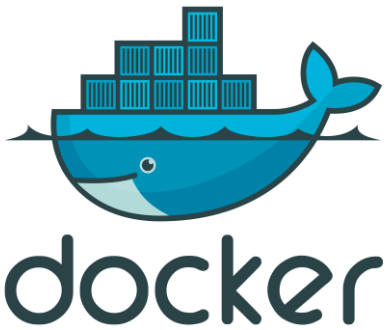
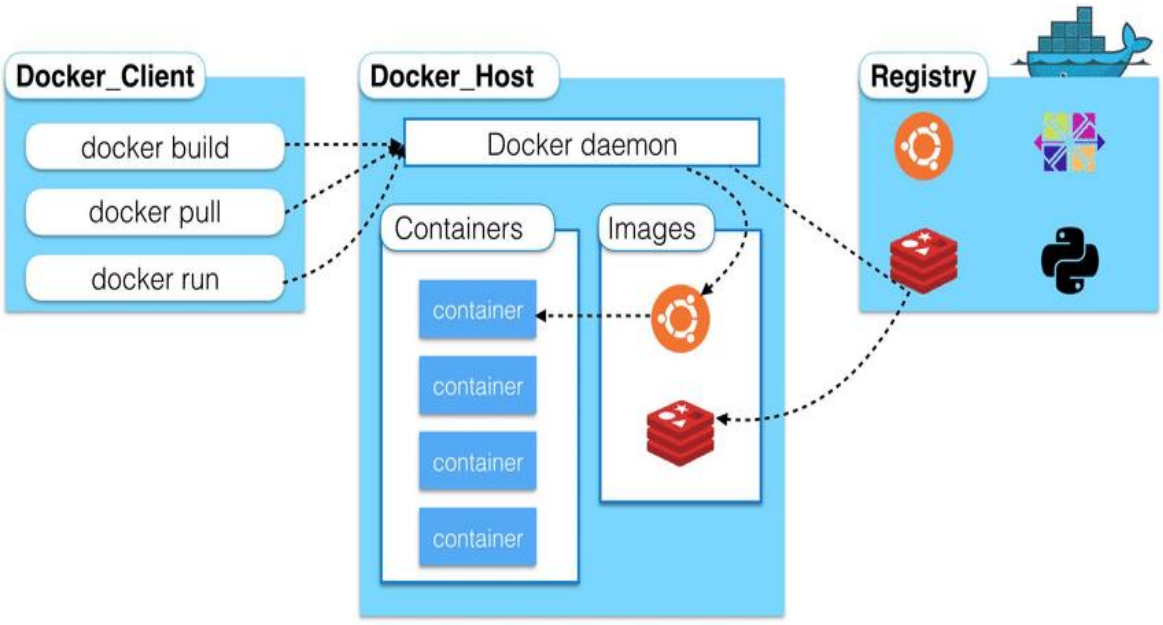

docker

# DOCKER

- Docker is an open platform tool for developing, shipping, & running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system.
- Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries.
- It is designed to benefit both developers and system administrators, making it a part of many DevOps tool chains.

## ❖ DOCKER ARCHITECTURE:

- Docker uses a **client-server** architecture. The Docker **client** talks to the **Docker daemon**, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote **Docker daemon**.



## DOCKER DAEMON:

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

## DOCKER CLIENT:

- The Docker client (docker) is the primary way that many Docker users interact with Docker.

## DOCKER REGISTRIES:

- 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.

## DOCKER OBJECTS:

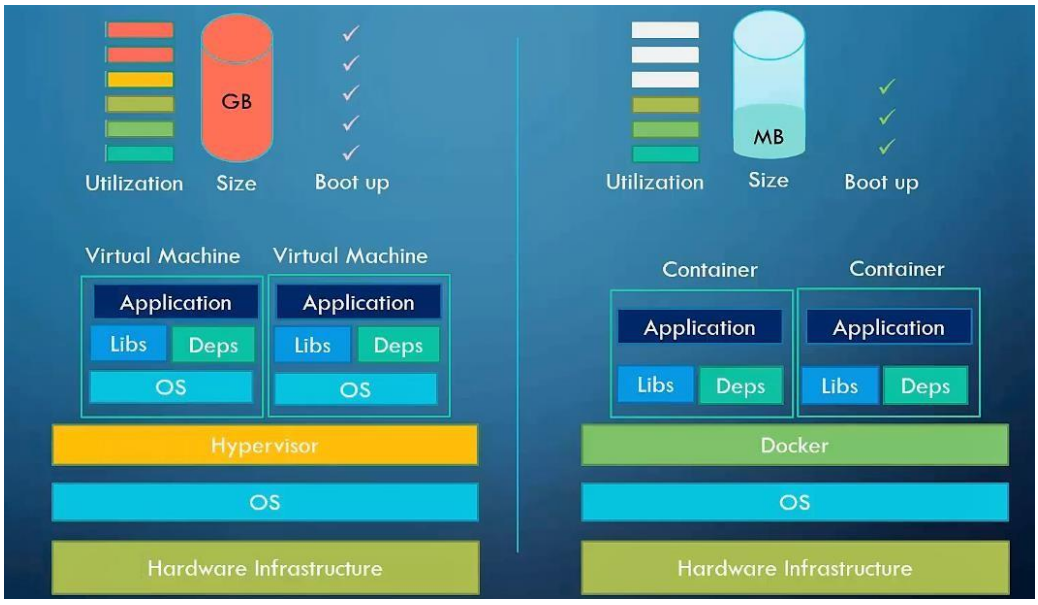
## IMAGES:

- An *image* is a read-only template with instructions for creating a Docker container.
- A container is launched by running an image. An **image** is an executable package that includes everything needed to run an application–the code, a runtime, libraries, environment variables, and configuration files.

## CONTAINERS:

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.
- Docker Containers are:
  - **Flexible** : Even the most complex app's can be containerized.
  - **Lightweight** : Containers leverage and share the host kernel.
  - **Interchangeable:** You can deploy updates and upgrades on-the-fly.
  - **Portable** : Can build locally, deploy to the cloud and run anywhere
  - **Scalable** : Can increase & automatically distribute container replicas.
  - **Stackable** : You can stack services vertically and on-the-fly.

# ❖ VIRTUAL MACHINES VS CONTAINERS:




## VIRTUAL MACHINES:

- A virtual machine (VM) is a virtual environment that functions as a virtual computer system with its own CPU, memory, network interface, and storage, created on a physical. In other words, creating a computer within a computer.
- Multiple virtual machines can run simultaneously on the same physical computer.

## CONTAINERS:

- A container is a running instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.

# ❖ DOCKER INSTALLATION:

## DOCKER ENGINE OVERVIEW:

- Docker Engine is an open-source containerization technology for building and containerizing your applications.
- Docker Engine acts as a client-server application with**:**
    - A server with a long-running daemon process dockerd.
    - APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
    - A command line interface (CLI) client docker.

## SUPPORTED PLATFORMS:

- Docker Desktop for Mac (macOS)
- Docker Desktop for Windows
- Linux distributions**:**
  Red Hat, Centos, Fedora, Debian, Ubuntu…etc.
- Cloud Platforms**:**
  AWS, AZURE, GCP, Digital Ocean…. etc.

## INSTALL DOCKER DESKTOP ON WINDOWS:

- Windows 10 64-bit: Home or Pro 2004 (build 19041) or higher, or Enterprise or Education 1909 or higher.
- Enable the WSL 2 feature on Windows. (Windows Subsystem for Linux, version 2)
- The following hardware prerequisites are required to successfully run WSL 2 on Windows 10:
    - 64-bit processor with Second Level Address Translation (SLAT)
    - 4GB system RAM
    - BIOS-level hardware virtualization support must be enabled in the BIOS settings.
    - Download and install the Linux kernel update package: https://docs.microsoft.com/en-us/windows/wsl/install-win10#step-4---download-the-linux-kernel-update-package

**NOTE:** https://docs.docker.com/engine/install/

# INSTALL DOCKER ENGINE ON LINUX:

## INSTALL ON RED HAT / CENTOS:

- To install Docker Engine, you need a maintained version:
  - **RedHat / CentOS 7, 8, 9**
  - **Ubuntu 18, 20, 21**

# DOCKER COMMANDS

| COMMAND | DESCRIPTION |
|---|---|
| **docker attach** | : Attach local standard input, output, & error streams to a running container |
| **docker build** | : Build an image from a Docker file |
| **docker checkpoint** | : Manage checkpoints |
| **docker commit** | : Create a new image from a container's changes |
| **docker config** | : Manage Docker configs |
| **docker container** | : Manage containers |
| **docker cp** | : Copy files/folders between a container and the local filesystem |
| **docker create** | : Create a new container |
| **docker deploy** | : Deploy a new stack or update an existing stack |
| **docker diff** | : Inspect changes to files or directories on a container's filesystem |
| **docker events** | : Get real time events from the server |
| **docker exec** | : Run a command in a running container |
| **docker export** | : Export a container's filesystem as a tar archive |
| **docker history** | : Show the history of an image |
| **docker image** | : Manage images |
| **docker images** | : List images |
| **docker import** | : Import the contents from a tar ball to create a filesystem image |
| **docker info** | : Display system-wide information |
| **docker inspect** | : Return low-level information on Docker objects |
| **docker kill** | : Kill one or more running containers |
| **docker load** | : Load an image from a tar archive or STDIN |
| **docker login** | : Log in to a Docker registry |
| **docker logout** | : Log out from a Docker registry |
| **docker logs** | : Fetch the logs of a container |
| **docker network** | : Manage networks |

**docker node** : Manage Swarm nodes

**docker pause** : Pause all processes within one or more containers

**docker plugin** : Manage plugins

**docker port** : List port mappings or a specific mapping for the container

**docker ps** : List containers

**docker pull** : Pull an image or a repository from a registry

**docker push** : Push an image or a repository to a registry

**docker rename** : Rename a container

**docker restart** : Restart one or more containers

**docker rm** : Remove one or more containers

**docker rmi** : Remove one or more images

**docker run** : Run a command in a new container

**docker save** : Save one or more images to a tar archive (streamed to STDOUT)

**docker search** : Search the Docker Hub for images

**docker secret** : Manage Docker secrets

**docker service** : Manage services

**docker stack** : Manage Docker stacks

**docker start** : Start one or more stopped containers

**docker stats** : Display a live stream of container(s) resource usage statistics

**docker stop** : Stop one or more running containers

**docker system** : Manage Docker

**docker tag** : Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

**docker top** : Display the running processes of a container

**docker unpause** : Unpause all processes within one or more containers

**docker update** : Update configuration of one or more containers

**docker version** : Show the Docker version information

**docker volume** : Manage volumes

**docker wait** : Block until one or more containers stop, then print their exit codes

# DOCKER IMAGES

#docker search centos

#docker pull centos

#docker images (or) #docker image ls

**Image History:**

#docker image history a9d583973f65

**Image Details:**

#docker image inspect ubuntu

**Removing dangling images:**

A dangling image is an image that is not tagged and is not used by any container.

**To remove dangling images:**

#docker image prune

**Remove Image:**

#docker rmi imageid or docker images rm imageid

**Removing all unused images**

#docker image prune -a

## MANIPULATING DOCKER IMAGES:

#docker run -i -t <imagename>:<tag> /bin/bash

**Options: -i** : Gives us an interactive shell into the running container

**-t :** Will allocate a pseudo-tty

**-d** : The daemon mode

#docker run -i -t centos:latest /bin/bash

#docker ps (or) #docker container ls [List running containers] [on another terminal]

#docker ps -a [all containers]

#docker stop cid  [Application shutdown gracefully]

#docker start -a cid  [-a attach mode]

**Rename Container:**

#docker rename <current_container_name> <new_container_name>

**Container stats:**

#docker stats <container_name>

#docker stats cid

**Monitor Container:**

#docker top cid/name

**Container Pause:**

#docker container pause containerID

Container Unpause:

#docker container unpause cid

**Kill one or more Containers:**

#docker kill cid  [no time for proper shutdown of Application]

**Removing Containers:**

#docker container rm cid  (or) #docker rm cid

**Removing all stopped containers:**

#docker container prune

**Docker System and stat:**

#docker logs cid

#docker logs -f cid

#docker system

#docker system df

#docker system events [Two terminals] #docker stop containerID / Launch new container

#docker system prune [Delete all stopped and unused containers]

#docker system prune -a [Delete all images and stopped containers]

## PORT FORWARDING:

#docker run -d -p <host_port>:<container_port> <image>:<tag>

#docker pull nginx

#docker run --name mynginx -d -p 8080:80 nginx/imageid

**Go To Web Browser**

http://host-ipaddress:8080/

#docker logs cid

#docker logs -f cid

Allocate Memory for Container:

#docker status ContID

#docker run -d -p 8000:80 -m 512m nginx  [Container memory limit 512m max]

#docker run -d -p 8000:80 -m 512m --cpu-quota=50000 nginx [cpulimit 50000(50%) total cpu size 100thousend]

# DOCKER FILE

- Dockerfile is the core file that contains instructions to be performed when an image is built.
- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- The docker build command builds an image from a Dockerfile and a context.
- Here syntax to write instruction and its arguments with in a Dockerfile.

```
# comment
Instruction  arguments
```

## DOCKER FILE INSTRUCTIONS & ARGUMENTS:

### FROM:

- A docker file must start with a FROM instruction.
- The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions.

**Ex:** FROM centos:latest

### MAINTAINER:

- MAINTAINER Instruction is used to specify about the author who creates this new docker image for the support.

**Ex:** MAINTAINER RAJU rnraju4u@gmail.com

### LABEL:

- It is used to specify metadata information to an Image. A LABEL is a key-value pair.

**Ex:** LABEL "App_Env"="Production"

## RUN:

- It is used to executes any commands on top of the current image and this will create a new layer.
- It has two forms: Shell form & Executable form

**Shell Form:**

**Ex:** RUN yum update

RUN yum install httpd -y

**Executable form:**

**Ex:** RUN ["yum","update"]

RUN ["Systemctl","start","sshd"]

## CMD:

- It is used to set a command to be executed when running a container.
- There must be only one CMD in a Dockerfile. If more than one CMD is listed, only the last CMD takes effect.
- It has two forms: Shell form & Executable form

**Ex:** CMD ping google.com

CMD python myapplication.py

## ENTRYPOINT:

- It is used to configure and run a container as an executable.
- It has two forms: Shell form & Executable form.

**Ex:** ENTRYPOINT ping google.com

ENTRYPOINT python myapplication.py

**NOTE:** If user specifies any arguments (commands) at the end of "docker run" command, the specified commands override the default in CMD instruction, But ENTRYPOINT instructional are not overwritten by the docker run command and ENTRYPOINT instruction will run as it is.

## EXPOSE:

- This instruction informs Docker that the container listens on the specified network ports at runtime.
- By default, EXPOSE assumes TCP. You can also specify UDP.
- To publish the port when running the container, use the -p flag on docker run

Ex: EXPOSE 80/tcp

## COPY:

- It is used to copy files, directories and Remote url filesto the destination (Docker image) within the file system of the docker images.
- It has two forms: Shell form & Executable form

EX: COPY src dest

COPY /root/file /tmp

**NOTE:** If the "src" argument is compressed file (tar, zip bzip2..etc), then it will copy exactly as a compressed file and will not extract.

## ADD:

- It is used to copy files, directories and Remote URL files to the destination within the file system of the docker images.
- It has two forms: Shell form & Executable form

EX: ADD src Dest

ADD /root/file /tmp

**NOTE:** If the "src" argument is compressed file (tar, zip bzip2..etc), then it will Extract it automatically inside a destination in the Docker image.

## WORKDIR:

- It is used to set the working directory.

  EX: WORKDIR /tmp

## USER:

- The USER instruction lets you specify the username to be used when a command is run.
- It is used to set the user name, group name, UID, GID for running subseqent commands. Else root user will be used.

**Ex:** USER webadmin

USER webadmin:webgroup

USER 1010

## ENV:

- It is used to set environmental variables with key and value set.

  ENV <key> <value>
  ENV username admin

  **(or)**

  ENV <key>=<value>
  ENV username=admin

**Ex:** ENV username=admin database=mydb tableprefix=pr2_

## ARG:

- It is also used to set environment variables with key and value, but this variable will set only during the image build not on the container.
  **Ex:** ARG tmp_ver 2.0

## ONBUILD:

- The ONBUILD instruction lets you stash a set of commands that will be used when the image is used again as a base image for a container.

  **Ex:** ONBUILD ADD

  ONBUILD RUN

# BUILDING IMAGES USING DOCKERFILE

- The docker build command builds Docker images from a Dockerfile and a "context".
- A build's context is the set of files located in the specified PATH or URL.

#docker build .

#docker build -f <path_to_Dockerfile> -t <REPOSITORY>:<TAG>

## EXAMPLE 1:

```
FROM centos:latest
MAINTAINER RAJU rnraju4u@gmail.com
RUN yum update -y
LABEL "Env"="Prod" \
      "Proj"="Airtel" \
      "Version"="8.4"
copy *.txt /opt/
ADD backup.tar /tmp/
ENTRYPOINT ping google.com
```

## PYTHON FLASK:

```
# Super simple example of a Dockerfile
FROM centos:latest
MAINTAINER Raju "rnraju4u@gmail.com"
RUN yum update -y
RUN yum install -y python3 python3-pip wget
RUN pip3 install Flask
ADD hello.py /home/hello.py
WORKDIR /home
#docker build -t python3:centos .
#docker ps
#docker run -d -p 5000:5000 python:centos python3 hello.py
```

## WEBSERVER:

```
FROM centos
RUN yum install httpd -y
COPY index.html /var/www/html/
CMD ["/usr/sbin/httpd","-D","FOREGROUND"]
EXPOSE 80

#docker run -d -p 8080:80 webserver
http://10.10.10.10:8080
```

# DOCKER HUB

- The Docker Hub is a location on the cloud, where you can store and share images that you have created.
- Can also link your images to the GitHub or Bitbucket repositories that can be built automatically based on web hooks.
- There are two types of repositories on the Docker Hub:

## PUBLIC REPOSITORIES:

- Users get access to free public repositories for storing and sharing images.
- Anyone can use the docker pull command to download an image to their local system and run or build further images from it.

## PRIVATE REPOSITORIES:

- Private repositories are just that private.
- Users can choose a subscription plan for private repositories.

## COMPARING DOCKER HUB TO DOCKER SUBSCRIPTION:

### DOCKER HUB:

- Shareable image, but it can be private
- No hassle of self-hosting
- Free (except for a certain number of private images)

### DOCKER SUBSCRIPTION:

- Integrated into your authentication services (that is, **AD/LDAP**)
- Deployed on your own infrastructure (or **cloud**)
- Commercial support

**NOTE:** By default, repositories are pushed as public. If you want to set them to private, log in to the Docker Hub website and set the repository to Make Private. You can also mark images as unlisted, so they don't show up in the Docker searches. You can also mark them as listed at a later date as well.

## DOCKER HUB ENTERPRISE:

- Docker Enterprise offers you is access to the software, access to updates/patches/security fixes, and support relating to issues with the software.
- The open-source Docker repository image doesn't offer these services at this level;

## DOCKER REGISTRY:

- A Docker registry is a storage and distribution system for Docker images. The same image might have multiple different versions, identified by their tags.
- A Docker registry is organized into Docker repositories, where a repository holds all the versions of a specific image.

**EXEC:** Run a command in a running container.

The docker exec command runs a new command in a running container.

**SYN:** #docker exec [options] container-id command [Arg….]

Ex: #docker exec -d ebac99faf1f2 touch /opt/php

#docker exec -it ebac99faf1f2 bash

#ls /opt

**COMMIT:** Create a new image from a container's changes.

It can be used to commit a container's file changes or settings into a new image.

The commit operation will not include any data contained in volumes mounted inside the container.

**SYN:** #docker commit [options] CONTAINER [Repository[:tag]]

Ex: # docker commit c3f279d17e0a rnraju/testimage:version3

#docker image ps

# DOCKER NETWORKING

- Docker includes support for networking containers through the use of **network drivers**.
- Docker's networking subsystem is pluggable, using drivers.

**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.**

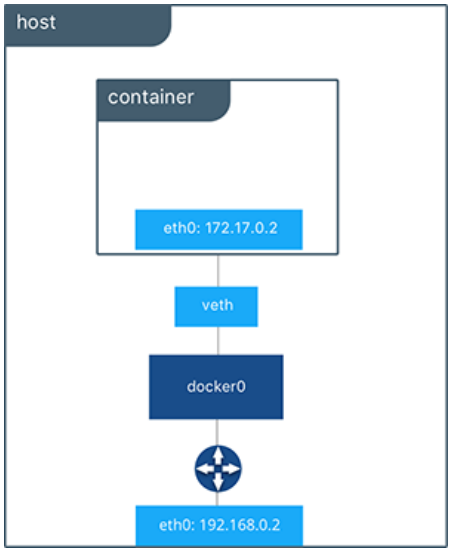
**HOST:**
For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly.

**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.

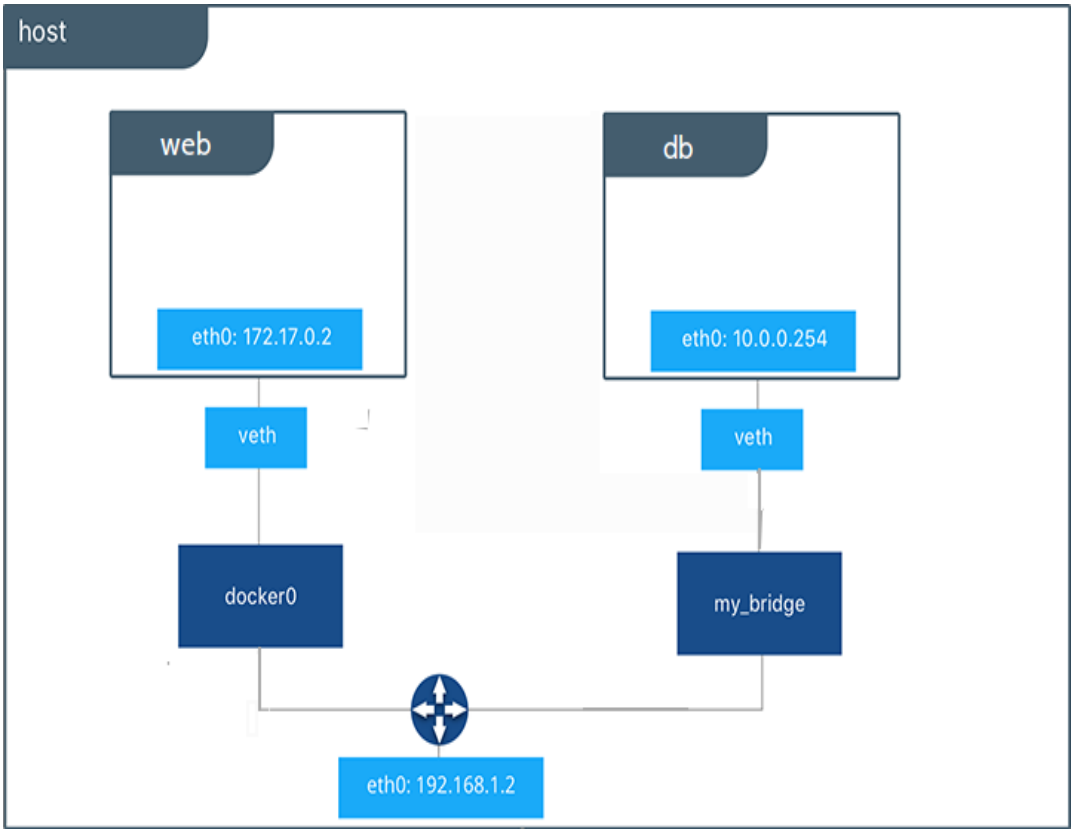
**MACVLAN:**
Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network. The Docker daemon routes traffic to containers by their MAC addresses. Using the macvlan driver is sometimes the best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.

**NONE:** For this container, disable all networking. Usually used in conjunction with a custom network driver. none is not available for swarm services.

# LAUNCH A CONTAINER ON THE DEFAULT NETWORK




# CREATE YOUR OWN BRIDGE NETWORK:



**NOTE:** The relationship between a host and containers is 1:N.

#ifconfig

#docker network ls

#brctl show

#apt-get install bridge-utils (yum install bridge-utils)

#brctl show docker0  [check it number of interfaces are running]

#docker network inspect bridge

#docker run --rm -it --name=test1 alpine sh

#ifconfig [it will get 172.x.0.2]

#brctl show docker0

#docker network inspect bridge [on another terminal]

#docker run --rm -it --name=test2 alpine sh

#ifconfg  [it will get 0.3]

#ping 172.17.0.2

[it will communicate baoth because both are running on same bridge(same network)]

#cat /etc/hosts

172.17.0.3  8a77e2b61724

#hostname

To change hostname:

#docker run --rm -it --name=test2 --hostname test2.example.com alpine sh

#hostname

#cat  /etc/hosts

#brctl show docker0

##################################################################

**Create a custom bridge network:**

#docker network create my-network

#docker network ls

#docker network inspect my-network

Connect a container to a user-defined bridge:

#docker run -it --name my-nginx --network my-network -p 8080:80 nginx:latest

#docker network rm my-network

#docker network create test-network --subnet 192.168.0.0/16 --gateway 192.168.0.1

#docker network inspect test-network

#docker run -it --net test-network --name test3 alpine sh

################ Communicate different containers ##################

#ping 172.17.0.3

#docker network connect bridge test3

#ping 172.17.0.3 [from 192.168.0.1 container]

Disconnect a container from a user-defined bridge:

#docker network disconnect my-network test3

################## HOST NETWORK ##########################

#docker run --rm -d --network host --name my_nginx nginx

Access Nginx by browsing to http://localhost:80/.

#ip addr show

#netstat -tulpn | grep :80

Disable networking for a container:

#docker run --rm -dit --network none --name no-net-alpine alpine sh

#docker exec no-net-alpine ip link show

#docker exec no-net-alpine ip route [empty ouput because there is no routing table]

**HOST NETWORK:**

#docker run -it --network host alpine sh

#ifconfig

**NONE NETWORK:**

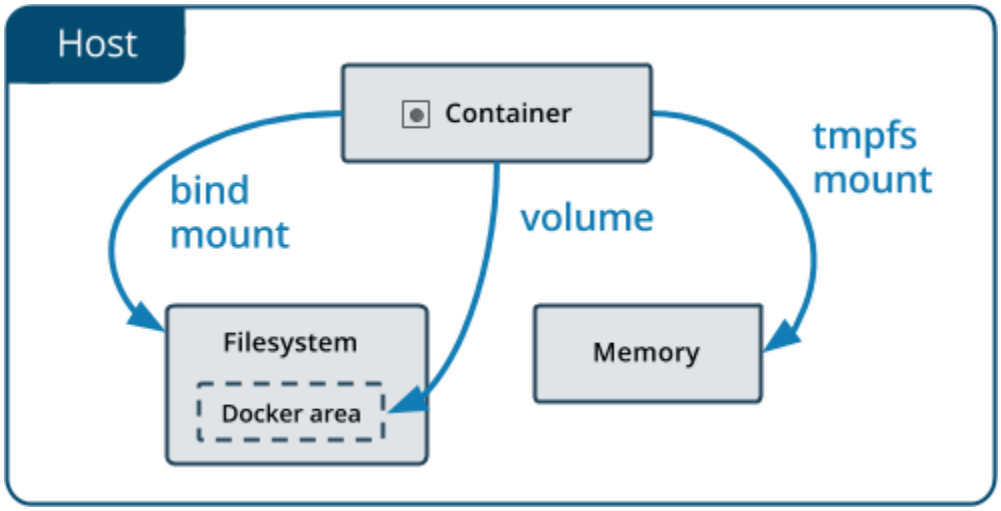
#docker run -it --network none alpine sh

#ifconfig

**NOTE:** only one instance of "host" and "null" networks are allowed.

# MANAGING DATA IN DOCKER

- By default, all files created inside a container are stored on a writable container layer. This means that:
  - The data doesn't persist when that container no longer exists.
  - You can't easily move the data somewhere else.
- Docker has two options for containers to store files in the host machine persistently.
  - **volumes**
  - **bind mounts**
- Running Docker on Linux you can also use a **tmpfs** mount.
- Running Docker on Windows you can also use a **named pipe**.




## VOLUMES:

- Volumes are stored in a part of the host filesystem which is managed by Docker **(/var/lib/docker/volumes/** on Linux).
- Volumes are the best way to persist data in Docker.

## BIND MOUNTS:

- **Bind mounts** may be stored *anywhere* on the host system. They may even be important system files or directories.

## TMPFS MOUNTS:

- **tmpfs mounts** are stored in the host system's memory only, and are never written to the host system's filesystem.

### USE A TMPFS:

#docker run -d -it --name tmptest --tmpfs /app ubuntu

#docker ps

#docker exec -it <CID> /bin/bash

#cd /app

#touch abc

#exit

#docker stop <CID>

#docker start <CID>

#docker exec -it <CID> /bin/bash

#cd /opt

#ls  [No files here]

### VOLUMES:

#docker pull ubuntu

#docker images

#docker run -it -v /my-data ubuntu bash

#cd /my-data

ls

**NOTE:** By default volume mounted to /var/lib/docker/volumes/someidname

#docker inspect <CONTAINER_ID>

can also use multiple -v volume switches on a single docker run line:

$docker run -it -v /my-data -v /data ubuntu /bin/bash

docker volume:

Creates a new volume that containers can consume and store data in. If a name is not specified, Docker generates a random name.

Syn: #docker volume create [options] [VOLUME]

#docker volume create my-vol

#docker volume ls

#cd /var/lib/docker/volumes

#ls

#docker run -it -v my-vol:/world ubuntu bash

#cd /world

#ls

# DOCKER COMPOSE

- Docker Compose is a tool for defining and running multiple containers as a single service.

## USING COMPOSE IS BASICALLY A THREE-STEP PROCESS:

**STEP1:** Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.
**STEP2:** Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.
**STEP3:** Run **docker compose up** and the **Docker compose command** starts and runs your entire app.

## DOCKER COMPOSE FILE STRUCTURE:

```
version: 'X'

services:

web:

  build: .

  ports:

   - "5000:5000"

  volumes:

   - .:/code

 redis:

  image: redis
```

## SERVICE:

- A **service** can be run by one or multiple containers.
- Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.

# INSTALLATION:

#curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compos

#chmod +x /usr/local/bin/docker-compose

#ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

(or)

#pip install -U Docker-compose

#docker-compose --version

## Upgrading:

**#docker-compose migrate-to-labels**

## Uninstallation:

**#rm /usr/local/bin/docker-compose**

**(or)**

**#pip uninstall docker-compose**

## Example 1:

**Step1: Create docker compose file at any location on your system.**

**#mkdir /dockercompose**

**#vim docker-compose.yml**

```
version: '3'   ###https://docs.docker.com/compose/compose-file/      [versions]
services:
  web:
    image: nginx
    ports:
       - 9090:80
  database:
    image: redis:
```

**Step2: Check the validity of file**

#docker-compose config

**Step3: Run the file**

#docker-compose up -d

#docker-compose ps or #docker ps

Step4: Bring down application

#docker-compose down

## SCALE A SERVICES:

**#docker-compose up -d --scale database=4**

**#docker-compose down**