**Docker**

**­­­­­­­­What is Docker?**

It is a containerization platform to simplify the process of building, running, managing the application by using containers.

Containers are lightweight, portable units.

In containers stores package of application and application needs like code, libraries, dependencies.

So that applications can work efficiently in different environment.

All the Linux work with same kernel.

It will work with under line host kernel.

We can create a docker image in one Linux and can run in another Linux ( here docker will directly talk to the kernel ).

**Docker on window?**

Internally docker will work on Linux kernel

But in windows we have Docker desktop

Docker desktop will work on windows by enabling WSL ( Windows SubSystem for Linux ).

**Docker has:**

VM1 VM2 VM3

lib bin lib bin lib bin

Docker

Kernel

Hardware

**Core Concepts**

**Docker Image vs. Container**

**Docker Image:** A read-only filesystem that serves as a template for creating containers. Images can be local (stored on your machine) or remote (e.g., Docker Hub).

**Docker Container:** A running instance of an image. Containers are ephemeral (short-lived) and lightweight, starting and stopping quickly. They are simple processes, not full operating systems.

**Relationship:**

Containers are always created from images.

One image can spawn multiple containers.

An image can be created from a container (e.g., using docker commit).

**Snapshot vs. Image**

**Snapshot:** In a virtual machine (VM), changes are automatically written to a snapshot, which captures the state at a given point.

**Image:** Unlike snapshots, Docker images are read-only and immutable. Changes in a container don’t affect the original image unless explicitly committed.

**Storage**

All images and containers are stored in the storage driver:

Default path: /var/lib/docker/

Images use the overlay2 storage driver, consisting of four layers:

**LowerDir:** Base layers of the image.

**UpperDir:** Changes made in the container.

**MergedDir:** Combined view of LowerDir and UpperDir (what you see in the container).

**WorkDir:** Temporary working directory for operations.

**Image Caching**

Docker caches image layers during builds for efficiency.

**Example:**

If a Dockerfile has 10 layers and you modify line 5, Docker reuses the cached layers for lines 1–4 and rebuilds from line 5 onward, even if lines 6–10 are unchanged.

**Key Docker Commands**

Image Management

* **Show all images:** docker images
* **Remove an image:** docker rmi image\_name
* **Pull an image from Docker Hub:** docker pull image\_name
* **Inspect an image:** docker image inspect image\_name
* **Tag an image:** docker tag image\_name dockeruserId/image\_name

**Push to Docker Hub:**

* docker login -u username (use password or token)
* docker push dockeruserId/image\_name

**Container Management**

Create and enter a container:

* docker run -it image\_name bash (or sh)
  + -i: Interactive, -t: Allocate a terminal.
  + Opens a terminal inside the container (e.g., Ubuntu).
* **List running containers:** docker ps
* **List all containers (including stopped):** docker ps -a
* **Start a stopped container:** docker start container\_id
* **Stop a running container:** docker stop container\_id
* **Enter a running container:** docker exec -it container\_id bash
* **Remove a container:** docker rm container\_id
* **Remove all stopped containers:** docker container prune
* **Stop all running containers:** docker stop $(docker ps -a -q)
* **Remove all containers:** docker rm $(docker ps -a -q)
* **View container processes:** docker top container\_id
* **View container stats:** docker stats
* **View container logs:** docker logs container\_id

**Running Containers**

**Run in detached mode:** docker run -d image\_name

**Port forwarding:** docker run -p host\_port:container\_port image\_name

Example: docker run -p 80:80 -d nginx (maps port 80 on host to port 80 in container).

**Auto-restart:** docker run -d --restart=always image\_name

**Memory reservation:** docker run -d --memory-reservation=750m nginx (reserves 750 MB).

**Commit container to image:** docker commit -m "message" container\_id image\_name

**Dockerfile Instructions**

**FROM:** Specifies the base image (e.g., FROM ubuntu).

**RUN:** Executes commands during the image build process (e.g., RUN apt-get update).

**CMD:** Defines the default command to run when a container starts. Overridden if a command is provided at runtime.

Example: CMD ["echo", "Hello"].

**ENTRYPOINT:** Sets a fixed command that always runs when the container starts. Arguments can be appended.

Example: ENTRYPOINT ["nginx", "-g", "daemon off;"].

**Bind Mounts**

Mounts a host directory into a container.

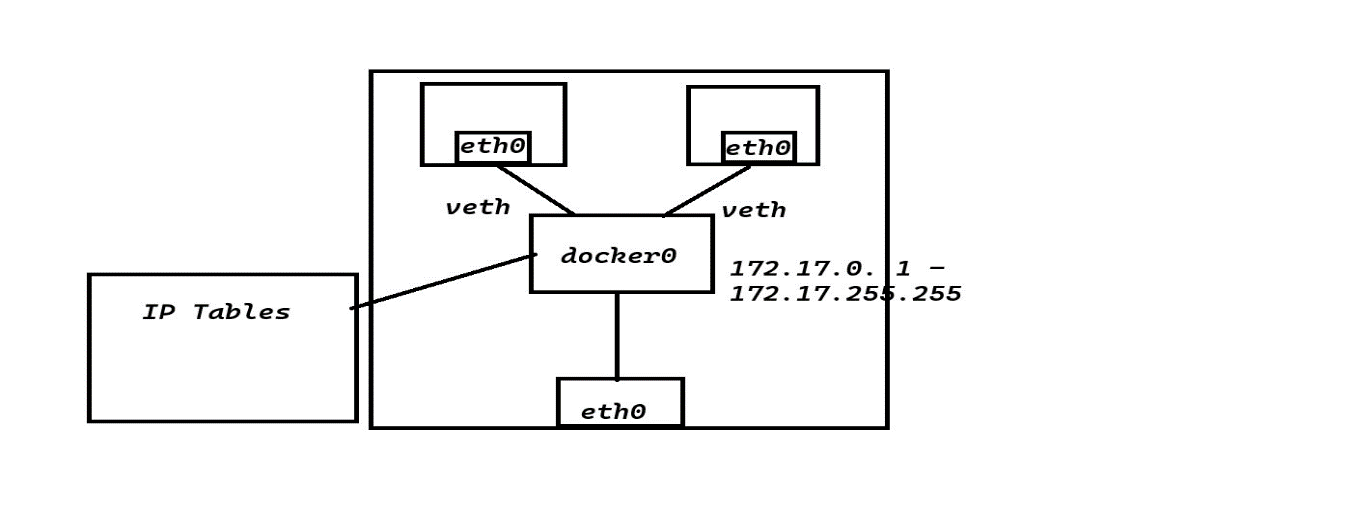
**Syntax:** docker run -it -v /host\_path:/container\_path image\_name bash

**Examples:**

docker run -it -v /hostdb:/contdb ubuntu bash

docker run -d -e MYSQL\_ROOT\_PASSWORD=admin -v /home/docker/mysql-data:/var/lib/mysql --name mysqlserver mysql

**Networking**



**List networks:** docker network ls

**Create a network:** docker network create network\_name

**Run container in a network:**

docker run -d --net network\_name image\_name

docker run -it --net network\_name image\_name bash

**Use host network:** docker run -d --net host nginx (container uses host’s network stack).

**Inspect container network:** docker container inspect container\_name

**Example Commands**

Run Nginx with port forwarding: docker run -d -p 8080:8080 nginx

Check running processes (system-level): ps -ef | grep runc

**Docker Networking**

**Networking involves a few basic things:** An IP address is a unique number given to a device so it can be found on a network. An Ethernet card is the part of a device that connects it to the network. An Ethernet cable is the wire that links devices. A gateway is like a door that lets data move between networks.

**NATing and DNATing** are ways to change addresses so devices can talk to the outside world. Subnetting splits a big network into smaller ones. A network switch is a device that helps devices share data.

An IP address is a special number that lets a device work on a network, following rules called network protocols. Networks are often split into smaller groups called subnets.

When you install Docker, it sets up something called "docker0." This is like a switch that helps two containers talk to each other. Docker has six different network types, but it mainly uses three: Bridge Network, Host Network, and None Network.

The Bridge Network is the default, and it’s handled by docker0.

Every container you create connects to docker0 unless you say otherwise. Each container gets its own virtual connection (called "veth") that links its network card (called "eth0") to docker0.

Docker0 has an IP address range from 172.17.0.1 to 172.17.255.255, and it uses a subnet called 172.17.0.0/16.

If a container doesn’t respond to a request, docker0 sends the data to the computer’s main network card (also called "eth0").

In the Host Network, a container connects straight to the computer’s network instead of using docker0.

This is fast, but there’s a catch: if one container is using a port (like port 80 for a website), another container on the same host network can’t use that same port—it won’t work because they’d clash.

The None Network means a container has no network at all. It’s cut off from everything, which can be useful sometimes.