Docker 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.

++++++++++++++++++++++++++
Create Ubuntu Machine on
AWS All Traffic - anywhere

Connect using git

bash

https://get.docker.co/

To Install Docker, Go to Root Account And execute below commands
$ sudo su -
# curl -fsSL https://get.docker.com -o get-docker.sh

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

when we start the docker engine "***Docker0*** " is created
Docker0 :- is the network related service and all the containers ip-address are managed by the Docker0 all this stuff we can see by making use of the command called "ifconfig"

How to check the docker is installed or not
docker --version

We should be comfortable with four terms

1) Docker Images

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

2) Docker Containers

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

3) Docker Host

Machine on which docker is installed, is called as Docker host.

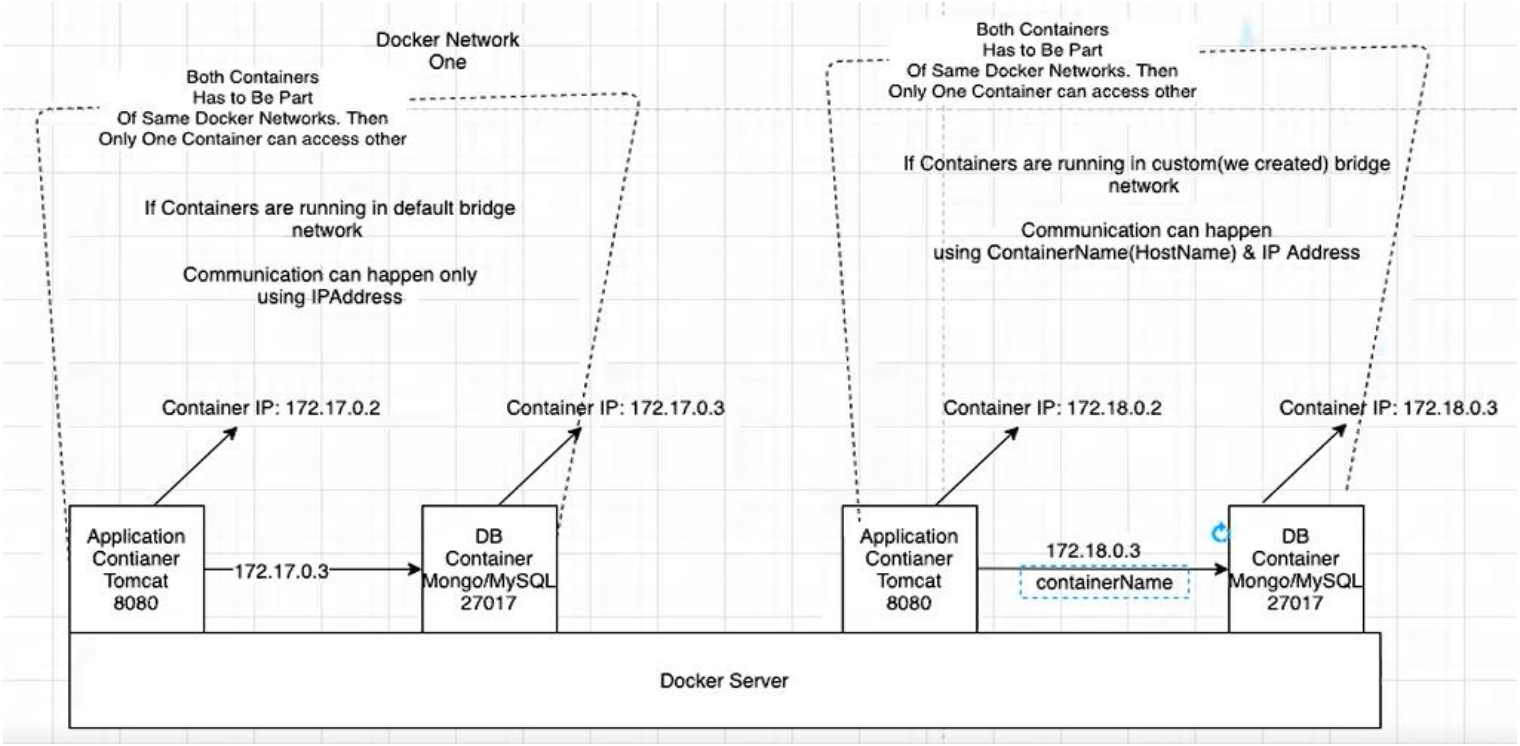
4) Docker Client

Terminal used to run docker run commands (Git bash)

On Linux machine, git bash will work like docker client

***Docker Networking*:**

- If we don't mention network name while creating the container. Containers will be created in default bridge network
- Docker community edition supports 3 kinds of networking (bridge, host, null)
- we can see this by making use of command
  docker network ls



BRIDGE:

it is a kind of network driver.

CIDR (Classless Inter-Domain Routing) IP range is set with their subnet. and basically, this bridge releases the virtual ip-address to each container.

it is a virtual bridge created by docker engine

here we can communicate one-container with another container through ip-address but we cannot communicate by using container-name/id

how we can create our own bridge

syntax

docker network create name-of-the-bridge driver=bridge

EX:

docker network create copart-dev --driver=bridge

how to create the container inside our own bridge

EX:

docker run -it -d --net=copart-dev ubuntu

it will create the ubuntu container inside the copart-dev bridge

can i ping /access the containers from the host machine? ====>***YES.***

if one container can ping another container ip-address? ==>***YES***.

for that we can install network related utility

step 1: docker attach container-id

step 2: update the repo--->apt-get update

step 3: apt-get install -y iputils traceroute net-tools

traceroute 8.8.8.8

**here the containers inside the bridge can communicate with each other but, cannot communicate with another bridge containers. why because bridges have the scope as "LOCAL"

and two different bridges are not communicated with each other here we can see the difference/flow how the ip-address are created and the gateways by making use of command called

- ifconfig
- inside one bridge we can create ***65536*** containers

HOST

this driver mode shares the networking of the host with the container

- docker run -it -d --net=host ubuntu

NULL

If we create containers in null/none network. Container will not have ip-address. we can not Access these containers from outside or from any other container

- docker run -it -d --net=none ubuntu

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.

overlay

whenever we create our own-network at that time overlay network will be created and its driver type is bridge

- docker network create name-of-the-bridge --driver=bridge

here we can communicate one container with another container by making use of ip-address, container-id & container-name

Docker container process in system:

- container is nothing but a isolated user space with their own network, process, mounts
- container is nothing but a writable layer of the running state of the image
- container is nothing but a running state of the image
- container is nothing but a process running on the host to keep a container PMAP alive

Docker Commands

Working on Images

1)To download a docker image
docker pull image-name

2)To see the list of docker images
docker image ls
(or)
docker images

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 customized container
docker commit container-name/container-id new-image-name

To create an image from docker file
docker build -t <image name>

Note:
Image name should have repository details along with Name and version

Public Repository(Docker Hub):
docker build -t <dockerHubUserName>/<AppName>:<version>

Note:
If we don't mention version information, by default it will use latest as version

EX: docker build -t dockerhandson/java-web-app:1

Private Repository(Nexus/JFrog/DTR(docker trusted registry)):

docker build -t <IP/HostNameOfRepos>:<RepoPort>/<AppName>:<version>

EX:

Docker build -t 178.92.34.12:8083/java-web-app:1

Authenticate with Repo:

Public Repo:

docker login -u <USER-NAME> -p <PASSWORD>

EX:

docker login -u dockerhudson -p password

Private Repo:

docker login -u <USER-NAME> -p <PASSWORD> <URL>

EX:

Docker login -u admin -p abcd123@ 178.92.34.12:8083

Push Docker image to Repo:

docker push <image name>

Public Repo:

docker push <dockerHubUserName>/<AppName>:<version>

Private Repo:

docker push <IP/HostNameOfRepos>:<Repo Port>/<AppName>:<version>

Download Image from Repo:

Syntax:

[image name= username/AppName]

docker pull <image name>

Public Repo:

docker pull dockerhandson/java-web-app:1

Private Repo:

Docker pull 178.90.34.12:8083/java-web-app:1

Docker run -d -p 8080:8080 - -name javawebapp dockerhudsun/java-web-app:1

To Access the application:

http://<DockerserverPublicIP>:8080/java-web-app

7)To search for a docker image

docker search image-name

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

docker system prune -a

What is dangling images in Docker?

The image in which doesn't have ***Repository Mapping*** *(or)* ***Tag***

Ex:- *docker rmi -f dockerhudson/java-web-app:1*

Repository	***Tag***	***Image ID***	***Created***	***size***
<none>	*<none>*	*6033567806df*	*23hours ago*	*499MB*

Here we tried to remove the image forcefully with image-name but there is a container, so it was not able to delete the image just it has ***untagged***

How we can see dangling images in docker?

docker images - -filter dangling=true

Note:

We cannot remove the images if there are running container for the image. We cannot force delete images if there is running container

If container is in stopped(exited) state we can force delete image for the stopped container.

What is the working directory of Docker?

/var/lib/docker

How we can move/copy images from one server to another server without repo?

- In source server(where we have image)
- Save image(All the layers) as a **tar file** by using the command
  docker save -o <filename>.tar <ImageName/ID>
- Then **scp tar file** from Source to Destination server
  scp /path/to/local/file username@remote-server:/path/to/destination/
- And in destination server we can run below command?
  docker lode -i <FileName>.tar

Working on containers:

1)To see the list of all running containers

docker container ls
(or)
docker ps

2)To see the list of running and stopped containers

docker ps -a

3)To start a container

docker start container-name/container-id

4)To stop a running container

docker stop container-name/container-id

5)To restart a running container

docker restart container-name/container-id

6)To restart after 10 seconds

docker restart -t 10 container-name/container-id

7)To delete a stopped container

docker rm container-name/container-id

8)To delete a running container

docker rm -f container-name/container id

9)To stop all running containers

docker stop $(docker ps -aq)

10)To restart all containers

docker restart $(docker ps -aq)

11)To remove all stopped containers

docker rm $(docker ps -aq)

12)To remove all container's (running and stopped)

docker rm -f $(docker ps -aq)

13)To see the logs generated by a container (or)
Trouble shoot/Debug application which is running as a container

docker logs container-name/container-id
docker logs - -tail <NoOfLines> <container-id/name>

14)To see the ports used by a container

docker port container-name/container-id

15)To get detailed info about a container

docker inspect container-name/container-id

16)Docker attach will attach container process or shell to host server
To go into the shell of a running container which is moved into background

docker attach <container-name/id>
if we want to come out without stopping the process ctrl+pq

17)To execute any command in a container

docker exec <container name/id> <cmd>
EX:
docker exec javawebapp ls
docker exec javawebapp pwd

how to go inside a container

docker exec -it <container-name/id> </bin/bash>
(or)
docker exec -it <container-name/id> </bin/sh>

Ex: To launch the bash shell in a container
docker exec -it container-name/container-id bash

18)To create a container from a docker image
docker run image-name

19) how to see only the layers of an image
docker history <image-name / imageid>

20) how to display process details in which is running inside a container
docker top <container-id/name>

21) it will display resource(RAM,CPU) consumption details
docker stats <container-id/name>

22) can we set the ram, cpu limit while creating the containers?
Yes we set using options while creating the container

- While creating the container we can set the limit by using -m or - -memory for memory in kb's it can't take more than that from the system.
- What will happen it requires more? There is a lot of lode on your container but you have set a limit

Suppose your server has 10GB ram, so while creating the container set the limit as 200mb, what will happen if lot of lode on that particular application?

Incase of memory the container will be stopped, but incase of CPU it will not stop, performance will be vary slow, because the speed of our process depends on number of CPU's

sudo docker run -it --memory="<memory_limit>" <docker_image>

For example, if you set **--memory** to **1 GB**, as in the example above, the amount of swap memory needs to be more than that. To run a container with an **additional 1 GB** of swap memory, set the swap memory to **2 GB**.

The syntax for running a container with limited memory and additional swap memory is:

sudo docker run -it --memory="<memory_limit>" --memory-swap="<memory_limit>" <docker_image>

Set Soft Limit to Container Memory

As an example, for an Ubuntu container to have the memory reservation of **750 MB** and the maximum RAM capacity of **1 gb**, use the command:

sudo docker run -it --memory="1g" --memory-reservation="750m" ubuntu

23) How to copy files from container to host system or host system to container?
docker cp

container to the system

docker cp <container name>:<path of the container file> <system path>/<file name>

system to container

docker cp <system path>/<filename> <container name>:<path of the container file>

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 docker-host port.

-P Used for automatic port mapping i.e., 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

How create a container?

docker create --name <container name> -p <host-port>:<container-port> <image-name>

docker run --name <container name> -p <host-port>:<container-port> <image-name>

#Create a container in Detached mode

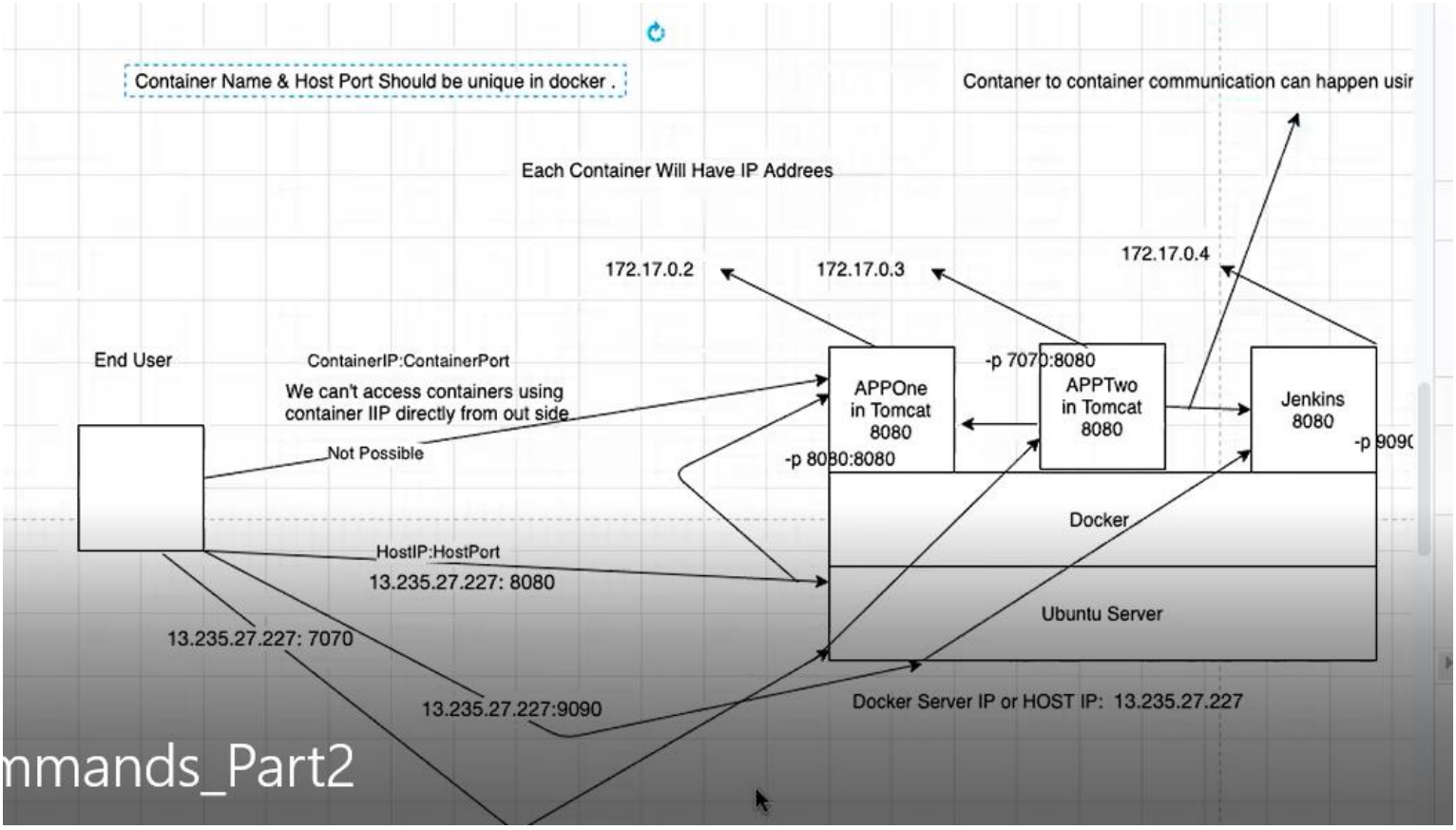
docker run -d --name <container name> -p <host-port>:<container-port> <image-name>

What is the difference between docker run & create?

- docker create will only create a container but it will not start the container
- docker run will create a container & start the container

What is port publish or port mapping in docker?

If we have to access the application which is running as container from outside of docker we can't Access using container-ip & containerPort. We can publish container port using host port using -p (or) - -publish. So that we can access using HostIp (Docker server ip) and Host port from outside docker.



docker images (***to see the list of images***)

To download tomcat image & ubuntu
docker pull tome
docker pull ubuntu

docker images

If you do not specify the version, by default, we get latest version & I want to download jenkins
docker pull jenkins

TO create a container from an image

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

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

TO check the tomcat is running or not http://13.250.47.90:7070

(7070 is port number mapped in docker host)

Let's remove the container
docker stop c1

docker rm -f c1

docker run --name mytomcat -p 7070:8080 tomee
Note:
(the above command runs tomcat container . it prompt log messages of a particular container. When we press the **Ctrl+c** it will exited from container. And check whether the container is running/not by using the command #**docker ps** it will show all the running containers)

docker run -d --name mytomcat -p 7070:8080 tomee
Note:
(The above command runs tomcat in detached mode, so we get **prompt back**)

docker container ls

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

To check for jenkins (Open browser) public-ip-Dockerserver:9090
Ex:

http://13.250.47.90:9090

How to create container in Interactive mode:

While creating container it self, it will enter in side the container

docker run --name myubuntu -it ubuntu

Observation: You have automatically entered into ubuntu
ls (To see the list of files in ubuntu)
exit (To come out of container back to 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

docker run --name appserver -P -d nginx

NOTE:
(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 which 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.

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

docker container ls

I want to open bash terminal of mysql
docker exec -it mydb bash

To connect to mysql database
mysql -u root -p
enter the password, we get mysql prompt TO see list of databases
show databases;

TO switch to a databse
use db_name
use mysql

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

exit

Multi container architecture using docker

This can be done in 2 ways

- --link
- docker-compose

1) --link option

Use case:

Start two busybox containers and create link between them

Create 1st busy box container

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

How to come out of the container without exit (ctrl + p + q)

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

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

How to check link is established for not?

/ # ping c10
Ctrl +c (to come out from ping) (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

TO start mysql as container

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

(if container is already in use , remove it # docker rm -f mydb)

Check whether the container is running or not

docker container ls

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

Ex 3: Create LAMP Architecture using docker

L -- linux
A -- Apache tomcat
M -- mysql
P -- php

(Linux os we already have)

Lets remove all the docker containers # docker rm -f $(docker ps -aq)

docker container ls (we have no containers now)

TO start mysql as container
docker run --name mydb -d -e MYSQL_ROOT_PASSWORD=sunil mysql:5

1) TO start tomcat as container
docker run --name apache -d -p 6060:8080 --link mydb:mysql tomcat

TO see the list of containers
docker container ls

To check if tomcat is linked with mysql
docker inspect apache (Apache is the name of the container)

2) TO start php as container
docker run --name php -d --link apache:tomcat --link mydb:mysqlphp

ex 4:
Create CI-CD environment, where jenkins container is linked with two tomcat containers.

Lets delete all the container
docker rm -f $(docker ps -aq)

To start Jenkins as a container
docker run --name devserver -d -p 7070:8080 jenkins/Jenkins

To enter into the jenkins container

#docker exec -it container-id/name /bin/bash

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

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
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

To start selenium/hub as container
docker run --name hub -d -p 4444:4444 selenium/hub

In hub.docker.com we have
selenium/node-chrome-debug (It is ubuntu container with chrome)

To start it as a container and link to hub (previous container)
docker run --name chrome -d -p 5901:5900 --link hub:selenium selenium/node-chrome-debug

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

To see the list of container
docker container ls

Note:
firefox and chrome containers are GUI containers. To see the GUI interface to chrome / firefox container

Download and install vnc viewer In VNC viewer in search bar place *public_ip_dockerhost:5901*
EX:
18.136.211.65:5901
Password - secret

++

All the commands we learnt till date are adhoc commands.

In the previous usecase 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 Left-hand-side - Key
Righthand side - Value Yaml file is space indented.

To validate the Yaml file Open
http://www.yamllint.com/

Installing Docker compose

1) Open https://docs.docker.com/compose/install/
2) Go to linux section
   Copy and paste the below two commands

```
#   sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(una me -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:
 mydb:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: sunilsunil

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

:wq
```

Lets remove all the running container

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

How to start the above services from dockerfile

```
# docker-compose up
```

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

```
# docker-compose stop
```

Remove the container

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

To check wordpress public_ip:5050

To stop both the containers
docker-compose stop

Create a docker compose file for setting up LAMP architecture

```
# vim docker-compose.yml

---
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
...


:wq

# docker-compose up -d
```

To see the list of the containers # docker container ls
(Observation - we are unable to see the php container) # docker ps -a

Ex:

Docker-compose file for setting up CI-CD Environment. jenkins container is linked with two tomcat containers

```
# vim docker-compose.yml


---
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 file

FROM: *From indicates the base image which are using to build our own image.*

FROM tomcat:8.0.20-jre8

FROM openjdk:8-alpine

MAINTAINER: It will be used as comment to describe the author/owner who is maintaining the docker file

COPY: It will copy local files from host server(docker server) where we are building the image to the image while creating a image

COPY <source-file> <destination-file>

EX:

COPY target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

ADD: Add also can copy files to the image while creating the image. Add can copy local files from host server and also it can download the files from remote HTTP/S locations while creating the image

ADD <URL> <Destination>

ADD <Source> <Destination>

EX:

ADD target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

RUN: RUN instruction will execute the commands. Run commands/instructions will be executed while creating an image.

EX:

- *RUN mkdir -p /opt/app*
- *RUN tar -xvzf /opt/apache-tomcat-8.5.54.tar.gz*

CMD: CMD instruction will execute the commands. CMD command/instructions will be executed while creating the container

EX:

CMD sh Catalina.sh run

CMD ["java"," -jar ","springapplication.jar"] (or) CMD java -jar springapplication.war

What is shell form & executable form in docker?

RUN/CMD/ENTRYPOINT *instructions/commands can be defind in shell form (or) executable form. When we use shell form our command will be running a child process under* ***bash/sh (****shell****)***

Shell form

CMD *java -jar springapplication.jar #SHELL FORM*

In Background above command will be executed as bellow

/bin/bash -c java -jar springapplication.jar

When we use exec from our command will be running as a main process.

CMD ["java","-jar","springapplication.jar"] #EXECUTABLE FORM

/bin/java -jar springapplication.war

NOTE:

When we use CMD & ENTRYPOINT exec form is preferable.

Because When we are stopping/killing the container, if we use shell form which process will be killed like the SIGNAL termination, will go to which process → bash
Basically we are killing the parent process, but chaild process doesn't know that is about to be killed So the java process will not release the resources what ever it was acquired

What is the difference between CMD/RUN?

Run instructions will be execute while creating a image.CMD instructions will be executed while Creating a container. we can have more than one RUN keyword in a docker-file. All RUN keyword's Will be processed while creating an image in the defind order (Top to Bottom).

Can we have more then one CMD keyword in the docker file?

Yes, we can have. but only the last one/resent one in the order will be processed while creating a container

ENTRYPOINT: *ENTRYPOINT instruction will execute the commands. ENTRYPOINT commands/instructions will be executed while creating the container*

ENTRYPOINT *java -jar springapplication.jar #SHELL FORM*

ENTRY PONT ["java","-jar"," springapplication.jar"] #EXECUTABLE FORM

What is the difference between CMD & ENTRYPOINT?

Cmd commands/instructions can be overridden while creating a container. Entry point commands/instructions cannot be overridden while creating a container.

Can we have both CMD & ENTRYPOINT in docker file?

Yes, we can have both in a docker file. Cmd instructions will not be executed if we have both Cmd and entrypoint. Cmd instructions will be passed as an argument for Entrypoint.

***EX**:*

CMD ls
ENTRYPOINT ["echo","HELLO"]

It will be executed as bellow

/bin/echo HELLO ls

OUTPUT:

HELLO ls

*Requirement always we have to execute **sh Catalina.sh**. But arguments by default it has to execute **"start"**. But dynamically I should have a option to pass different argument while creating a container.*

CMD start
ENTRYPOINT ["sh"," Catalina.sh"]

WORKDIR: *we can set working directory for an image /container. All subsequent instructions will be processed under working directory.*

EX:
***WORKDIR** /usr/local/tomcat*

EXPOSE: *Expose indicate which port is opened/used in the image.*

EXPOSE <port>

EXPOSE 8080

ENV: *Env is used to set an environment variable. these env variables will be available for image & Container*

ENV <NAME> <VALUE>

EX: *ENV CATALINA_HOME /usr/local/tomcat*
ENV JAVA_HOME /usr/bin/jdk8

USER: *User is used to set user for the container (or)image*

USER <username>

By default we can't mention USER. Container will be running as root user of the container

LABLE: *we add LABLE to the image*
LABLE <key> <value>

EX: ***LABLE branch develop***

ARG: *Using* ***ARG*** *we can define variables for docker file. We can use this in any instruction file. We can dynamically pass ARG while creating image.*

ARG branchname=develop

While creating an image we can pass ARG as below

docker build -t <imagename> --build-arg branchname=develop

VOLUME: *VOLUME keyword is used to mount container folder with host system folder*

VOLUME <folder/file>

What is the best practice while creating the docker image?

- Use alpineLinux images where ever it's possible.
- Try to combine multiple run instruction's into single run instructions by using && operator where it's possible.
- Avoid installing/downloading un-necessary packages/software's which are not required to run our applications.
- Remove un-necessary files.

How to create read only volume?

How do you do the os patches in the docker swarm/worker machines with out impact your applications/Containers?

We can drain the machine so that container will move to another machine and we can do performance like whatever the activity You want to do then again we make it active

Dockerfile
==========

```
FROM ubuntu:20.04

RUN apt-get -y update
RUN useradd -d /home/jenkins -u 1001 -m -s /bin/bash jenkins
RUN apt-get -y update && apt-get install -y unzip wget curl net-tools git vim xz-utils docker.io
RUN apt-get -y update && DEBIAN_FRONTEND=noninteractive apt-get install -y openjdk-11-jdk
#ln -fs /usr/share/zoneinfo/America/New_York /etc/localtime
#RUN mkdir -p /var/lib
RUN wget https://services.gradle.org/distributions/gradle-6.9-bin.zip -O /opt/gradle.zip
RUN (cd /opt && unzip gradle.zip)
USER jenkins
ENV JAVA_HOME /usr/lib/jvm/java-11-openjdk-amd64
ENV PATH /usr/local/bin:/usr/bin:/bin:/opt/gradle-6.9/bin
WORKDIR /home/jenkins
```

==========

In Docker, the CMD instruction is used to specify the command that should be run when a container is started. However, you can only have one CMD instruction in a Dockerfile. If you specify multiple CMD instructions, only the last CMD instruction will be used and the others will be ignored.

How you can see/Generate the MAC address in docker container ?
==========

When I start new containers, Docker automatically assigns some MAC address to them.

docker network inspect bridge


```
"Containers": {
        "3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c": {
            "EndpointID": "647c12443e91faf0fd508b6edfe59c30b642abb60dfab890b4bdccee38750bc1",
            "MacAddress": "02:42:ac:11:00:02",
            "IPv4Address": "172.17.0.2/16",
            "IPv6Address": ""
        },
        "94447ca479852d29aeddca75c28f7104df3c3196d7b6d83061879e339946805c": {
            "EndpointID": "b047d090f446ac49747d3c37d63e4307be745876db7f0ceef7b311cbba615f48",
            "MacAddress": "02:42:ac:11:00:03",
            "IPv4Address": "172.17.0.3/16",
            "IPv6Address": ""
        }
```


Verifying the MAC Address,
After starting the container, you can verify the MAC address inside the container:

ip link show eth0

This command will display details about the 'eth0' interface, including the MAC address.

docker inspect <container_name_or_id>

This will output a JSON object with all the details of the container. Look for the NetworkSettings section, where you can find the MacAddress field.

(OR)

docker inspect -f '{{.NetworkSettings.MacAddress}}' my_container

This will output the MAC address.

What is Docker Daemon?

The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Life cycle of Docker ?

docker build -> builds docker images from Docker file
docker run -> runs container from docker images
docker push -> push the container image to public/private regestries to share the docker images

The default size of a Docker container?

The default size of a Docker container depends on several factors, including the base image used and any additional software installed within the container. Docker itself does not impose a strict "default size" on containers; instead, the size is determined by the image layers that make up the container.

However, here are some general considerations:

Base Image Size: The base image you choose significantly impacts the size. For example, alpine is a minimal base image (~5 MB), while ubuntu or debian images are larger (typically between 50 MB and 200 MB).

what is Docker Scout?

Docker Scout is a tool used for managing and improving Docker images. It helps developers analyze their Docker images in terms of security, efficiency, and performance.

how we can scann the docker images?

Docker Scout helps you analyze your Docker images to optimize them and find potential security vulnerabilities.

Install Docker Scout (if not already installed): docker plugin install docker-scout

Scan an Image: Run the following command to scan a specific image: docker scout image <image_name>

For example: docker scout image my-app:latest

Review the Results: Docker Scout will provide details about vulnerabilities, image size, unused layers, and recommendations for optimization.

Diff b/w USER & useradd in Docker file?

useradd command and the USER instruction in a Docker file serve different purposes related to user management

1. useradd Command:- creates the new user in the containers filesystem
2. USER instruction:- USER switches to that user for execution

An OCI label in Docker refers to standardized metadata annotations defined by the Open Container Initiative (OCI). These are key–value labels that describe your container image

OCI labels are stored in an image's metadata under the org.opencontainers. They provide standard fields like:

Who built the image, When it was created, Source repository, Version, Documentation links, Licensing info

Label	Meaning
org.opencontainers.image.title :-	Name of the image/project
org.opencontainers.image.description:-	Short description
org.opencontainers.image.version:-	Version (e.g., v1.0.0)
org.opencontainers.image.revision:-	Git commit SHA
org.opencontainers.image.created:-	Build timestamp
org.opencontainers.image.source:-	Repo URL
org.opencontainers.image.licenses:-	License (e.g., MIT)
org.opencontainers.image.authors :-	Image authors

Viewing OCI Labels

```
docker inspect <image-name> --format '{{ json .Config.Labels }}' | jq
```