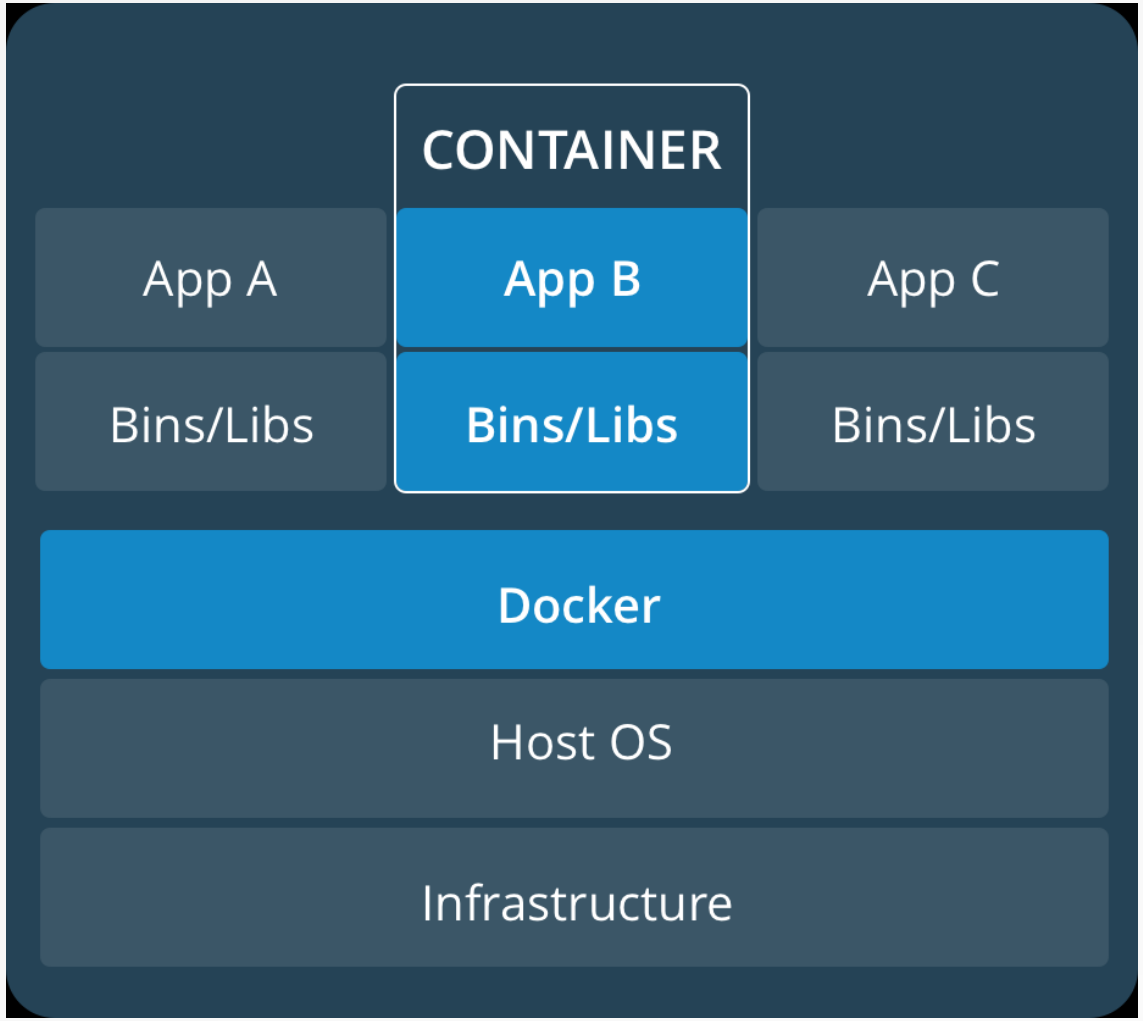
DOCKER

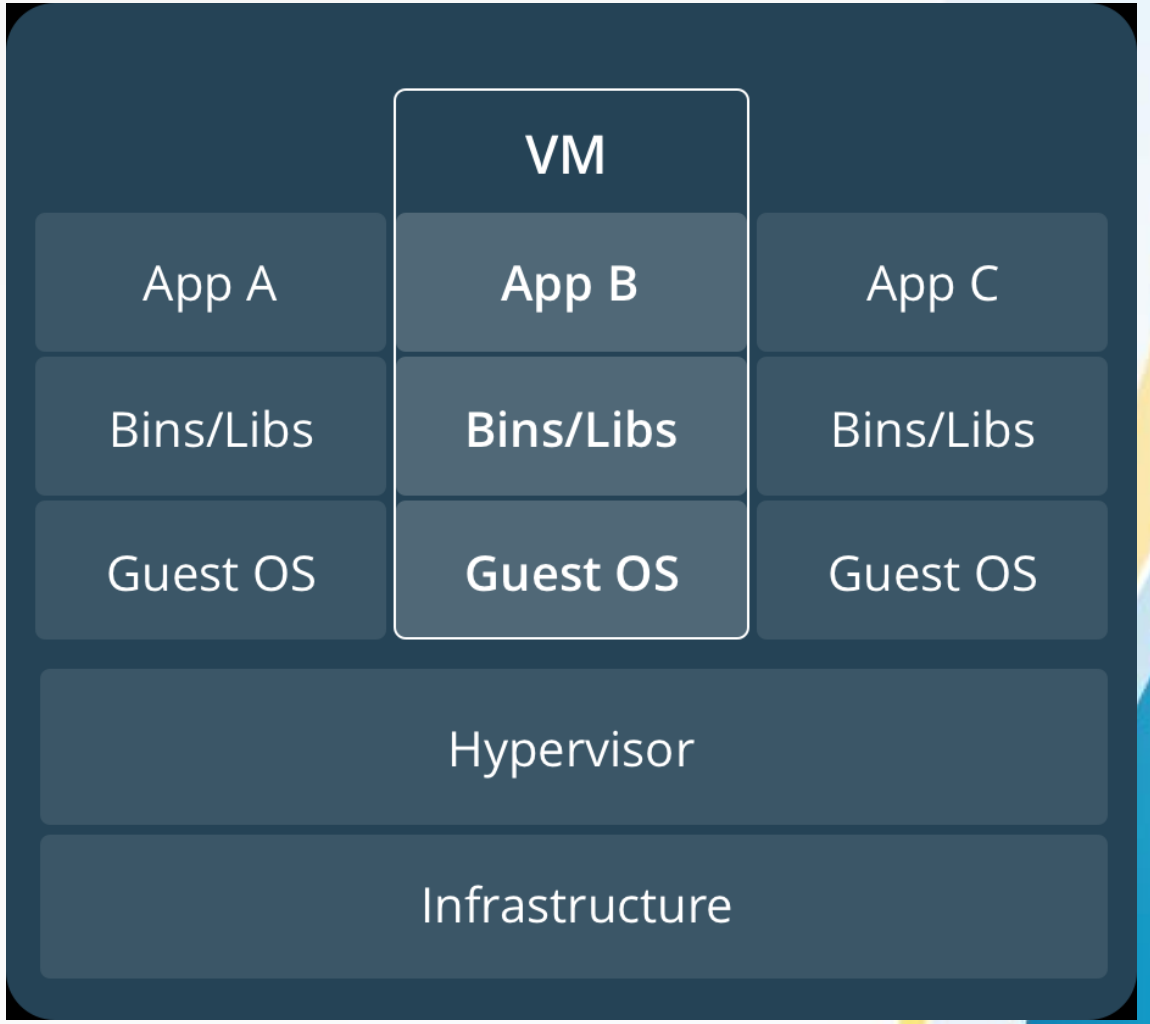
Why Docker?

- Compatability
- Portability
- Easy to develop and deploy
- Efficient use of system resource
- Lightweight containers
- Faster development life cycle

Container vs Virtual Machine



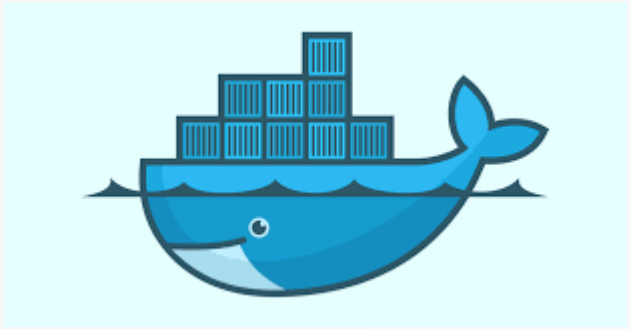
CONTAINER

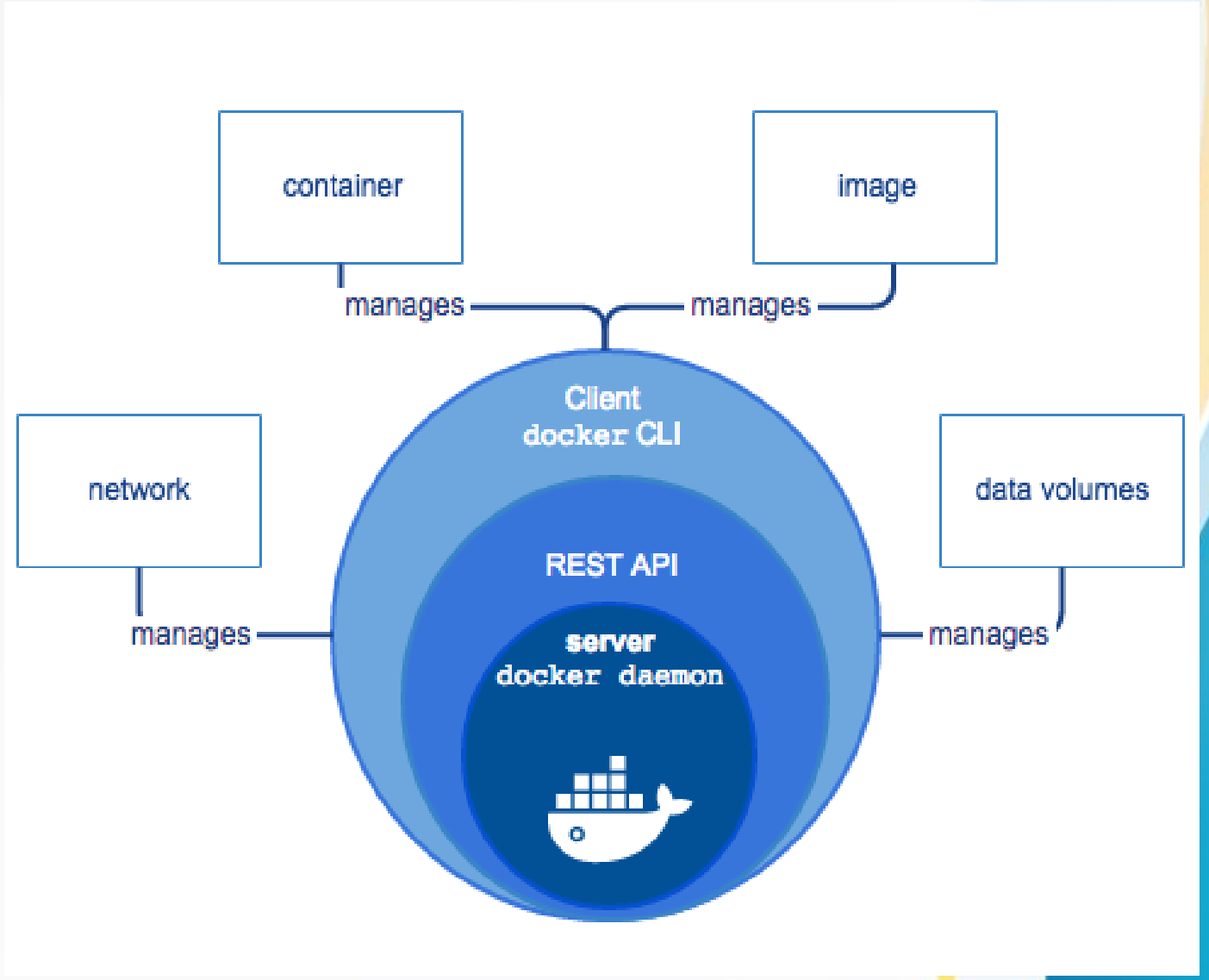


VIRTUAL MACHINE

Docker

- Docker is an open-source engine that automates the deployment of application inside the containers.
- Provides an isolated environment from the infrastructure.





Image

Docker image is a read-only template to create docker container. Often, image is based on the another image.

Container

Docker container is the runnable instance of the image. By default, a container is relatively well isolated from other containers and its host machine.

Registry

Docker registry stores Docker images. Docker hub is a public repository that anyone can use. Docker is configured to look for images on Docker hub by default.

Run Docker Image

- Docker Hub provides official image for many application.

Pull docker image

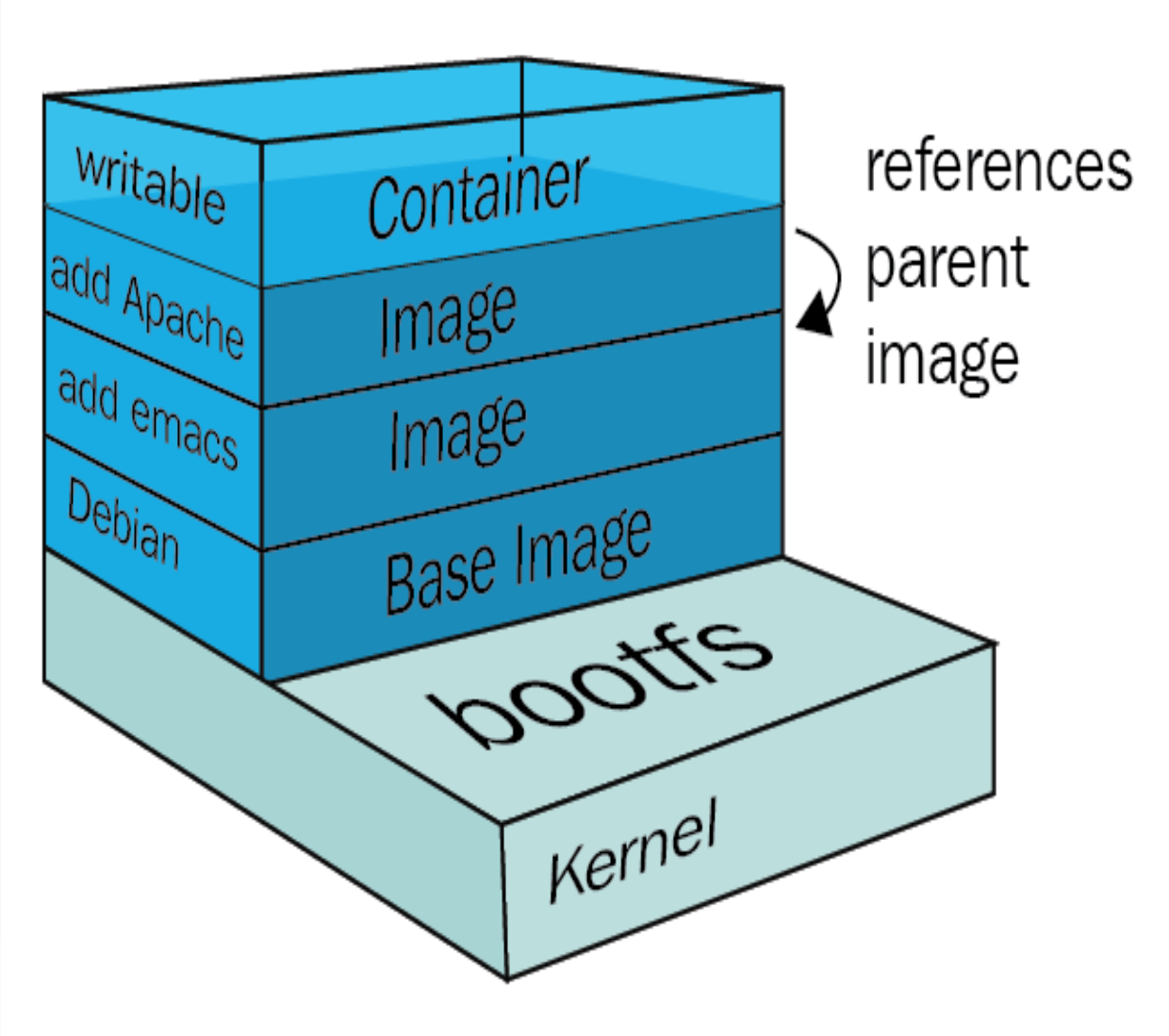
docker pull <image_name>
docker pull ubuntu
docker pul ubuntu:16.04

Run docker image

docker run -it --name <container_name> <image_name>
docker run -it --name ubuntudemo ubuntu

Build own docker image

- Each instructions in a Dockerfile creates a layer in the image.
- Docker images are immutable, but you can add an extra layer and save them as a new image.
- **Container/Writable layer**: Containers may share access to the underlying layer of a docker images but this layer is unique to the containers.



Dockerfile Commands

- **FROM** - initializes a new build stage and sets the Base Image for subsequent instructions. (FROM ubuntu)
- **RUN** - execute any commands in a new layer on top of the current image and commit the results.(RUN apt-get install python)
- **LABEL** - adds metadata to an image.(LABEL descripition - “Sample image”)
- **MAINTAINER** - sets the author field of the generated images.(MAINTAINER maintainer = “xyz@abc.co.in”)
- **ENV** - sets the environment variable.() (ENV <key> <value>)

ADD and COPY

- **COPY** - copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
- **ADD** - copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.

CMD and ENTRYPOINT

- ENTRYPOINT - Configure a container that will run as an executable.
- CMD - to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well.

FROM UBUNTU
ENTRYPOINT [“echo”]
CMD [“Welcome”]

1) docker run -t ubuntu
Welcome
2) docker run -t ubuntu “Hello World”
Hello World

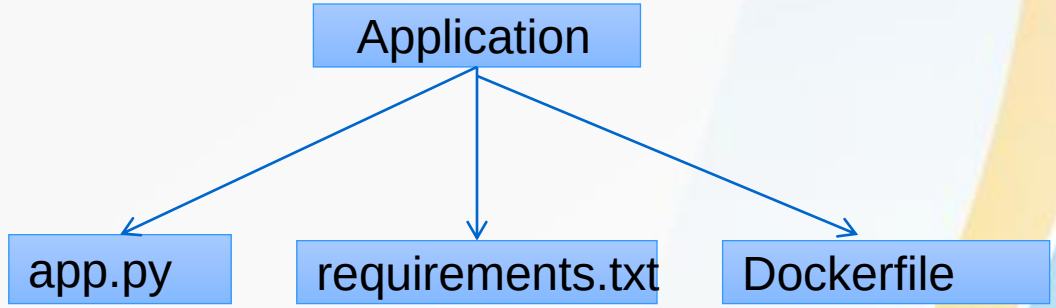
FROM UBUNTU
CMD [“echo”,“Welcome”]

1) docker -t ubuntu
Welcome
2) docker run -t ubuntu echo “Hello World”
Hello World

Dockerfile

- Sample docker file for creating docker images.

```
FROM python:latest
RUN pip install -r requirements.txt
RUN mkdir /var/www
COPY . /var/www
WORKDIR /var/www
ENTRYPOINT ["python"]
CMD ["app.py"]
```



- Create docker image.

docker build -t <image_name>:<version> <path_to_dockerfile>

docker build -t pythonapp:1.0.0 .

Container and Logs

-it -> Interactive
-d -> Background -> Returns container id.

- To run, start and stop containers.

Run Container
docker run <-it/-d> --name <container_name> <image_name>
docker run -it --name pyapplication -p 5000:5000 pythonapp:1.0.0

docker run -d --name pyapplication -p 5000:5000 pythonapp:1.0.0
docker attach pyapplication

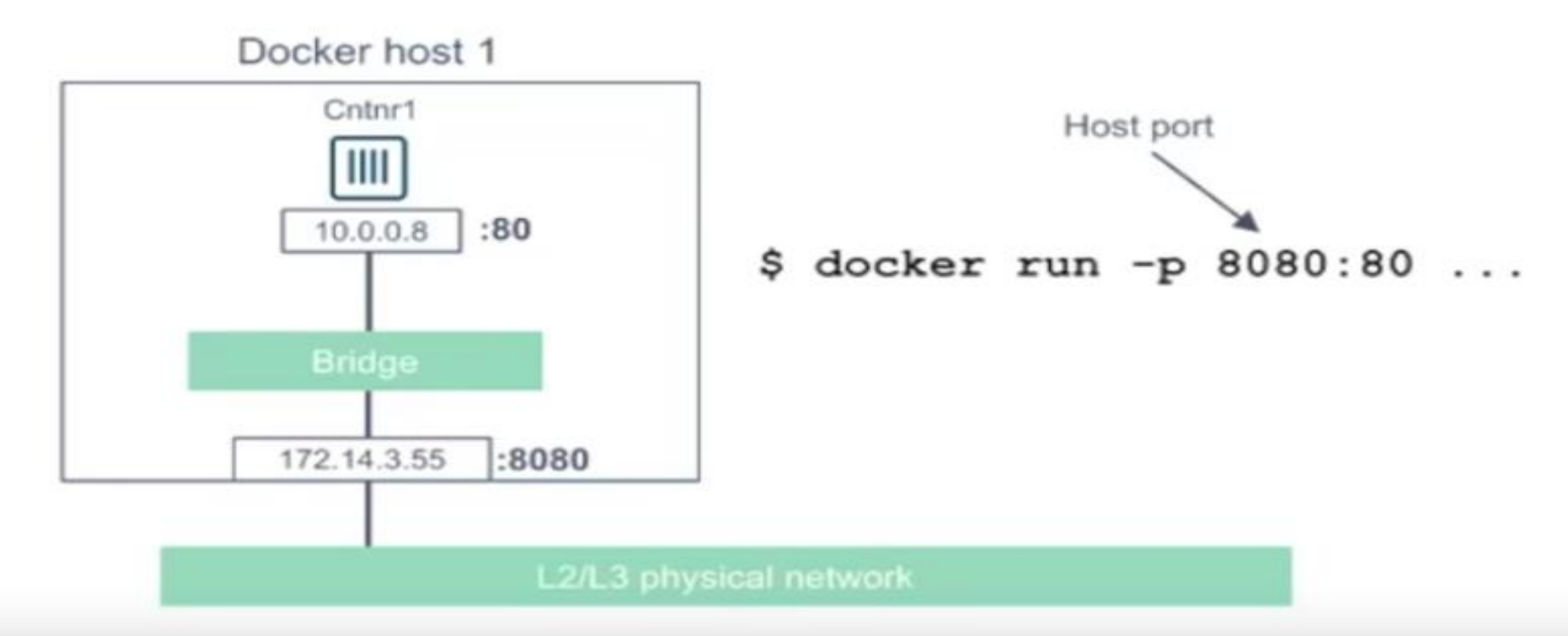
Stop container
sudo docker stop <container_name>
sudo docker stop pyapplication

Start container
sudo docker start <container_name>
sudo docker start pyapplication

Docker Logs
docker logs <container_name/container_id>

Port Mapping

docker run -it --name nginxapp -p 8080:80 nginx



Delete Container/Image

- Delete an image

  Remove image
  sudo docker rmi <image_name:version>
  sudo docker rmi pythonapp:1.0.0

- Delete a container

  Remove container
  sudo docker rm <container_name>
  sudo docker rm pyapplication

Basic Docker commands

To list Docker Images
docker images

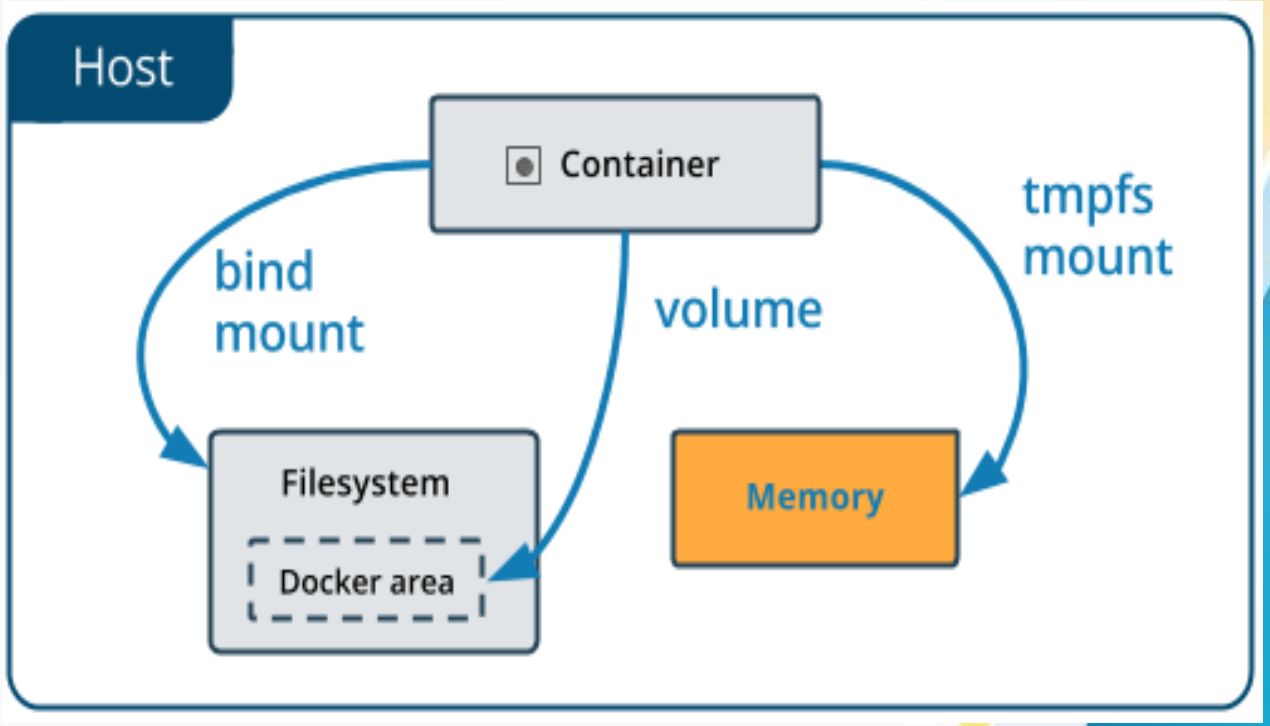
To list running containers
docker ps

To list all containers
docker ps -a

To create new image from an existing container
docker commit <container_id/container_name> <new_image_name>

Docker Volume

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- Easy to backup.
- Shared among multiple containers.
- Managed using docker commands.



Docker volume commands

Create Volume
docker volume create <volume_name>
docker volume create mystorage

Inspect volume
docker volume inspect <volume_name>
docker volume inspect mystorage

Remove specified volume
docker volume rm <volume_name>
docker volume rm mystorage

Remove unused volume
docker volume prune

Attach volume

Volume:

docker run --name <container_name> -v <volume_name>:/data/db -p 27017:27017 <image_name>

docker run --name mongo1 -v mystorage:/data/db -p 27017:27017 mongo

Bind Mount:

docker run --name <container_name> -v <system_path>:/data/db -p 27017:27017 <image_name>

docker run --name mongo1 -v /mywork/dockerstorage/mongoinfo:/data/db -p 27018:27017 mongo

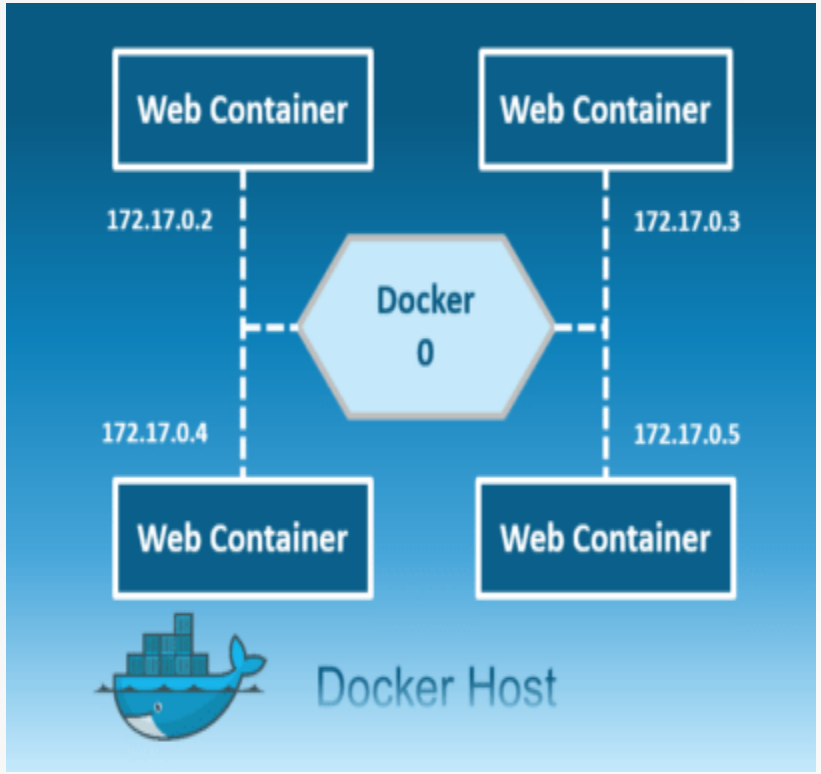
Tmpfs:

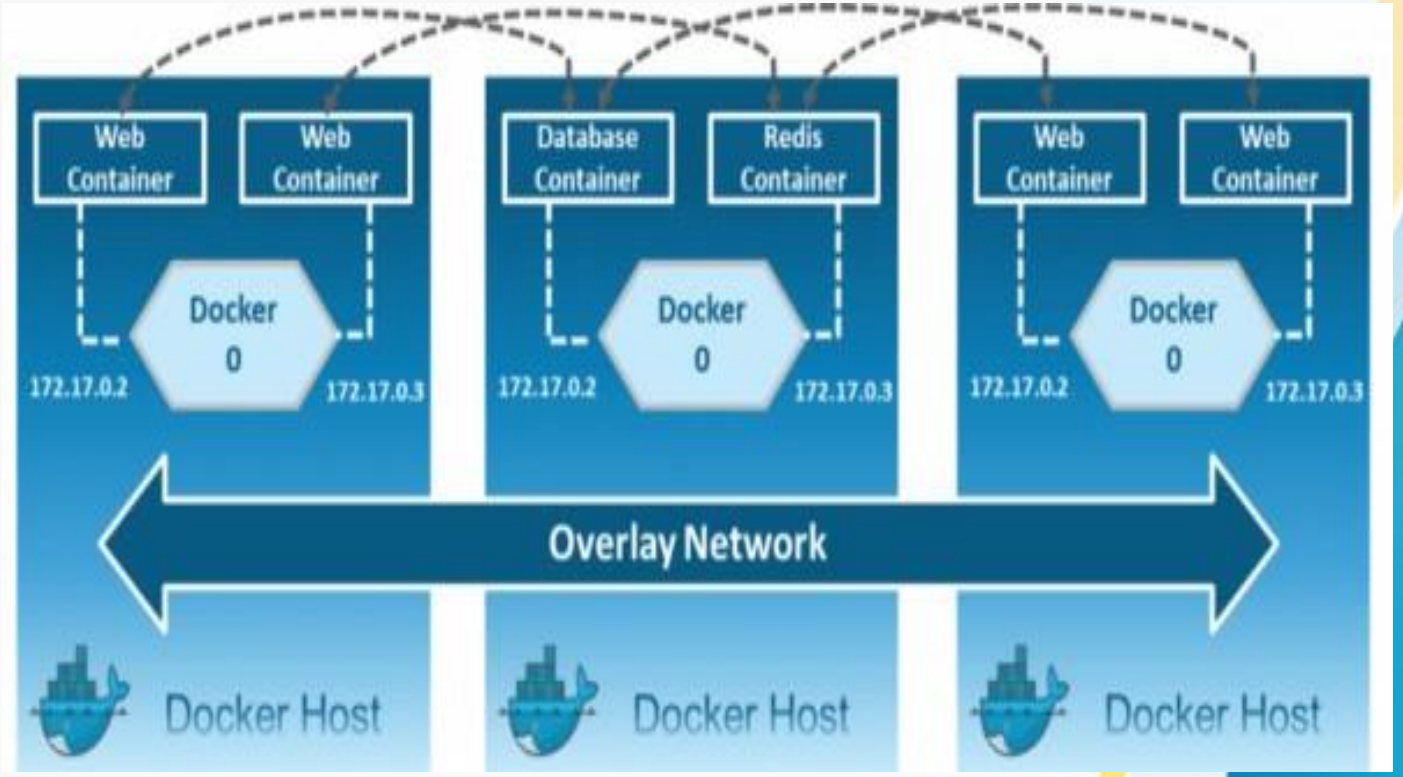
docker run --name mongo1 --tmpfs /data/db -p 27018:27017 mongo

docker run -d --name devtest --mount source=myvol2,target=/app nginx

Docker Network

- Connect docker containers together, or connect docker containers to non-Docker workloads.





Docker Network

- **bridge**: Communication among containers in the same docker daemon.
- **overlay**: Communication among containers on the different docker daemon.
- **host**: remove network isolation between the container and the Docker host, and use the host’s networking directly.
- **macvlan**: allow you to assign a MAC address to a container, making it appear as a physical device on your network.
- **none**: disable all networking.
- **network plugins**: install and use third-party network plugins with Docker.

Network commands

List Network drivers
docker network ls

Inspect driver
docker network inspect bridge

Create Bridge Network

docker network create --driver=bridge <bridge_name>
docker network create --driver=bridge my-bridge

docker run -d --network <bridge_name> <image_name>
docker run -d --network my-bridge myapp

Connect running container to the network

docker network connect <network_name> <container name>
docker network connect my-bridge pyapplication

Docker compose

- tool for defining and running multi-container Docker applications.
- Command : docker-compose up

```
version: '3'

services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    depends_on:
      - db
```