**CREATE, SHARE, RUN**

**Docker Installation**

-------------------------------------------------------------------------------------------------------------------------------------------------------

RedHat rhel8

1. yum install yum-utils -y

2. yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

3. yum install docker-ce -y

4. systemctl enable --now docker

5. systemctl status

6. docker -v

--------------------------------------------------------------------------------------------------------------------------------------------------------

**WEB HOSTING THROUGH DOCKER FILE**

#vim Dockerfile

FROM centos

RUN yum install httpd -y

COPY index.html /var/www/html

EXPOSE 80/tcp

CMD [“httpd”, “-DFORGROUND”]

# docker image build . -t mywebsite

#docker container run -d -p 80:80 mywebsite

--------------------------------------------------------------------------------------------------------------------------------------------------------

**4 OBJECTS OF DOCKER**

docker container

docker image

docker network

docker volume

---------------------------------------------------------------------------------------------------------------------------------------------------------

**SOME COMMANDS**

docker container ls

docker image ls

docker network ls

docker volume ls

docker container ls --->you will get running container

docker container ls -a --->you will get all running and stopped containers

docker container run -itd httpd --->it= interactive terminal, d= detached mode

//if you run command with only "it" then if you want to come out from container then you have two options

//1. exit --->your container will be terminate

//2. ctrl+p+q --->you will come out from container and container also run in background

docker container stop NAME/ID --->to stop a running container

docker container rm NAME/ID --->remove stopped containers

docker container exec NAME/ID COMMAND --->exec= use to run a command in container from outside.

docker container exec -it NAME/ID bash --->to enter in container and access the container

docker container ls -aq --->a= all, q=display only container ID

\*docker container rm $(docker container ls -aq) --->removing container by container ID

docker container stop $(docker container ls -aq) --->stop running container by container ID

\*docker container run -d --name deepak httpd --->assign name to container

docker container run -d -e MYSQL\_ROOT\_PASSWORD=PASSWORD --name CONTAINER\_NAME mysql --->e= environment variable for database related containers. Agar aap bina variable ke container banaoge to vo exit ho jayega

docker container rename deepak mohit --->renaming container

docker container inspect NAME/ID --->show all information

curl IP

docker container stats --->show uses of RAM and RESOURCES

docker container logs ID --->show log msgs

docker container logs -f ID --->to follow log msgs

docker container top ID --->show live running processes

docker container pause ID --->to pause container

docker diff ID --->Inspect changes to files or directories on a container's filesystem

-------------------------------------------------------------------------------------------------------------------------------------------------------

**PORT MAPING COMMAND**

\*docker container run -d --name vivan -p 80:80 centos/httpd --->publish port

-------------------------------------------------------------------------------------------------------------------------------------------------------

docker container cp /deepak.txt vivan:/var/www/html --->copy a file from local machine to container

docker system events --since 60m --->Get real time events from the server

docker container kill --signal=9 vivan --->Kill one or more running containers

docker container prune --->Remove all stopped containers

\*docker container attach ID --->OS based containers se hi attach ho skte h

docker image history NAME/ID --->Show the history of an image

docker container run -itd --hostname=grras.com ubuntu ---> Set hostname for container

docker image prune -a --->to remove all images

-------------------------------------------------------------------------------------------------------------------------------------------------------

**CONTAINER RESTART POLICY**

**4 TYPES OF RESTART POLICIES**

1. **NO** = Do not automatically restart the container(default)
2. **ON-FALIURE**[:max-retries]= Restart container if it exiits due to an error.

docker container run -d --restart=on-faliure httpd

1. **ALWAYS**= Always restart container if it stops. If it is manually stopped, it is restarted only when docker daemon restart or the conatiner itself is manually restarted.

Docker container run -d --restart=always httpd

1. **UNLESS-STOPPED**= Similar to always, except when the container is stopped (manually or otherwise) it is not started even after docker daemon restarts

**DOCKER ARCHITECTURE**

**DOCKER CLIENT**

* Docker user can interact with docker daemon through the client (CLI).
* The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

**DOCKER HOST**

* Docker host (H/W) is used to provide an environment to execute and run application. It contains the docker daemon, images, containers, networks and volumes.
* Docker host, it is the server machine on which Docker daemon runs.

**DOCKER REGISTRY**

* Docker registry manages and stores the docker images.
* A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.
* When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

**CONTAINER IMAGE MANAGEMENT**

**CONTAINER REGISTRYS THOSE STORE CONTAINER IMAGES.**

Google container registry

Amazon container registry

Azure container registry

1. **Types of Image File**
2. **Official Images --** In Images ko docker khud manage krta h.
3. **Verified Images --** Third parties in images ko bnati h or fr docker se verify krva leti h.
4. **User Created Images --** In Images ko local user khud bnate h.

**TWO METHODS OF MAKING CONTAINER IMAGE**

1. **RUNNING CONTAINER SE IMAGE BNANA**
2. **DOCKER FILE SE IMAGE BNANA**
3. **RUNNING CONTAINER SE IMAGE BNANA**

* **docker container commit -a “AUTHER\_NAME” CONTAINER\_NAME IMAGE\_NAME** --->making a image from a container
* docker container run -d -- name CONTAINER\_NAME -P 80:80 IMAGE\_NAME --->making a container from image
* docker image pull IMAGE\_NAME --->to pull a image from repository
* docker image rm IMAGE\_NAME ---> to remove image

registry/useraccount/image:tag

* docker image pull rupeshsaini09/httpd:latest --->to pull a image from a personal account
* docker image pull gcr.io/httpd/httpd:latest ---> to pull a image from google repo.
* docker login ---> to login on hub.docker.com
* docker image push rupeshsaini09/IMAGE\_NAME:TAG ---> to push your image on docker registry(push krne se phle aapki image ka name repeshsaini09/centos is formate me hona chahiye)
* docker image tag rupeshsaini09/website:latest rupeshsaini09/mywebsite:latest ---> to rename image name
* docker system df ---> to show ho much space using a image
* docker image inspect IMAGE\_NAME --->to show all detail of image
* docker image history IMAGE\_NAME---> to show image history

**IMAGE KI TAR FILE BANANA**

* docker image save IMAGE\_NAME -o NAME.tar ---> o= output, image se tar file banana

**TAR FILE SE VAPAS IMAGE KO IMPORT KRNA**

* docker image load -i NAME.tar

**CONTAINER KO EXPORT KRA KE TAR FILE BANANA**

* docker container export CONTAINER\_NAME > NAME.tar

**AB IS TAR FILE SE IMAGE IMPORT KRNA FIR IMAGE SE CONTAINER BNANA**

* docker image import NAME.tar IMAGE\_NAME:TAG

1. **DOCKER FILE SE IMAGE BNANA**

**DOCKER FILE**

**INSTRUCTIONS ARGUMENTS**

FROM: for base image, this command must be on the top of the dockerfile.

RUN: to execute commands, it will create a layer in image

MAINTAINER: author/ owner/ description

COPY: copy files from local system (docker vm) we need to provide source, destination (we can’t download file from internet and any remote repo.)

ADD: similar to copy but it provides a feature to download files from internet, also extract file at docker image side.

EXPOSE: to expose ports such as port 8080 for tomcat , port 80 for nginx etc.

CMD: execute commands but during container creation.

The CMD instruction has three forms:

1. CMD ["executable","param1","param2"] (*exec* form, this is the preferred form)
2. CMD ["param1","param2"] (as *default parameters to ENTRYPOINT*)
3. CMD command param1 param2 (*shell* form)

ENTRYPOINT: similar to CMD but has higher priority over CMD, first commands will be executed by

ENTRYPOIN only.

ENV: environment variables

Dockerfile

* Create a file named Dockerfile
* Add instructions in Dockerfile
* Build dockerfile to create image
* Run image to create container

# vi Dockerfile

FROM ubuntu

RUN echo “Nagarjuna hota” > /tmp/testfile

To create image out of Dockerfile

# docker image build . -t myfile

#docker container ls -a

# docker image ls

Now create container from the above image

#docker container run -d -p 80:80 myfile

#cat /tmp/testfile

#vi dockerfile

FROM ubuntu

WORKDIR /tmp

RUN echo “thank you” > /tmp/testfile

ENV myname naga

COPY testfile1 /tmp

ADD test.tar.gz /tmp

**KISI BHI SERVICE KI CONFIG FILE DEKHNE KE LIE systemctl cat SERVICE\_NAME**

**EX.- Systemctl cat httpd**

**DOCKER NETWORK**

Docker networking is **primarily used to establish communication between Docker containers and the outside world via the host machine where the Docker daemon is running**.

**NETWORKING TYPES:-**

1. **None --** isme container isolate ho jayega use kahi se access nhi kr skte. For testing purpose.
2. **Bridge (default) --** jab bhi hum container launch krte h to use by default IP, bridge network se milta h.do container bridge IP ke through hi communicate krte h.

Containers apne name se bhi ek dusre se communicate kr skte h. name se communicate krte h to IP ki jarurat nhi hoti.

1. **Host --** isme container ko jis bhi port no. se launch kiya vo host ke usi port no. se connect ho jayega.
2. **MacVlan**
3. **Overlay**
4. **IPVlan**

**Network drivers SAME AS ABOVE AND BUT EXPLAINED, GOT IT FROM GOOGLE**

Docker’s networking subsystem is pluggable, using drivers. Several drivers exist by default, and provide core networking functionality:

* bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. **Bridge networks are usually used when your applications run in standalone containers that need to communicate.** See [bridge networks](https://docs.docker.com/network/bridge/).
* host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly. See [use the host network](https://docs.docker.com/network/host/).
* overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers. See [overlay networks](https://docs.docker.com/network/overlay/).
* ipvlan: IPvlan networks give users total control over both IPv4 and IPv6 addressing. The VLAN driver builds on top of that in giving operators complete control of layer 2 VLAN tagging and even IPvlan L3 routing for users interested in underlay network integration. See [IPvlan networks](https://docs.docker.com/network/ipvlan/).
* macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host’s network stack. See [Macvlan networks](https://docs.docker.com/network/macvlan/).
* none: For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services. See [disable container networking](https://docs.docker.com/network/none/).

**DOCKER NETWORK COMMANDS**

* docker network ls ---> to list network
* docker container inspect CONTAINER \_NAME ---> output me last me aapko network section milega jha aap network ki sari details dekh skte h
* docker network create --driver bridge --subnet 192.168.10.0/24 NETWORK\_NAME --->network creating
* docker container run -d --network=NETWORK\_NAME CONTAINER\_NAME ---> hamare bnaye hue network se container banana
* docker network disconnect NETWORK\_NAME CONTAINER\_NAME --->to disconnect through network
* docker network connect bridge CONTAINER\_NAME --->to connect again with bridge network
* docker network rm NETWORK\_NAME ---> to remove a network
* docker network inspect NETWORK\_NAME ---> to inspect network

**DOCKER VOLUME**

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While [bind mounts](https://docs.docker.com/storage/bind-mounts/) are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:

* Volumes are easier to back up or migrate than bind mounts.
* You can manage volumes using Docker CLI commands or the Docker API.
* Volumes work on both Linux and Windows containers.
* Volumes can be more safely shared among multiple containers.
* Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
* New volumes can have their content pre-populated by a container.
* Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.
* cd /var/lib/docker --->all docker related objects data store at this location
* cd /var/lib/docker/ overlay2/CONTAINER\_ID --->init ke just upar wali directory jhan aapko container ka content show hota h

**Container ka file system overlay2 h phle overlay hota tha**

1. **BIND MOUNT**
2. **VOLUME MOUNT**

**HOW TO MAKE CUSTOM VOLUME DIRECTORY (BIND MOUNT)**

**For mounting the data follow these steps**

* mkdir /root/webcode --->make a directory(you can make anywhere with any name)
* docker container run -d --name web -p 80:80 -v /root/webcode:/var/www/html IMAGE\_NAME ---> v = volume mount, container ko local machine ki dir. pe mount krna
* docker container inspect CONTAINER\_NAME --->to inspect container. In this detail you will see a mount point where all details shown about mount

**NOTE -- You can put index.html data at /root/webcode location that will be show in container /var/www/html location.**

**Agar aapka container delete ho jata h to aap dobara isi location pe new container mount krva skte h. kyoki aapka data to container ke bahar local machine pe rakha hua tha islie vo delete nhi hua h.**

**Now if your container get deleted then you can make another container and mount this /root/webcode location in that new container then your data will be persistent. You will not loos your data.**

* docker container run -d --name db -v /root/database:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=PASSWORD mysql --->is container me Volume ke sath Environment Variable bhi set ho jayega

**THE PREVIOUS METHOD “BIND MOUNT” IS OLDER BUT NOWDAYS WE MAKE VOLUMES IN DOCKER.**

**IN VOLUMES KO DOCKER MANAGE KRTA H. OR CONTAINER KO BYDEFAULT READ, WRITE KI PERMISSION HOTI H IN VOLUMES PE.**

**HOW TO MAKE DOCKER VOLUME (VOLUME MOUNT)**

* docker volume create VOLUME\_NAME --->to make a docker volume
* docker volume ls --->to list docker volumes
* docker volume rm VOLUME\_NAME --->to delete the volume
* \*docker container run -d --name CONTAINER\_NAME -v VOLUME\_NAME:/var/www/html IMAGE\_NAME --->yhan hum only volume name denge jisse docker volume ban jayega. Agar hum already bane hue volumes se name na de kr container banate time hi new name de dete h to new volume usi name se ban jayega.

DOCKER VOLUME LOCATION **/var/lib/docker/volumes/VOLUME\_NAME/\_data**

* docker volume inspect VOLUME\_NAME --->volume ki all detail show ho jayegi with volume location, or Mounts option me aapko volume type show hognge jese Volume Mount ke lie = “Type” : “volume” or Bind Mount ke lie “Type”: “bind” or bhi options show honge jese Source, Destination, Mode, etc…

**THIS \*-V OPTION IS ALSO OLDER NOW WE USE THIS NEW OPTION TO MAKE A NEW VOLUME AT CONTAINER MAKING TIME**

* docker container run -d --mount source=VOLUME\_NAME,destination=/var/www/html IMAGE\_NAME
* docker container run -d --mount source=VOLUME\_NAME,destination=/var/www/html,readonly IMAGE\_NAME --->agar aap chahte h ki container only read kr paye to aap readonly option dete h

NOTE : Agar aapka container delete ho jata h to new container me old container ka source volume de dene se new container me old container ka data aa jayega.

**Multi Tier Application in Container**

Multi-tier application – 1 application ka content alag-alag container pe launch krna. Har component ko separate container me dalna. Jese 1 container pe httpd webserver chalana dusre pe database launch krna third pe kuch or……. Ese 1 application ka data alag-alag container pe rhta h.

Search on google = sample code for docker

**DOCKER COMPOSE**

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see [the list of features](https://docs.docker.com/compose/#features).

Compose works in all environments: production, staging, development, testing, as well as CI workflows. You can learn more about each case in [Common Use Cases](https://docs.docker.com/compose/#common-use-cases).

Using Compose is basically a three-step process:

1. Define your app’s environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker compose up and the [Docker compose command](https://docs.docker.com/compose/#compose-v2-and-the-new-docker-compose-command) starts and runs your entire app. You can alternatively run docker-compose up using the docker-compose binary.

A docker-compose.yml looks like this:

version: "3.9" # optional since v1.27.0

services:

web:

build: .

ports:

- "8000:5000"

volumes:

- .:/code

- logvolume01:/var/log

links:

- redis

redis:

image: redis

volumes:

logvolume01: {}

Multi-tier applications mostly docker compose ke through hi launch ki jati h.

**INSTALL DOCKER COMPOSE**

Search on google = install docker compose

And follow this article <https://docs.docker.com/compose/install/>

* curl -SL https://github.com/docker/compose/releases/download/v2.5.0/docker-compose-linux-x86\_64 -o /usr/local/bin/docker-compose
* chmod +x /usr/local/bin/docker-compose
* docker-compose –version --->is command tak docker compose install ho gya h ab aage 1 practical h
* vim docker-compose.yml

version: "3.9"

services:

wordpress:

image: wordpress

ports:

- "80:80"

depends\_on:

- db

- redis

db:

image: mysql

environment:

MYSQL\_ROOT\_PASSWORD: password

MYSQL\_DATABASE: name

* docker-compose up -d --->d = deteched mod

**yhan aapke 2 container launch ho jayenge 1 wordpress ka dusra mysql database ka**