Docker

hub.docker.com

Running a docker image :-

docker run <image-name>.

Ex:

docker run nginx.

PS

docker ps :- It lists all running containers and some basic information about them.

ps-a

It will list all containers whether they are running or not.

Stop :-

docker stop <image-id/container-id / container name>

Ex.

docker stop nginx.

Listing Available Images

docker images :- It will list images and their sizes.

docker container prune -f
to remove all stopped containers.

Date :
Page No :

Removing Image

First ensure that no containers are running of that image and then following command is used to remove the image:-

docker rmi <image name>
It will remove the image

Stopping & removing the container:-

docker stop <container-id>
docker rm <container-id>

docker rm -f <container-id> —
to remove docker forcefully.

Docker Pull:

To just download the docker image.

docker pull <image-name> <version (optional).

Ex: docker pull alpine

→ docker only lives as long as the process inside it is alive.

Ex:-

docker run ubuntu

after giving above command docker image instance is created and exited ASAP because inside it there is no process going on.

ubuntu is just a base image.

Now.

if we want to see contents of file inside a running container.

docker exec <docker-container name>.
exec <command>.

Ex: docker exec pedantic_moore cat /etc/hosts.

Note: before accessing the file container shud be running.

Attach and Detach Mode

Containers run in two modes :-
i) Attach Mode (foreground)
ii) Detach Mode. (Background)

→ By default docker runs the container in attached mode means docker starts the process and attach the ~~process~~ console to the process's standard input, standard output and standard error.

→ We can switch to detach mode by using -d flag while running the container.

docker run -d <docker-image>

→ To again attach the detached container docker attach command is used.

docker attach <name of container>
(or)
docker attach <docker-id>.

to attach to the terminal and run image in interactive mode following command is used.

`docker run -it <image-name>`

Here,

i - refers to Interactive

t - refers to terminal.

Port mapping

`docker run -p 80:5000 <image-name>`

Here we are mapping port 80 to port 5000.

Volume Mapping:-

While working with containers if a container gets deleted accidentally then all data stored in it will be deleted. So to

`docker volume create <volume_name>`

used to create volume under /var/lib/docker/volumes.

Date :
Page No :

To mitigate this problems we used docker volume.

So what we exactly do is...

We mount a specific directory on the host to the container.

Ex:-

Do Mysql image/container

Data is stored inside

`/var/lib/mysql.`

Now in our system we will create a directory /opt/data.

Now, we will map the above created director.

```
docker run -v /opt/data:var/lib/mysql
                                mysql
```

Now even if the container or image will be deleted the

docker run --mount type=bind/volume, source=path, target=path

data will be persistent in /opt/data directory.

Note:-
To map volume we use -v which is old school way now normally we use --mount.

Inspect:-

inspect command is used to see the additional details about a container in JSON format.

Ex:-
docker inspect pedantic_roooore.

Container Logs:-

docker logs <containerName/. ContainerID>.

to see the logs of that container

docker logs --since time(second) <id>.

It prints the log of that specific container since that given time.

Environment Variables

After inspecting the container we find the environment variable.

We can manipulate the environment variables by using following command.

Ex:
A container named Simple-webapp has environment variable

APP_COLOR.= blue

and we have to changed it to green so following command will be used.

```
docker run -e APP_COLOR=green
                    Simple-webapp.
```

Docker Image :-

Creating Dockerfile

Syntax:

INSTRUCTION ARGUMENT

Ex:

```
FROM    ubuntu  →  Base image
RUN     apt-get update                  } - Dependency
RUN     apt-get install python          }
RUN     pip install flask               }
RUN     pip install flask-mysql         } - Some other dependencies
COPY    . /opt/source-code              } - Copy source code
ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
                                        ↓
                                    Entry point specifications.
```

→ Docker image is created in layered Architecture.

→ After writing the dockerfile we can build the image using following command:-

docker build <directory path>

→ Note: Name of dockerfile should always be "Dockerfile"

→ After building the Docker image we have to provide tag for that

```
docker build -t username/repositoryName
```

CMD vs Entry Point in Docker

→ Unlike VM containers are not meant to host ~~virtual~~ operating systems. They are meant to run specific tasks or servers. Such as to host a instance of application or database or to simply carry out any calculation.

→ Once the task is completed the container ~~exited~~ exits. ASAP.

We can specify commands with CMD. There are two ways of specifying commands with CMD:-

i) Shell form
ii) JSON array form.

Shell form

Syntax:

CMD <command> <parameter>
Ex: CMD sleep 5

JSON array format

CMD ["command", "Parameter"]
Ex: CMD ["echo", "Hello Shubham"]

Date :
Page No :

If we want to pass the required parameter while giving docker run command then we use Entrypoints.

Ex:

If we want to make ubuntu container sleep after some specific seconds then at the end of docker file we will include the entrypoint like following

```
FROM Ubuntu

ENTRYPOINT ["sleep"]
```

Now:

we will give the following command

```
docker run ubuntusleep 10.
```

Here we are passing seconds

Here we are just passing the no. of seconds and sleep command invokes automatically. Here the Entrypoint

comes into play.

Docker Networking :-

After installing Docker it creates 3 networks automatically :-

`Sudo docker network ls`

i) Bridge
ii) none
iii) host.

Bridge:
Bridge is the private internal network created by the docker on the host.

All container attache to this network by default and get internal IP address usually in the range 172.17 series.

Container can access each other using this internal IP if required.

To access these containers from outside world we map port of the container with port of the host.

Another way to access the container externally is to associate the container to the host network

Date :
Page No :

None:-

Containers are not attached to any network and containers doesn't have any access to the external networks or other containers.

They Run in isolated Networks.

Inspecting Networks :-

We use following command to inspect the network :-

```
docker network inspect <network Name>
```

Creating own network

Before launching container one can create the network with the following command :-

```
Sudo docker network create --driver <driver-name>
                           <network name>
```

Ex:-

Sudo docker network create --driver bridge new_nw

// It will give a new id as output.

Now:
Container can be attached to this network.

Ex:-

Sudo docker run --network=new_nw ubuntu.

Now:
if you will inspect the network you will find the container attached with it.

Note:-
You will find the network related information of the container by inspecting it.

Docker file System :-

When we install docker it creates the following folders structure:

`/var/lib/docker`

There will be multiple folders in it:-

/aufs
/containers
/image
/volume

In these folders/directories datas are stored.

Date :
Page No :

Docker Compose:-

Compose is a tool for defining and running multiple container docker applications.

With compose YAML file is used to configure Applications service.

We define all the services in a file named `docker-compose.yaml`

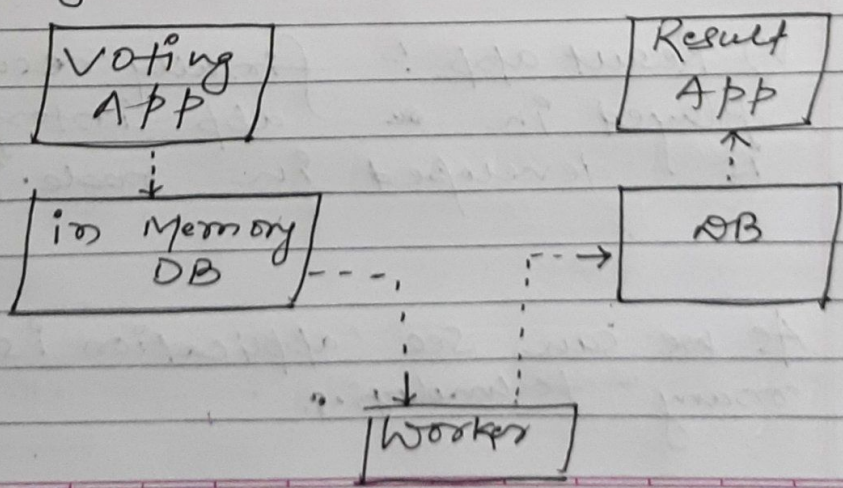
after that we use following command

```
docker compose up
```

It will start and run the entire app.

Let's Understand use of docker compose with a simple example:-

Voting Application



This application provides two interfaces.
i) To vote
ii) To show result.

It consists 5 components:-

i) Voting App - developed in python
It provides interface to users to vote.

ii) In Memory Database:- In this application redis serves as Database in memory. which stores the vote

iii) Worker:- The vote which is made be user is then processed by worker which is developed in .net

iv) Database:- worker takes the vote and updates it in persistent database which is potgresql.

v) Result app:- finally result is displayed in app interface which is developed in node.js.

As we can see application is built using many technologies.

Now we will see how we can integerate the different components of the applications.

first we will run all the containers.

docker run -d --name=redis redis

docker run -d --name=db postgres

docker run -d --name=vote -p 5000:80 voting-app

docker run -d --name=result -p 5001:80 result-app

docker run -d --name=worker worker.

Now all instances are running but issue is they are not linked with each other means there is no communication between them.

Here Links come into play :-

Link is a command line option to link 2 containers together.

NOW we will link the containers :-

docker run -d --name=vote -p 500:80 --link redis:redis voting-app

Name of container — hostname.

```
docker run -d --name=result -p 5001:80
    --link db:db result-app

docker run -d --name=worker --link
    db:db --link redis:redis worker
```

Now with the help of above command let's write the docker compose file in YAML :-

```
redis:
    image:redis

db:
    image : postgres:9.4

vote:
    image : voting-app
    ports:
        - 5000:80
    links:
        -redis

result:
    image: result.
    ports:
        -5001:80
    links:
        -db:
```

worker:
    image : worker
    links:
        - db
        - redis

So, this is how we will write a docker compose file.

Now,
Let's consider we want to build an application locally and we don't have image of that application available on docker hub then,

we can replace the image line with build line as follows:-

build : <directory-path>

Ex:-
build : ./vote

Now we will use docker compose up command to run and start the applications.

Docker compose versions:-
We are free to use any versions of docker file but we have to explicitly declare the version in docker compose file as:-

version : <version no.>.

Note:
if we want to add dependency to any container then we can add depends_on section in version 2 of docker compose.
Example:

if we want to ensure that redis should run first then vote container should run the we will add following line to the docker compose file:-

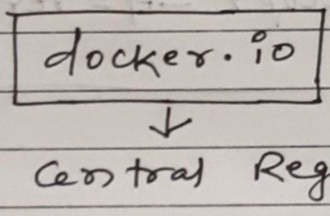
```
vote:
    image : vote
    ports:
        - 5000:80

    depends_on:
        - redis.
```

Date :
Page No :

Docker Registry :-

It's central repository where all docker images are stored.

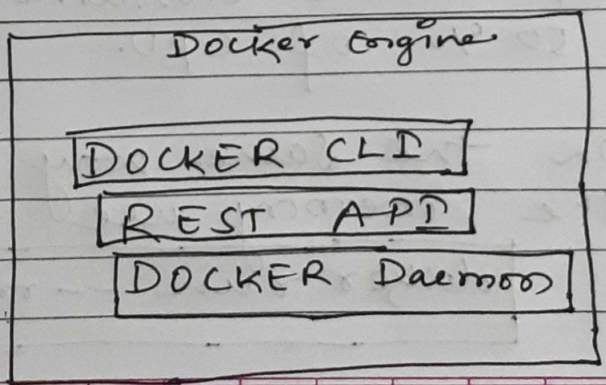
→ All images are pulled from Docker Registry.



→ if we have to access the image of any private registry then first we have to login using docker credentials.

Docker Engine :-

Simply docker engine is reffered as a host with docker installed in it.



Docker CLI can be accessed remotely to.

we have to use -H flag and specify the port.

Ex:

```
docker -H=remote-docker-engine:2375
```

Resource Used by Coontainers :-

By default there is no restriction on how much amount of resource one coontainer can utilize.

But explicitly we can provide by the following ways:-

```
docker run --cpus=.5 ubuntu
```

It means coontainer can just use 50 % of CPU.

In the same way we can limit the memory use too :-

```
docker run --memory=100m ubuntu
```

Container orchestration :-

Consider if we have to scale up our application then we have to create more number of instances of that application. We have to keep eye on health of the containers too means if a container crashes it should respawn (or) new instance should be created.

Here Container orchestration comes into play.

Container orchestration is automates the deployment, management, scaling and networking of containers.

There are multiple Container orchestration solutions available today like:-

Docker Swarm, Kubernetes, Mesos etc.