d Jenkins/jenkins
```



```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker run --name mytomcat -p 7070:8080 -d tomee
acadebd8202a7e879839b9b690b71558f35eda9a6eaab6ab356811aee76fc5a5
root@ip-172-31-14-89:~#
```

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)

```
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:~# |
```

Ubuntu image is already there.

docker run --name myubuntu -it ubuntu

-it= interactive terminal.

```
root@ip-172-31-14-89:~# docker run --name myubuntu -it ubuntu
root@8c686bc663dd:/# |
```

```
root@8c686bc663dd:/#
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:~# docker run --name myubuntu -it ubuntu
root@8c686bc663dd:/#
root@8c686bc663dd:/#
root@8c686bc663dd:/#
root@8c686bc663dd:/# |
```

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

ls (To see the list of files in ubuntu)

```
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:~# docker run --name myubuntu -it ubuntu
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:~# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot etc  lib   lib64  media  opt  root  sbin  sys  usr
root@8c686bc663dd:~# |
```

exit (To comeout of container back to host)

```
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:~# docker run --name myubuntu -it ubuntu
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:#
root@8c686bc663dd:~# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot etc  lib   lib64  media  opt  root  sbin  sys  usr
root@8c686bc663dd:~# exit
exit
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

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

The screenshot shows the Docker Hub search results for 'nginx'. At the top, there's a banner for Docker Subscriptions. Below it, the search bar shows 'nginx'. The results list the 'nginx' official image from NGINX, Inc. with a star icon. It includes details like '1B+' in size and various build options: Container, Linux, x86-64, ARM, ARM 64, IBM Z, mips64le, PowerPC 64 LE, 386. There are also links for Application Infrastructure and Official Image. On the right, there's a 'Copy and paste to pull this image' button with the command 'docker pull nginx' and a clipboard icon. A link 'View Available Tags' is also present.

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:~# 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:~# docker run --name appserver -P -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
69692152171a: Pull complete
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
1a91f416f3d21d80add47f732547c821750d5e242ceac85b863edc0738fd965
```

```

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
```

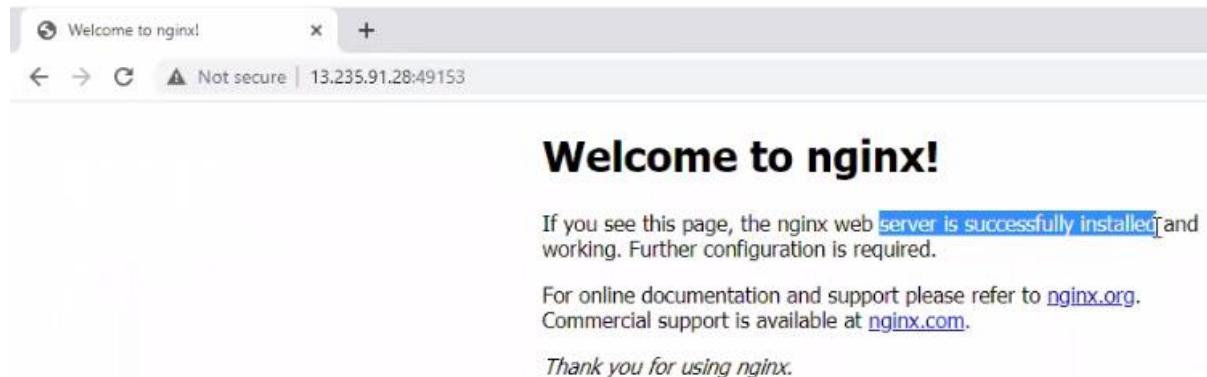
```

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 nginx and docker host)

```

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

The screenshot shows a terminal window with the following text:

```
root@ip-172-31-14-89:~# docker run --name mycentos -it centos
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
7a0437f04f83: Pull complete
Digest: sha256:5528e8b1b1719d34604c87e11dc1c0a20bedf46e83b5632cdeac91b8c04efc1
Status: Downloaded newer image for centos:latest
[root@68e0e5b8bcdd /]#
[root@68e0e5b8bcdd /]#
[root@68e0e5b8bcdd /]#
[root@68e0e5b8bcdd /]# ls
bin  etc  lib  lost+found  mnt  proc  run  srv  tmp  var
dev  home  lib64  media  opt  root  sbin  sys  usr
[root@68e0e5b8bcdd /]# |
```

++++++

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 nginx 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.

The screenshot shows the Docker Hub search interface. The search bar at the top contains 'mysql'. Below the search bar, there's a banner for Docker Subscriptions with a 20% discount code. The main search results are displayed under the heading '1 - 25 of 26,992 results for mysql'. On the left, there are filters for 'Images' (including 'Verified Publisher' and 'Official Images') and 'Categories' (Analytics, Application Frameworks, Application Infrastructure). The first result is 'mysql' by MySQL, which is marked as an 'OFFICIAL IMAGE'. It has over 10M+ downloads and 10K+ stars. A brief description states: 'MySQL is a widely used, open-source relational database management system (RDBMS)'. The second result is 'mariadb' by MySQL, also marked as an 'OFFICIAL IMAGE', with over 10M+ downloads and 4.1K stars. The description for mariadb is partially visible.

```
# 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) click on description

The screenshot shows the Docker Hub page for the MySQL Docker Official Images. At the top, there's a logo for MySQL and the text 'mysql ☆ Docker Official Images'. Below this, a description states: 'MySQL is a widely used, open-source relational database management system (RDBMS)'. There's a download button labeled '1B+'. Below the download button are tabs for 'Container', 'Linux', 'x86-64', 'Databases', and 'Official Image'. The 'Official Image' tab is selected. At the bottom of the page, there are tabs for 'Description', 'Reviews', and 'Tags', with 'Description' being the active tab.

MySQL is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications, covering the entire range from personal projects and websites, via e-commerce and information services, all the way to high profile web properties including Facebook, Twitter, YouTube, Yahoo! and many more.

For more information and related downloads for MySQL Server and other MySQL products, please visit www.mysql.com.

MySQL™

How to use this image

Start a mysql server instance

Starting a MySQL instance is simple:

```
$ docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

... where `some-mysql` is the name you want to assign to your container, `my-secret-pw` is the password to be set for the MySQL root user and `tag` is the tag specifying the MySQL version you want. See the list above for relevant tags.

Connect to MySQL from the MySQL command line client

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

mysql ☆

Docker Official Images

MySQL is a widely used, open-source relational database management system (RDBMS).

Copy and paste to pull this image

```
docker pull mysql
```

View Available Tags

↓ 1B+ Container Linux x86-64 Databases Official Image

Description Reviews Tags

← → 🔍 hub.docker.com/_/mysql?tab=tags&page=1&ordering=last_updated

Filter Tags Sort by Newest

TAG	DIGEST	OS/ARCH	COMPRESSED SIZE
latest	68b207d01891	linux/amd64	154.5 MB
8.0.25	68b207d01891	linux/amd64	154.5 MB
8.0	68b207d01891	linux/amd64	154.5 MB

TAG	DIGEST	OS/ARCH	COMPRESSED SIZE
5	92ad1d7e3f8e	linux/amd64	147.28 MB

```
root@ip-172-31-14-89:~# docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=snil 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
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
S	PORTS			NAMES
d30a3a50857a	mysql:5	"docker-entrypoint.s..."	15 seconds ago	Up 13s mydb
1a91f416f3d2	nginx	"/docker-entrypoint..."	9 minutes ago	Up 9m appserver
fefab04d9fb9	jenkins/jenkins	"/sbin/tini -- /usr/..."	16 minutes ago	Up 16m myjenkins
acadebd8202a	tomee	"catalina.sh run"	18 minutes ago	Up 18m mytomcat

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

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker exec -it mydb bash
root@d30a3a50857a:#
root@d30a3a50857a:~# mysql -u root -p
Enter password: |
```

```
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# docker exec -it mydb bash
root@d30a3a50857a:#
root@d30a3a50857a:~# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.34 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

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 database

```
> 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/>

Happy SQLing.

```
DROP TABLE IF EXISTS emp;

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
);

DROP TABLE IF EXISTS dept;

CREATE TABLE dept (
    deptno decimal(2,0) default NULL,
    dname varchar(14) default NULL,
    loc varchar(13) default NULL
);

INSERT INTO emp VALUES ('7369','SMITH','CLERK','7902','1980-12-17','800.00'
INSERT INTO emp VALUES ('7499','ALLEN','SALESMAN','7698','1981-02-20','1600'
INSERT INTO emp VALUES ('7521','WARD','SALESMAN','7698','1981-02-22','1250.
INSERT INTO emp VALUES ('7566','JONES','MANAGER','7839','1981-04-02','2975.
INSERT INTO emp VALUES ('7654','MARTIN','SALESMAN','7698','1981-09-28','125
INSERT INTO emp VALUES ('7698','BLAKE','MANAGER','7839','1981-05-01','2850.
```

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

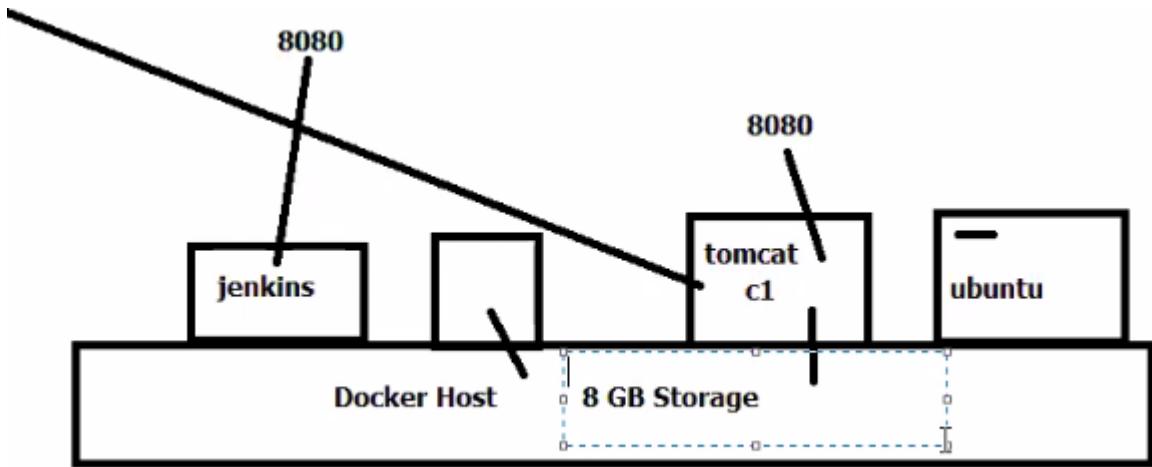
exit

```
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
[{"Id": "92aa85adb206d03d95", "ContainerId": "c9f10f91f10060ee0bce7afc9fe9498883f0d13284d8afde034e49b32fcceb622", "Name": "mydb", "Image": "mysql:5.7", "ImageID": "sha256:3a722de5b2cb274dab2ef9c8c7e46da82f5685761", "Created": "2018-05-10T10:45:40.000000000Z", "Status": "Up 10 seconds", "RunningSince": "2018-05-10T10:45:40.000000000Z", "RestartCount": 0, "Ports": [{"HostPort": 3306, "ContainerPort": 3306}], "HostConfig": {"Binds": null, "Links": null, "Mounts": null, "PortBindings": [{"HostPort": 3306, "ContainerPort": 3306}], "Cgroup": null, "Memory": null, "MemoryReservation": null, "Blkio": null, "CpuPeriod": null, "CpuQuota": null, "CpuShares": null, "CpuRealtimePeriod": null, "CpuRealtimeQuota": null, "Iomtu": null, "IomtuPeriod": null, "IomtuQuota": null, "Ulimits": null, "Dns": null, "DnsOptions": null, "DnsSearch": null, "ExtraHosts": null, "VolumeDriver": null, "ExtraArgs": null, "CapAdd": null, "CapDrop": null, "Privileged": false, "ReadonlyRootfs": false, "SecurityOpt": null, "ShmSize": null, "Utsname": null, "PidType": null, "Label": null}, "NetworkSettings": {"Bridge": "bridge", "NetworkMode": "bridge", "IPAMConfig": null, "Links": null, "Aliases": null, "NetworkID": "435623a722de5b2cb274dab2ef9c8c7e46da82f5685761", "EndpointID": "c9f10f91f10060ee0bce7afc9fe9498883f0d13284d8afde034e49b32fcceb622", "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}}, {"Id": "92aa85adb206d03d95", "ContainerId": "c9f10f91f10060ee0bce7afc9fe9498883f0d13284d8afde034e49b32fcceb622", "Name": "mydb", "Image": "mysql:5.7", "ImageID": "sha256:3a722de5b2cb274dab2ef9c8c7e46da82f5685761", "Created": "2018-05-10T10:45:40.000000000Z", "Status": "Up 10 seconds", "RunningSince": "2018-05-10T10:45:40.000000000Z", "RestartCount": 0, "Ports": [{"HostPort": 3306, "ContainerPort": 3306}], "HostConfig": {"Binds": null, "Links": null, "Mounts": null, "PortBindings": [{"HostPort": 3306, "ContainerPort": 3306}], "Cgroup": null, "Memory": null, "MemoryReservation": null, "Blkio": null, "CpuPeriod": null, "CpuQuota": null, "CpuShares": null, "CpuRealtimePeriod": null, "CpuRealtimeQuota": null, "Iomtu": null, "IomtuPeriod": null, "IomtuQuota": null, "Ulimits": null, "Dns": null, "DnsOptions": null, "DnsSearch": null, "ExtraHosts": null, "VolumeDriver": null, "ExtraArgs": null, "CapAdd": null, "CapDrop": null, "Privileged": false, "ReadonlyRootfs": false, "SecurityOpt": null, "ShmSize": null, "Utsname": null, "PidType": null, "Label": null}, "NetworkSettings": {"Bridge": "bridge", "NetworkMode": "bridge", "IPAMConfig": null, "Links": null, "Aliases": null, "NetworkID": "435623a722de5b2cb274dab2ef9c8c7e46da82f5685761", "EndpointID": "c9f10f91f10060ee0bce7afc9fe9498883f0d13284d8afde034e49b32fcceb622", "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

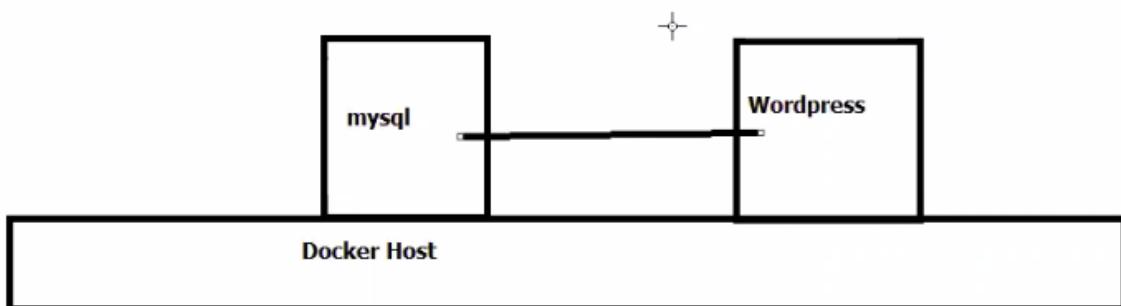
Ex: mysql, wordpress

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

Google search results for "busybox":

- [What is BusyBox used for?](https://play.google.com/store/apps/details?id=ster....)
- [Is BusyBox an operating system?](https://play.google.com/store/apps/details?id=ster....)
- [Is Busy Box Safe?](https://play.google.com/store/apps/details?id=ster....)
- [What is BusyBox Android NDK?](https://play.google.com/store/apps/details?id=ster....)
- <https://en.wikipedia.org/wiki/BusyBox>
- <https://hub.docker.com/r/busybox>

BusyBox - Apps on Google Play

Busybox Pro is on sale now for a limited time only! Root required for this application! The fastest, most trusted, and #1 BusyBox installer and uninstaller! Over 20 ...
★★★★★ Rating: 4.3 · 159,830 votes · Free · Android · Utilities/Tools

BusyBox - Wikipedia

BusyBox is a software suite that provides several Unix utilities in a single executable file. It runs in a variety of POSIX environments such as Linux, Android, and ...
History · Features · Examples · Appliances and reception

BusyBox

Software

BusyBox is a software suite that provides several Unix utilities in a single executable file. It runs in a variety of POSIX environments such as Linux, Android, and FreeBSD, although many of the tools it provides are designed to work with interfaces provided by the Linux kernel. Wikipedia

Operating system: Unix-like
Size: 2.1 MB (compressed "tar.bz2")
License: GPLv2
Stable release: 1.33.1 (May 3, 2021; 22 days ago)
Initial release: November 4, 1999; 21 years ago
Original author(s): Bruce Perens

Create 1st busy box container

```
# docker run --name c10 -it busybox
```

```
root@ip-172-31-14-89:~# docker run --name c10 -it busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
92f8b3f0730f: Pull complete
Digest: sha256:b5fc1d7b2e4ea86a06b0cf88de915a2c43a99a00b6b3c0af731e5f4c07ae8eff
Status: Downloaded newer image for busybox:latest
/ #
/ # 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:~# |
```

/ #

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.

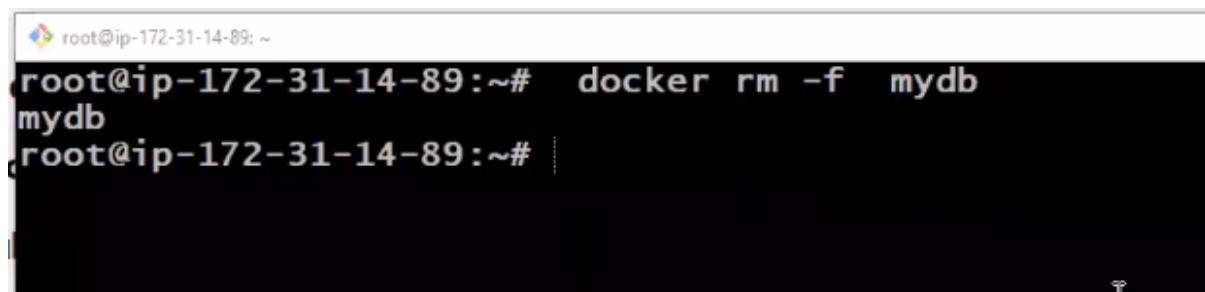
Developer should be able to create wordpress website

1) TO start mysql as container

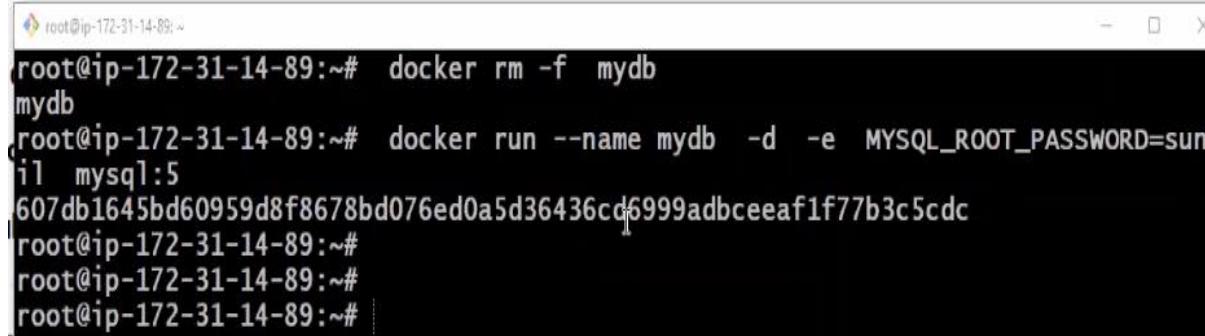
```
# docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5
```

(if container is already in use , remove it

```
# docker rm -f mydb )
```



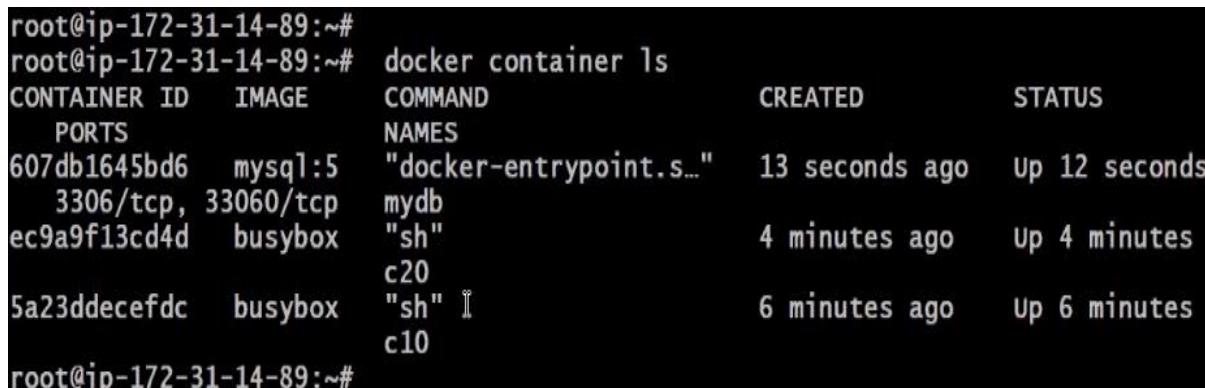
```
root@ip-172-31-14-89:~# docker rm -f mydb
mydb
root@ip-172-31-14-89:~#
```



```
root@ip-172-31-14-89:~# docker rm -f mydb
mydb
root@ip-172-31-14-89:~# docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5
607db1645bd60959d8f8678bd076ed0a5d36436cd6999adbceeaaf1f77b3c5cdc
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

Check whether the container is running or not

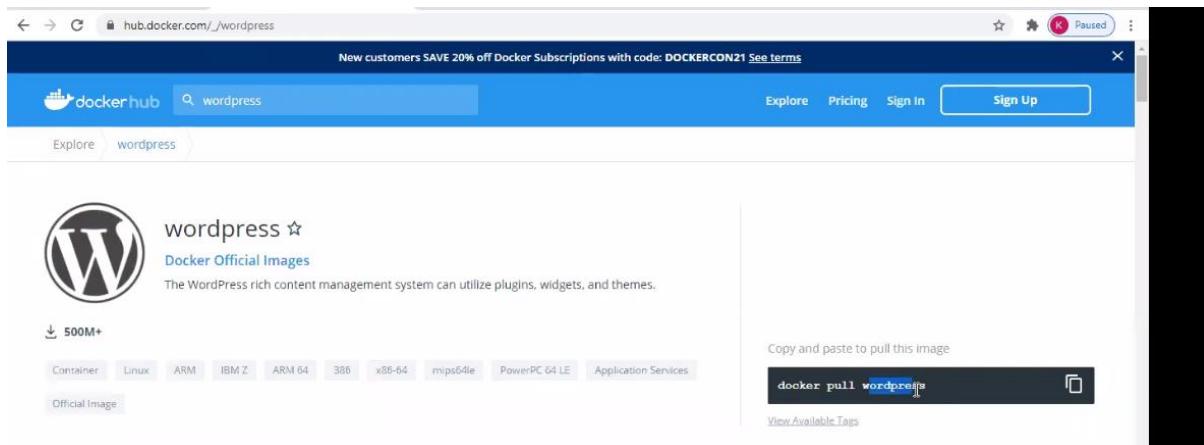
```
# docker container ls
```



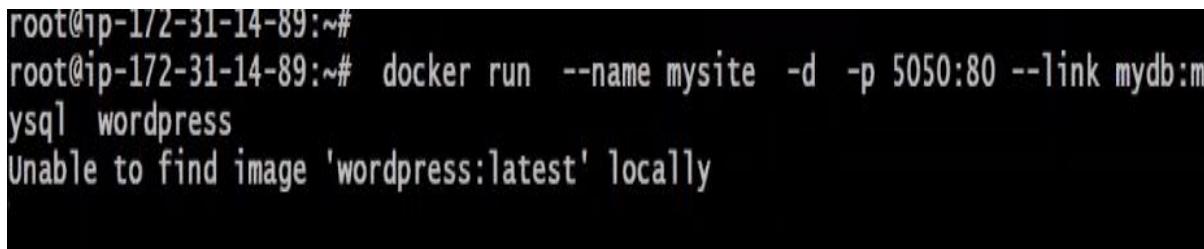
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
607db1645bd6	mysql:5	"docker-entrypoint.s..."	13 seconds ago	Up 12 seconds
3306/tcp, 33060/tcp	mydb			
ec9a9f13cd4d	busybox	"sh"	4 minutes ago	Up 4 minutes
c20				
5a23ddecefcd	busybox	"sh" ↵	6 minutes ago	Up 6 minutes
c10				

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

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)

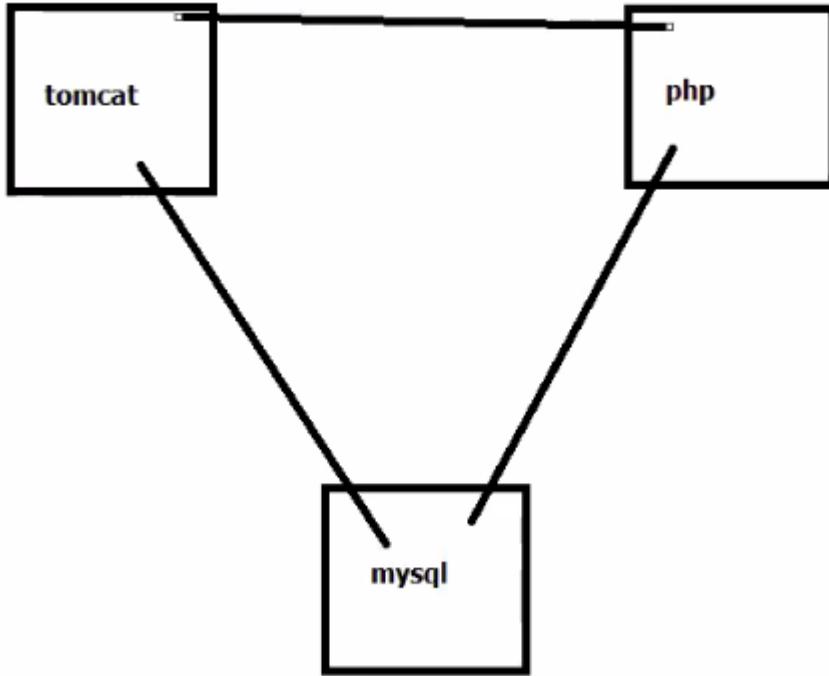
Lets remove all the docker containers

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

```
root@ip-172-31-14-89:~# docker rm -f $(docker ps -aq)
d6aef13f383f
607db1645bd6
ec9a9f13cd4d
5a23ddecefcd
68e0e5b8bcdd
1a91f416f3d2
8c686bc663dd
fefab04d9fb9
acadebd8202a
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~#
```

`# docker container ls (we have no containers now)`

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



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=sunil mysql:5
02c289fd91855a5b23c662df601ffb5f9b65072ceee2d58ce315aa9493234b16
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# |
```

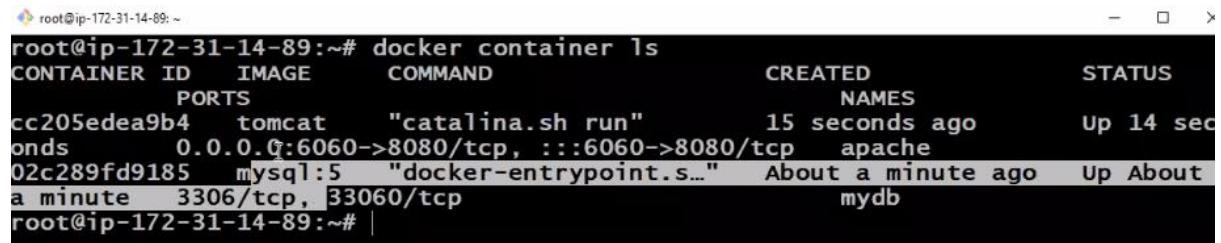
2) TO start tomcat as container

```
# 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 mydb: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: sha256:71703331e3e7f8581f2a8206a612dbeedfbcc7bb8caeee972eadca1cc4a72e6b1
Status: Downloaded newer image for tomcat:latest
cc205eadea9b493eceda6e12e7111f520802886e6901ff450a37c134733b6ccf1
```

TO see the list of containers

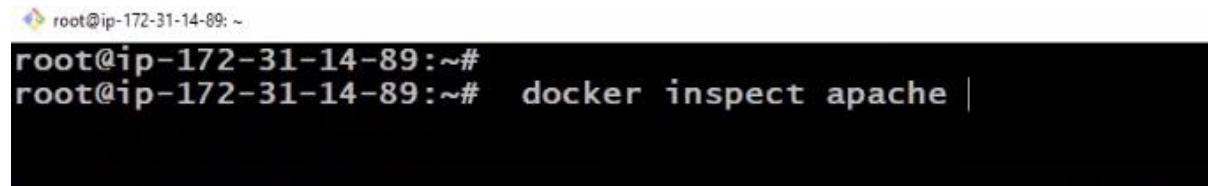
```
# docker container ls
```



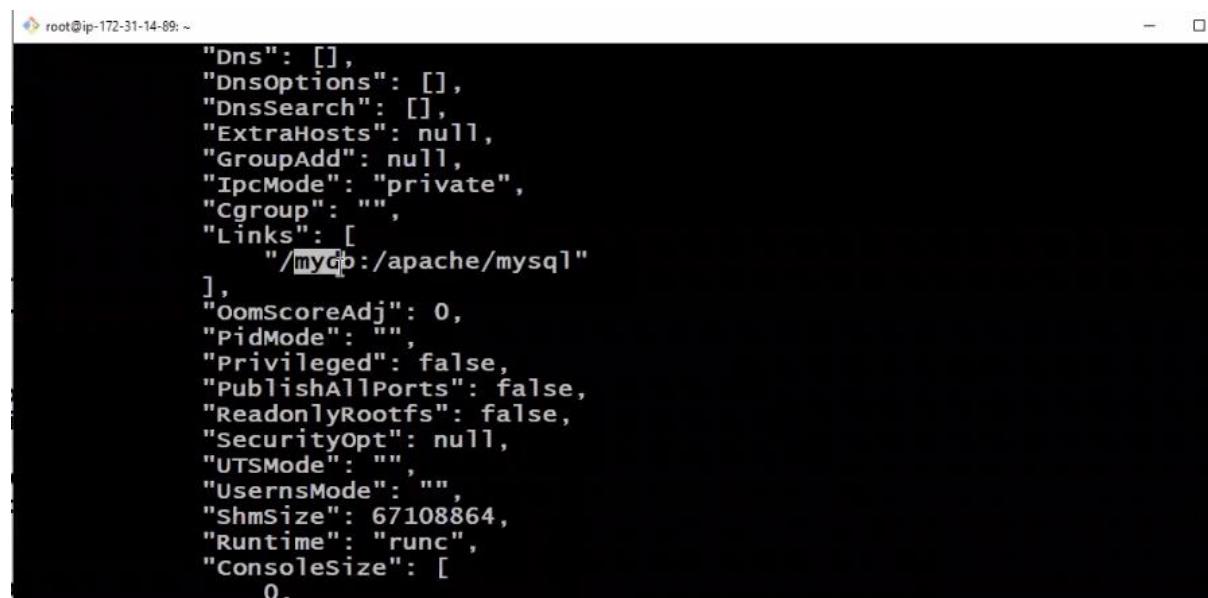
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
		PORTS	NAMES	
cc205edea9b4	tomcat	"catalina.sh run"	15 seconds ago	Up 14 sec
onds		0.0.0.0:6060->8080/tcp, :::6060->8080/tcp	apache	
02c289fd9185	mysql:5	"docker-entrypoint.s..."	About a minute ago	Up About a minute
		3306/tcp, 33060/tcp		mydb

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	



```
"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,
    0
  ]}
```

3) TO start php as container

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

We can attached n number of containers by using --link

```
root@ip-172-31-14-89:~# docker run --name php -d --link apache:tomcat --link mydb:mysql
root@ip-172-31-14-89:~# docker run --name php -d --link apache:tomcat --link mydb:mysql
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)
```

```
root@ip-172-31-14-89:~# docker rm -f $(docker ps -aq)
546f491a17e3
cc205edea9b4
02c289fd9185
root@ip-172-31-14-89:~#
```

To start jenkins as a container

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

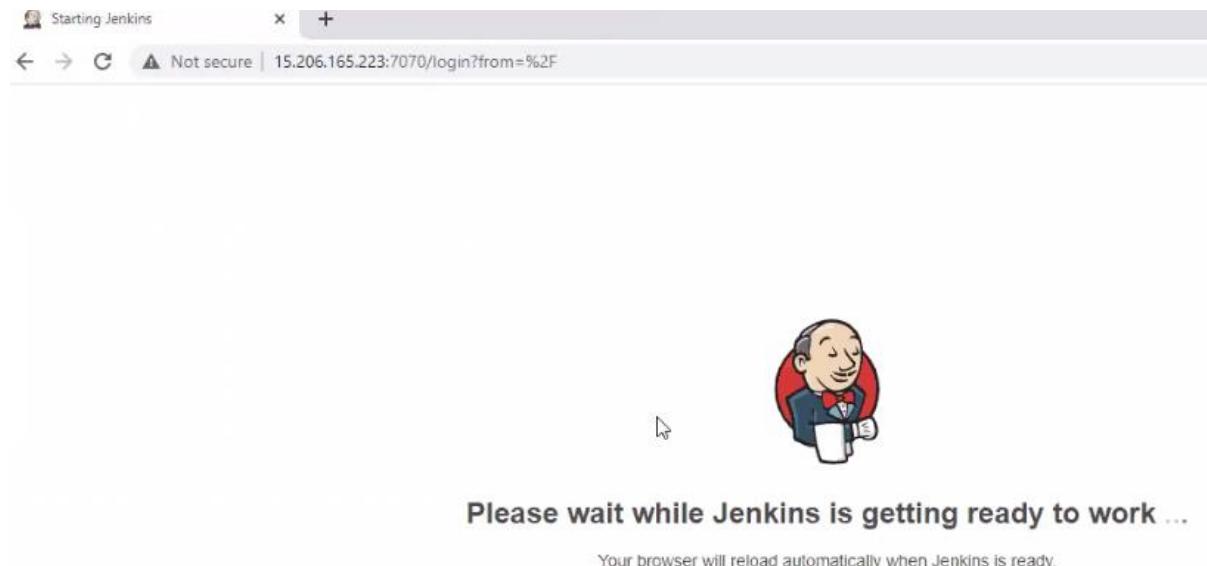
```
root@ip-172-31-14-89:~# docker run --name devserver -d -p 7070:8080 jenkins/jenkins
8c36ee851427635dadf3a19b79bee41ead0a6df2f2304e54a0e4944814fe995d
root@ip-172-31-14-89:~#
root@ip-172-31-14-89:~# |
```

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

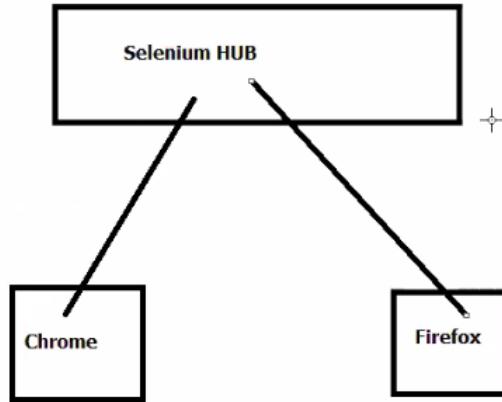
to check the tomcat of prodserver

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

The screenshot shows the Docker Hub search interface. The URL in the address bar is `hub.docker.com/search?q=selenium%2Fhub&type=image`. The search bar contains the query `selenium/hub`. Below the search bar, there's a message for new customers: "New customers SAVE 20% off Docker Subscriptions with code: DOCKERCON21 See terms". The main search results area shows one result: "selenium/hub" by `selenium`, updated 2 days ago. The result is a "Container" for Linux x86-64. It has 10M+ Downloads and 374 Stars. On the left, there are filters for "Images" and "Verified Publisher" and "Official Images". The "Containers" tab is selected. At the bottom, there are links for "Explore", "Pricing", "Sign In", and "Sign Up".

To start selenium/hub as container

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

```
root@ip-172-31-14-89:~# docker run --name hub -d -p 4444:4444 selenium/hub
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:ebcbc572f52820546a0e6094e7a0d8c4d9ffc4ec9c5557cb7af63
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)

The screenshot shows the Docker Hub search interface. The search term 'node-chrome-debug' is entered in the search bar. The results page displays a list of Docker images. The first result is 'selenium/node-chrome-debug' by selenium, which was updated 13 hours ago. It has 10M+ downloads and 63 stars. The second result is 'kurento/node-chrome-debug' by kurento, which was updated 2 years ago. It has 100K+ downloads and 0 stars.

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:selenium 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:selenium 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             NAMES
3eda066a9531        selenium/node-firefox-debug   "/opt/bin/entry_poin..."   13 seconds ago   firefox
21fb177a1467        selenium/node-chrome-debug    "/opt/bin/entry_poin..."   About a minute ago   chrome
233459fae8b3        selenium/hub                "/opt/bin/entry_poin..."   2 minutes ago    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

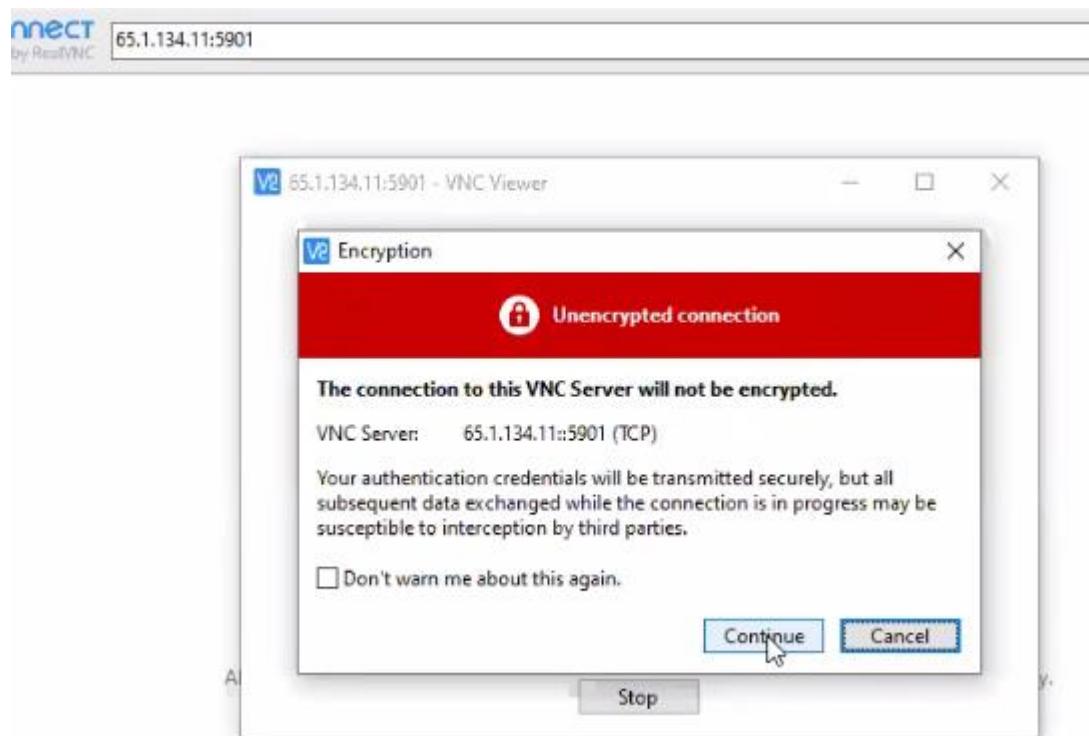
The screenshot shows the RealVNC VNC Connect website. At the top, there's a navigation bar with links for Products, Company, Contact us, and EN. Below it is a secondary navigation bar with links for Discover, Pricing, Download, Support, Partners, Try (button), and Buy (button). The main content area features a heading 'VNC® Connect consists of VNC® Viewer and VNC® Server' and a sub-instruction about downloading the viewer. Below this are icons for various platforms: Windows (selected), macOS, Linux, Raspberry Pi, iOS, Android, Solaris, HP-UX, and AIX. A large blue button labeled 'Download VNC Viewer' is prominently displayed.

In VNC viewer search bar

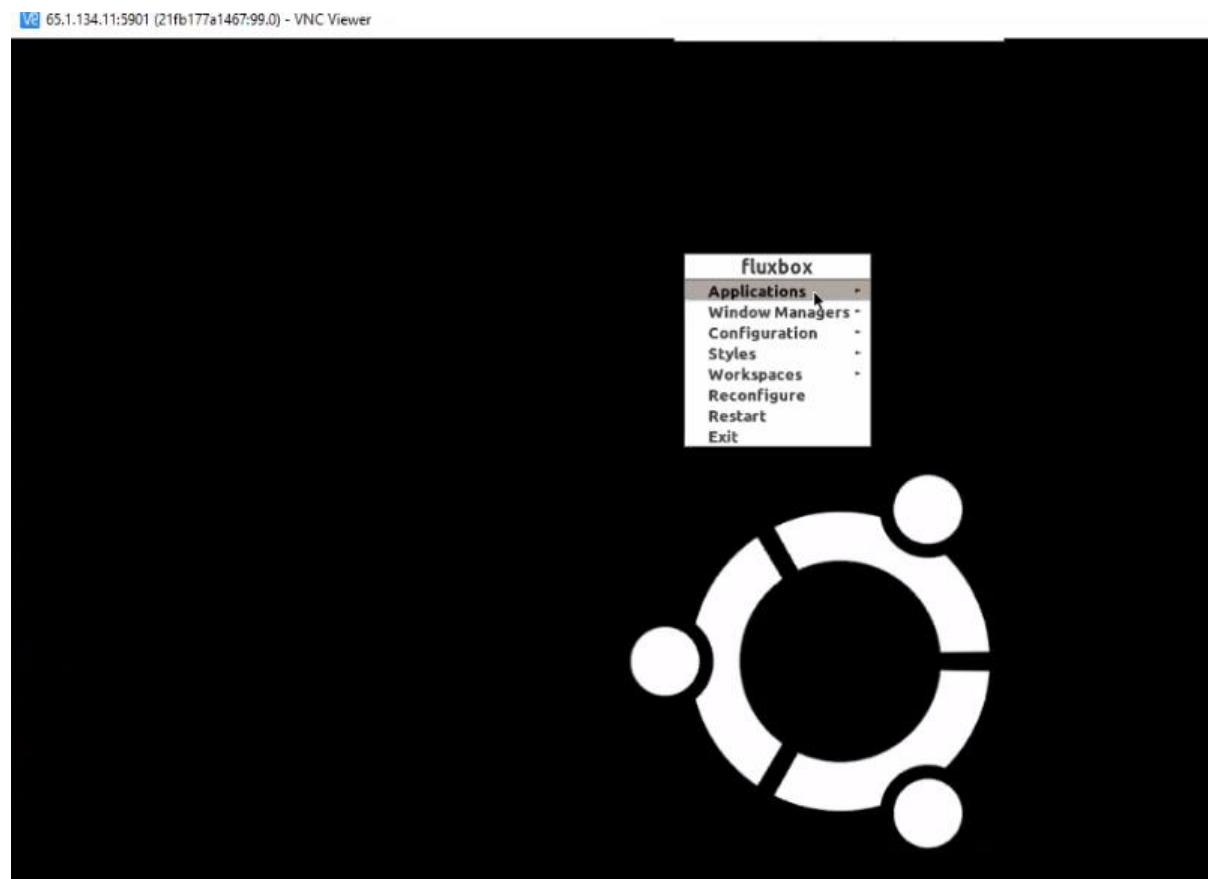
public_ip_dockerhost:5901

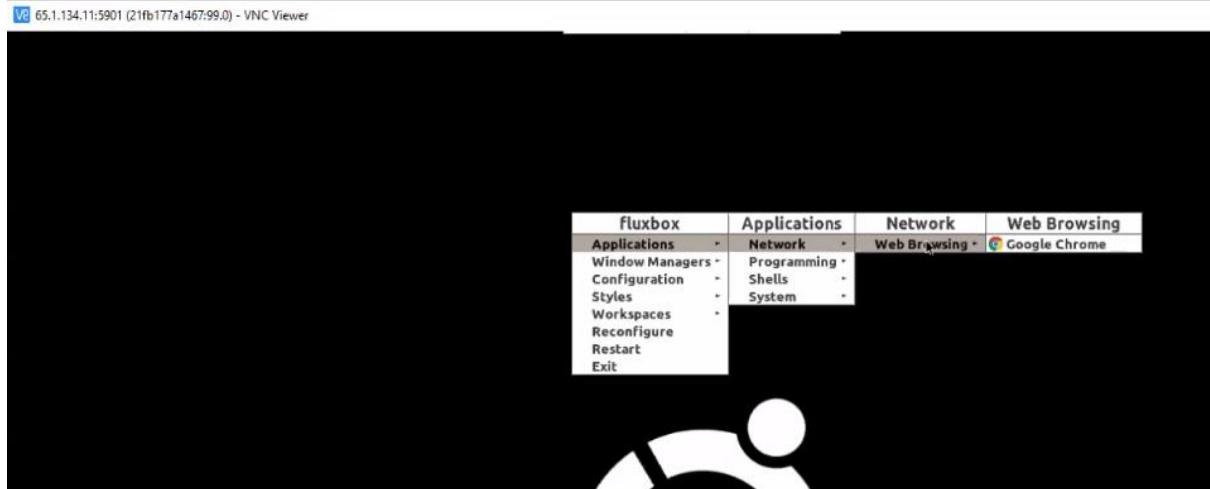
18.136.211.65:5901





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

YAMLint - The YAML Validator +

Not secure | yamllint.com

YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or not, and give you a nice clean UTF-8 version of it. Optimized for Ruby.

```
1 ---  
2 logilabs:  
3   Coordinators:  
4     lakshmi: Devops  
5     rani: AWS  
6   trainers:  
7     raj: Python  
8     sunil: Devops  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

Go

Vaid YAMLI

Exclusively install Docker compose on docker

Installing Docker compose

- 1) Open <https://docs.docker.com/compose/install/>
- 2) Go to linux section

← → ⌂ ⌂ docs.docker.com/compose/install/compose-plugin/#installing-compose-on-linux-systems

 docker docs Home Guides Manuals Reference Samples

Manuals / Docker Compose / Install Compose / Install Docker Compose Plugin

Install Compose plugin through Desktop
Install Docker Compose Plugin
Uninstall Docker Compose

Getting started
Environment variables in Compose
Environment file
Using service profiles
GPU support in Compose
Extend services in Compose
Networking in Compose
Using Compose in production
Control startup order
Sample apps with Compose

Note

Compose standalone: If you need to use Compose without installing the Docker CLI, the instructions for the standalone scenario are similar. Note the target folder for the binary's installation is different as well as the compose syntax used with the plugin (*space compose*) or the standalone version (*dash compose*).

1. To download and install Compose standalone, run:

```
$ curl -SL https://github.com/docker/compose/releases/download/v2.7.0/docker-compose-linux-x86_64
```

2. Apply executable permissions to the standalone binary in the target path for the installation.

3. Test and execute compose commands using `docker-compose`.

Note

If the command `docker-compose` fails after installation, check your path. You can also create a symbolic link to `/usr/bin` or any other directory in your path. For example:

```
$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

1. Run this command to download the current stable release of Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

To install a different version of Compose, substitute `1.29.2` with the version of Compose you want to use.

If you have problems installing with `curl`, see [Alternative Install Options](#) tab above.

2. Apply executable permissions to the binary:

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

Copy and paste 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
```

```
root@ip-172-31-11-3:~# 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
% Total    % Received % Xferd  Average Speed   Time   Time   Current
          Dload  Upload   Total   Spent    Left  Speed
100  633  100  633    0     0   2190      0 --:--:-- --:--:-- 2190
100 15.4M  100 15.4M    0     0  11.1M      0  0:00:01  0:00:01 --:--:-- 20.1M
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

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

```
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~# sudo chmod +x /usr/local/bin/docker-compose  
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~#
```

How to check docker compose is installed or not?

```
# Docker-compose --version
```

```
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~# docker-compose --version  
docker-compose version 1.24.0, build 0aa59064  
+-----+  
+-----+
```

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:~# vim docker-compose.yml
```

```
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 completed 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 events
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)
Gracefully 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 create compose without detachment by we use -d option

So Remove the container which just installed

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

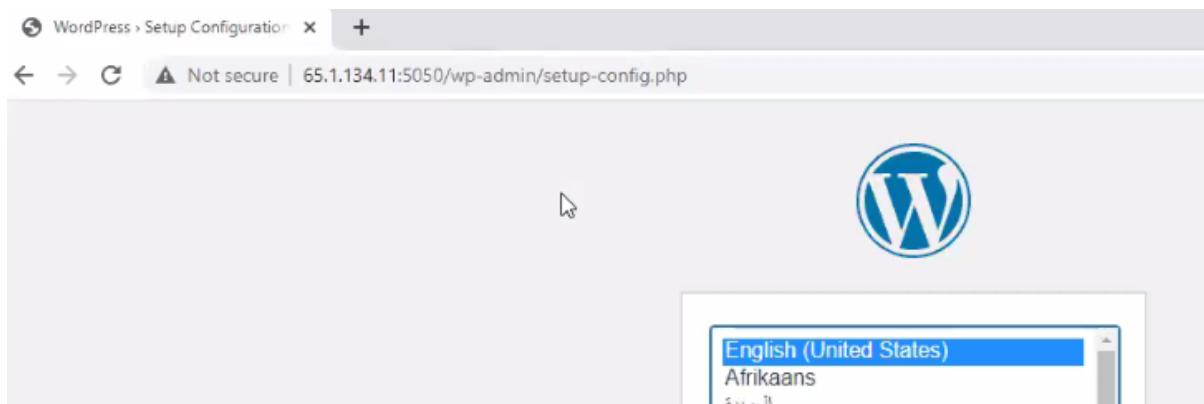
```
[root@ip-172-31-11-3:~# docker rm -f $(docker ps -aq)
27b3caa46cab
86613c15d2b3
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# ]
```

```
# 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

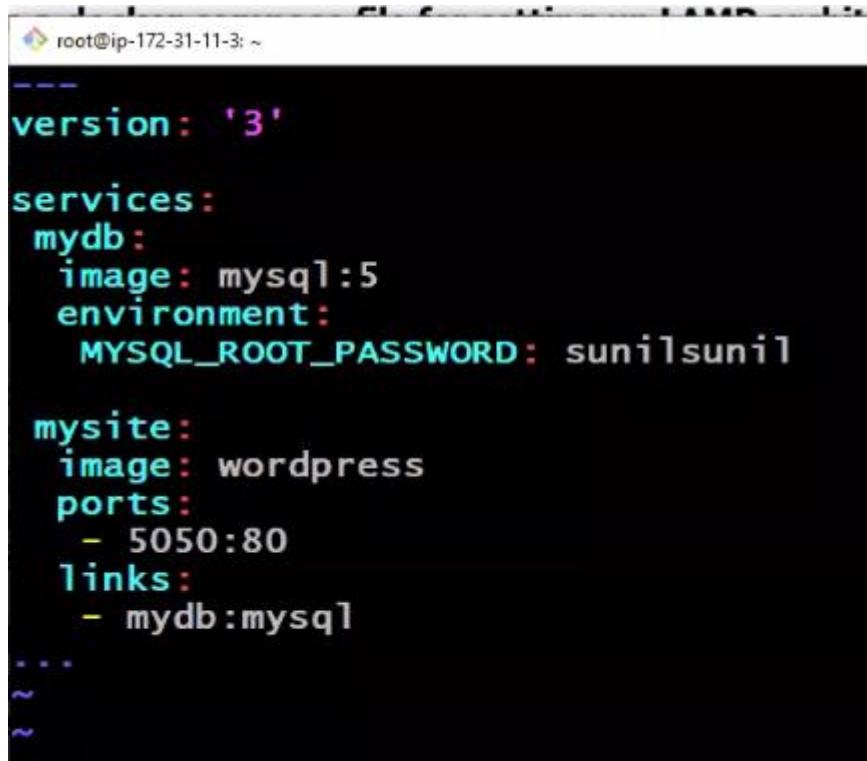
```
# docker-compose stop
```

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

```
+++++
```

Create a docker compose file for setting up LAMP architecture

```
# 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
...
~
```

Delete old version n paste the below yaml file

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

services:
  mydb:
    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
...
-- INSERT --
```

version: '3'

```
services:
  mydb:contain 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
```

```
root@ip-172-31-11-3:~# docker-compose up -d  
Pulling apache (tomee:)...  
latest: Pulling from library/tomee  
b9a857cbf04d: Pulling fs layer  
b9a857cbf04d: Downloading [>  
 518.5kB/50.4MB]  
d557ee20540b: Downloading [>  
b9a857cbf04d: Downloading [=====>  
 5.143MB/50.4MB]  
d557ee20540b: Downloading [=====>  
 2.8MB/7.81MB]  
b9a857cbf04d: Extracting [=====>  
35.13MB/50.4MB]  
  
6cee913589ff: Download complete  
 4.742MB/5.53MB  
3f2724b4ff1a: Downloading [=====>  
 15.81MB/42.16MB]
```

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
```

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

services:
  mydb:
    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
```

Remove old and pate yaml file

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

Please remove old containers bcz our docke host has low mem

```
root@ip-172-31-11-3:~# docker rm -f $(docker ps -aq)  
e0f49945eb5b  
9e747c363b77  
85d7ef038d8c  
16d37ecc5a8a  
root@ip-172-31-11-3:~# |
```

```
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/103.6MB download complete
50.85MB/54.53MB download complete
1845b40ff179: Download complete
63560e572de5: Download complete
2778c0fb363c: Download complete
b1bfe419e51c: Download complete
70.77MB/71.38MB download complete
1ba4a3f12fb0: Download complete
5d997a158691: Download complete
1.924kB/1.924kB download complete
e3f0fcf07941: Download complete
    380B/380B waiting
21bafef74bc6: Waiting
```

Jenkins image not available so it is pulling

```
root@ip-172-31-11-3: ~
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:dff60a7e1f6e476be850c0abdaa9c1ad783c68e9abc0e3907de2a24ef274bfd2
Status: Downloaded newer image for jenkins/jenkins:latest
Creating root_devserver_1 ... done
Creating root_prodserver_1 ... done
Creating root_qaserver_1 ... done
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# |
```

docker container ls

To check

public_ip:7070 (To check jenkins)

public_ip:8899 (Tomcat qa server)

public_ip:9090 (Tomcat prod server)

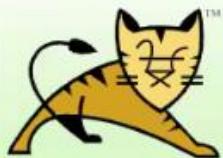
13.126.58.183:7070



13.126.58.183:8899

Apache Tomcat (TomEE)/9.0.41 (8.0.6)

If you're seeing this, you've successfully installed Tomcat.



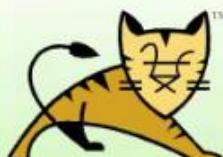
Recommended Reading:

- [Security Considerations How-To](#)
- [Manager Application How-To](#)
- [Clustering/Session Replication How-To](#)

13.126.58.183:9090

Instances | EC2 Management Cc Apache Tomcat (TomEE)/9.0.41 (8.0.6)

If you're seeing this, you've successfully installed Tomcat.



Recommended Reading:

- [Security Considerations How-To](#)
- [Manager Application How-To](#)
- [Clustering/Session Replication How-To](#)

+++++

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.

```
root@6ade908417d4:/# docker run --name c1 -it -v /data centos  
Unable to find image 'centos:latest' locally  
latest: Pulling from library/centos  
7a0437f04f83: Pull complete  
Digest: sha256:5528e8b1b1719d34604c87e11dc1c0a20bedf46e83b5632cdeac91b8c04efc1  
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:~  
[{"Mounts": [{"Config": {"Hostname": "6ade908417d4", "Domainname": "", "User": "", "AttachStdin": true, "AttachStdout": true, "AttachStderr": true}, "Destination": "/data", "Driver": "local", "Mode": "", "RW": true, "Propagation": ""}], "Name": "overlay2"}]
```

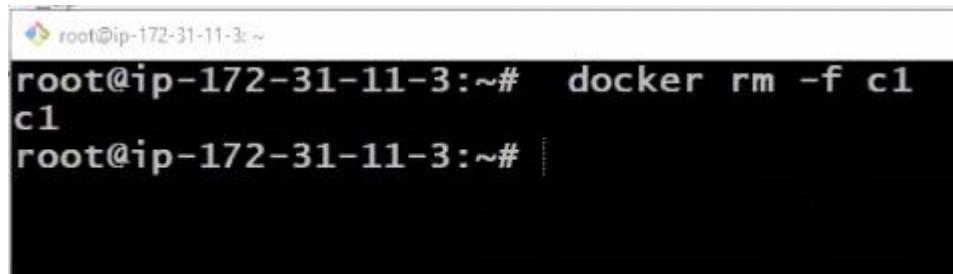
We can see under mounts "data" folder it located in the host machine.

Copy the path

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

Now, lets delete the container

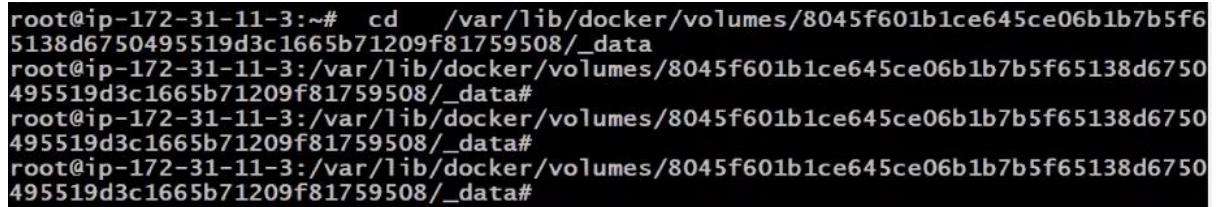
```
# docker rm -f c1
```



```
root@ip-172-31-11-3:~# docker rm -f c1
c1
root@ip-172-31-11-3:~#
```

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

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



```
root@ip-172-31-11-3:~# cd /var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
```

```
# ls ( we can see file1 file2 )
```



```
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data
root@ip-172-31-11-3:~# docker rm -f c1
c1
root@ip-172-31-11-3:~# cd /var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data# ls
file1 file2
root@ip-172-31-11-3:/var/lib/docker/volumes/8045f601b1ce645ce06b1b7b5f65138d6750495519d3c1665b71209f81759508/_data#
```

(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/d867766f70722eaf8cba651bc1d64c60e9f49c5b1f1ebb9e781260f777f3c7  
e8/_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:

```
root@ip-172-31-11-3:~# rm -r /data  
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~# |
```

```
# sudo su -
```

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

```
# mkdir /data
```

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

```
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~# docker run --name c1 -it -v /data centos  
[root@8415bf9df5d0 /]# ls  
bin dev home lib64 media opt root sbin sys usr  
data etc lib lost+found mnt proc run srv tmp var  
[root@8415bf9df5d0 /]#
```

```
# 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 container inside c2

```
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:~# 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
[root@0dcf92ac9c1a data]#
```

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:~#
```

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

```
"UpperDir": "/var/lib/docker/overlay2/8f04207da8b35b390e56c4740248cabbc2a6a0b3af196aa1b80efd3fb0afa404/diff",
  "WorkDir": "/var/lib/docker/overlay2/8f04207da8b35b390e56c4740248cabbc2a6a0b3af196aa1b80efd3fb0afa404/work"
},
  "Name": "overlay2"
},
  "Mounts": [
  {
    "Type": "volume",
    "Name": "0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12",
    "Source": "/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data"
      "Destination": "/data",
      "Driver": "local",
      "Mode": "",
      "RW": true,
      "Propagation": ""
  }
]
```

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

Docker inspect c2—same path

```
root@ip-172-31-11-3:~#
5a4a6424dfdc103541632bc70c6ca923ed632/work"
},
  "Name": "overlay2"
},
  "Mounts": [
  {
    "Type": "volume",
    "Name": "0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12",
    "Source": "/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data",
      "Destination": "/data",
      "Driver": "local",
      "Mode": "",
      "RW": true,
      "Propagation": ""
  }
]
```

Let's remove all the containers

```
# docker rm -f c1 c2 c3
```

```
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 rm -f c1 c2 c3
c1
c2
c3
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# |
```

Lets go to the source path

```
# cd  
/var/lib/docker/volumes/e22a9b39372615727b964151b6c8108d6c02b13114  
a3fcce255df0cee7609e15/_data
```

```
root@ip-172-31-11-3:~# cd /var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data  
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data#  
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data#  
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data#
```

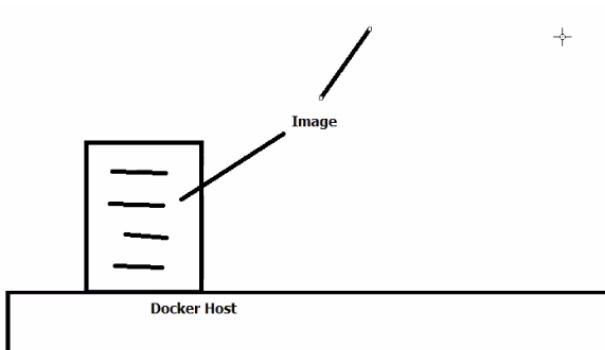
```
# ls ( we can see all the files )
```

```
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_data# ls  
file1 file2 file3 file4 file5 file6  
root@ip-172-31-11-3:/var/lib/docker/volumes/0478ac57b60c9ebe4f144c1c29361c1f209c333a01bb87a50deb30cb37d83c12/_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 whenever 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:adf73ca014822ad823623d388cedf4d5346aa72c270c5acc01431cc93e18e2d
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

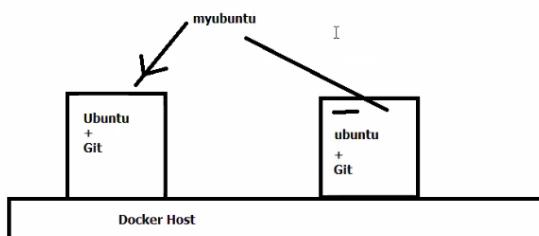
To see the list of images

```
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   dd3827ec8fcac  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
```

```
root@ip-172-31-11-3:~# docker run --name c2 -it myubuntu
root@c07304516d24:#
root@c07304516d24:#
root@c07304516d24:#

```

```
# git --version ( git is pre installed )
```

```
root@c07304516d24:#
root@c07304516d24:# git --version
git version 2.25.1
root@c07304516d24:#

```

Note:

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

1.stop

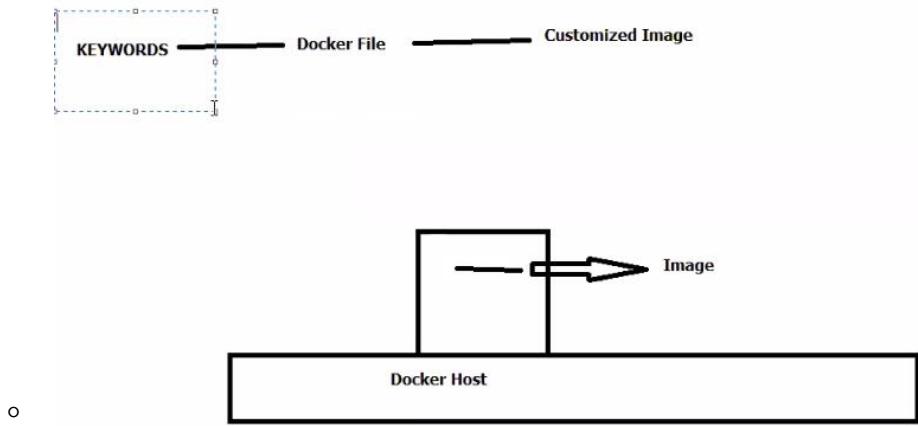
2.create image

3.creating images from running containers 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

STOP SIGNAL -- 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 logilabs. 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

```
 root@ip-172-31-11-3: ~  
FROM nginx  
MAINTAINER logiclabs
```

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:~# 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
```

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

FROM centos

MAINTAINER logiclabs

CMD ["date"]

:wq

```
root@ip-172-31-11-3: ~  
FROM centos  
MAINTAINER logiclabs  
CMD ["date"]
```

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 container 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

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

```
FROM centos
MAINTAINER logiclabs
CMD ["date"]
CMD ["ls", "-la"]
```

```
# 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
```

```
root@ip-172-31-11-3:~# ls -la
total 56
drwxr-xr-x  1 root root 4096 Jun 14 05:29 .
drwxr-xr-x  1 root root 4096 Jun 14 05:29 ..
-rw xr-xr-x  1 root root    0 Jun 14 05:29 .dockerenv
lrwxrwxrwx  1 root root     7 Nov  3 2020 bin -> usr/bin
drwxr-xr-x  5 root root  360 Jun 14 05:29 dev
drwxr-xr-x  1 root root 4096 Jun 14 05:29 etc
drwxr-xr-x  2 root root 4096 Nov  3 2020 home
lrwxrwxrwx  1 root root     7 Nov  3 2020 lib -> usr/lib
lrwxrwxrwx  1 root root     9 Nov  3 2020 lib64 -> usr/lib64
drwxr-xr-x  2 root root 4096 Dec  4 2020 lost+found
drwxr-xr-x  2 root root 4096 Nov  3 2020 media
drwxr-xr-x  2 root root 4096 Nov  3 2020 mnt
drwxr-xr-x  2 root root 4096 Nov  3 2020 opt
dr-xr-xr-x 165 root root    0 Jun 14 05:29 proc
dr-xr-xr-x  2 root root 4096 Dec  4 2020 root
drwxr-xr-x 11 root root 4096 Dec  4 2020 run
lrwxrwxrwx  1 root root     8 Nov  3 2020 sbin -> usr/sbin
drwxr-xr-x  2 root root 4096 Nov  3 2020 srv
dr-xr-xr-x 13 root root    0 Jun 14 05:29 sys
drwxrwxrwt  7 root root 4096 Dec  4 2020 tmp
drwxr-xr-x 12 root root 4096 Dec  4 2020 usr
drwxr-xr-x 20 root root 4096 Dec  4 2020 var
root@ip-172-31-11-3:~#
```

(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 [114 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

```

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

```
docker# git init
```

```
docker# git status
```

```
root@ip-172-31-11-3:~/docker
root@ip-172-31-11-3:~# mkdir docker
root@ip-172-31-11-3:~# mv dockerfile docker
root@ip-172-31-11-3:~# cd docker/
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker# ls
dockerfile
root@ip-172-31-11-3:~/docker# git init
Initialized empty Git repository in /root/docker/.git/
root@ip-172-31-11-3:~/docker# git status
On branch master
I
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    dockerfile

nothing added to commit but untracked files present (use "git add" to track)
root@ip-172-31-11-3:~/docker#
```

```
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"
```

```
root@ip-172-31-11-3:~/docker# git add .
root@ip-172-31-11-3:~/docker# git config --global user.name "sunildevops77"
root@ip-172-31-11-3:~/docker# git config --global user.email "sunildevops77@g
.com"
root@ip-172-31-11-3:~/docker#
root@ip-172-31-11-3:~/docker# git commit -m "a"
[master (root-commit) 10841c3] a
 1 file changed, 4 insertions(+)
 create mode 100644 dockerfile
root@ip-172-31-11-3:~/docker#
```

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

```
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 )
```

```
root@ip-172-31-11-3: ~/docker  
FROM ubuntu  
MAINTAINER logiclabs  
RUN apt-get update  
RUN apt-get install -y git  
~  
~  
~  
~  
~  
~  
~  
~
```

```
+++++
```

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
```

```
root@ip-172-31-11-3:~/docker
FROM ubuntu
MAINTAINER logiclabs
RUN apt-get update
RUN apt-get install -y git
RUN apt-get install -y tree
~
~
```

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 disadvantage 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
```

```
root@ip-172-31-11-3: ~/docker
FROM ubuntu
MAINTAINER logiclabs
RUN apt-get update && apt-get install -y git tree
~
```

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 kB]
```

(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 default 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)
```

```
root@ip-172-31-11-3:~# docker rm -f $(docker ps -aq)
d22e004a89be
3d040f557a40
cf5d8b638509
c07304516d24
24adc066a694
a87ab5477497
0ccfd9375a39
033e930a2362
root@ip-172-31-11-3:~# c|
```

Observation 1:

When we start ubuntu container, we use below command

```
# docker run --name c1 -it ubuntu
```

```
/#
```

```
root@01e92861becc:/#
root@ip-172-31-11-3:~# docker run --name c1 -it ubuntu
root@01e92861becc:/#
root@01e92861becc:/#
root@01e92861becc:/#
```

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

```
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker run --name c1 -it ubuntu
root@01e92861becc:/#
root@01e92861becc:/#
root@01e92861becc:/# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
```

docker container ls

(our container c1 is running in the background)

```
root@ip-172-31-11-3:~# docker run --name c1 -it ubuntu
root@01e92861becc:/#
root@01e92861becc:/#
root@01e92861becc:/# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
01e92861becc ubuntu "/bin/bash" 33 seconds ago Up 32 seconds
c1
root@ip-172-31-11-3:~#
```

Observation 2:

When we start jenkins container, we use below command

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

```
root@ip-172-31-11-3:~# docker run --name c1 -it ubuntu
root@01e92861becc:/#
root@01e92861becc:/#
root@01e92861becc:/# root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~#
root@ip-172-31-11-3:~# docker container ls
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
 NAMES
01e92861becc ubuntu "/bin/bash" 33 seconds ago Up 32 seconds
c1
root@ip-172-31-11-3:~# docker run --name j1 -d -P jenkins/jenkins
decdd17b6b1ed66505f8b57aefd4cbf64295d6dfac0e484848bdd181ac3ff14f
root@ip-172-31-11-3:~#
```

Now, I want to open interactive terminal to enter jenkins

```
# docker exec -it j1 bash
```

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

(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:~#
```

ctrlpq

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.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:~# 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            STATUS              PORTS
NAMES
8f7e038c01f4        tomee              "catalina.sh run"
seconds             0.0.0.0:49155->8080/tcp, :::49155->8080/tcp
                   t1
decdd17b6b1e        jenkins/jenkins    "/sbin/tini -- /usr/..."
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"
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)
```

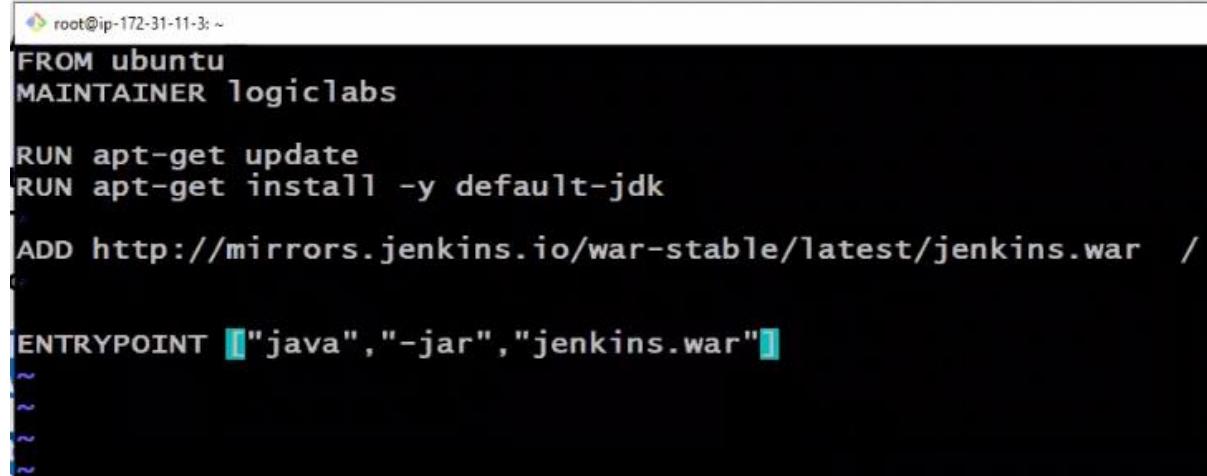
```
RUN apt-get update (installed softwares in creating image)
```

```
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
```



```
root@ip-172-31-11-3: ~
FROM ubuntu
MAINTAINER logiclabs

RUN apt-get update
RUN apt-get install -y default-jdk

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

ENTRYPOINT ["java","-jar","jenkins.war"]
```

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#d
oStart: 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.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 jenkins.war

RUn the below command

docker ps -a

For myubuntu the command is java -jar jenkins.war

For ubuntu the commans is /bin/bash

```

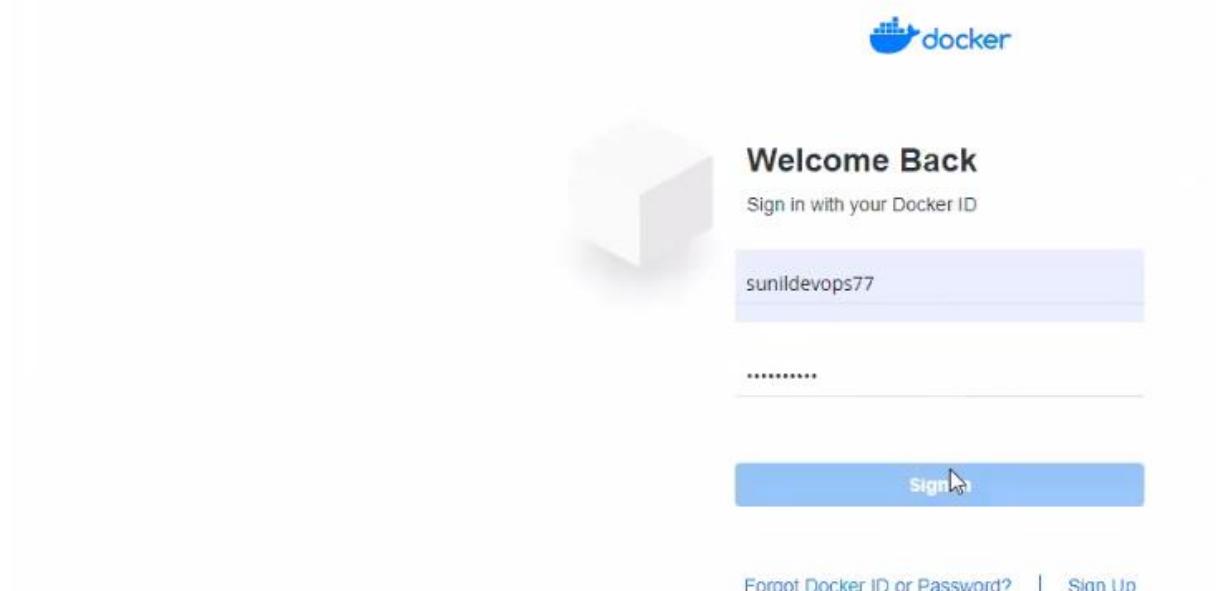
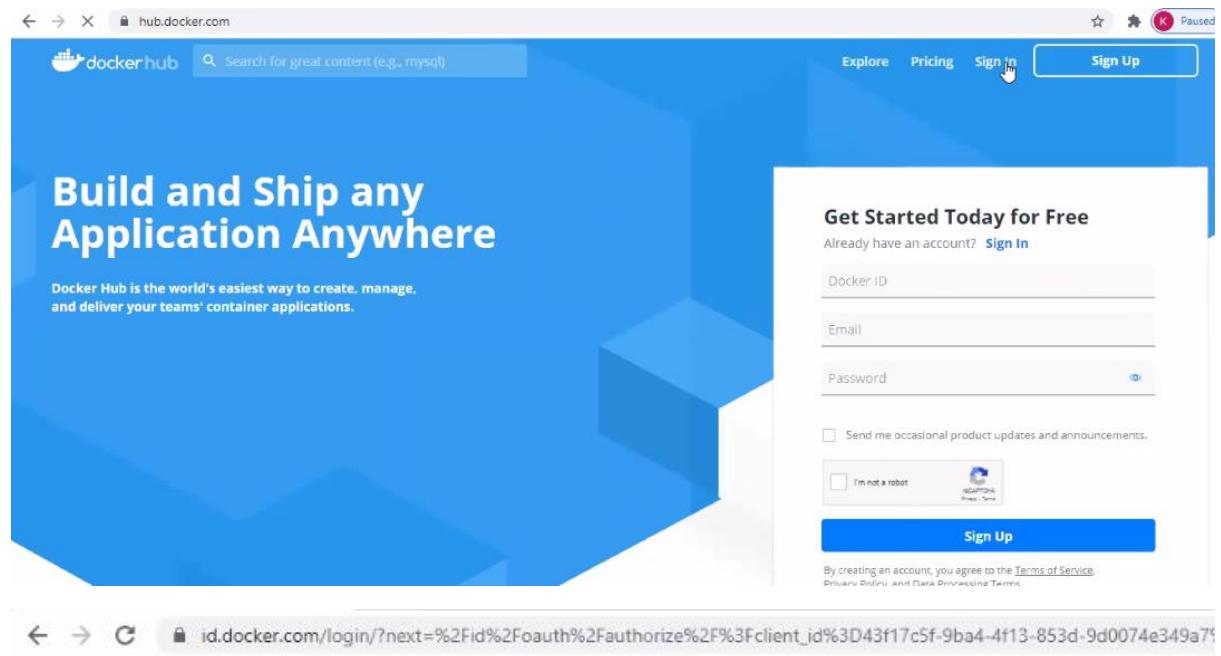
root@ip-172-31-11-3:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED      STATUS
S              PORTS         NAMES
499724ccda8a   newubuntu     "java -jar jenkins.war"   32 seconds ago   Exited
d (130) 11 seconds ago
                                         hopeful_babbage
8f7e038c01f4   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
```

```
root@bdc6c4320a49:/# docker run --name c5 -it ubuntu
root@bdc6c4320a49:/#
root@bdc6c4320a49:/#
```

Lets install tree package in this container

```
/# apt-get update
```

```
root@bdc6c4320a49:/#
root@bdc6c4320a49:/# 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]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
0% [4 InRelease 5497 B/101 kB 5%] [Waiting for headers]
```

```
/# 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
```

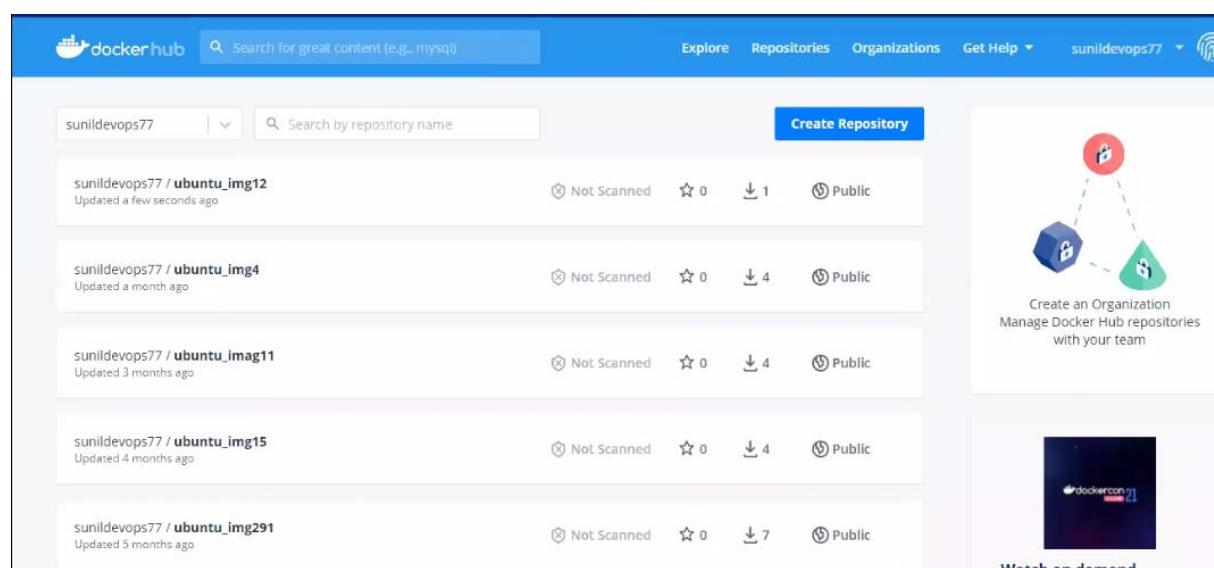
```
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~#  
root@ip-172-31-11-3:~# docker login  
Login with your Docker ID to push and pull images from Docker Hub. If you don't  
have a Docker ID, head over to https://hub.docker.com to create one.  
Username: sunildevops77  
Password:  
WARNING! Your password 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

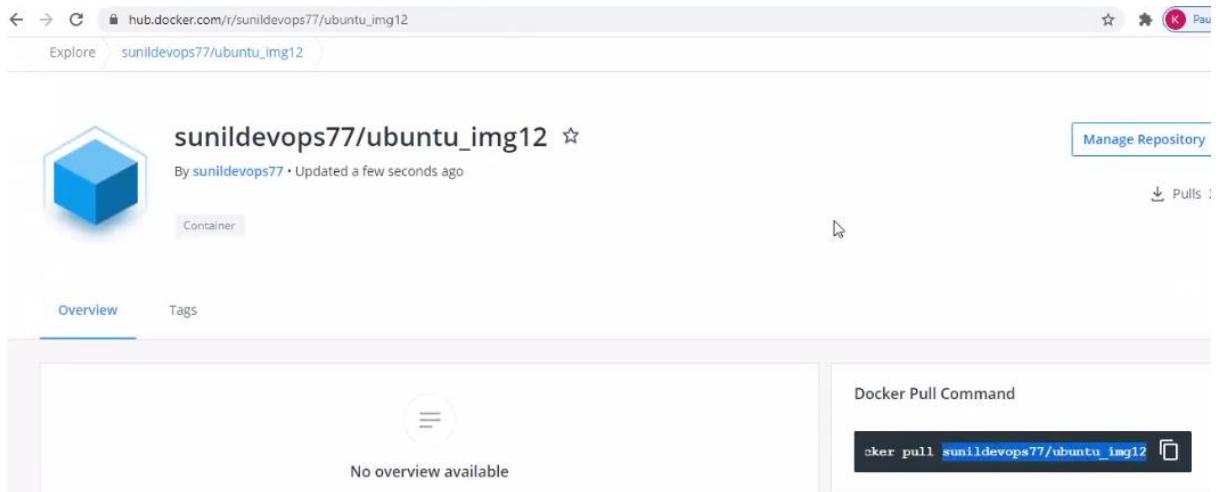
```
# docker push <image_name>
```

```
# 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 containers in single docker host then all containers stopped due to this. Use orchestration- any Hardware failure on docker hosts then the particular containers 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

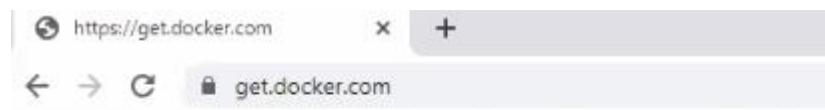
Name is as Manager, Worker1, Worker2

ID	IP Address	Container ID	Status	Type	Ports
Manager	i-0c4fe638f1c0b2429	Running	t2.micro	-	
Worker1	i-06de0238aa92af54a	Running	t2.micro	-	
Worker2	i-03611ade63be424c2	Running	t2.micro	-	

Connect mchines by git bash/putty

All the above machines should have docker installed in it.

Install docker using get.docker.com



```
#!/bin/sh
set -e
# Docker CE for Linux installation script
#
# See https://docs.docker.com/engine/install/ for the installation steps
#
# This script is meant for quick & easy install via:
# $ curl -fsSL https://get.docker.com -o get-docker.sh
# $ sh get-docker.sh
#
# For test builds (ie. release candidates):
# $ curl -fsSL https://test.docker.com -o test-docker.sh
# $ sh test-docker.sh
#
# NOTE: Make sure to verify the contents of the script
#       you downloaded matches the contents of install.sh
#       located at https://github.com/docker/docker-install
#       before executing.
#
# Git commit from https://github.com/docker/docker-install when
#       this file was generated
```

Change to root user if you dnt see

Sudo su -

```
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: 7cae5f8b0decc17d6571f9f52eb840fb
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.

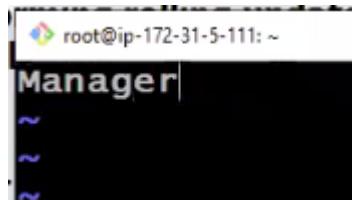
After installing docker in the 1st machine (Manager), Lets change the host name.

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

```
# vim /etc/hostname
```

Manager

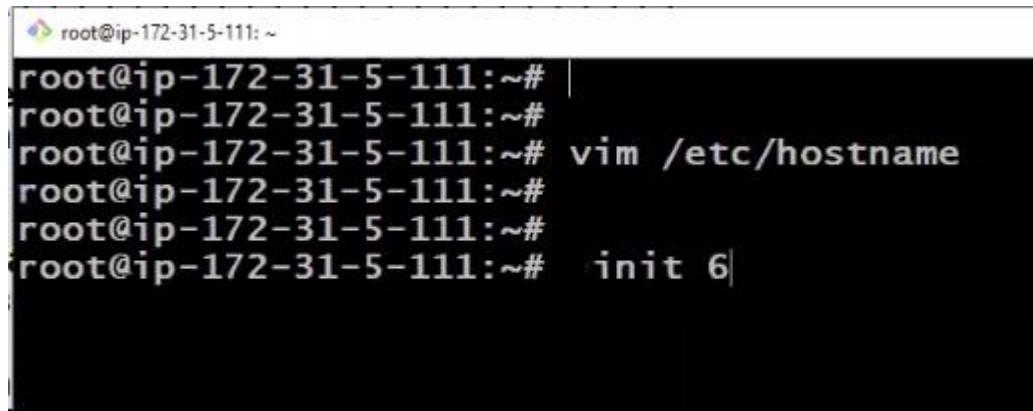
```
:wq
```



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

After changing the hostname, lets restart the machine

```
# init 6
```



```
root@ip-172-31-5-111: ~#
root@ip-172-31-5-111: ~#
root@ip-172-31-5-111: ~# vim /etc/hostname
root@ip-172-31-5-111: ~#
root@ip-172-31-5-111: ~#
root@ip-172-31-5-111: ~# init 6
```

```
+++++
```

Similary repeat the same in worker1 and worker2

```
+++++
```

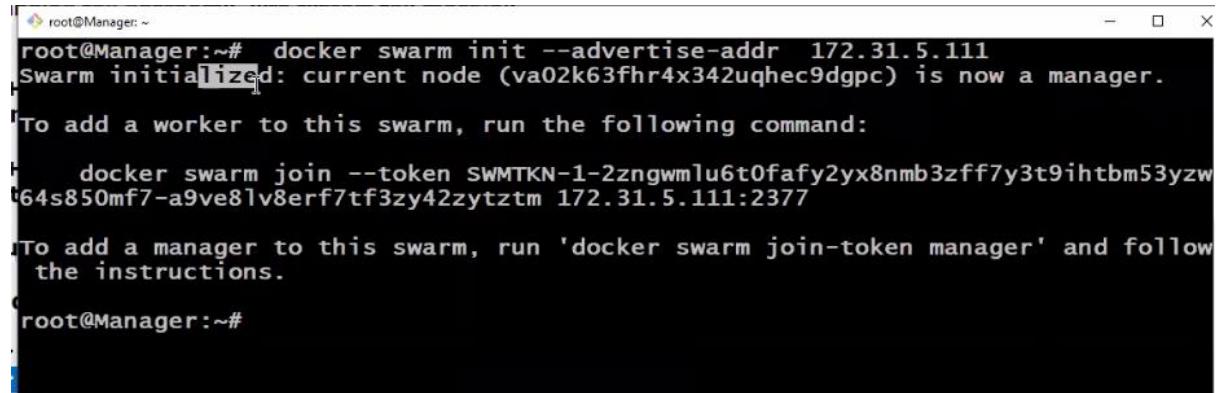
Connect to Manager, install docker swarm in it.

```
$ 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
```



```
root@Manager:~# docker swarm init --advertise-addr 172.31.5.111
Swarm initialized: current node (va02k63fhr4x342uqhec9dgpc) is now a manager.

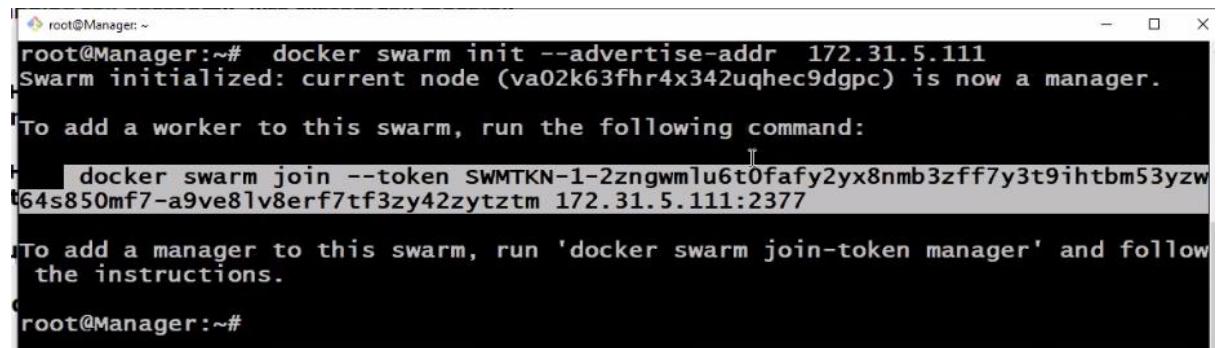
To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw
  64s850mf7-a9ve81v8erf7tf3zy42zyztm 172.31.5.111:2377

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

root@Manager:~#
```

Please read the log messages and take note of the token



```
root@Manager:~# docker swarm init --advertise-addr 172.31.5.111
Swarm initialized: current node (va02k63fhr4x342uqhec9dgpc) is now a manager.

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

  docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw
  64s850mf7-a9ve81v8erf7tf3zy42zyztm 172.31.5.111:2377

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

root@Manager:~#
```

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
```

```
root@Worker1:~# docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw64s850mf7-a9ve81v8erf7tf3zy42zyztm 172.31.5.111:2377
This node joined a swarm as a worker.
root@Worker1:~#
```

```
ubuntu@Worker1:~#
root@Worker1:~# docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw64s850mf7-a9ve81v8erf7tf3zy42zyztm 172.31.5.111:2377
This node joined a swarm as a worker.
root@Worker1:~#
root@Worker1:~#
root@Worker1:~# exit
logout
ubuntu@Worker1:~$ exit
```

Repeat for worker2

```
ubuntu@Worker2:~#
root@Worker2:~# docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw64s850mf7-a9ve81v8erf7tf3zy42zyztm 172.31.5.111:2377
This node joined a swarm as a worker.
root@Worker2:~# exit
logout
ubuntu@Worker2:~$ exit
logout
Connection to ec2-13-126-196-213.ap-south-1.compute.amazonaws.com closed.
```

+++++

TO see the no of nodes from the manager

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

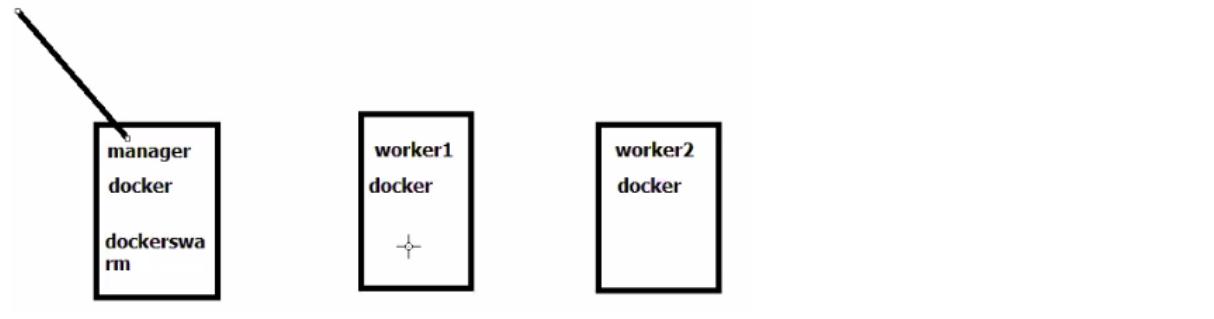
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
va02k63fhrr4x342uqhec9dgpc * 20.10.7	Manager	Ready	Active	Leader
j98tea7gt2tiovjru45040ui4 20.10.7	Worker1	Ready	Active	
xtxqptswvqje191cdfgv5082b 20.10.7	worker2	Ready	Active	

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 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**

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-  
0etsmfa26vreetyq278q8ohhi73il7j1lpnrzzlowuld1r8yex-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 containers with the same service, distributed load in 3 ec2 machines)

```
root@Manager:~# docker service create --name webserver -p 9090:8080 --replicas 5 tomee
u2buu3rxyye0ikhrng7c05mjt
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
root@Manager:~# |
```

How to see where they are running?

```
Manager# docker service ps webserver
```

```

root@Manager:~#
root@Manager:~# docker service ps webserver
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT ST
ATE         webserver.1  tomee:latest  Manager   Running     Running 48
6tex9opvvb3i seconds ago
c9ctol7tmuhq seconds ago
ysuq37miosnj seconds ago
q8obqofu1gtg seconds ago
mabgi6s46sjz seconds ago
root@Manager:~#

```

Lets take the note

Manager - 2 container

Worker1 - 1 container

Worker2 - 2 container

We cant decide how the number of containers 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 swarm will distribute the load to the available docker hosts

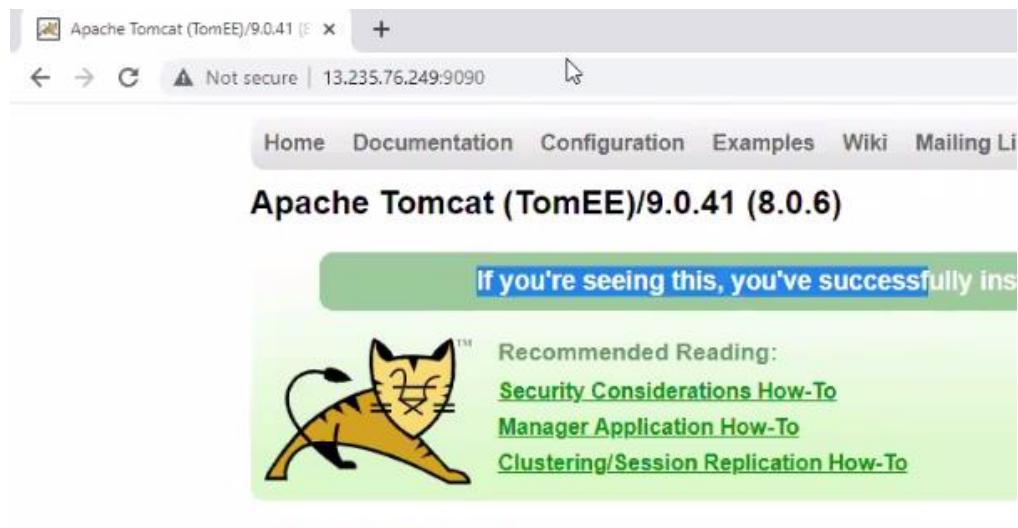
+++++

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 service as containers 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
```

```
root@Manager:~# docker service create --name mydb --replicas 3 -e MYSQL_ROOT_PASSWORD=sunil mysql:5  
uddge25mpmbmr906esm7yvwv  
overall progress: 3 out of 3 tasks  
1/3: running  
2/3: running  
3/3: running  
verify: Service converged  
root@Manager:~#  
root@Manager:~#  
root@Manager:~# docker service ps mydb  
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE  
ERROR      PORTS  
2qg8nrxaf89  mydb.1   mysql:5    Manager    Running      Running 16 seconds  
ago  
2feqau9w9btc  mydb.2   mysql:5    worker1    Running      Running 17 seconds  
ago  
wpctkj10gp2z  mydb.3   mysql:5    worker2    Running      Running 16 seconds  
ago  
root@Manager:~#
```

How to see where they are running?

```
Manager# docker service ps mydb
```

```

root@Manager:~# docker service create --name mydb --replicas 3 -e MYSQL_ROOT_PASSWORD=sunil mysql:5
uddge25mpmbmr906esm7yvww
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
root@Manager:~#
root@Manager:~#
root@Manager:~# docker service ps mydb
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
ERROR      PORTS
2qg8nrxaf89  mydb.1   mysql:5    Manager   Running      Running 16 seconds
ago
2feqau9w9btc  mydb.2   mysql:5    worker1   Running      Running 17 seconds
ago
wpctkj10gp2z  mydb.3   mysql:5    worker2   Running      Running 16 seconds
ago
root@Manager:~#

```

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

1. tomee service

2. mydb

Manager# docker service ls

```

root@Manager:~# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
uddge25mpmb  mydb     replicated  3/3      mysql:5
u2buu3rxyye0  webserver  replicated  5/5      toomee:latest  *:9090->8080/t
cp
root@Manager:~#

```

Docs.docker.com –ca se service related commands

+++++

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

Now,

Manager# docker service ps mydb

```
root@Manager:~# docker service ps mydb
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
ERROR      PORTS
2qg8nrxaf89  mydb.1    mysql:5   Manager   Running       Running 2 minutes ago
go
2feqau9w9btc  mydb.2    mysql:5   Worker1   Running       Running 2 minutes ago
go
wpctkj10gp2z  mydb.3    mysql:5   Worker2   Running       Running 2 minutes ago
go
root@Manager:~#
```

We can see one container is running in Manager machine

Manager# docker container ls

```
root@Manager:~# docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
           NAMES
67238f47bc60      mysql:5            "docker-entrypoint.s..." 3 minutes ago     Up 3 minutes
utes      3306/tcp, 33060/tcp
8851189bc4b4      tomee:latest       "catalina.sh run"      16 minutes ago   Up 16 minutes
nutes      8080/tcp
7a7e20bac140      tomee:latest       "catalina.sh run"      16 minutes ago   Up 16 minutes
nutes      8080/tcp
root@Manager:~# |
```

(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

```
root@Manager:~# docker rm -f 67238f47bc60
67238f47bc60
root@Manager:~#
```

Now lets check the mydb service

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

ID	NAME	IMAGE	NODE PORTS	DESIRED STATE	CURRENT STATE
zpu7pnyehe5s	mydb.1	mysql:5	Manager	Running	Running 8 second
2qg8nrxaf89	_ mydb.1	mysql:5	Manager	shutdown	Failed 14 second
2feqau9w9btc	mydb.2	mysql:5	worker1	Running	Running 4 minute
wpctkj10gp2z	mydb.3	mysql:5	worker2	Running	Running 4 minute

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
```

```
root@Manager:~# docker service create --name appserver -p 8080:80 --replicas 5 nginx
nnh09pgy5b5rm60wwdczqx94k
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
root@Manager:~#
```

```
# docker service ps appserver
```

```
root@Manager:~# docker service ps appserver
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT ST
ATE         appserver.1  nginx:latest  worker1   Running     Running  25
t5d9vvb5f21k
seconds ago
s00j2ukobi2h
seconds ago
xx51c43p4mgx
seconds ago
130t2jyydp3r
seconds ago
c77f3801eup3
seconds ago
root@Manager:~# |
```

Scale up

Command to scale

```
# docker service scale appserver=10
```

```
root@Manager:~# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
nnh09pgy5b5r  appserver  replicated  10/10    nginx:latest *:8080->80/tcp
uddge25mpmbb  mydb      replicated  3/3      mysql:5
u2buu3rxyye0  webserver replicated  5/5      tomee:latest *:9090->8080/t
cp
root@Manager:~#
```

To check

```
# docker service ps appserver
```

Scale down: Now I want only two containers

```
# docker service scale appserver=2
```

```
root@Manager:~# docker service scale appserver=2
appserver scaled to 2
overall progress: 0 out of 2 tasks
1/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
overall progress: 2 out of 2 tasks
1/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
overall progress: 2 out of 2 tasks
1/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
overall progress: 2 out of 2 tasks
1/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
overall progress: 2 out of 2 tasks
2/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
overall progress: 2 out of 2 tasks
overall progress: 2 out of 2 tasks
1/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
2/2: Failed to find a load balancer IP to use for network: ixclgzooouol4tj6xqllh.
verify: Waiting 3 seconds to verify that tasks are stable...
```

To check

```
# docker service ps appserver
```

```
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      *:3306->3306/tcp
u2buu3rxyye0  webserver replicated  5/5       tomee:latest *:9090->8080/tcp
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:~# docker node ls
ID          HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
va02k63fhr4x342uqhec9dgpc *  Manager  Ready   Active        Leader
  20.10.7
j98tea7gt2tiovjru45040ui4  Worker1  Ready   Active
  20.10.7
xtxqptswvqje191cdfgv5082b  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
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
va02k63fhr4x342uqhec9dgpc * 20.10.7	Manager	Ready	Active	Leader
j98tea7gt2tiovjru45040ui4 20.10.7	Worker1	Ready	Drain	
xtxqptswvqje191cdfgv5082b 20.10.7	Worker2	Ready	Active	

To add the node

```
# docker node update --availability active Worker1
```

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

```
# docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
va02k63fhr4x342uqhec9dgpc * 20.10.7	Manager	Ready	Active	Leader
j98tea7gt2tiovjru45040ui4 20.10.7	Worker1	Ready	Active	
xtxqptswvqje191cdfgv5082b 20.10.7	Worker2	Ready	Active	

2nd Way (Node can leave)

Lets Connect to worker2 from git bash

```
Worker2# docker swarm leave
```

```
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
va02k63fhr4x342uqhec9dgpc *  Manager  Ready   Active        Leader
j98tea7gt2tiovjru45040ui4  Worker1  Ready   Active
xtxqptswvqje191cdcfgv5082b  Worker2  Down    Active
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 performed 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:~# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE      PORTS
giqgtvfy3rsd  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

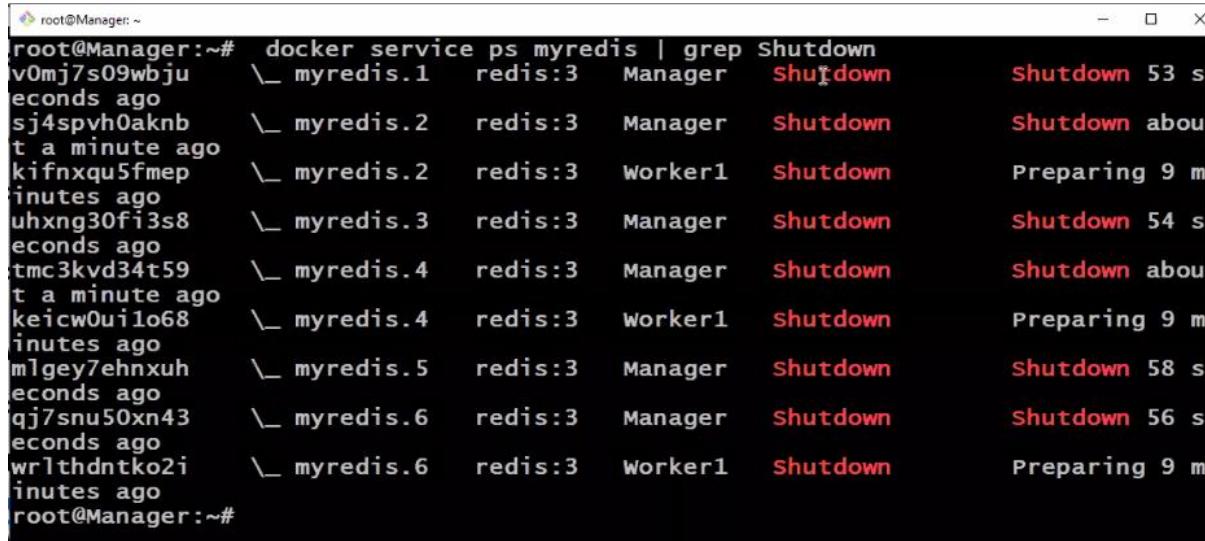
```
root@Manager:~#
root@Manager:~# docker service ps myredis
ID          NAME      IMAGE      NODE      DESIRED STATE  |
root@Manager:~#
```

Time Ago	ID	Name	Image	Node	Desired State
1 second ago	kifnxqu5fmep	_ myredis.2	redis:3	Worker1	Shutdown
1 minute ago	yjvh9bzoaqbr	myredis.3	redis:4	Manager	Running
1 second ago	uhxng30fi3s8	_ myredis.3	redis:3	Manager	Shutdown
1 second ago	y5mkvak9s48n	myredis.4	redis:4	Manager	Running
1 second ago	tmc3kvd34t59	_ myredis.4	redis:3	Manager	Shutdown
1 second ago	keicw0ui068	_ myredis.4	redis:3	Worker1	Shutdown
1 minute ago	uviw8wx7gsi4	myredis.5	redis:4	Manager	Running
1 second ago	m1gey7ehnxuh	_ myredis.5	redis:3	Manager	Shutdown
1 second ago	10pdbtcysiu4	myredis.6	redis:4	Manager	Running
1 second ago	qj7snu50xn43	_ myredis.6	redis:3	Manager	Shutdown
1 minute ago	wrlthdntko2i	_ myredis.6	redis:3	Worker1	Shutdown
	root@Manager:~#				Preparing 8 m

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 )
```



```
root@Manager:~# docker service ps myredis | grep Shutdown
v0mj7s09wju    \_ myredis.1    redis:3   Manager  Shutdown      Shutdown 53 s
econds ago
sj4spvh0aknb  \_ myredis.2    redis:3   Manager  Shutdown      Shutdown about
t a minute ago
kifnxqu5fmep  \_ myredis.2    redis:3   Worker1 Shutdown      Preparing 9 m
inutes ago
uhxng30fi3s8  \_ myredis.3    redis:3   Manager  Shutdown      Shutdown 54 s
seconds ago
tmc3kvdi34t59  \_ myredis.4    redis:3   Manager  Shutdown      Shutdown about
t a minute ago
keicw0ui1o68  \_ myredis.4    redis:3   Worker1 Shutdown      Preparing 9 m
inutes ago
mlgey7ehnxuh  \_ myredis.5    redis:3   Manager  Shutdown      Shutdown 58 s
seconds ago
qj7snu50xn43  \_ myredis.6    redis:3   Manager  Shutdown      Shutdown 56 s
seconds ago
wr1thdntko2i  \_ myredis.6    redis:3   Worker1 Shutdown      Preparing 9 m
inutes ago
root@Manager:~#
```

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

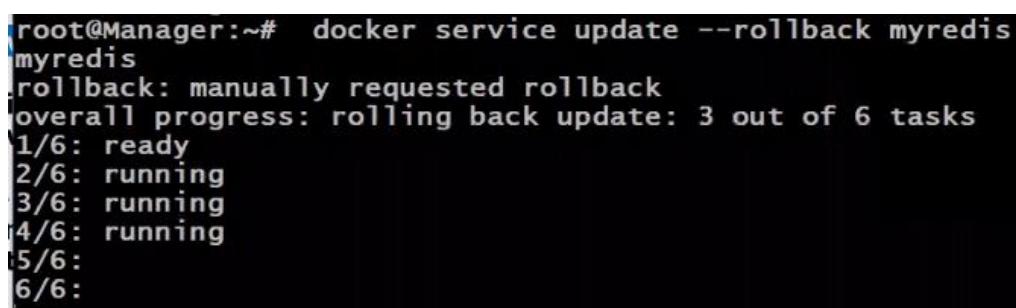


ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
mzgk1gbj3es3	myredis.1	redis:4	Manager	Running	Running about
a minute ago	ee0kezcrm0vy	myredis.2	redis:4	Manager	Running
a minute ago	yjh9bzoaqbr	myredis.3	redis:4	Manager	Running
a minute ago	y5mkvak9s48n	myredis.4	redis:4	Manager	Running
a minute ago	uviw8wx7gsi4	myredis.5	redis:4	Manager	Running
a minute ago	10pdbtciyu4	myredis.6	redis:4	Manager	Running
a minute ago	root@Manager:~#				

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

Performing rolling rollback , to downgrade to redis:3 version

```
# docker service update --rollback myredis
```



```
root@Manager:~# docker service update --rollback myredis
myredis
rollback: manually requested rollback
overall progress: rolling back update: 3 out of 6 tasks
1/6: ready
2/6: running
3/6: running
4/6: running
5/6:
6/6:
```

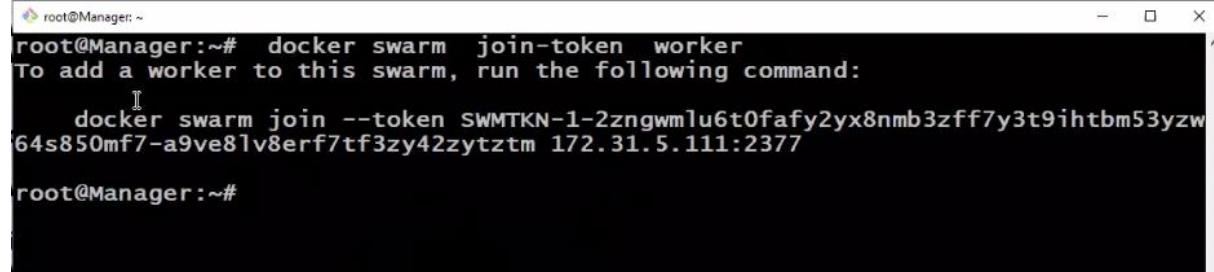
To check redis:3 is running with 6 replicas and other version are shutdown.

```
# docker service ps myredis
```

```
+++++
```

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

To generate the command



```
root@Manager:~# docker swarm join-token worker
To add a worker to this swarm, run the following command:
  docker swarm join --token SWMTKN-1-2zngwmlu6t0fafy2yx8nmb3zff7y3t9ihtbm53yzw
64s850mf7-a9ve8lv8erf7tf3zy42zyztm 172.31.5.111:2377
root@Manager:~#
```

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

```
docker swarm join --token SWMTKN-1-
0etsmfa26vreetyq278q8ohhi73il7j1lpnrzzlowuld1r8yex-9x04pjmiq85jxjzjayzglh1c
172.31.27.151:2377
```

```
+++++
```

To add a new machine as a manager

```
# docker swarm join-token manager
```

```
docker swarm join --token SWMTKN-1-
5wbamgr8x7gxabwtlm1j1i91bm5ilzotgna6bc0edubtwtxi1-3jmzi67qdn5aawvielcng2e4
172.31.34.112:2377
```

```
+++++
```

If there are two managers, one will be leader

If hardware failed in one manager then 2nd manager will come online(high availability)

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

Decision of which machine should be leader is automatic.

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

We cant decide which one is on priority

```
+++++
```

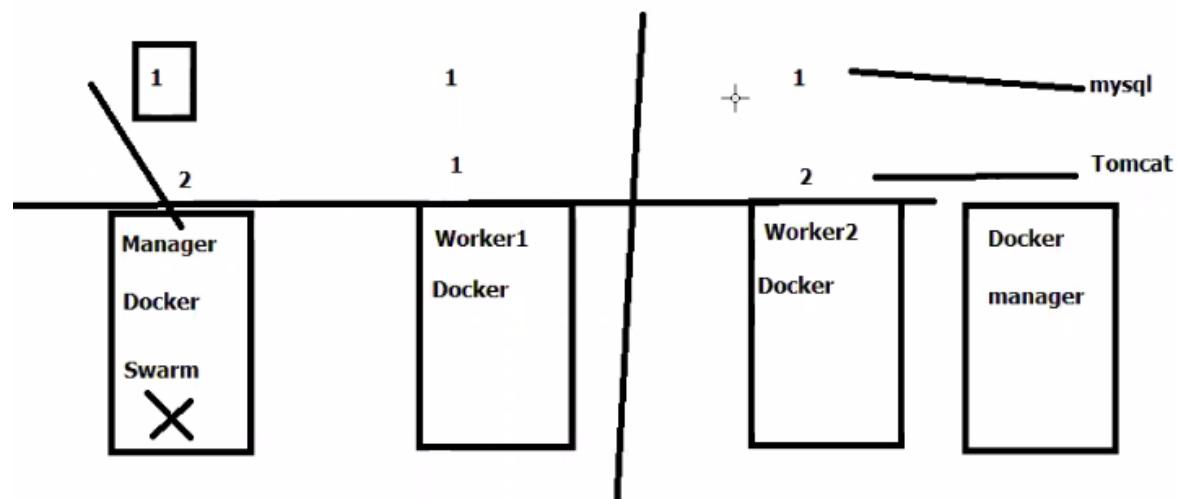
To promote worker1 as a manager node

```
# 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

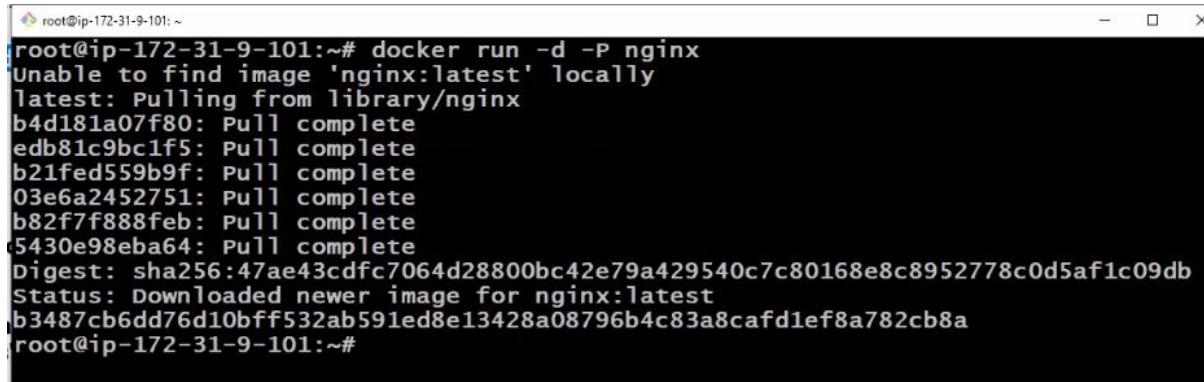
- Understanding Docker Networking

- 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 so creates its with own name)

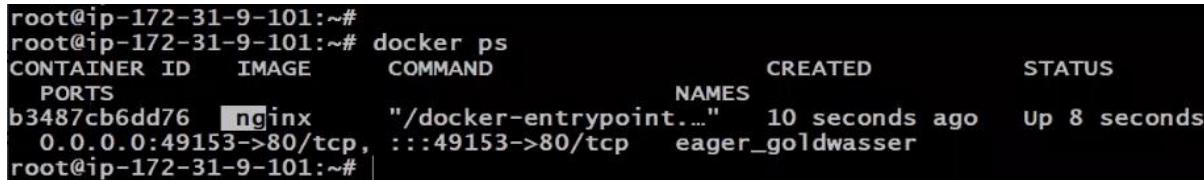


```
root@ip-172-31-9-101:~# docker run -d -P nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
b4d181a07f80: Pull complete
edb81c9bc1f5: Pull complete
b21fed559b9f: Pull complete
03e6a2452751: Pull complete
b82f7f888feb: Pull complete
5430e98eba64: Pull complete
Digest: sha256:47ae43cdfc7064d28800bc42e79a429540c7c80168e8c8952778c0d5af1c09db
Status: Downloaded newer image for nginx:latest
b3487cb6dd76d10bff532ab591ed8e13428a08796b4c83a8cafd1ef8a782cb8a
root@ip-172-31-9-101:~#
```

docker ps

(we can see nginx container running)

Take note of container id and inspect the container



```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              NAMES
          Ports
b3487cb6dd76        nginx              "/docker-entrypoint...."   10 seconds ago    Up 8 seconds   eager_goldwasser
root@ip-172-31-9-101:~#
```

docker inspect b3487cb6dd76

eager_goldwasser= name of container

Note: we can see the IP address

```

root@ip-172-31-9-101:~#
{
    "MacAddress": "02:42:ac:11:00:02",
    "Networks": [
        {
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID": "74079e0a3e75f3e8f250bf3c8f5a81a7ccad7f65941477db80d3666f97fa0ffb",
                "EndpointID": "217fffe5508b993100c15b7b292d2298617892cabaf9593f0bd74e3383b1a70",
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.2",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:11:00:02",
                "DriverOpts": null
            }
        }
    ]
}
root@ip-172-31-9-101:~#

```

"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

Name	Instance ID	Instance state	Instance type
DockerHost	i-07b4053729c6d266e	Running	t2.micro

Instance: i-07b4053729c6d266e (DockerHost)											
Details Security Networking Storage Status checks Monitoring Tags											
Instance summary <table border="1"> <tr> <td>Instance ID</td> <td>Public IPv4 address</td> <td>Private IPv4 addresses</td> </tr> <tr> <td>i-07b4053729c6d266e (DockerHost)</td> <td>15.206.128.242 open address</td> <td>172.31.9.101</td> </tr> <tr> <td>Instance state</td> <td>Public IPv4 DNS</td> <td>Private IPv4 DNS</td> </tr> </table>			Instance ID	Public IPv4 address	Private IPv4 addresses	i-07b4053729c6d266e (DockerHost)	15.206.128.242 open address	172.31.9.101	Instance state	Public IPv4 DNS	Private IPv4 DNS
Instance ID	Public IPv4 address	Private IPv4 addresses									
i-07b4053729c6d266e (DockerHost)	15.206.128.242 open address	172.31.9.101									
Instance state	Public IPv4 DNS	Private IPv4 DNS									

Do you think, they both are the same?

NO

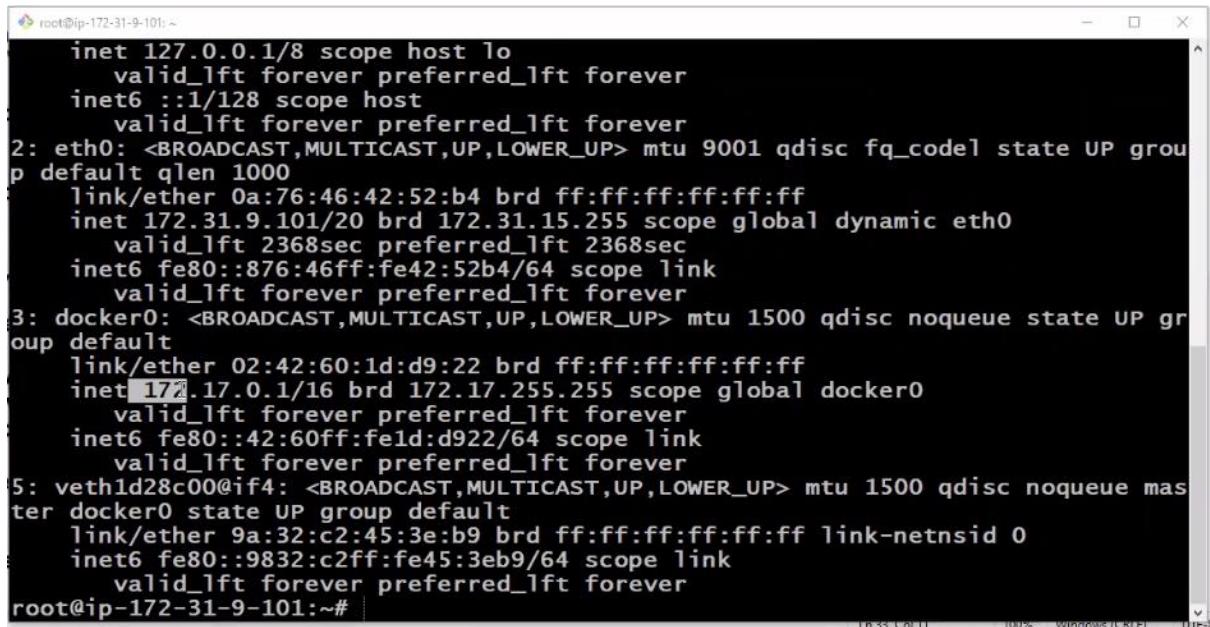
How container get IP address?

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 )
```



```
root@ip-172-31-9-101: ~
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default qlen 1000
    link/ether 0a:76:46:42:52:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.31.9.101/20 brd 172.31.15.255 scope global dynamic eth0
        valid_lft 2368sec preferred_lft 2368sec
    inet6 fe80::876:46ff:fe42:52b4/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:60:1d:d9:22 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:60ff:feld:d922/64 scope link
        valid_lft forever preferred_lft forever
5: veth1d28c00@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 9a:32:c2:45:3e:b9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::9832:c2ff:fe45:3eb9/64 scope link
        valid_lft forever preferred_lft forever
root@ip-172-31-9-101:~#
```

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:~# 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
e069dbb52ddd064e15642590dd20f5d7431592e9b33c0d8f8577132889e81a7
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:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS
PORTS
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",

```

root@ip-172-31-9-101:~#
"MacAddress": "02:42:ac:11:00:03",
"Networks": [
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "74079e0a3e75f3e8f250bf3c8f5a81a7ccad7f65941477
db80d3666f97fa0ffb",
        "EndpointID": "c2c363453e7e1dfffb18c3605fa3b37ec0d5606ef9e1fd
39af5de0e80e62e2f4f",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.31",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:03",
        "DriverOpts": null
    }
}
]
root@ip-172-31-9-101:~#

```

+++++

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 contains ping utility.

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

```

root@ip-172-31-9-101:~# docker run -d --name tester1 busybox:1.28 sleep 3600
Unable to find image 'busybox:1.28' locally
1.28: Pulling from library/busybox
07a152489297: Pull complete
Digest: sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416dea4f41046e0f37d47
Status: Downloaded newer image for busybox:1.28
bfb32e53b26017e6cf40f68ad4adb83fa52b6a04e085ee0e0d00bdfce35149c7
root@ip-172-31-9-101:~#

```

docker ps (we can see 3 container running)

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS			NAMES	
bfb32e53b260	busybox:1.28	"sleep 3600"	13 seconds ago	Up 13 se
conds			tester1	
e069dbb52ddd	redis	"docker-entrypoint.s..."	2 minutes ago	Up 2 min
utes	0.0.0.0:49154->6379/tcp, :::49154->6379/tcp		sad_rosalind	
b3487cb6dd76	nginx	"/docker-entrypoint..."	8 minutes ago	Up 8 min
utes	0.0.0.0:49153->80/tcp, :::49153->80/tcp		eager_goldwasser	

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

```
root@ip-172-31-9-101:~# docker exec bfb32e53b260 ping e069dbb52ddd
ping: bad address 'e069dbb52ddd'           |
root@ip-172-31-9-101:~# |
```

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              NAMES
PORTS
bfb32e53b260        busybox:1.28     "sleep 3600"           2 minutes ago      Up 2 min          tester1
e069dbb52ddd        redis              "docker-entrypoint.s..."   4 minutes ago      Up 4 min          sad_rosalind
b3487cb6dd76        nginx             "/docker-entrypoint..."  10 minutes ago     Up 10 min         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"
    ]
  }
]
```

```
root@ip-172-31-9-101: ~
"MacAddress": "02:42:ac:11:00:03",
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "74079e0a3e75f3e8f250bf3c8f5a81a7ccad7f65941477
db80d3666f97fa0ffb",
        "EndpointID": "c2c363453e7e1dffbb18c3605fa3b37ec0d5606ef9e1fd
39af5de0e80e62e2f4f",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.3",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:03",
        "DriverOpts": null
    }
}
```

Take note of container IP of redis

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

```
root@ip-172-31-9-101:~# docker exec bfb32e53b260 ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.117 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.078 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.076 ms
64 bytes from 172.17.0.2: seq=3 ttl=64 time=0.078 ms
64 bytes from 172.17.0.2: seq=4 ttl=64 time=0.075 ms
64 bytes from 172.17.0.2: seq=5 ttl=64 time=0.078 ms
^C
root@ip-172-31-9-101:~#
```

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

```
root@ip-172-31-9-101:~# docker exec bfb32e53b260 ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.343 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.090 ms
64 bytes from 172.17.0.3: seq=2 ttl=64 time=0.093 ms
64 bytes from 172.17.0.3: seq=3 ttl=64 time=0.096 ms
^C
root@ip-172-31-9-101:~# |
```

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

```
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:60:1d:d9:22 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:60ff:fe1d:d922/64 scope link
        valid_lft forever preferred_lft forever
```

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 container 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)
```

```
root@ip-172-31-9-101:~# docker rm -f $(docker ps -aq)
bf32e53b260
e069dbb52ddd
lb3487cb6dd76
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# |
```

To create a new network

```
# docker network create sunil
```

we get network id

```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# docker network create sunil
5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af
root@ip-172-31-9-101:~#
```

5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af

To get the more network information

```
# docker inspect sunil
```

```
root@ip-172-31-9-101: ~
"EnableIPv6": false,
"IPAM": {
    "Driver": "default",
    "options": {},
    "Config": [
        {
            "Subnet": "172.18.0.0/16",
            "Gateway": "172.18.0.1"
        }
    ]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {},
"Options": {},
"Labels": {}
]
]
root@ip-172-31-9-101: ~#
```

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
```

```
root@ip-172-31-9-101:~# docker run -d -P --name app --net sunil nginx f01c08280a3db4a1983a305f3eec61a5602d29cc5905e9e45f0109955f1054b3 root@ip-172-31-9-101:~# root@ip-172-31-9-101:~# |
```

(--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              NAMES
f01c08280a3d        nginx              "/docker-entrypoint..."   21 seconds ago    Up 19 seconds   app
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",
    "EndpointID": "1fc716b2185d9f1985e1ff92aaaf2b6f5693dea149bb1d",
    "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 sleep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~#

```

docker ps (we can see two containers)

```

root@ip-172-31-9-101:~# docker run -d -P --name mybox --net sunil busybox:1.28 sleep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~# docker ps
CONTAINER ID IMAGE COMMAND NAMES CREATED STATUS
cb54d7b76d1a busybox:1.28 "sleep 3600" mybox 8 seconds ago Up 7 seconds ago
f01c08280a3d nginx "/docker-entrypoint..." 2 minutes ago Up 2 minutes ago
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:~# docker run -d -P --name mybox --net sunil busybox:1.28 sleep 3600
cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe
root@ip-172-31-9-101:~# docker ps
CONTAINER ID IMAGE COMMAND NAMES CREATED STATUS
cb54d7b76d1a busybox:1.28 "sleep 3600" mybox 8 seconds ago Up 7 seconds ago
f01c08280a3d nginx "/docker-entrypoint..." 2 minutes ago Up 2 minutes ago
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
```

```
root@ip-172-31-9-101:~
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS
cb54d7b76d1a	busybox	"sleep 3600"	NAMES	8 seconds ago	Up 7 seconds
cb54d7b76d1a	busybox:1.28	"sleep 3600"	mybox	2 minutes ago	Up 2 minutes
f01c08280a3d	nginx	"/docker-entrypoint..."	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 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
/ # ping 172.18.0.2
PING 172.18.0.2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.101 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.090 ms
```

Yes!! It pings

I want to ping with name

```
root@ip-172-31-9-101:~
```

```
valid_lft forever preferred_lft forever
/ # ping 172.18.0.2
PING 172.18.0.2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.101 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.090 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.080 ms
^C
--- 172.18.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.080/0.090/0.101 ms
/ #
/ #
/ # ping app
PING app (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.056 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.080 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.074 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.078 ms
64 bytes from 172.18.0.2: seq=4 ttl=64 time=0.071 ms
^C
--- app ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.056/0.071/0.080 ms
/ # |
```

```
/# 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
[{"Id": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af",
 "Created": "2021-06-28T05:28:14.078873399Z",
 "Scope": "local",
 "Driver": "bridge",
 "EnableIPv6": false,
 "IPAM": {
     "Driver": "default",
     "Options": {},
     "Config": [
         {
             "Subnet": "172.18.0.0/16",
             "Gateway": "172.18.0.1"
         }
     ]
 }, "Name": "sunil", "NetworkSettings": {"Bridge": "bridge", "ContainerID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af", "EndpointID": "cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "IPv6Address": ""}, "ConfigFrom": {}, "ConfigOnly": false, "Containers": {"cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe": {"Name": "mybox", "EndpointID": "c5b2f487de4e1647fbab197cd5381c7a26fcc13795ed5a85ff0f9e0c9771c2c8", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "IPv6Address": ""}, "f01c08280a3db4a1983a305f3eec61a5602d29cc5905e9e45f0109955f1054b3": {"Name": "app", "EndpointID": "1fc716b2185d9f1985e1ff92aaaf2b6f5693dea149bb1d67c74d66678bf487845", "MacAddress": "02:42:ac:12:00:02", "IPv4Address": "172.18.0.2/16", "IPv6Address": ""}}}], "NetworkSettings": {"Bridge": "bridge", "ContainerID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af", "EndpointID": "cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "IPv6Address": ""}}
```

```
root@ip-172-31-9-101:~# docker inspect sunil
[{"Id": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af",
 "Created": "2021-06-28T05:28:14.078873399Z",
 "Scope": "local",
 "Driver": "bridge",
 "EnableIPv6": false,
 "IPAM": {
     "Driver": "default",
     "Options": {},
     "Config": [
         {
             "Subnet": "172.18.0.0/16",
             "Gateway": "172.18.0.1"
         }
     ]
 }, "Name": "sunil", "NetworkSettings": {"Bridge": "bridge", "ContainerID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af", "EndpointID": "cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "IPv6Address": ""}, "ConfigFrom": {}, "ConfigOnly": false, "Containers": {"cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe": {"Name": "mybox", "EndpointID": "c5b2f487de4e1647fbab197cd5381c7a26fcc13795ed5a85ff0f9e0c9771c2c8", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "IPv6Address": ""}, "f01c08280a3db4a1983a305f3eec61a5602d29cc5905e9e45f0109955f1054b3": {"Name": "app", "EndpointID": "1fc716b2185d9f1985e1ff92aaaf2b6f5693dea149bb1d67c74d66678bf487845", "MacAddress": "02:42:ac:12:00:02", "IPv4Address": "172.18.0.2/16", "IPv6Address": ""}}}], "NetworkSettings": {"Bridge": "bridge", "ContainerID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c3edf61f742bd1507af", "EndpointID": "cb54d7b76d1af204107f8243fa72584d11f778b959d5cc61d933a6df593b9efe", "MacAddress": "02:42:ac:12:00:03", "IPv4Address": "172.18.0.3/16", "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 mentioned network information, by default it takes docker0 network.

```
# docker ps ( to see the list of container )
```

```
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 seconds nervous_tereshkova  
cb54d7b76d1a busybox:1.28 "sleep 3600" 3 minutes ago Up 3 minutes mybox  
f01c08280a3d nginx "/docker-entrypoint..." 5 minutes ago Up 5 minutes 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:~# docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS  
PORTS NAMES  
145c4da0603c redis "docker-entrypoint.s..." 11 seconds ago Up 10 seconds nervous_tereshkova  
cb54d7b76d1a busybox:1.28 "sleep 3600" 3 minutes ago Up 3 minutes mybox  
f01c08280a3d nginx "/docker-entrypoint..." 5 minutes ago Up 5 minutes 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
root@ip-172-31-9-101:~# slesd
```

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

```
# docker inspect 14fc4da0603c
```

```
root@ip-172-31-9-101:~#
```

```
        "sunil": {
            "IPAMConfig": {},
            "Links": null,
            "Aliases": [
                "145c4da0603c"
            ],
            "NetworkID": "5bed09831ad7dcc2075d6011c110c4f11fad839720705c
3edf61f742bd1507af",
            "EndpointID": "905566bf641275cd0a55ba74ed1e114d8bc7ccc799f39
4862c6dec4d94aebea70",
            "Gateway": "172.18.0.1",
            "IPAddress": "172.18.0.4",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "02:42:ac:12:00:04",
            "DriverOpts": {}
        }
    }
}
]
```

Now, its part of two networks

1.sunil

2.docker0

Bcz one container can be part of multiple n/w.

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
```

```
root@ip-172-31-9-101:~# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
74079e0a3e75    bridge    bridge      local
496daa604740    host      host       local
04b9051d7673    none      null       local
5bed09831ad7    sunil     bridge      local
root@ip-172-31-9-101:~# |
```

+++++

Types of networks in Docker

```
root@ip-172-31-9-101:~# docker network ls
NETWORK ID     NAME      DRIVER      SCOPE
74079e0a3e75   bridge    bridge      local
496daa604740   host      host       local
04b9051d7673   none      null       local
5bed09831ad7   sunil    bridge      local
root@ip-172-31-9-101:~# |
```

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

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

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 containers.

```
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
```

```
root@ip-172-31-9-101:~# docker run -d -P --net host nginx
f1716a08eaff52bac4eb8978d5a16c0e71bbc78924077827224bb8e9b8fb93ad
root@ip-172-31-9-101:~#
```

```
# docker container ps
```

```
root@ip-172-31-9-101:~# docker run -d -P --net host nginx
f1716a08eaff52bac4eb8978d5a16c0e71bbc78924077827224bb8e9b8fb93ad
root@ip-172-31-9-101:~# docker container ps
CONTAINER ID   IMAGE      COMMAND          CREATED        STATUS
PORTS NAMES
f1716a08eaff  nginx     "/docker-entrypoint..."  5 seconds ago  Up 5 seconds
stupefied_archimedes
root@ip-172-31-9-101:~#
```

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.

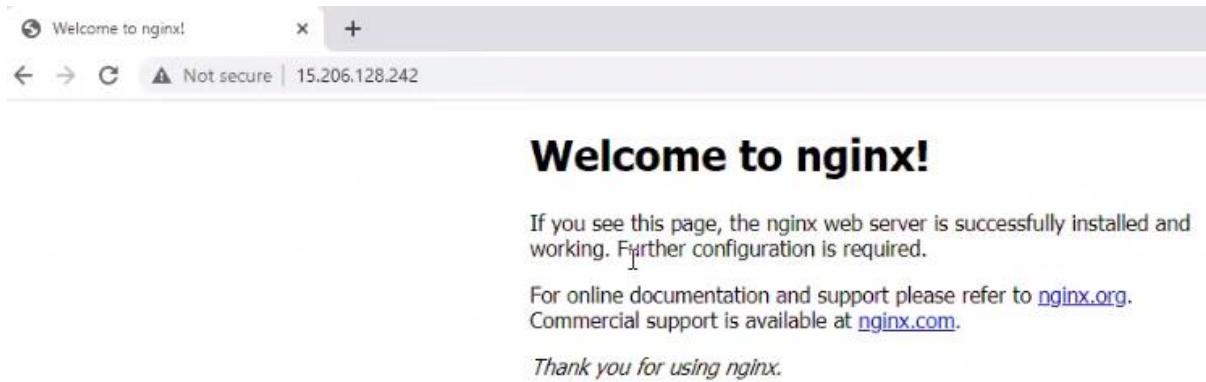
Take the public IP of the Dockerhost and access using default port.

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:~# docker ps
CONTAINER ID        IMAGE       COMMAND                  CREATED             NAMES
f1716a08eaff      nginx      "/docker-entrypoint..."   About a minute ago   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
481345d3c7e6        nginx      "/docker-entrypoint..."   23 seconds ago    Exited (1) 2
seconds ago
f1716a08eaff        nginx      "/docker-entrypoint..."   2 minutes ago     Up 2 minutes
stupified_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:~# docker logs 481345d3c7e6
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf

```

```

2021/06/28 05:49:43 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already 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 in 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 already 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 in 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 already 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 in 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
```

```
root@ip-172-31-9-101:~# docker run -d -P --net none redis
582f1f42843141e722334de20bb13d23e19e8b245f739803f99de00e53d35ad5
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# |
```

```
# docker ps
```

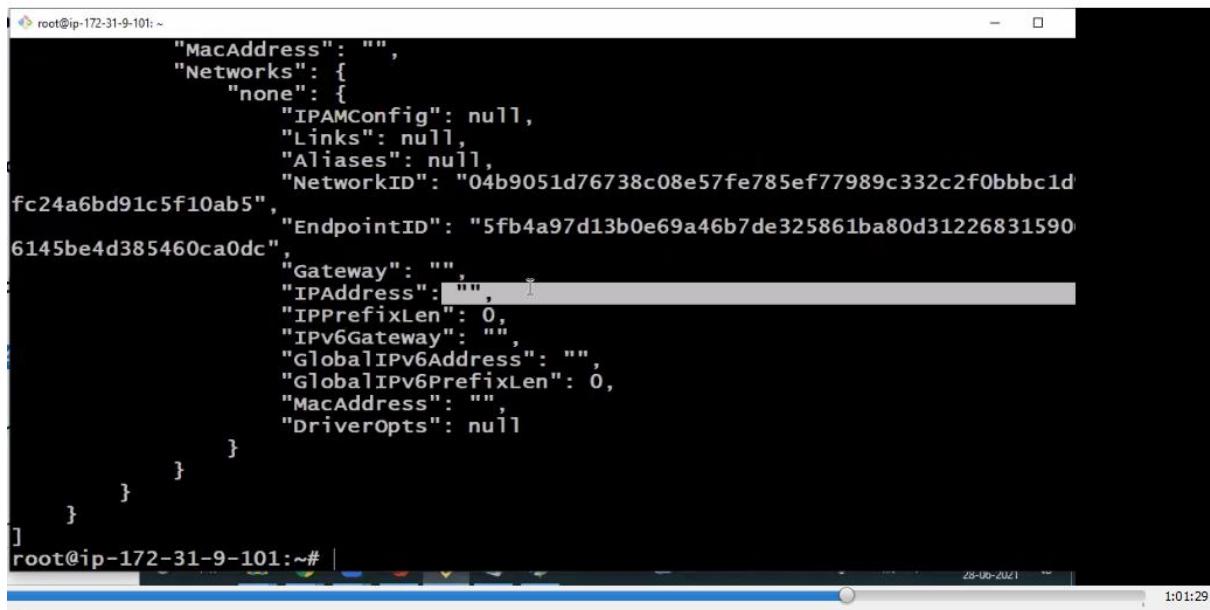
Take note of container id

```
fadebe0ad3cf
```

```
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
582f1f428431        redis              "docker-entrypoint.s..."   8 seconds ago      Up 7 seconds
gracious_goldberg
f1716a08eaff        nginx              "/docker-entrypoint...."  4 minutes ago     Up 4 minutes
stupefied_archimedes
root@ip-172-31-9-101:~# |
```

```
# docker inspect fadebe0ad3cf
```

We do not see any IP for the container



A screenshot of a terminal window titled "root@ip-172-31-9-101: ~". The window displays a JSON object representing a Docker network configuration. The object includes fields for MacAddress, Networks (specifically "none"), IPAMConfig, Links, Aliases, NetworkID, EndpointID, Gateway, IPAddress (redacted), IPPrefixLen, IPv6Gateway, GlobalIPv6Address, GlobalIPv6PrefixLen, MacAddress, and DriverOpts. The IPAddress field is explicitly shown as an empty string. The terminal window has a dark background and light-colored text. The bottom right corner shows the date and time as "28-06-2021" and "1:01:29".

```
"IPAddress": "",  
  "Networks": {  
    "none": {  
      "IPAMConfig": null,  
      "Links": null,  
      "Aliases": null,  
      "NetworkID": "04b9051d76738c08e57fe785ef77989c332c2f0bbbc1d  
fc24a6bd91c5f10ab5",  
      "EndpointID": "5fb4a97d13b0e69a46b7de325861ba80d31226831590  
6145be4d385460ca0dc",  
      "Gateway": "",  
      "IPAddress": "",  
      "IPPrefixLen": 0,  
      "IPv6Gateway": "",  
      "GlobalIPv6Address": "",  
      "GlobalIPv6PrefixLen": 0,  
      "MacAddress": "",  
      "DriverOpts": null  
    }  
  }  
}  
]  
root@ip-172-31-9-101:~#
```

"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 inactive

Swarm: inactive

```
root@ip-172-31-9-101: ~
Log: awslogs fluentd gcplogs gelf journalctl json-file local logentries splunk syslog
Swarm: inactive
Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
Default Runtime: runc
Init Binary: docker-init
containerd version: d71fcfcd7d8303cbf684402823e425e9dd2e99285d
runc version: b9ee9c6314599f1b4a7f497e1f1f856fe433d3b7
init version: de40ad0
Security Options:
  apparmor
  seccomp
    Profile: default
Kernel Version: 5.4.0-1045-aws
Operating System: Ubuntu 18.04.5 LTS
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 978.6MiB
Name: ip-172-31-9-101
ID: 74D4:XUZY:RASG:YC2J:DBQN:T3VN:VJPT:2GAO:ANY5:P6YH:JJBW:ZE30
Docker Root Dir: /var/lib/docker
Debug Mode: false
Registry: https://index.docker.io/v1/
```

I want to initialize swarm

```
# docker swarm init
```

Now, lets see the list of networks

```
root@ip-172-31-9-101: ~# docker swarm init
Swarm initialized: current node (sdd60bwvq1bg7t2ndj5spylud) is now a manager.

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

  docker swarm join --token SWMTKN-1-07ztinrmtnb1it598plzbkiae063rcymxnu9wm3h
  eecwzxfoy-b1rvhlu8cnqqep51sh35n5q9 172.31.9.101:2377

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

root@ip-172-31-9-101:~# |
```

```
# docker network ls
```

```
root@ip-172-31-9-101:~# docker swarm init
Swarm initialized: current node (sdd60bwvq1bg7t2ndj5spylud) is now a manager.

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

    docker swarm join --token SWMTKN-1-07ztinrmtnb1it598plzbkiae063rcymxnuf9wm3h
    eecwzxfoy-b1rvhlu8cnqkeep51sh35n5q9 172.31.9.101:2377

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

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
5bed09831ad7   sunil    bridge      local
root@ip-172-31-9-101:~#
```

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

```
root@ip-172-31-9-101: ~
{
    "ConfigOnly": false,
    "Containers": [
        "ingress-sbox": {
            "Name": "ingress-endpoint",
            "EndpointID": "6cf367b47f4aafc56ce9cc687d13435ed5f1a76b941cb14b412b9e9149fb452",
            "MacAddress": "02:42:0a:00:00:02",
            "IPv4Address": "10.0.0.2/24",
            "IPv6Address": ""
        }
    ],
    "Options": {
        "com.docker.network.driver.overlay.vxlanid_list": "4096"
    },
    "Labels": {},
    "Peers": [
        {
            "Name": "e571c8902f3e",
            "IP": "172.31.9.101"
        }
    ]
}
root@ip-172-31-9-101:~#
```

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
```

```
root@ip-172-31-9-101: ~# docker service create --name s1 --replicas 5 -p 1234:80 nginx
orzf1lnmckivpbsayfe04oxxu
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
root@ip-172-31-9-101:~#
root@ip-172-31-9-101:~# I
root@ip-172-31-9-101:~# |
```

As we have one machine right now.

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

```
# docker ps
```

```
root@ip-172-31-9-101:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
b0cf017c9f78        nginx:latest       "/docker-entrypoint..."   22 seconds ago     Up 16 sec
conds              80/tcp             s1.2.wn1myg09hmxbdnovphrx6ven0
e1b4a32678cb        nginx:latest       "/docker-entrypoint..."   22 seconds ago     Up 17 sec
conds              80/tcp             s1.5.odgnu1nd6anln062d4lssdkxo
6f6d877e9b27        nginx:latest       "/docker-entrypoint..."   22 seconds ago     Up 17 sec
conds              80/tcp             s1.1.tk99dhb1ctih3koq0ile1fpmp
6e90ae2f9475        nginx:latest       "/docker-entrypoint..."   22 seconds ago     Up 19 sec
conds              80/tcp             s1.3.fqrnk5zde6dy2ejkeqd21c368
18f2261b7f84        nginx:latest       "/docker-entrypoint..."   22 seconds ago     Up 19 sec
conds              80/tcp             s1.4.vvvwzus4sqz2lmtauy7mj0z0q
582f1f428431        redis              "docker-entrypoint.s..."  5 minutes ago      Up 5 min
utes               gracious_goldberg
f1716a08eaff        nginx              "/docker-entrypoint..."   9 minutes ago      Up 9 min
utes               stupefied_archimedes
root@ip-172-31-9-101:~#
```

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

Take note of the any (1st) container ID

eaa1e645ae48

docker inspect 96631cf17237

```
root@ip-172-31-9-101:~#
{
    "IPAMConfig": {
        "IPv4Address": "10.0.0.5"
    },
    "Links": null,
    "Aliases": [
        "b0cf017c9f78"
    ],
    "NetworkID": "145uv3idu94qx9iw8dqx4jnq",
    "EndpointID": "f509c8d385e59a87b69603a43d1d48672657e34fbccff
2d8bf2932d61c372910",
    "Gateway": "",
    "IPAddress": "10.0.0.5",
    "IPPrefixLen": 24,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "MacAddress": "02:42:0a:00:00:05",
    "DriverOpts": null
}
}
}
]
root@ip-172-31-9-101:~#
```

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

```
root@ip-172-31-9-101: ~
    "Driver": "default",
    "Options": null,
    "Config": [
        {
            "Subnet": "10.0.1.0/24",
            "Gateway": "10.0.1.1"
        }
    ],
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": null,
    "Options": {
        "com.docker.network.driver.overlay.vxlanid_list": "4097"
    },
    "Labels": null
}
]
root@ip-172-31-9-101:~# |
```

+++++

Lets delete the existing service

```
# docker service rm s1
```

Lets remove all the existing containers

```
# docker rm -f `docker ps -aq`
```

```
root@ip-172-31-9-101: ~
root@ip-172-31-9-101:~# docker service rm s1
s1
root@ip-172-31-9-101:~# docker rm -f `docker ps -aq`
e1b4a32678cb
6f6d877e9b27
6e90ae2f9475
18f2261b7f84
582f1f428431
481345d3c7e6
f1716a08eaff
root@ip-172-31-9-101:~# |
```

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  
ouu7c8j3qy2x0ld4pmmon18nwb  
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:~# docker ps  
CONTAINER ID IMAGE COMMAND  
PORTS NAMES  
35b7d1a30038 nginx:latest "/docker-entrypoint...."  
conds 80/tcp mynginx.3.tp8sgqh6gtfx5z8sq0hjlclal  
8c7071da6e99 nginx:latest "/docker-entrypoint...."  
conds 80/tcp mynginx.2.lz497izbp6k0opfys7ved48ej  
2c7517937c5a nginx:latest "/docker-entrypoint...."  
conds 80/tcp mynginx.1.ydurlalaj4nkwiuq7sickrutex  
7c674846840f nginx:latest "/docker-entrypoint...."  
conds 80/tcp mynginx.4.rt54sc5iohqr3qcyrft8e5lwh  
1c34b0d9526f nginx:latest "/docker-entrypoint...."  
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": "h9w81braafdnacpom6a71y7z9",  
"EndpointID": "6f5a6e024316fc41e8375a3c2d8a222b575721e1c767  
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.

```
+++++
```