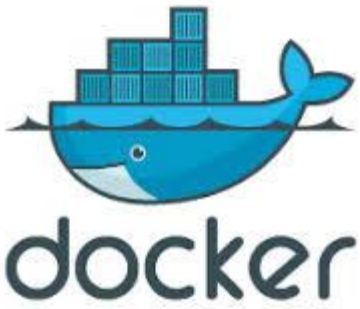


What is Docker?

→ **Docker is a open-source platform that allows you to build ,ship ,and run applications inside containers.**

- **A container is a lightweight, standalone and portable environment that include everything your application needs to run --> like code, runtime, librraies and dependencies.**
- **With docker, developers can ensure their applications runs the same way everywhere – whether on their laptop, a testing server or in the cloud.**
- **It solves the problem of "It works on my machine" , because containers carry all dependencies with them.**

In Short

- **Docker = Platform to create and manage containers.**
- **Container = Small, portable environment to run applications with all dependencies.**

Restart Policy

In Docker, when we talk about policy, it usually refers to the restart policies of containers.
These policies define what should happen to a container when it stops, crashes, or when Docker itself restarts.

Types of restart Policy

1. No (default)
2. Always
3. On-failure
4. Unless-stopped

Always policy:- `If container manual is off and always policy is set to on then container will only start when "docker daemon restart"`

`Command-->`

```
docker container run -d --restart always httpd
```

Unless-stopped:- `If a container is down due to an error and has an Unless-stopped policy, it will only restart when you "restart docker daemon"`

`Command -->`

```
docker container run -d --restart unless-stopped
httpd
```

On-failure:- When a container shuts down due to an error and has a no-failure policy, the container will restart itself.

```
Command -->
```

```
Docker container run -d --restart on-failure httpd
```

Max Retry in on-failure Policy (English)

When you use the on-failure restart policy in Docker, you can set a maximum retry count.

- This tells Docker how many times it should try to restart a failed container before giving up.
- If the container keeps failing and reaches the retry limit, Docker will stop trying.

```
docker run -d --restart=on-failure:5 myapp
```

```
                    Always Policy
```

```
ubuntu:~$ docker container run -d --name dev --restart always httpd
607c6b3e998c5c881f930a3e75c24a1d131fd10d53dc8de3afac97c90754b9d9
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
607c6b3e998c   httpd     "httpd-foreground"   6 seconds ago   Up 5 seconds   80/tcp    dev
ubuntu:~$ docker container stop dev
dev
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu:~$ systemctl restart docker
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED              STATUS         PORTS     NAMES
607c6b3e998c   httpd     "httpd-foreground"   About a minute ago   Up 2 seconds   80/tcp    dev
ubuntu:~$
```

Unless-stopped policy

```
ubuntu:~$ docker run -d --name myapp --restart=unless-stopped busybox sh -c "while true; do echo Hello; sleep 2; done"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
80bfbb8a41a2: Pull complete
Digest: sha256:ab33eacc8251e3807b85bb6dba570e4698c3998eca6f0fc2ccb60575a563ea74
Status: Downloaded newer image for busybox:latest
b189bdb5b821ef6bcb01e5188ea2ce9e53f710e6a3d6440532b05faa86517313
ubuntu:~$ docker exec myapp sh -c "exit 1"
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
b189bdb5b821   busybox   "sh -c 'while true; …"   27 seconds ago   Up 26 seconds             myapp
231ca0307dd8   httpd     "httpd-foreground"       17 minutes ago   Up 9 minutes    80/tcp    prod
ubuntu:~$ sudo systemctl restart docker
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS          PORTS     NAMES
b189bdb5b821   busybox   "sh -c 'while true; …"   About a minute ago   Up 19 seconds             myapp
231ca0307dd8   httpd     "httpd-foreground"       18 minutes ago       Up 19 seconds   80/tcp    prod
```

On -failure police

```
ubuntu:~$ docker container run -d --name play --restart on-failure httpd
ac9afa92f26cf18749d248cd97f78c511884db9578ded5be6dc1aa9ff30b58f7
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
ac9afa92f26c   httpd     "httpd-foreground"   3 seconds ago   Up 3 seconds   80/tcp    play
```

```
ubuntu:~$ kill -9 9159
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
ac9afa92f26c   httpd     "httpd-foreground"   2 minutes ago   Up 4 seconds   80/tcp    play
ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
ac9afa92f26c   httpd     "httpd-foreground"   2 minutes ago   Up 8 seconds   80/tcp    play
ubuntu:~$
```

Port Mapping

→ **Every Docker container has its own network namespace (like a mini-computer).**

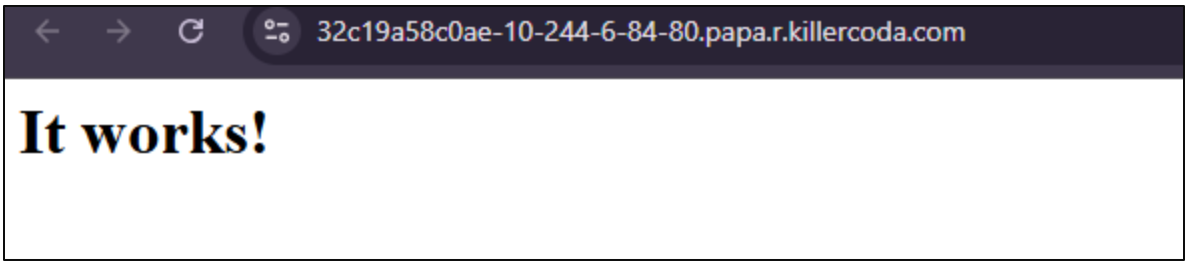
→ **By default, services inside a container are not accessible from outside the host machine.**

→ **Port Mapping is the process of exposing a container's internal port to the host machine's port so that external users can access it.**

It uses the `-p` or `--publish` option:-

docker container run –d –p <host port>:<container port> httpd

```
ubuntu:~$ docker container run -d --name port -p 80:80 httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
ce1261c6d567: Pull complete
aeb6d226161f: Pull complete
4f4fb700ef54: Pull complete
56926e6ce68f: Pull complete
4938babf7b43: Pull complete
307fcc49c641: Pull complete
Digest: sha256:027c678f36d3cd3dd2b44ad1e963e81be66f9eba065381c1126d3019fffeb01a
Status: Downloaded newer image for httpd:latest
a91cd8834d4587a2f0be5d40ccaa13ddd4d32b61999c36c109af5a528b159b6b
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                   NAMES
a91cd8834d45   httpd     "httpd-foreground"   4 seconds ago   Up 4 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   port
ubuntu:~$
```



Networking

Docker networking is how containers communicate with each other, with the host machine, and with the outside world (internet).

When Docker is installed, it creates some default networks. Containers can be attached to these networks depending on how you want them to communicate.

Default Docker Networks

1. **bridge (default)**
    a. **If you run a container without specifying a network, it connects to the bridge network.**
    b. **Containers on the same bridge network can communicate using IP addresses.**
    c. **You can also create your own user-defined bridge for name-based communication.**
2. **host**
    a. **Removes the isolation between the container and the host network.**
    b. **Container uses the host's network directly.**
    c. **Example: If container exposes port 80, it will be directly available on host port 80.**

3. **none**
    a. **Completely isolates the container from all networks.**
    b. **No internet, no container-to-container communication.**

How Containers Communicate

- **Container ↔ Container (same bridge network) → via container name or IP.**
- **Container ↔ Host → via port mapping (`-p hostPort:containerPort`).**
- **Container ↔ Internet → via NAT (Network Address Translation) on the host.**

Command -->

docker network ls

```
ubuntu:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
aeb86a9c03b1   bridge    bridge    local
f0ecc2302d43   host      host      local
6805a1e354ac   none      null      local
ubuntu:~$
```

Create a network-->

docker network create – -driver bridge network name

```
ubuntu:~$ docker network create --driver bridge grras
3a9b9a37b85fc8f2d04160d2db859ec875596fa38606bdc632059f9e47a5a739
ubuntu:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
aeb86a9c03b1   bridge    bridge    local
3a9b9a37b85f   grras     bridge    local
f0ecc2302d43   host      host      local
6805a1e354ac   none      null      local
ubuntu:~$
```

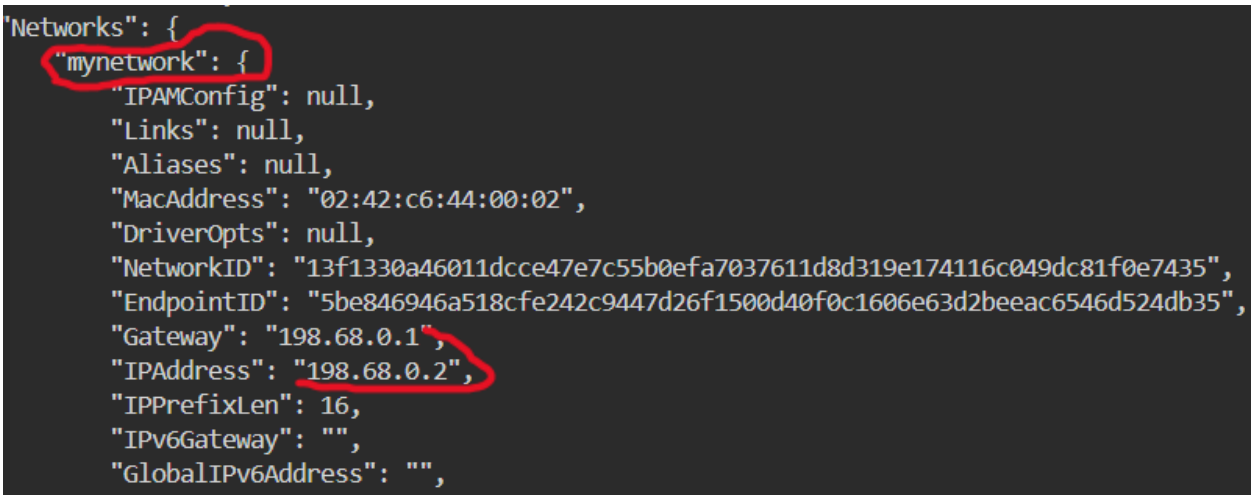
Docker network create – -driver bridge – -subnet 198.68.0.0/16 mynetwork

```
ubuntu:~$ docker network create --driver bridge --subnet 198.68.0.0/16 mynetwork
13f1330a46011dcce47e7c55b0efa7037611d8d319e174116c049dc81f0e7435
ubuntu:~$ docker ns
docker: 'ns' is not a docker command.
See 'docker --help'
ubuntu:~$ docker ns ls
docker: 'ns' is not a docker command.
See 'docker --help'
ubuntu:~$ docker network ls
NETWORK ID     NAME        DRIVER    SCOPE
aeb86a9c03b1   bridge      bridge    local
3a9b9a37b85f   grras       bridge    local
f0ecc2302d43   host        host      local
13f1330a4601   mynetwork   bridge    local
6805a1e354ac   none        null      local
```

Craete a container in our custom network

```
ubuntu:~$ docker container run -d --network mynetwork httpd
0a0a0c877e8d7cedefef7b784ffe804ad5b3de39473007dab7a69053f7101186
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
0a0a0c877e8d   httpd     "httpd-foreground"   6 seconds ago   Up 6 seconds   80/tcp    gifted_cohen
ubuntu:~$
```

docker container inspect container name



Volume

→ **By default, anything you save inside a container is temporary.**

→ **If the container is deleted, all data inside it is lost.**

→ **Volumes are Docker's way to store data permanently (persistent storage).**

A Docker Volume is a storage location outside the container's filesystem but managed by Docker.
This way, data remains safe even if the container is removed or recreated.

Why Use Volumes?

1. **Data Persistence → Data won't be lost if the container is deleted.**

2. **Sharing Data → Multiple containers can share the same volume.**
3. **Performance → Better than bind mounts for production workloads.**

Types of volume:-

1 Bind volume

2 Local Mount/ Volume mount

Bind Mount:-

→ **A Bind Mount directly connects a host machine's directory/file to a container's directory.**

→ **This means whatever changes you make inside the container will reflect on the host, and vice versa.**

→ **It's different from a Volume because:**

→ **Volumes are managed by Docker (stored in `/var/lib/docker/volumes/...`)**

→ **Bind mounts are managed by you (stored anywhere on your host).**

Command :-

```
docker container run –d –p 80:80  –v
/directory_name:/usr/local/apache2/htdocs
```

```
ubuntu:~$ mkdir /my_mount
ubuntu:~$ docker container run -d --name linux -p 80:80 -v /my_mount:/usr/local/apache2/htdocs httpd
3fe3c3e64a3e5ad66f5a812cbf3ec71e874428a554d69580a8c76c1cfa7f043f
ubuntu:~$ docker ps
CONTAINER ID   IMAGE     COMMAND              CREATED         STATUS         PORTS                                   NAMES
3fe3c3e64a3e   httpd     "httpd-foreground"   6 seconds ago   Up 5 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp   linux
ubuntu:~$ cd /my_mount/
ubuntu:/my_mount$ vim index.html
ubuntu:/my_mount$ docker container exec -it ls linux /usr/local/apache2/htdocs
Error response from daemon: No such container: ls
ubuntu:/my_mount$ docker container exec -it  linux ls  /usr/local/apache2/htdocs
index.html
ubuntu:/my_mount$ docker container exec -it  linux cat  /usr/local/apache2/htdocs/index.html
<center><h1>THIS IS MY MOUNT POINT</h1><center>
ubuntu:/my_mount$
```

Local mount

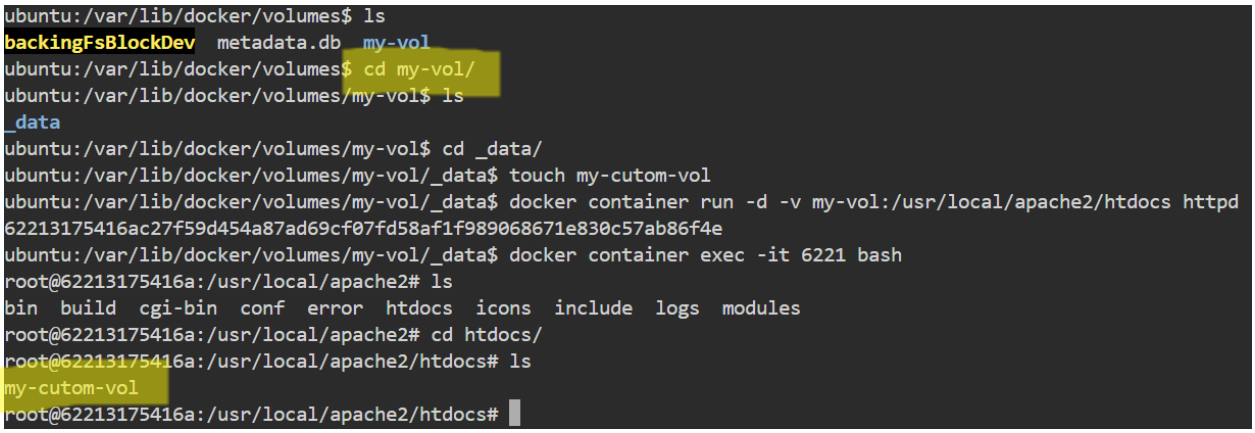
Create a Volume

Command:-

docker volume create my-vol

```
ubuntu:~$ docker volume create my-vol
my-vol
ubuntu:~$ docker volume ls
DRIVER    VOLUME NAME
local     my-vol
ubuntu:~$
```

```
ubuntu:~$ cd /var/lib/docker/
ubuntu:/var/lib/docker$ ls
buildkit  containers  engine-id  image  network  overlay2  plugins  runtimes  swarm  tmp  volumes
ubuntu:/var/lib/docker$ cd volumes/
ubuntu:/var/lib/docker/volumes$ ls
backingFsBlockDev  metadata.db  my-vol
ubuntu:/var/lib/docker/volumes$
```



Docker Image

→ A Docker Image is a blueprint (template) used to create Docker containers.

→ It contains:

→ Application code

→ Dependencies (libraries, packages)

→ Configuration files

→ Environment settings

→ You can think of an image like a snapshot or read-only template.

→ When you run an image → it becomes a container.

Docker pull nginx

```
ubuntu:~$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
d107e437f729: Pull complete
cb497a329a81: Pull complete
f1c4d397f477: Pull complete
f72106e86507: Pull complete
899c83fc198b: Pull complete
a785b80f5a67: Pull complete
6c50e4e0c439: Pull complete
Digest: sha256:d5f28ef21aabddd098f3dbc21fe5b7a7d7a184720bc07da0b6c9b9820e97f25e
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Types of Image Creation

1 Commit Method

2 Dockerfile Method

Commit mehtod:-

→ The `docker commit` command is used to create a new image from an existing container.

→ This is helpful when you:

→ Run a container

→ Make changes inside it (install packages, edit files, configure apps)

→ Then save those changes as a new Docker Image.

Push the Image on your DockerHub

Command :-

docker login –u username(dockerHub username)

```
ubuntu:~$ docker login -u mannu0
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/#credential-stores

Login Succeeded
ubuntu:~$
```

Create a Image for Docker Commit Method

Commands :-

Vim index.html

--> this is my commit method

- **Docker container run –d – name web httpd**
- **Docker container cp index.html web:/usr/local/apache2/htdocs**
- **Docker container commit –a "grras" web team:latest(team=image name)**

```
ubuntu:~$ docker container run -d --name web httpd
659de9e8527924d9cdf199c79074e9a6b7bc3059062964169c7d636e5d041eff
ubuntu:~$ docker container cp index.html web:/usr/local/apache2/htdocs
Successfully copied 2.05kB to web:/usr/local/apache2/htdocs
ubuntu:~$ docker container commit -a "grras" web taem:latest
sha256:88cfb09cd5ca47a2e279dec65513d9916f09c20810dfc39c461b489a9b278002
```

```
ubuntu:~$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED              SIZE
taem         latest    88cfb09cd5ca   About a minute ago   117MB
nginx        latest    41f689c20910   4 weeks ago          192MB
httpd        latest    65005131d37e   5 weeks ago          117MB
ubuntu:~$
```

Create a new container from custom image and hit the IP on browser and show the contant

```
ubuntu:~$ docker container run -d --name prod -p 80:80 taem:latest
078cd68da9d482ac1071688f695d2fcb4bb3c56ce8842994d40a6e3ca5b0a209
ubuntu:~$ docker container exec -it prod bash
root@078cd68da9d4:/usr/local/apache2#
```

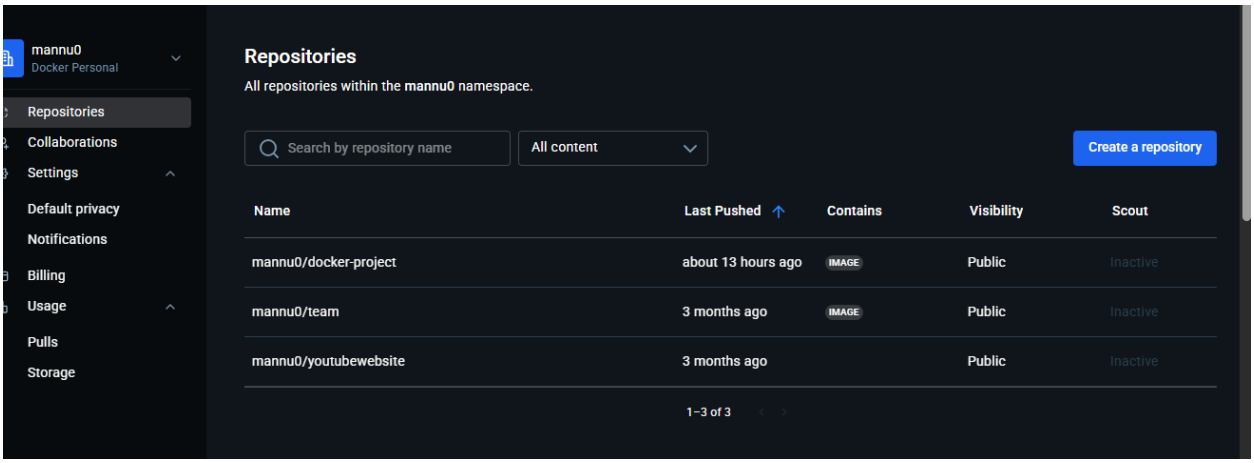
Push the Image on DockerHub

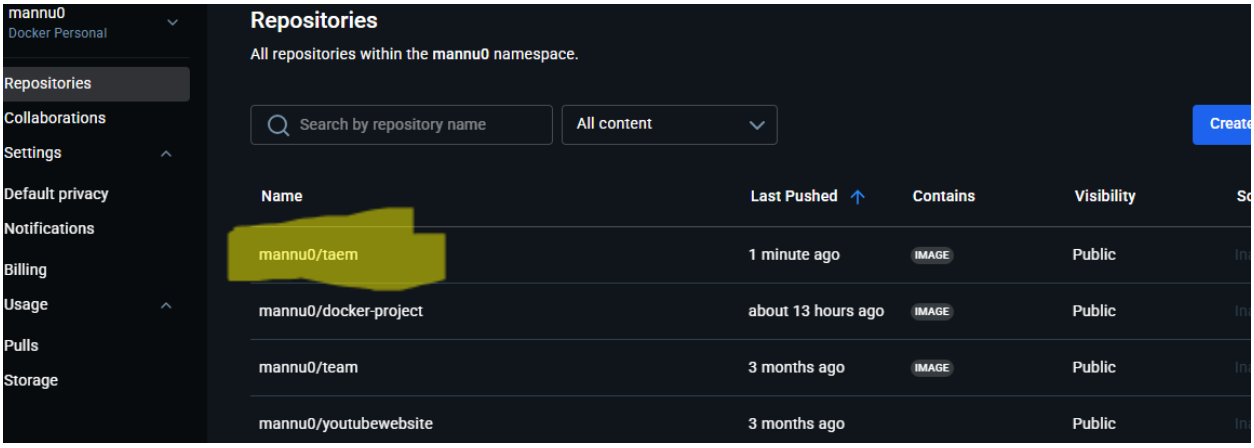
Command :-

docker image tag team:latest username/team

docker image push username/team:latest

```
ubuntu:~$ docker image tag taem:latest mannu0/taem
ubuntu:~$ docker image push mannu0/taem:latest
The push refers to repository [docker.io/mannu0/taem]
479455b1828f: Pushed
4ed3261aa08c: Mounted from library/httpd
664c74752319: Mounted from library/httpd
09c56534a346: Mounted from library/httpd
5f70bf18a086: Mounted from library/httpd
d694d07f5d65: Mounted from library/httpd
daf557c4f08e: Mounted from library/httpd
latest: digest: sha256:d6e98d3444f60fe5236c7bc554fb46bfadb2c2cc5380d065023592582d9fe09e size: 1779
ubuntu:~$
```





Dockerfile

→ A Dockerfile is a text file that contains a set of instructions to build a Docker Image.

→ Instead of making changes in a container and committing (using `docker commit`), we write instructions

in a Dockerfile → so the image can be built automatically and repeatedly.

→ It ensures consistency (same image every time you build).

Common Instructions in Dockerfile:

- `FROM` → Base image (e.g., ubuntu, alpine, nginx)
- `RUN` → Run commands (install packages)
- `COPY` → Copy files from host to image
- `WORKDIR` → Set working directory
- `CMD` → Default command to run when container starts
- `EXPOSE` → Inform which port container will use

mkdir docker

cd docker

Vim index.html

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y curl
COPY index.html /var/www/html/
WORKDIR /var/www/html
CMD ["cat", "index.html"]
~
```

Docker image build . tag web

Docker container run –d web:test

```
ubuntu:~/docker$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
<none>       <none>    02e3bfd1bb81   42 seconds ago   147MB
<none>       <none>    77b3af0d71ea   21 minutes ago   170MB
ubuntu       20.04     b7bab04fd9aa   5 months ago     72.8MB
```

Complete Docker