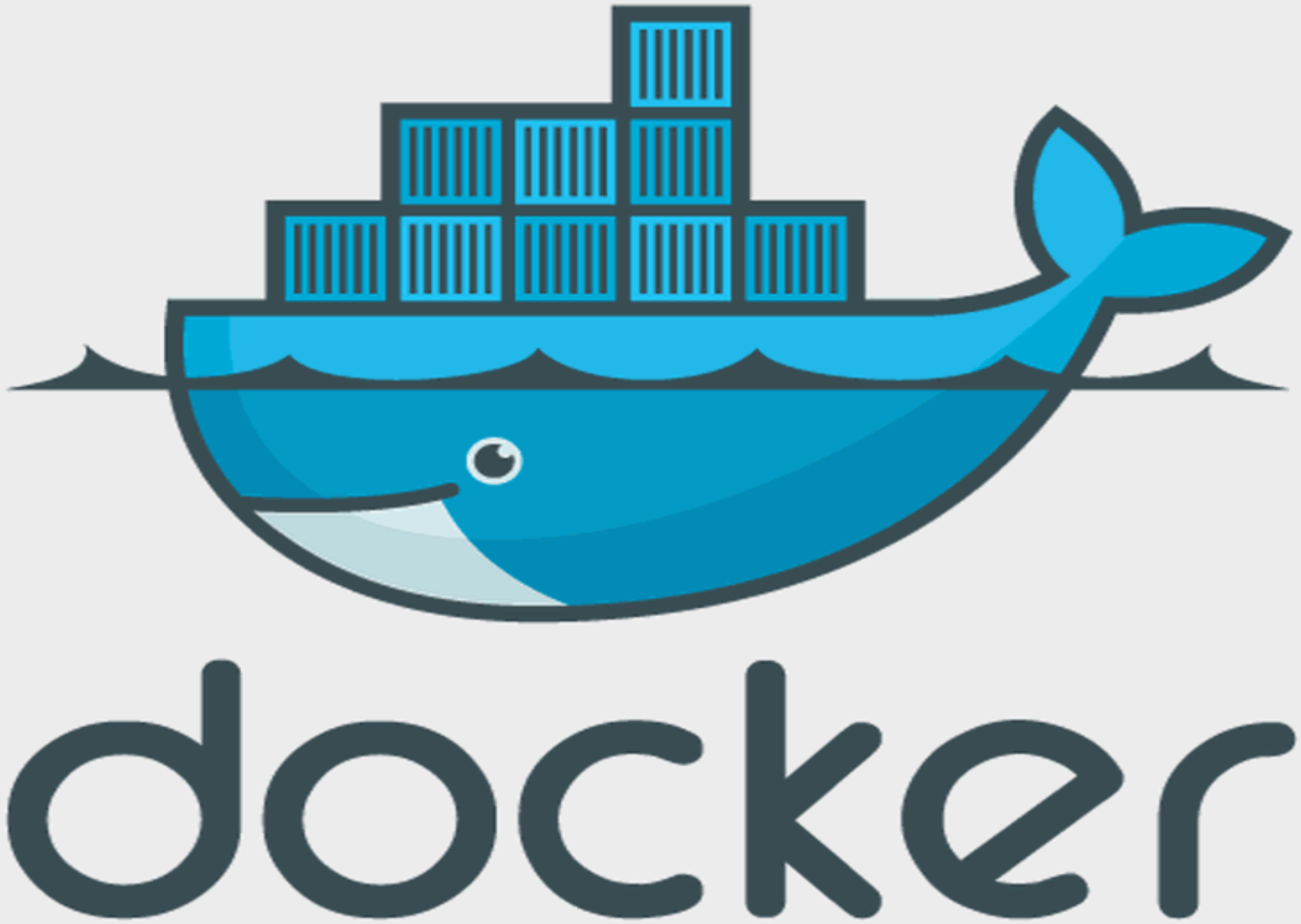

docker

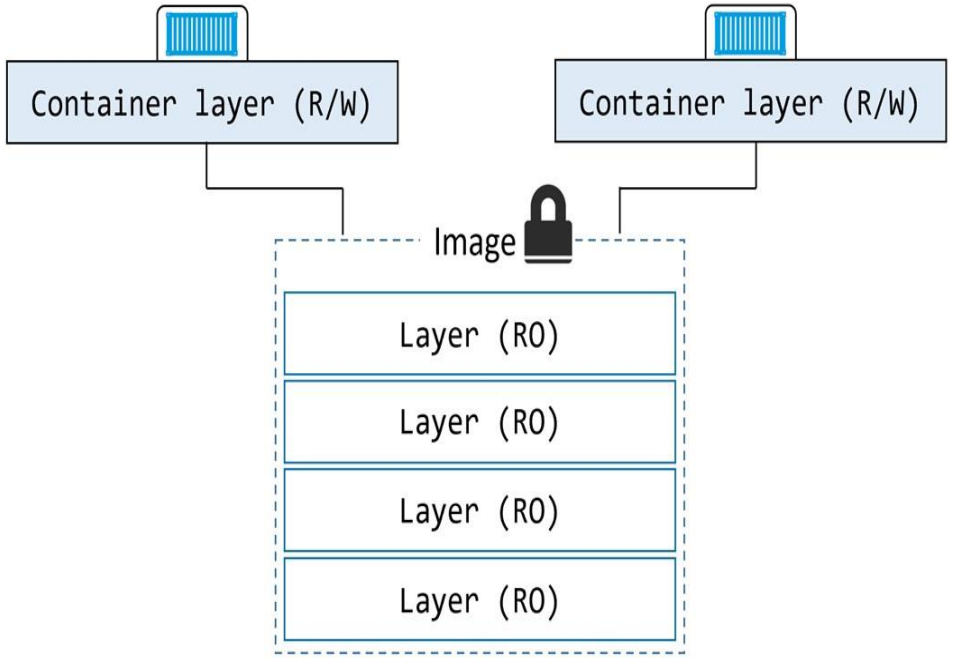
Agenda

- Docker Volume
- Docker Networking
- Docker Compose
  - Docker Compose Basics
  - Docker Compose Commands

Docker Volume

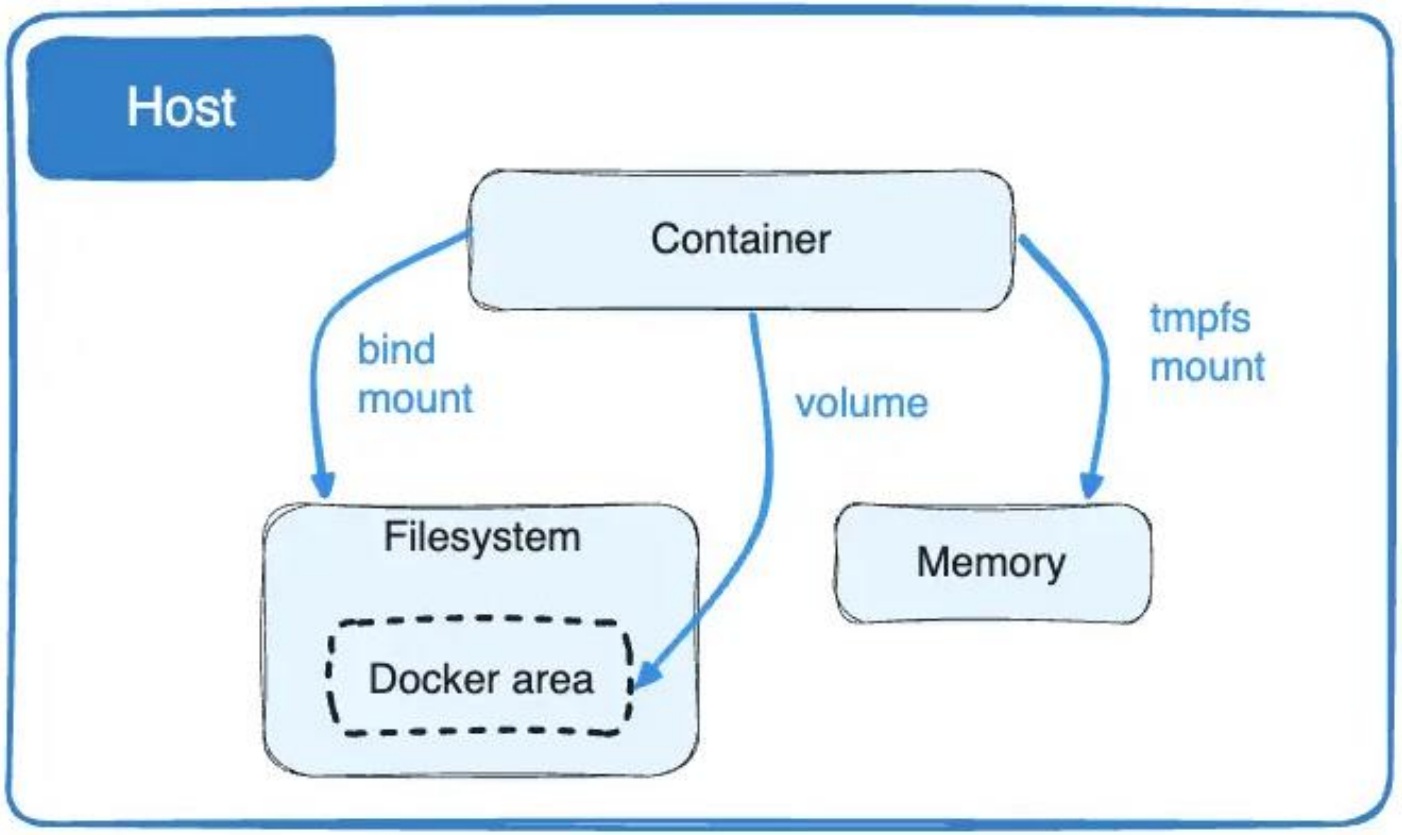
Introduction

- Container storage is said to be ephemeral, meaning its contents are not preserved after the container is removed. Containerized applications work on the assumption that they always start with empty storage.



Docker Volume cont'd

- Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.



Docker Volume cont’d

Type of mount

Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

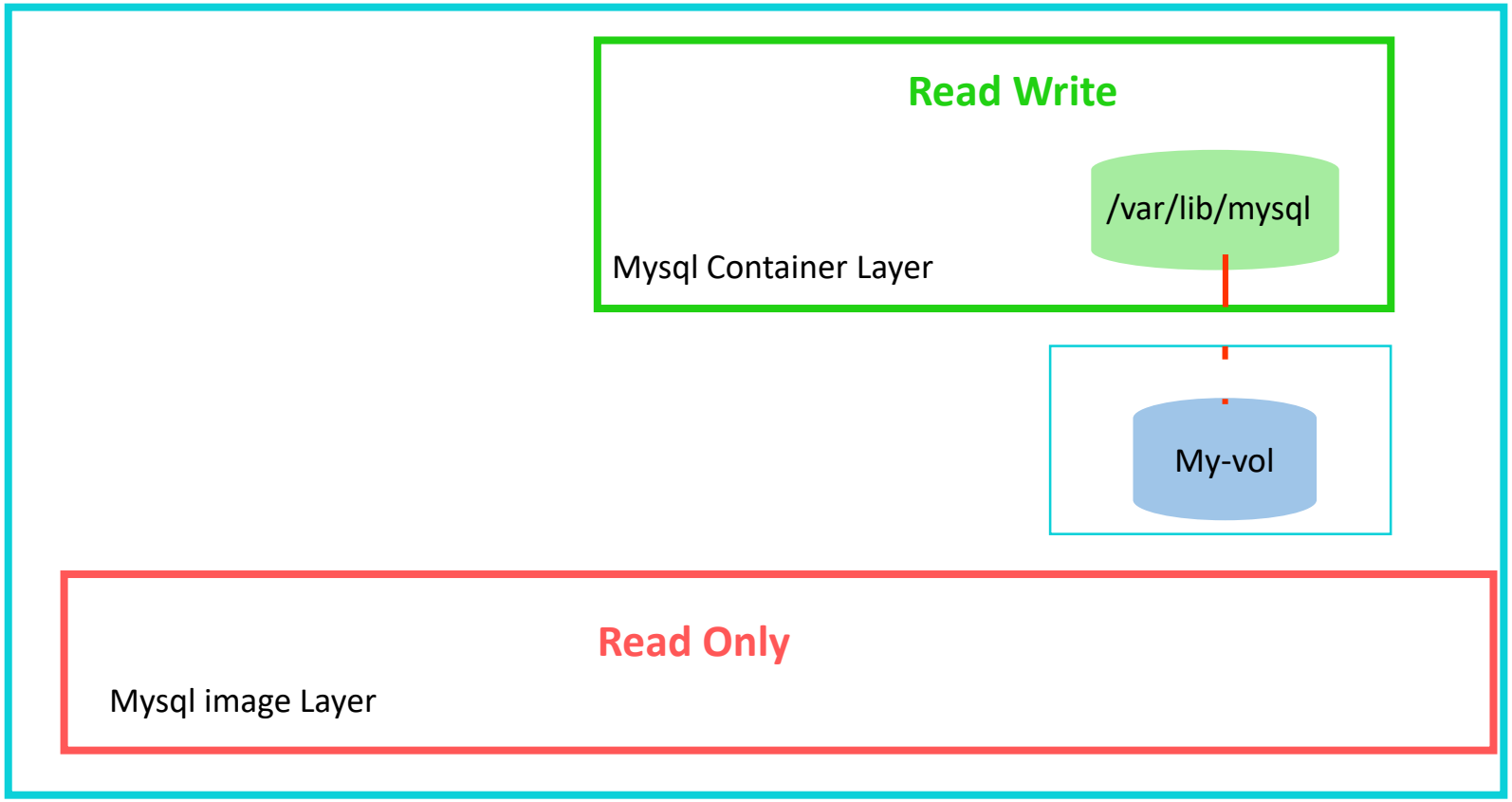
Tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Docker Volume cont’d

Volumes

Commands:

docker volume create my-vol
docker run -v
my-vol:/var/lib/mysql mysql



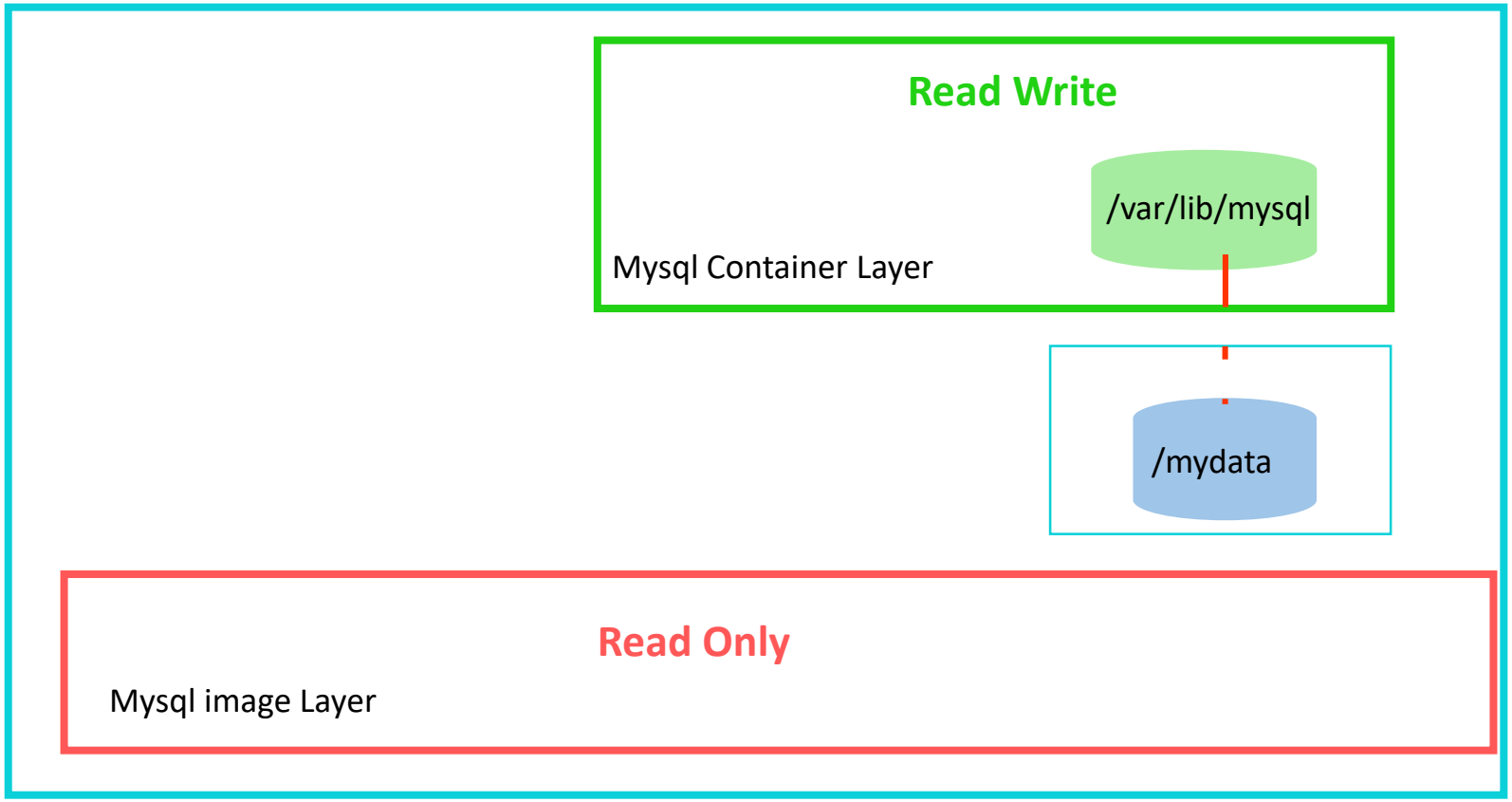
Docker Volume cont’d

Bind mounts

Commands:

mkdir /mydata

docker run -v /mydata:/var/lib/mysql mysql



Docker Volume cont’d

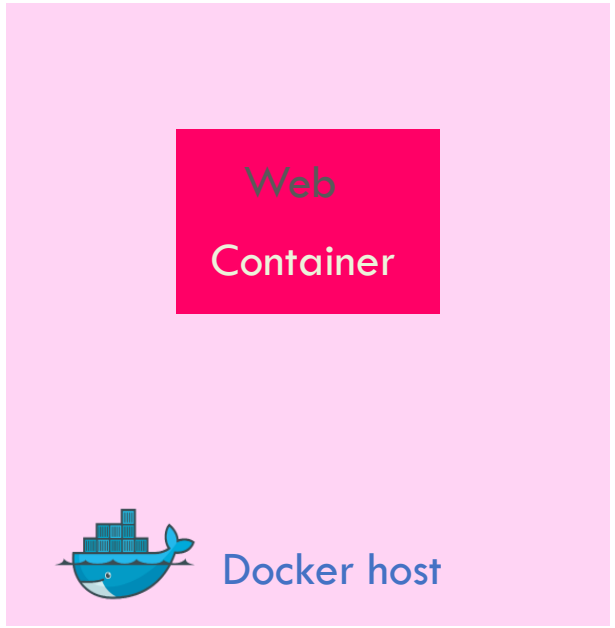
Commands:

docker volume create myvol
docker volume ls
docker volume inspect myvol
docker run –d --name devtest --mount source=myvol,target=/app nginx:latest

Docker Networking

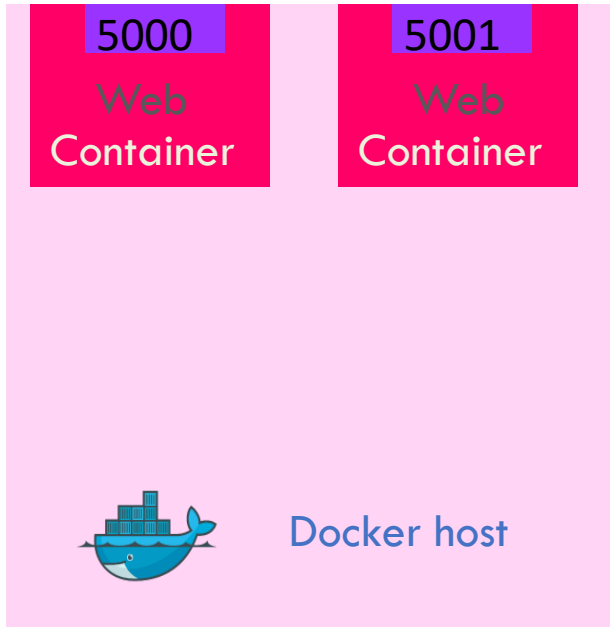
There are main five docker networks:

- **none**: disable all networking for the container.



Docker Networking Cont'd

- **Host**: For standalone containers, it removes the network isolation between the container and the Docker host but other aspects of the container are isolated. It doesn't require port mapping because the host network driver automatically uses the "eth0" when running on linux/unix systems.



Docker Networking Cont’d

Commands:

Run container with host network:

docker container run −d −−network host −−name container_name container_image

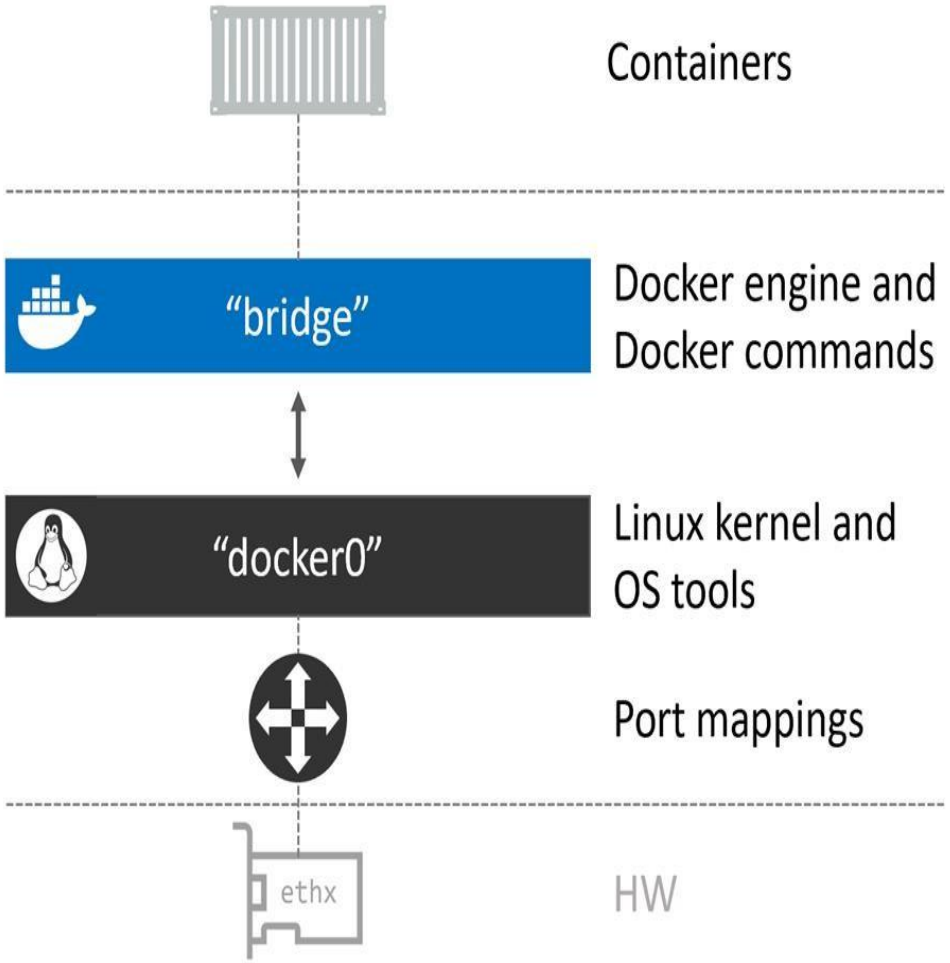
Inspect the host network:

sudo docker network inspect host (range of ip and ports)

Docker Networking Cont'd

Bridge

The default network driver. It requires port mapping to communicate. It is used when you need multiple containers to communicate on the same Docker host.(Run by default)

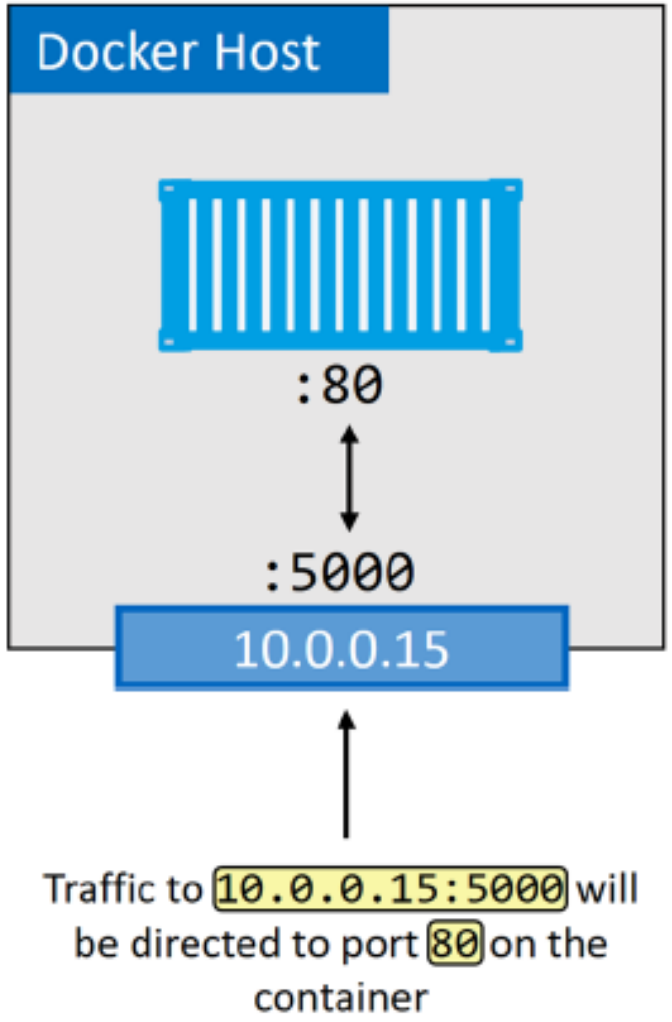


Docker Networking Cont'd

Bridge cont'd

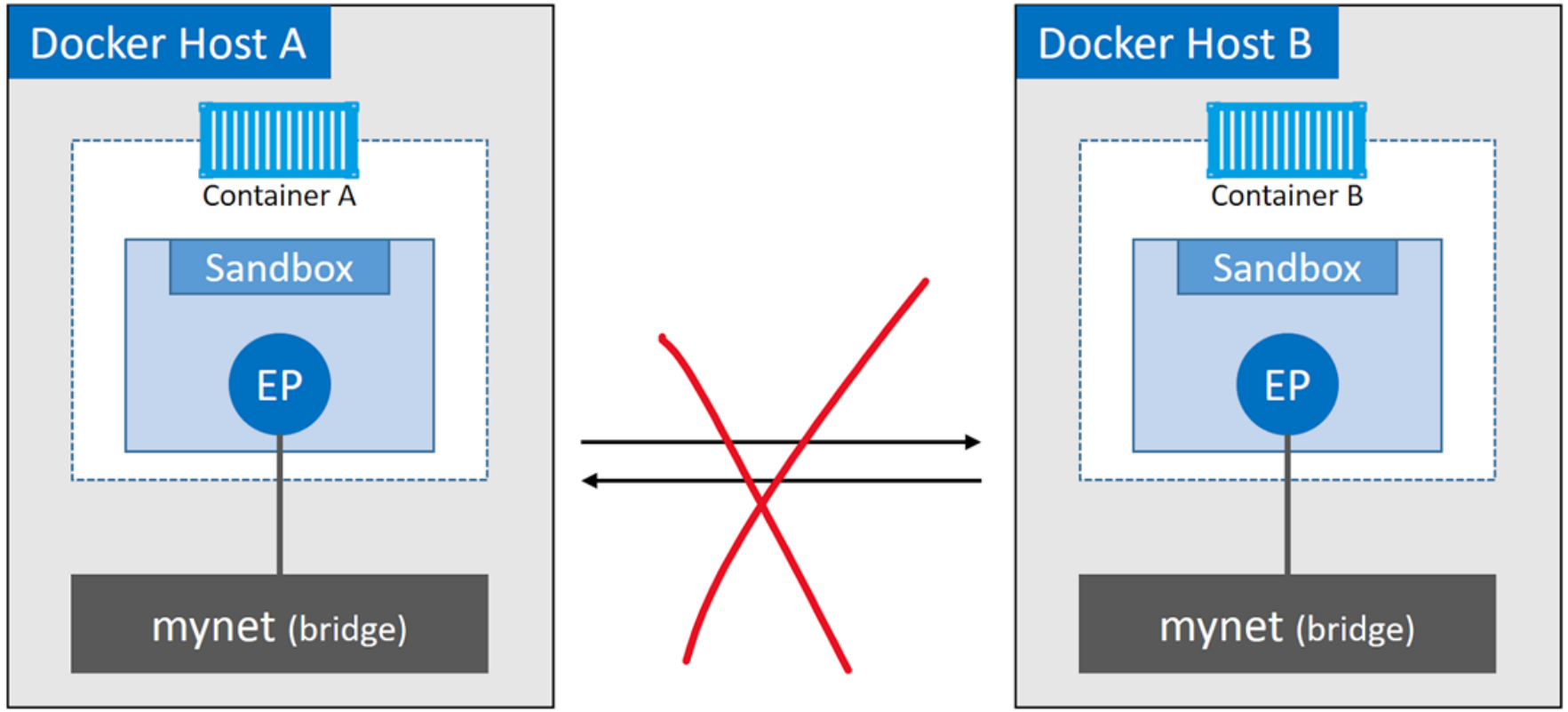
- port mapping

docker run -d --name apache2
-p 10.0.0.15:5000:80 nginx



Docker Networking Cont'd

Bridge cont'd



Docker Networking Cont’d

Bridge cont’d:

Commands:

Run container with bridge network:

docker network create my-network

docker run -d --network=my-network   --name apache -p 8080:80   nginx

docker container run –d ––network bridge ––name nginx02 –p 80:80 nginx:alpine

Inspect the bridge network:

sudo docker network inspect bridge

Docker Networking Cont’d

Overlay: It is used when you need multiple containers to communicate on different Docker hosts.

Macvlan: It is the best when containers are needed to look like physical hosts on your network, each with a unique MAC address.

To get the container IP:

sudo docker inspect <container−id>

To check if the host network is using port mapping or not:

sudo docker container ls.

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.

How To Install Docker Compose

Docker Compose Basics

- **Environment variables**
- # environment vars
  environment:
    RACK_ENV: development
  environment:
    - RACK_ENV=development

- # environment vars from file
  env_file: .env
  env_file: [.env, .development.env]

Docker Compose Basics cont’d

- **Ports**

- ports:
  - "3000"
  - "8000:80" # Guest:Container

- # expose ports to linked services (not to host)
  expose: ["3000"]

Docker Compose Basics cont'd

- **Commands**
- # command to execute
  command: bundle exec thin -p 3000
  command: [bundle, exec, thin, -p, 3000]

- # override the entrypoint
  entrypoint: /app/start.sh
  entrypoint: [php, -d, vendor/bin/phpunit]

Docker Compose Basics cont’d

- **Dependencies**
- # makes the `db` service available as the hostname `database`
  # (implies depends_on)
  links:
    - db:database
    - redis

- # make sure `db` is alive before starting
  depends_on:
    - db

Docker Compose Basics cont'd

- **Building**
- web:
  # build from Dockerfile
  build: .

- # build from custom Dockerfile
  build:
    context: ./dir
    dockerfile: Dockerfile.dev

- # build from image
  image: ubuntu
  image: ubuntu:14.04
  image: example-registry:4000/postgresql

Docker Compose Basics cont’d

- **Volumes**

- volumes:
  - /var/lib/mysql
  - ./_data:/var/lib/mysql

Docker Compose Sample


```yaml
version: "3.0"
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    volumes:
      - wordpress_data:/var/www/html
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
volumes:
  db_data: {}
  wordpress_data: {}
```

Docker Compose Commands

docker-compose start

docker-compose stop

docker-compose pause

docker-compose unpause

docker-compose ps

docker-compose up

docker-compose down

docker-compose run

docker-compose scan