

DOCKER CĂN BẢN

PRESENTATION BY NGUYEN NHU THUONG

NỘI DUNG CHÍNH

GIỚI THIỆU VÀ CÀI ĐẶT DOCKER

Tìm hiểu về tổng quan, kiến trúc của Docker; lý do tại sao Docker lại quan trọng trong các hệ thống ngày nay?

IMAGE & CONTAINER

Hiểu rõ về 2 thành phần Image & Container trong Docker. Khác nhau và vai trò giữa chúng.

BIND MOUNT & VOLUME

Nắm bắt về cách triển khai stateful service trong Docker, các cách lưu giữ dữ liệu của service trong Docker

DOCKER NETWORK

Tìm hiểu thành phần network trong Docker, làm sao để kết nối giữa hai container với nhau?



GIỚI THIỆU

Tại sao phải sử dụng Docker?

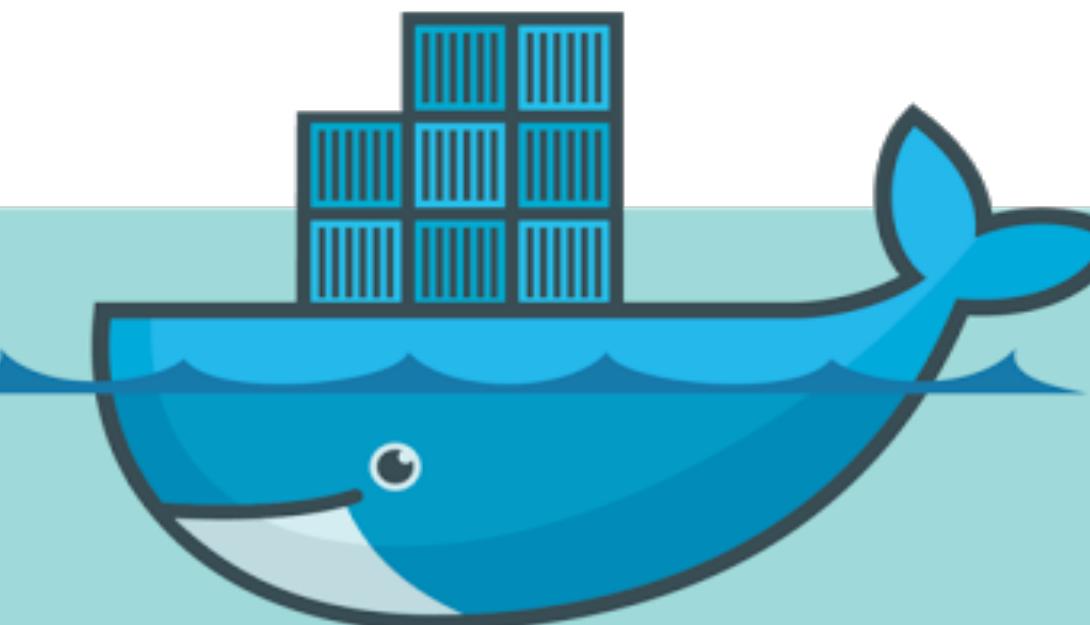
65%

use Docker to deliver development agility.



48%

use Docker to control app environments.



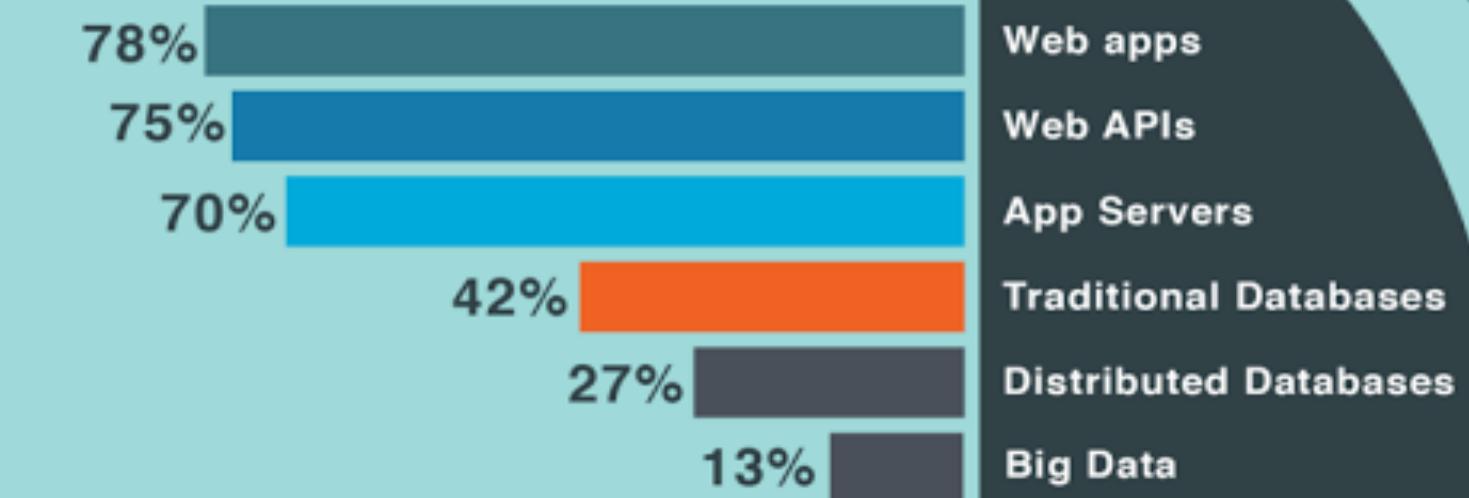
90%

use Docker for apps in development.

41%

use Docker to achieve app portability.

Docker Workloads



58%

use Docker for apps in production.



90%

plan dev environments around Docker.



80%

plan DevOps around Docker.

Tại sao phải sử dụng Docker?



PACKAGING

Docker Container cung cấp một cách đáng tin cậy để gộp các thành phần ứng dụng và đóng gói chúng lại với nhau thành một bản artifact. Điều này rất quan trọng vì các ứng dụng hiện đại thường bao gồm nhiều phần khác nhau, không chỉ code, mà còn các phần phụ thuộc, binary files hay thư viện hệ thống.



PORTABILITY

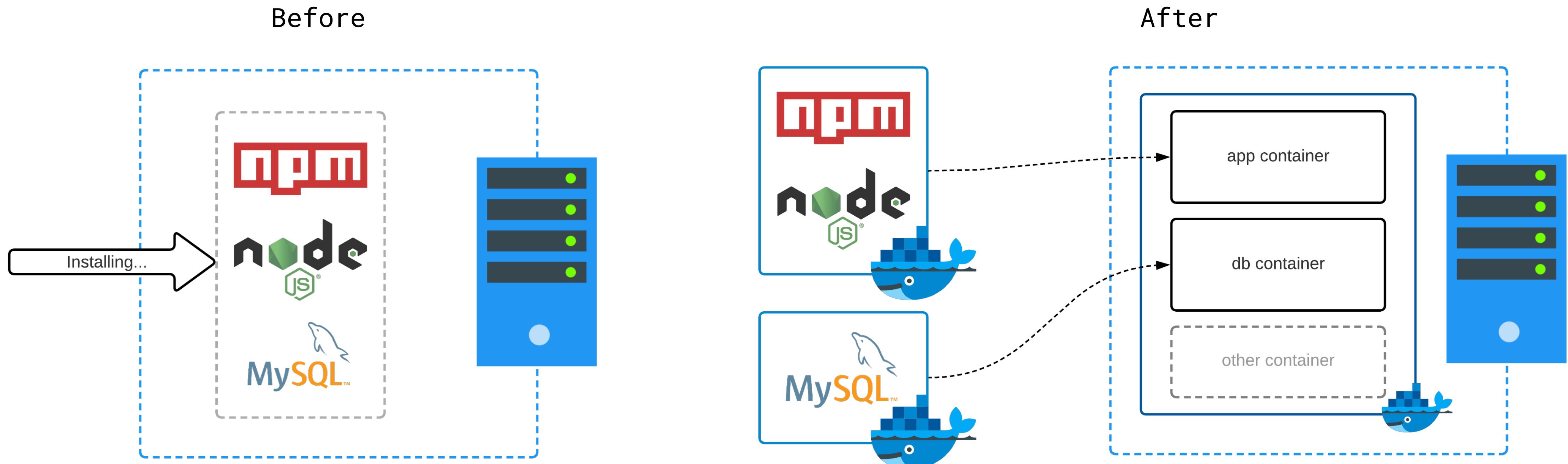
Bởi vì Docker Container cho phép ứng dụng chứa tất cả các phụ thuộc của nó cùng với nó, điều này cho phép container được đưa đến bất cứ nơi để ứng dụng chạy một cách đáng tin cậy. Có thể triển khai ở bất cứ đâu như: máy cá nhân, máy ảo, hệ thống cloud,...



EFFICIENCY

Docker container cải thiện hiệu quả bằng cách cung cấp một mô hình cô lập hiệu quả, nhẹ. Không giống như một máy ảo (khá nặng), có thể chạy nhiều docker container trên một máy duy nhất. Không có gì lạ khi một server có thể triển khai với 10-20 Docker container.

Tại sao phải sử dụng Docker?

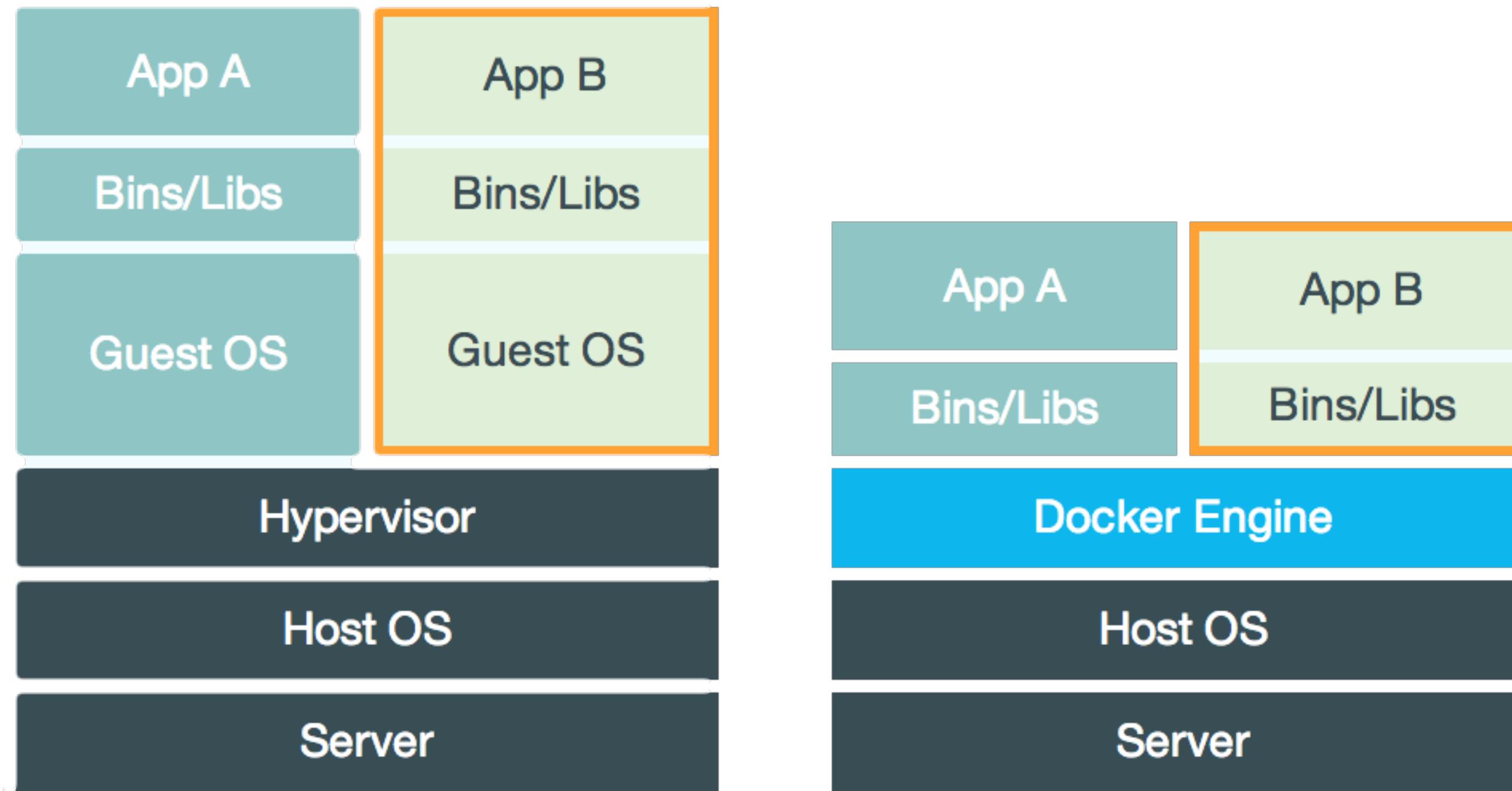




Docker là gì?

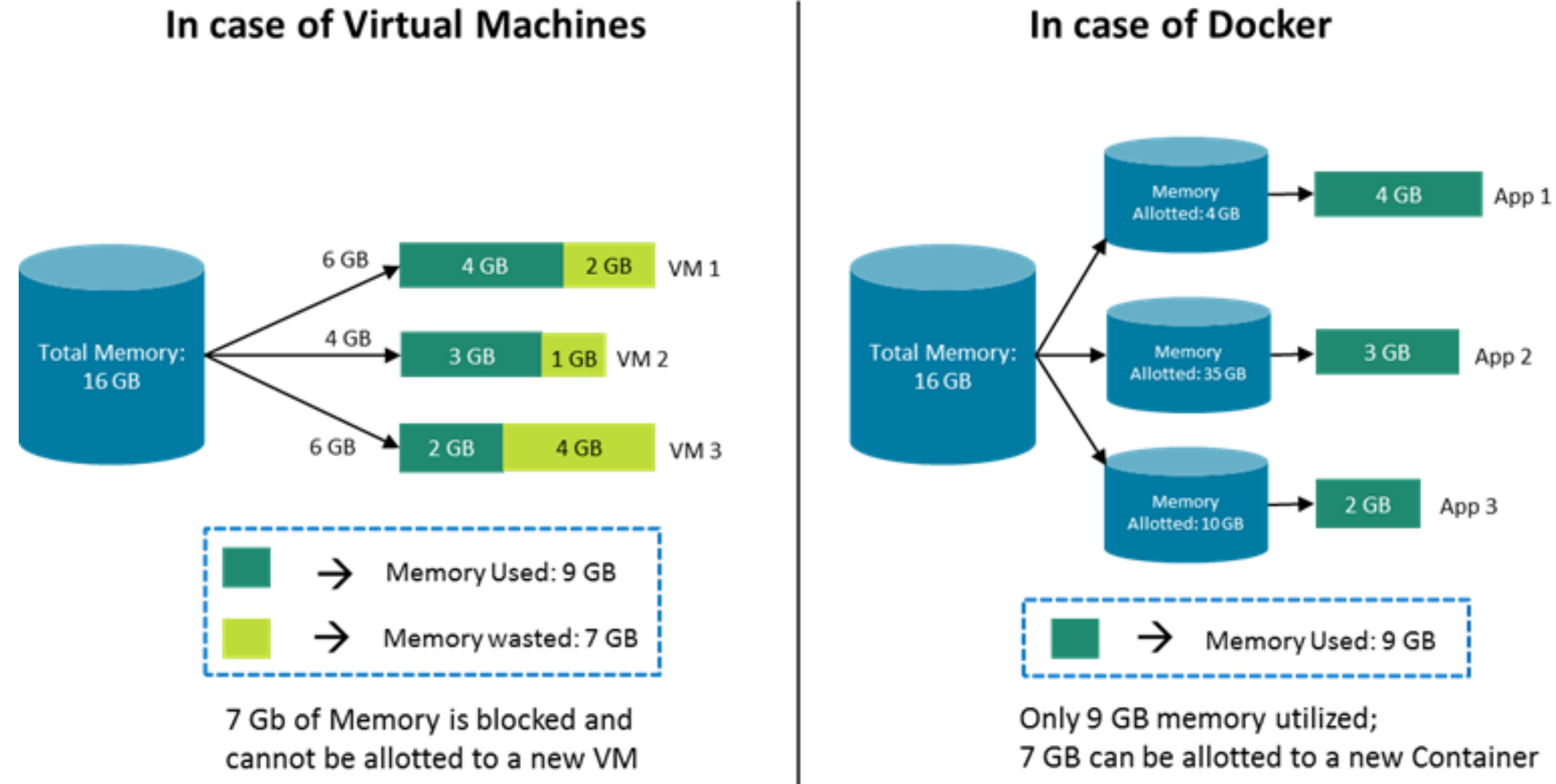
Là một hệ thống mã nguồn mở, hỗ trợ đóng gói và tự động triển khai phần mềm trong các container. Nó cung cấp một cách nhanh, gọn để tạo môi trường hoạt động cho phần mềm, giảm thiểu rủi ro giữa dev và ops khi developer phát triển trên cùng một môi trường như môi trường vận hành, giúp dễ dàng tự động hóa và tăng tốc chu trình phát triển phần mềm (dev, test, deploy...)

Khác nhau giữa Virtual Machine và Docker

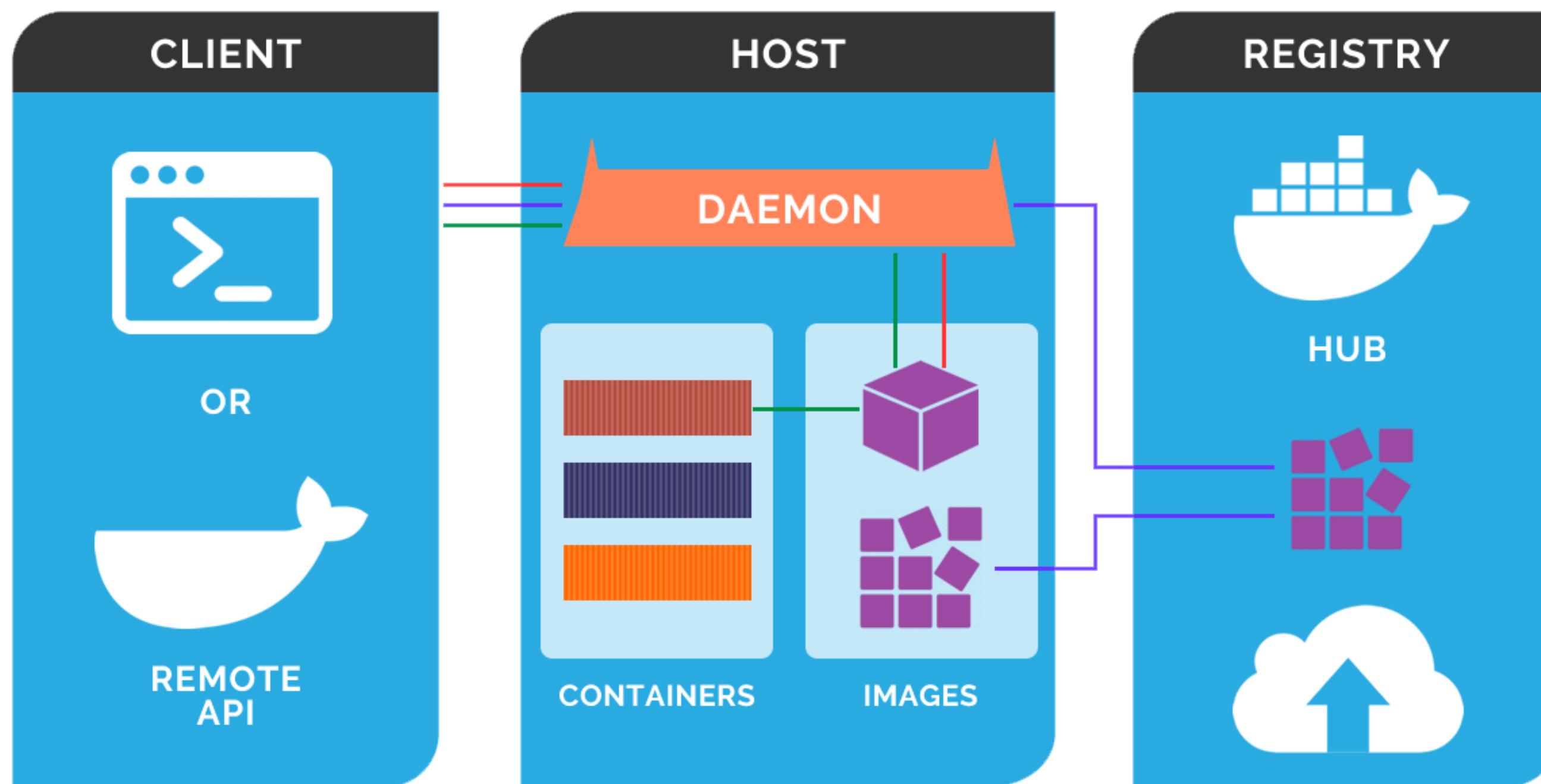


- **Docker:** Dùng chung kernel, chạy độc lập trên Host Operating System và có thể chạy trên bất kỳ hệ điều hành nào cũng như cloud.
- **Virtual Machine:** Cần thêm một Guest OS cho nên sẽ tốn tài nguyên hơn và làm chậm máy thật khi sử dụng.

Khác nhau giữa Virtual Machine và Docker

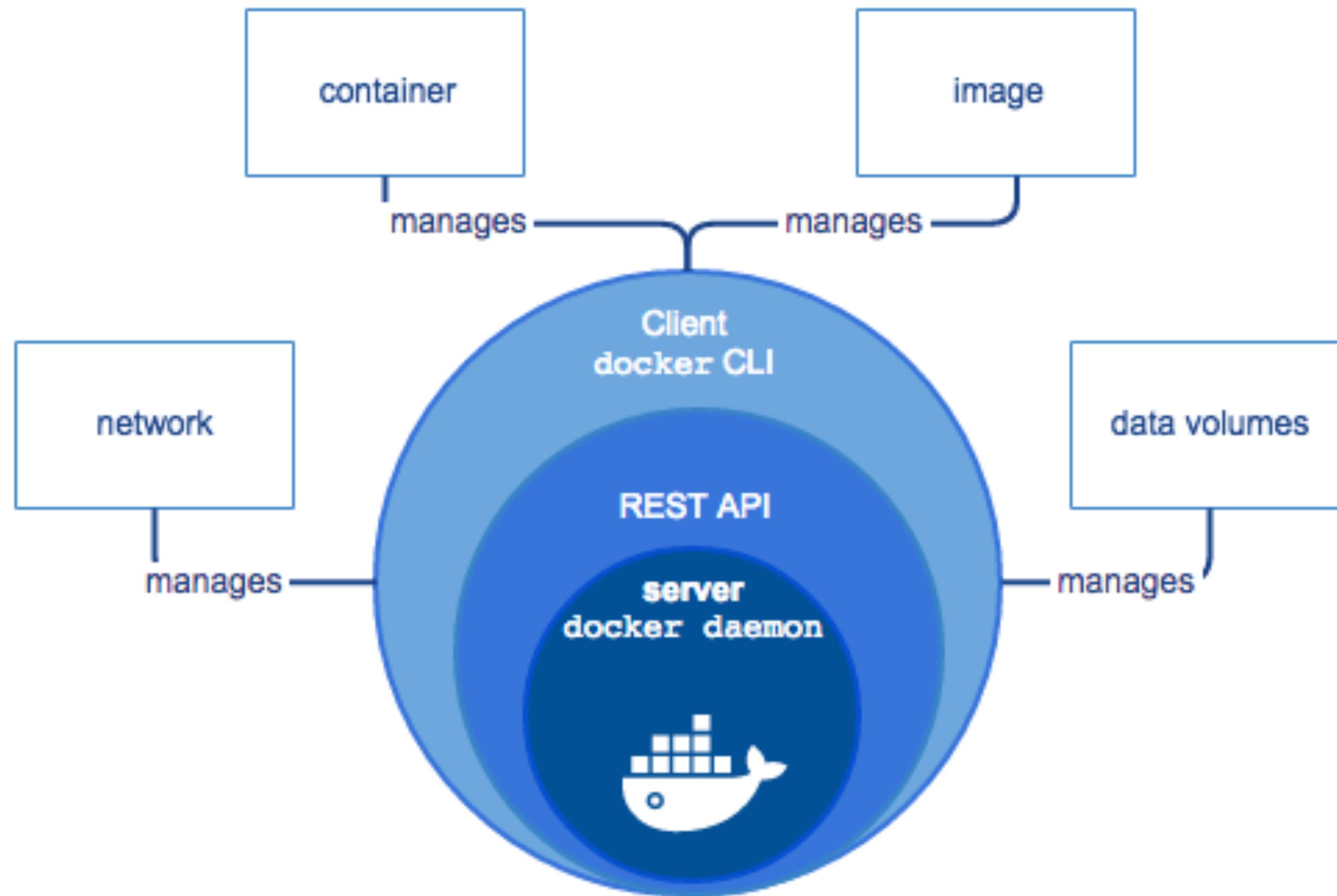


Kiến trúc của Docker



- **Docker Daemon**: đóng vai trò server, nhận các RESTful requests từ Docker Client và thực thi.
- **Docker CLI**: vai trò là client, cung cấp giao diện tương tác với người dùng (CLI) và gửi các RESTful requests tương ứng đến Docker Daemon.
- **Docker Registry**: private hay public registry, là nơi lưu trữ và chia sẻ các Docker Image.

Kiến trúc của Docker



Docker Engine quản lý 4 đối tượng chính.

- Image
- Container
- Network
- Volume

Cài đặt Docker

Desktop

Platform	x86_64 / amd64
Docker Desktop for Mac (macOS)	✓
Docker Desktop for Windows	✓

Server

Docker provides `.deb` and `.rpm` packages from the following Linux distributions and architectures:

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64
CentOS	✓		✓
Debian	✓	✓	✓
Fedora	✓		✓
Raspbian		✓	✓
Ubuntu	✓	✓	✓

Chạy Docker trên trình duyệt Web.

1. Đăng ký tài khoản trên Docker Hub [ở đây](#)
2. Truy cập trang [labs.play-with-docker.com](#)
3. Đăng nhập bằng tài khoản Docker Hub
4. Bấm Start
5. Chọn Add New Instance
6. Kiểm tra bằng lệnh `docker --version`



DOCKER IMAGE & CONTAINER

Docker Image

- Docker Images là một “**read-only template**”; Chẳng hạn, một image chứa hệ điều hành Ubuntu đã cài đặt sẵn Apache và ứng dụng web.
- Nói ngắn gọn là có thể gói các cài đặt môi trường (OS, package, ứng dụng, ...) lại thành 1 cục duy nhất, đó chính là Docker image.
- Để tạo ra một image thì phải viết Dockerfile, build Dockerfile để tạo ra Docker image (có dung lượng từ vài MB đến vài GB)

```
1 FROM node:alpine
2
3 RUN mkdir -p /usr/src/app
4 WORKDIR /usr/src/app
5
6 COPY package.json /usr/src/app/
7 RUN npm install
8
9 COPY ./ /usr/src/app
10 RUN npm run build
11
12 EXPOSE 3000
13
14 CMD [ "npm", "run", "start" ]
```

Docker Image

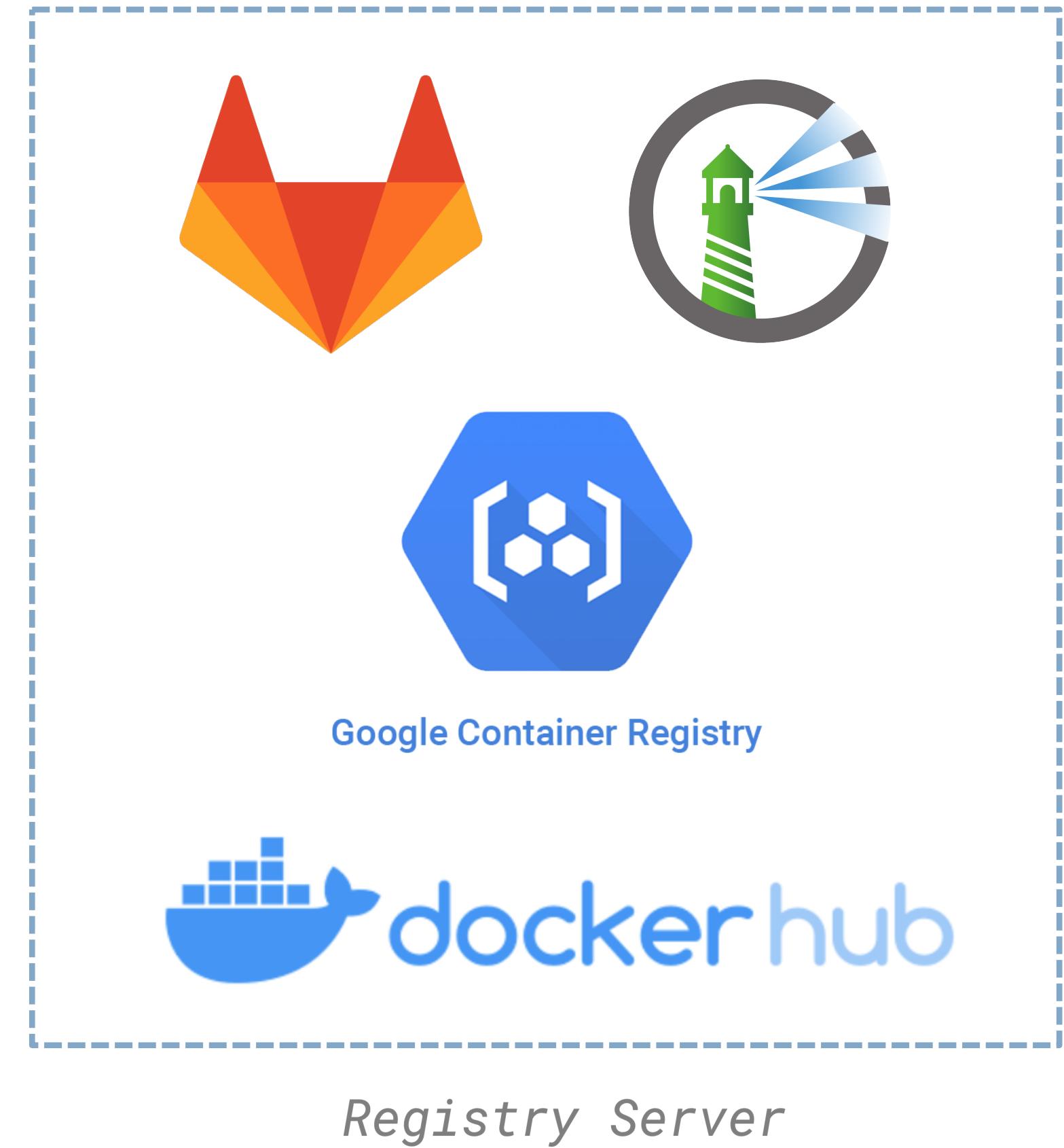
```
1 FROM node:alpine
2
3 RUN mkdir -p /usr/src/app
4 WORKDIR usr/src/app
5
6 COPY package.json /usr/src/app/
7 RUN npm install
8
9 COPY ./usr/src/app
10 RUN npm run build
11
12 EXPOSE 3000
13
14 CMD [ "npm", "run", "start" ]
```

Dockerfile

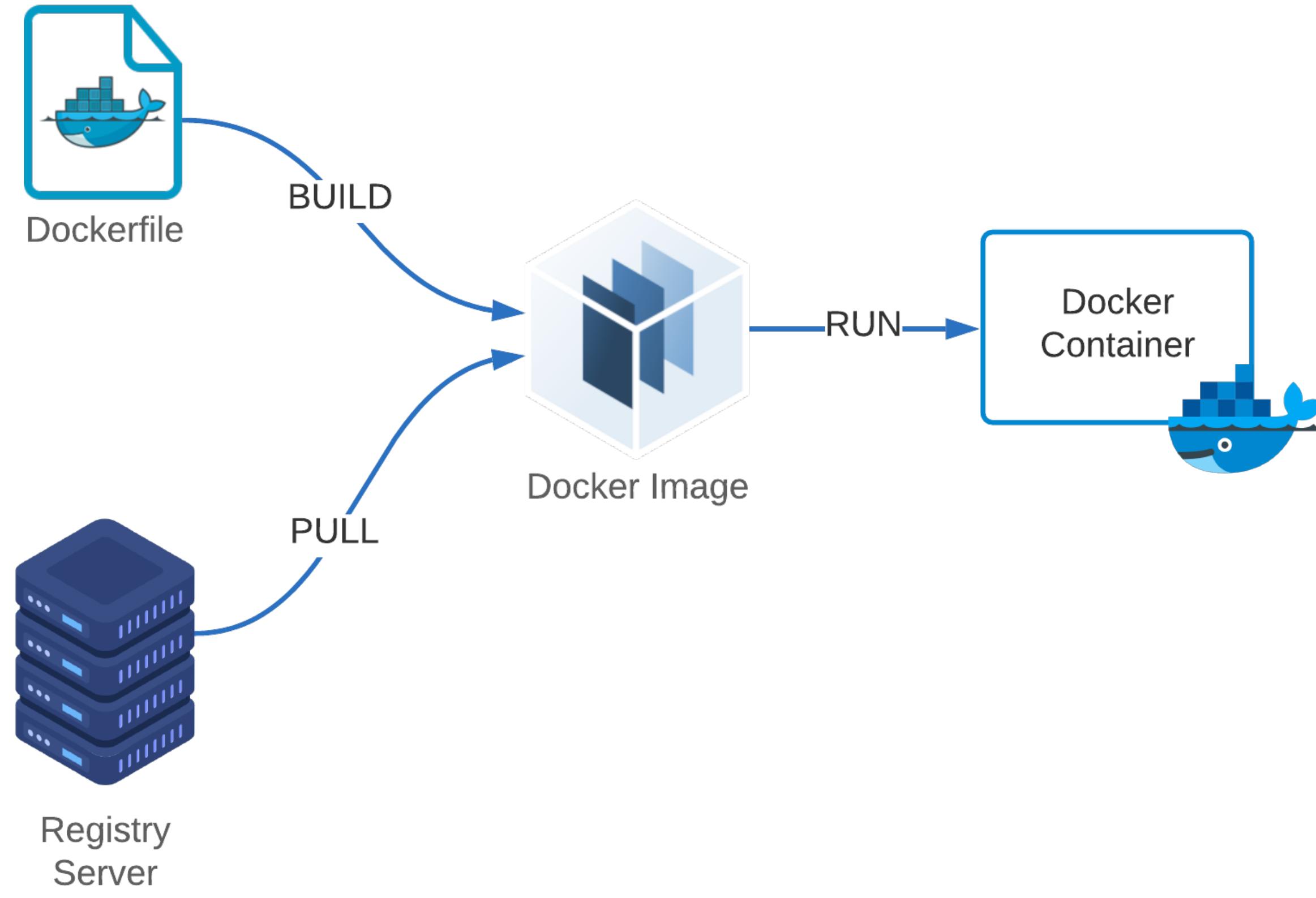
build



push

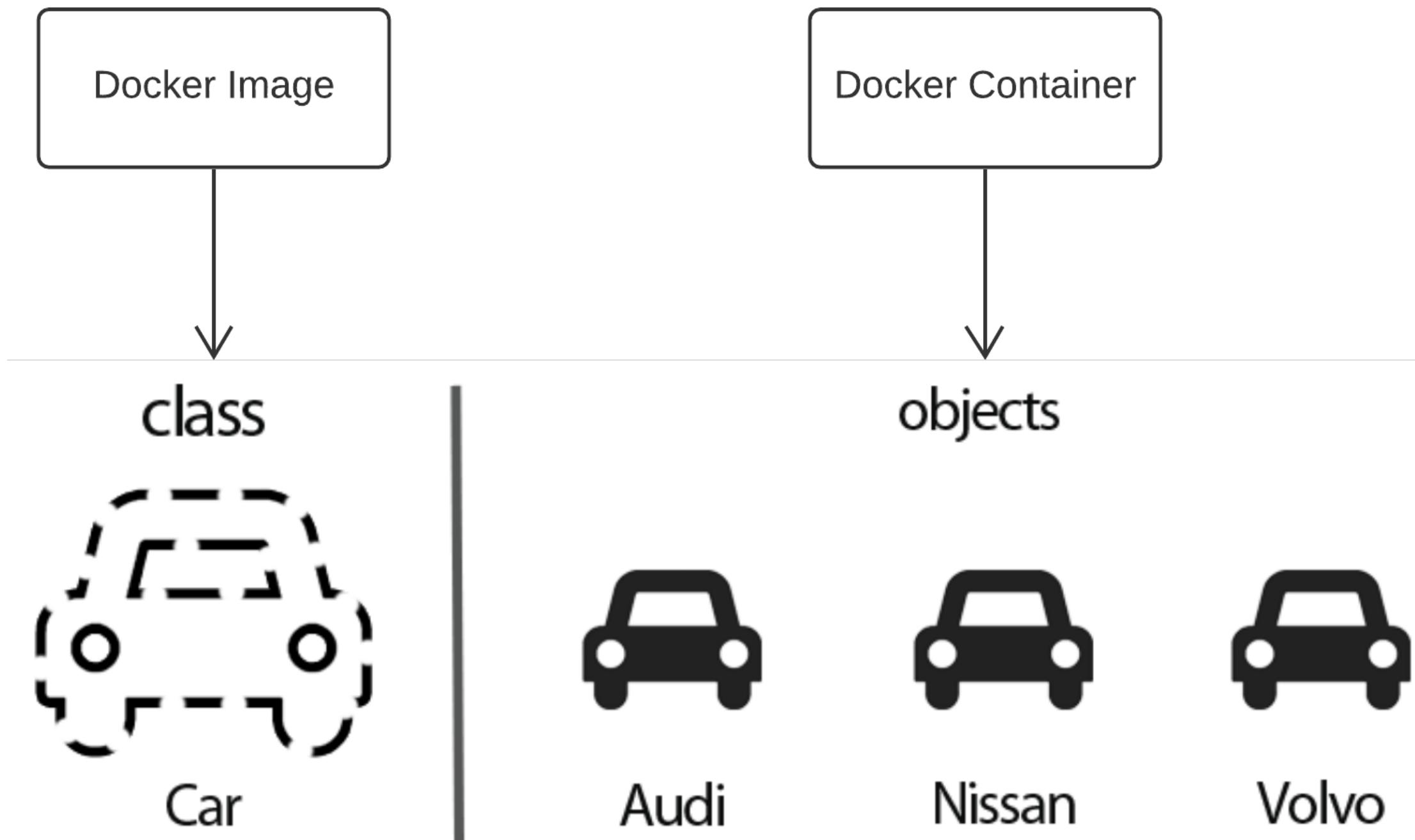


Docker Container



- **Docker Container** là một instance của một image. Có thể create, start, stop, move hoặc delete container dựa trên Docker API hoặc Docker CLI.
- Image có thể tồn tại mà không cần container, trong khi container chạy thì cần có image đã tồn tại.
- Cả image và container đều là thành phần quan trọng để tạo ra 1 running container. Docker image là cực kì quan trọng để chỉ phôi và định hình 1 Docker container.

Docker Container



- Nếu liên tưởng đến OOP, có thể xem Docker image là class, còn Docker container là object / instance của class đó.
- Vì vậy từ một docker image trên máy, có thể tạo một hoặc nhiều container có môi trường bên trong chúng giống hệt nhau.

TASK #1

CÀI ĐẶT NGINX BẰNG DOCKER

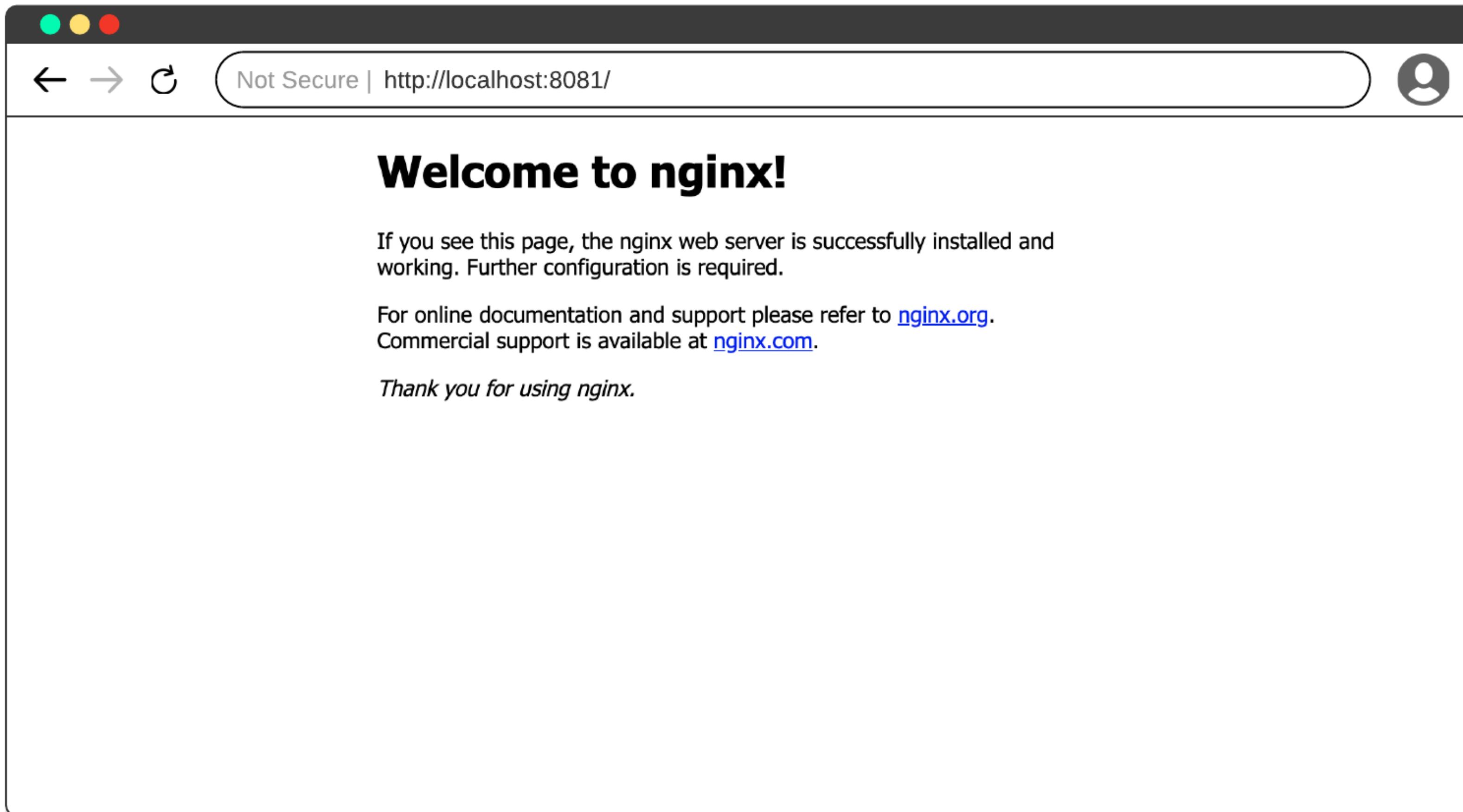
`docker run nginx`

```
● ● ●

$ docker run -d --name nginx-1 -p 8081:80 nginx:latest
eca609b2d5cbe666deb5355110734205a8b1a0f8b415bf0940902defe1ebfe6
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
eca609b2d5cb nginx:latest "/docker-entrypoint..." 7 seconds ago Up 5 seconds 0.0.0.0:8081->80/tcp nginx-1
```

Tham số chạy docker run	Ý nghĩa
-d	Chạy container ở chế độ ngầm
--name nginx-1	Đặt tên cho container là nginx-1
-p 8081:80	Expose cổng 80 của container ra cổng 8081 của máy host
nginx:latest	Container được khởi chạy từ image là nginx:latest

```
docker run nginx
```



Truy cập vào địa chỉ
<http://localhost:8081>
để kiểm tra

TASK #2

TRIỂN KHAI NHIỀU CONTAINER NGINX

yêu cầu

Triển khai thêm 2 container

- Sử dụng image **nginx:latest**
- Chạy ở chế độ ngầm
- Expose ra hai cổng khác nhau trên host

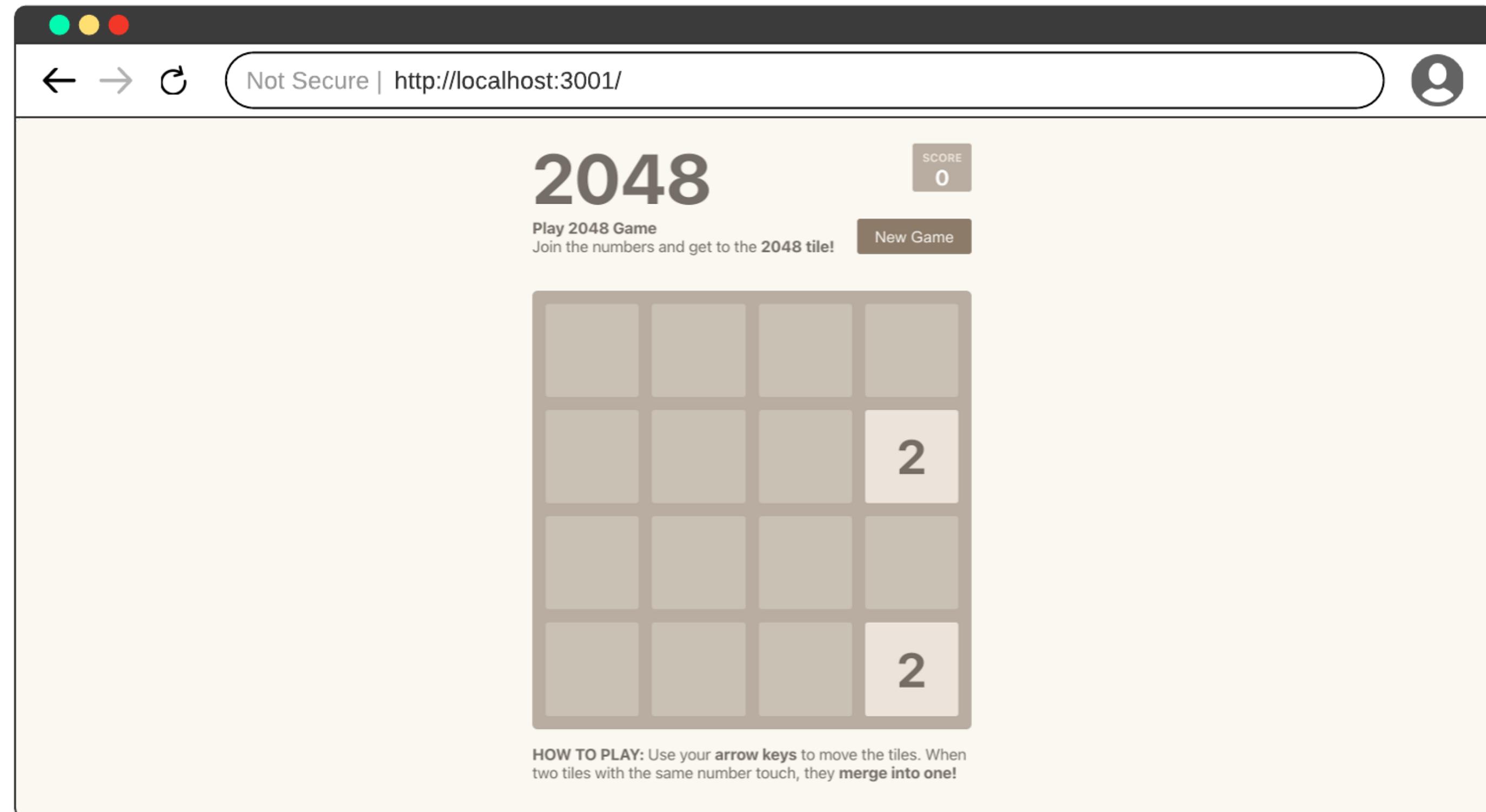


*Sau khi triển khai, truy cập vào cổng tương ứng trên localhost để kiểm tra

TASK #3

TRIỂN KHAI MỘT ỨNG DỤNG REACT BẰNG DOCKER

yêu cầu

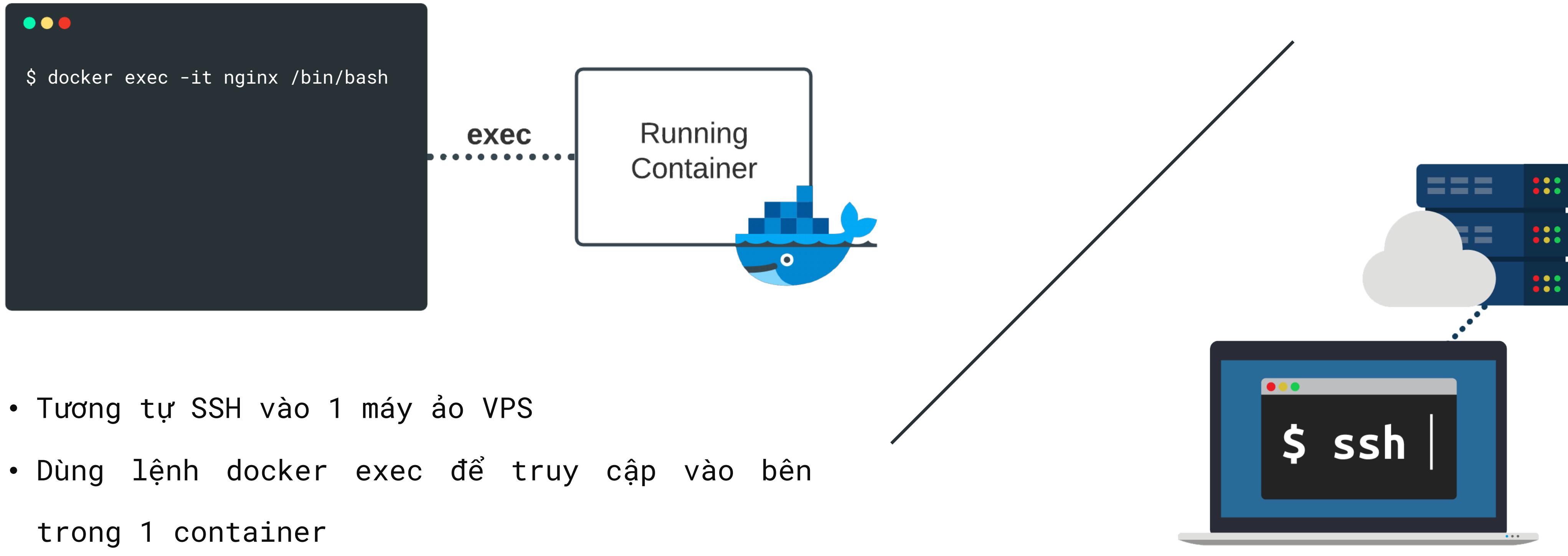


- Sử dụng image là [thuongnn1997/react-app:latest](#)
- Chạy container ở chế độ ngầm với tên của container là [react-app](#).
- Expose cổng 80 của container ra cổng 3001 của máy host.
- Sau khi triển khai, truy cập vào localhost:3001 để kiểm tra.

TASK # 4

TRUY CẬP VÀO BÊN TRONG MỘT CONTAINER

docker exec vào trong 1 container



yêu cầu

- Chạy 1 container nginx ở chế độ ngầm, tên container là `nginx-1` expose ra cổng `8081` (nếu đã có rồi thì thôi). Có thể kiểm tra bằng lệnh `docker ps`
- Truy cập vào trong container `nginx-1` bằng lệnh: `docker exec -it nginx-1 /bin/bash`
- Cài đặt trình soạn thảo `nano` (hoặc `vi`)
- Thực hiện `cd` vào thư mục `/usr/share/nginx/html` bên trong container `nginx-1`
- Mở file `index.html` bằng `nano` (hoặc `vi`) và tiến hành chỉnh sửa
- Reload lại trang `localhost:8081` để kiểm tra

TASK #5

QUẢN LÝ DOCKER BẰNG PORTAINER

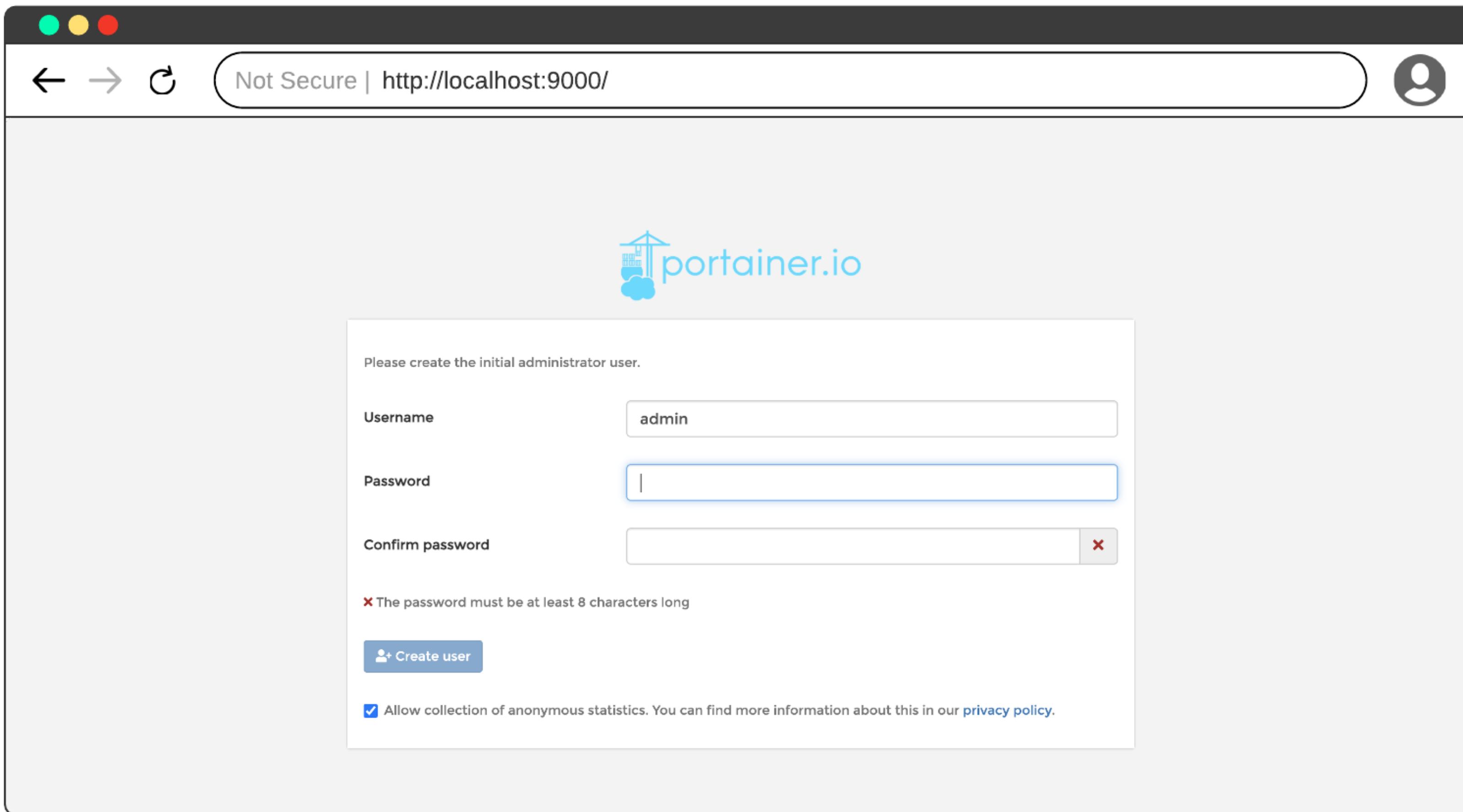
cài đặt portainer

- Portainer là công cụ quản lý Docker Container miễn phí với kích thước gọn nhẹ và giao diện quản lý trực quan, đơn giản để triển khai cũng như sử dụng.
- Hướng dẫn cài đặt

```
$ docker volume create portainer_data
$ docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce
```

[XEM CHI TIẾT PORTAINER](#)

kết quả

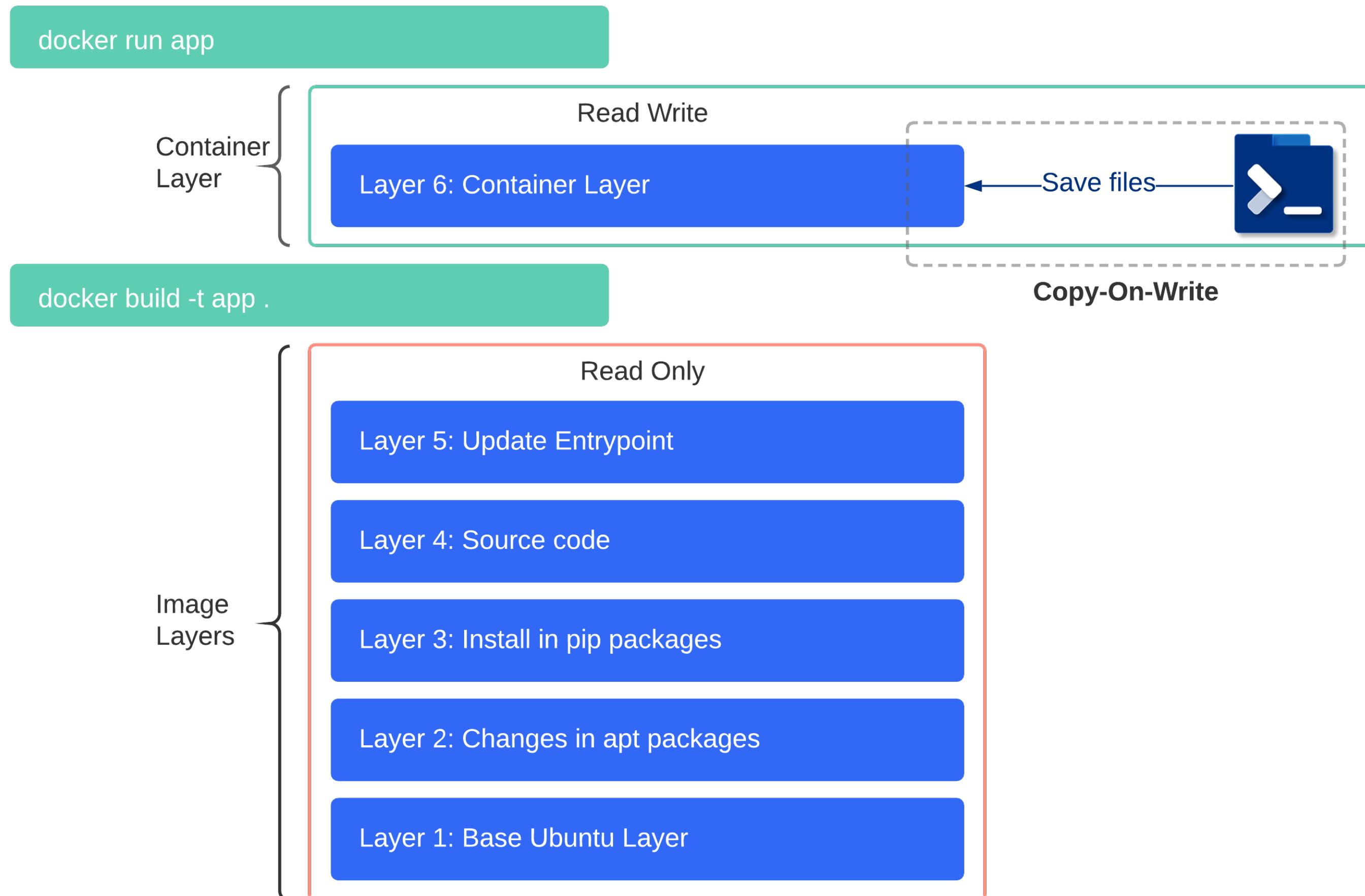


Truy cập vào địa chỉ
<http://localhost:9000>
để kiểm tra



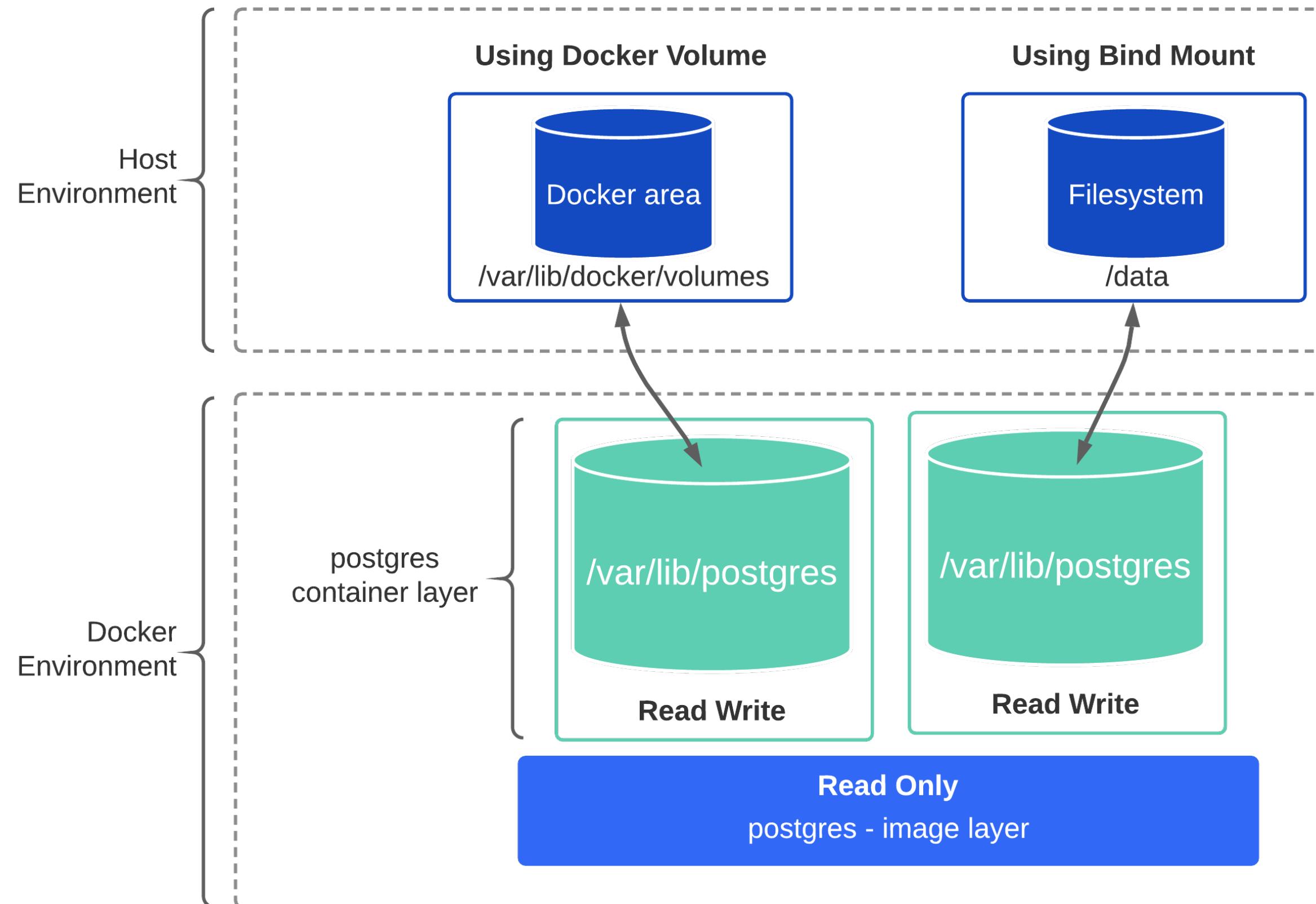
BIND MOUNT & VOLUME

Dữ liệu của một Docker container



- Dữ liệu trong quá trình chạy container được ghi vào Container Layer. Cơ chế này được gọi là Copy-On-Write.
- Tất cả dữ liệu trong Container Layer sẽ tồn tại khi container đó còn sống, nếu container bị xoá thì dữ liệu trong đó cũng bị xoá theo.

Làm sao để giữ data container không bị mất?



Mount một folder từ Docker Container vào Docker Host.

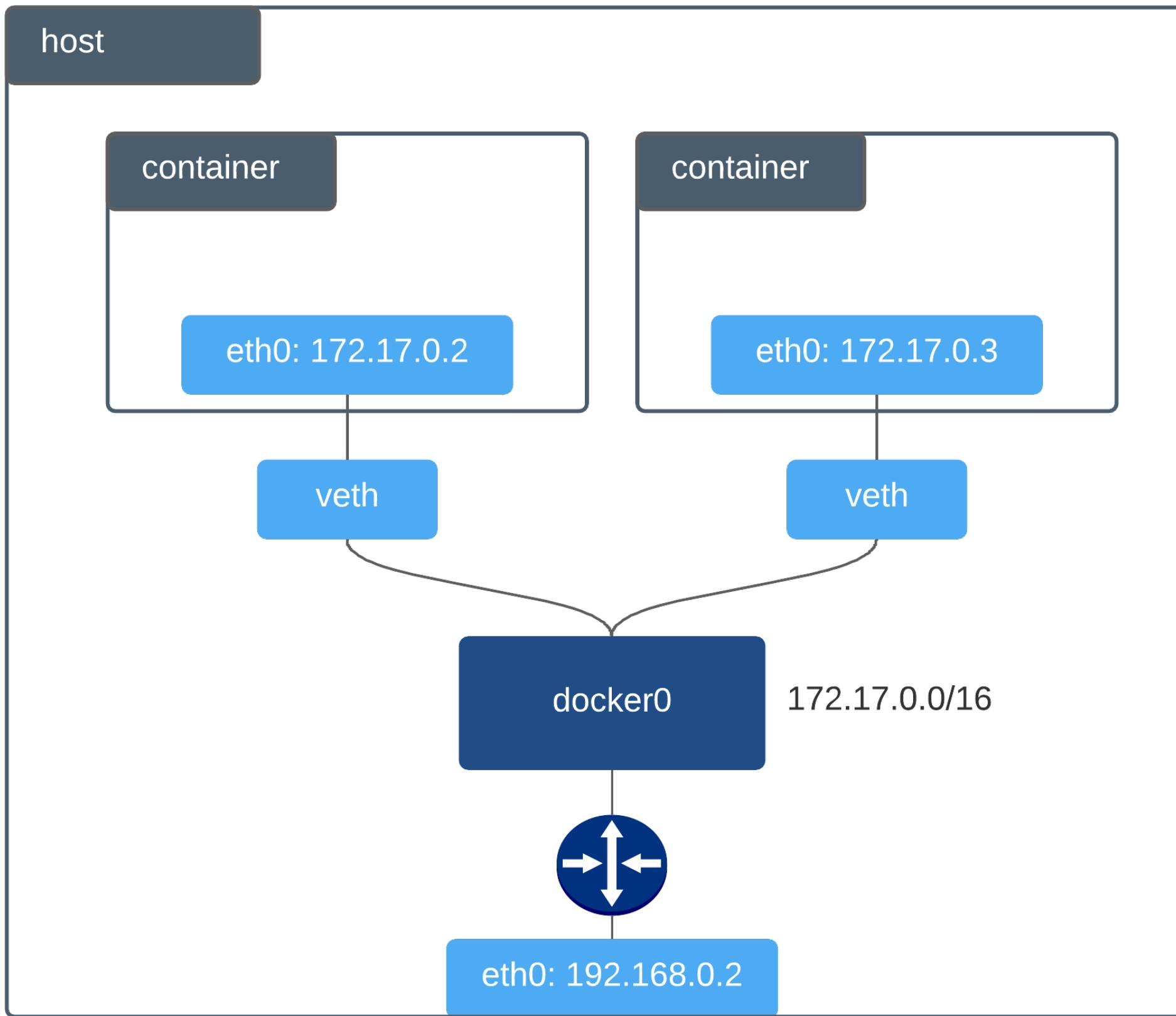
Có hai kiểu mount:

- **Volume:** Thư mục mount nằm ở `/var/lib/docker/volumes/` của Docker Host.
- **Bind mount:** Thư mục mount có thể nằm ở bất kỳ đâu trong Docker Host và không được quản lý bởi Docker.



DOCKER NETWORK

Kết nối giữa các container

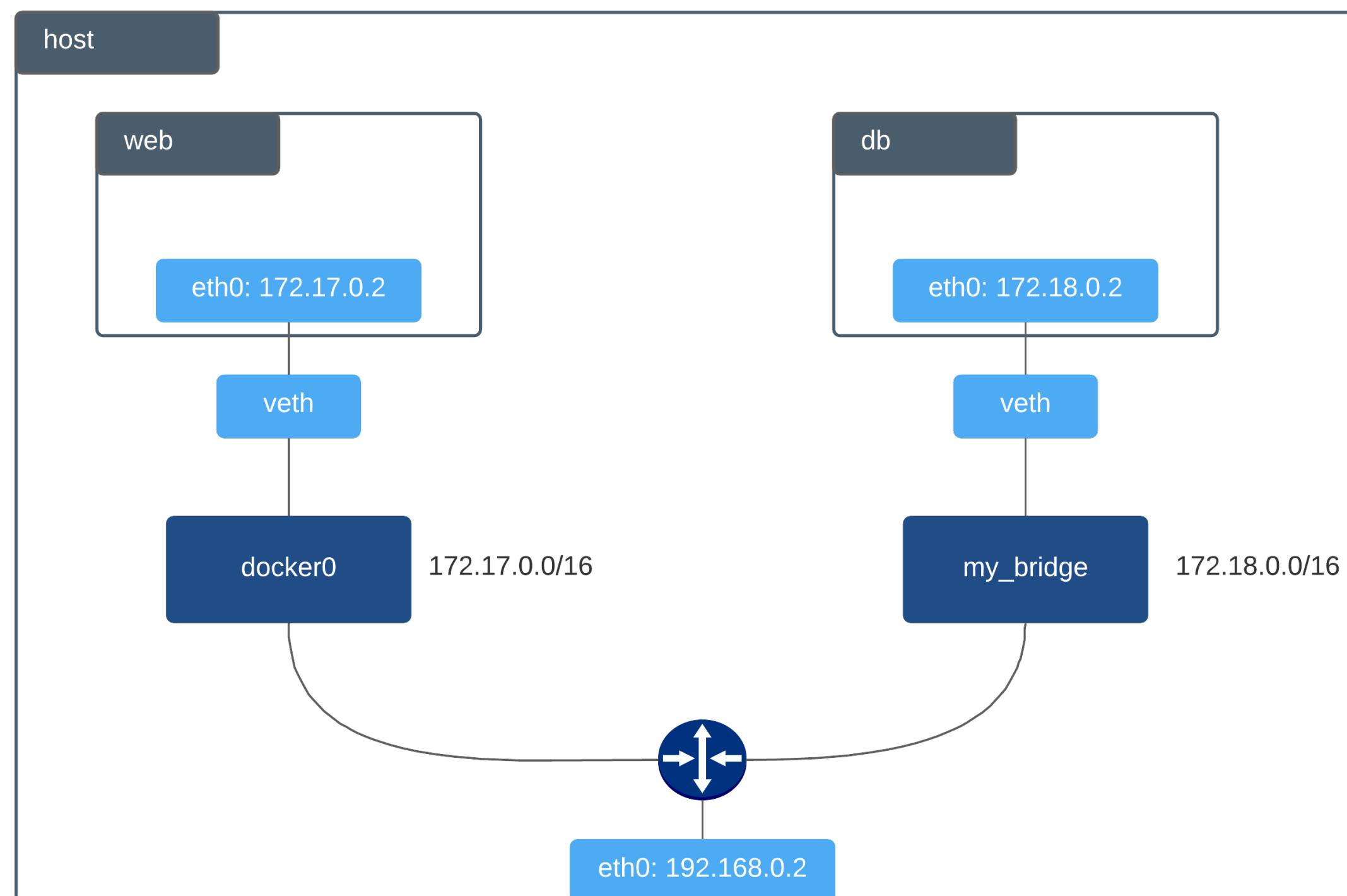


Chạy container dưới đây và kiểm tra chi tiết bridge network của docker:

```
$ docker run -d --name nginx-1 -p 8081:80 nginx:latest
$ docker network inspect bridge
```

- Mọi container khi chạy nếu không chỉ định network cho nó sẽ được tự động gắn vào docker default network (docker0)
- 2 container gắn cùng default network chỉ có thể giao tiếp với nhau qua địa chỉ IP chính xác của nhau.
- Có thể sử dụng tham số --link để setup kết nối thông qua alias (không cần đến container IP)

Kết nối giữa các container



Khởi tạo một mạng network riêng với mode là bridge và chạy db container gắn với network đó:

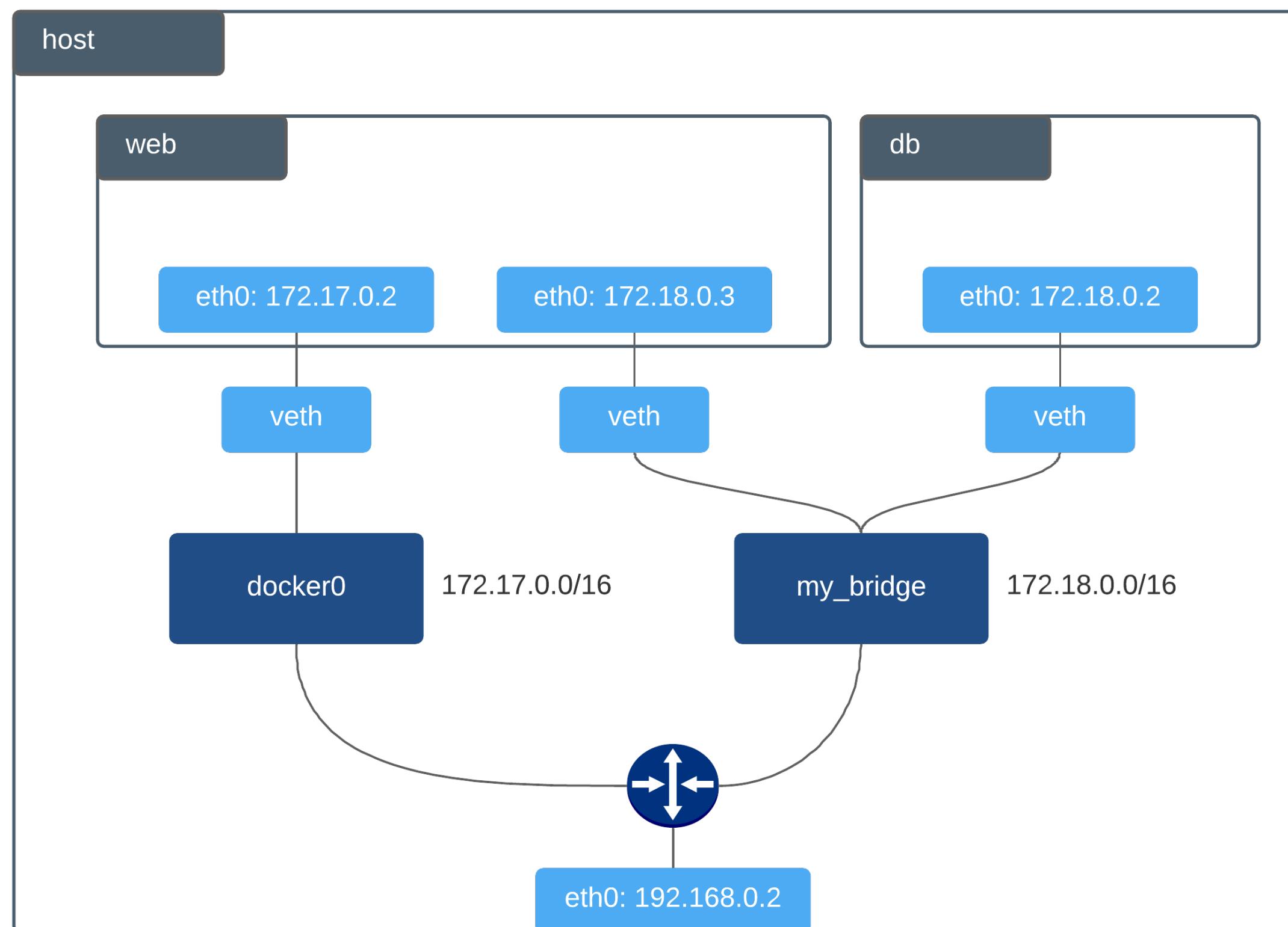
```
$ docker network create -d bridge my_network
$ docker run -d --net=my_network --name db
training/postgres
```

Chạy web-app container không cần gắn network:

```
$ docker run -d --name web training/webapp python app.py
```

*Kiểm tra sẽ thấy được web và db container vẫn chưa kết nối được với nhau vì đang không dùng chung một mạng.

Kết nối giữa các container



Thực hiện đính web container vào mạng network (`my_bridge`) đã tạo ở trên:

```
$ docker network connect my_bridge web
```

Thực hiện exec vào db container và sử dụng ping command để kiểm tra kết nối:

```
$ docker container exec -it db bash
root@a205f0dd33b2:/# ping web
PING web (172.18.0.3) 56(84) bytes of data.
64 bytes from web (172.18.0.3): icmp_seq=1 ttl=64...
64 bytes from web (172.18.0.3): icmp_seq=2 ttl=64...
```

Các loại network trong Docker

Có ba loại network hay gặp nhất

➤ bridge network

Network driver mặc định, nếu không chỉ định network cho container thì nó được mặc định gắn. Loại network này thường được sử dụng nhất, dung cho kết nối an toàn giữa các container với nhau và có thể gọi qua tên container thay vì địa chỉ IP.

➤ host network

Dành cho standalone container, sẽ xoá bỏ sự độc lập của Docker container và Docker host. Container sẽ sử dụng trực tiếp host network mà không cần đi qua docker0 .

➤ overlay network

Được dùng cho Docker mode swarm, giả sử mỗi VM instance chạy một Docker daemon – loại này cho phép Docker daemon trên các instance thiết lập chung một network. Từ đó các service swarm mode có thể giao tiếp với nhau.

TASK # 6

TRIỂN KHAI 2 CONTAINER TRÊN CÙNG 1 NETWORK

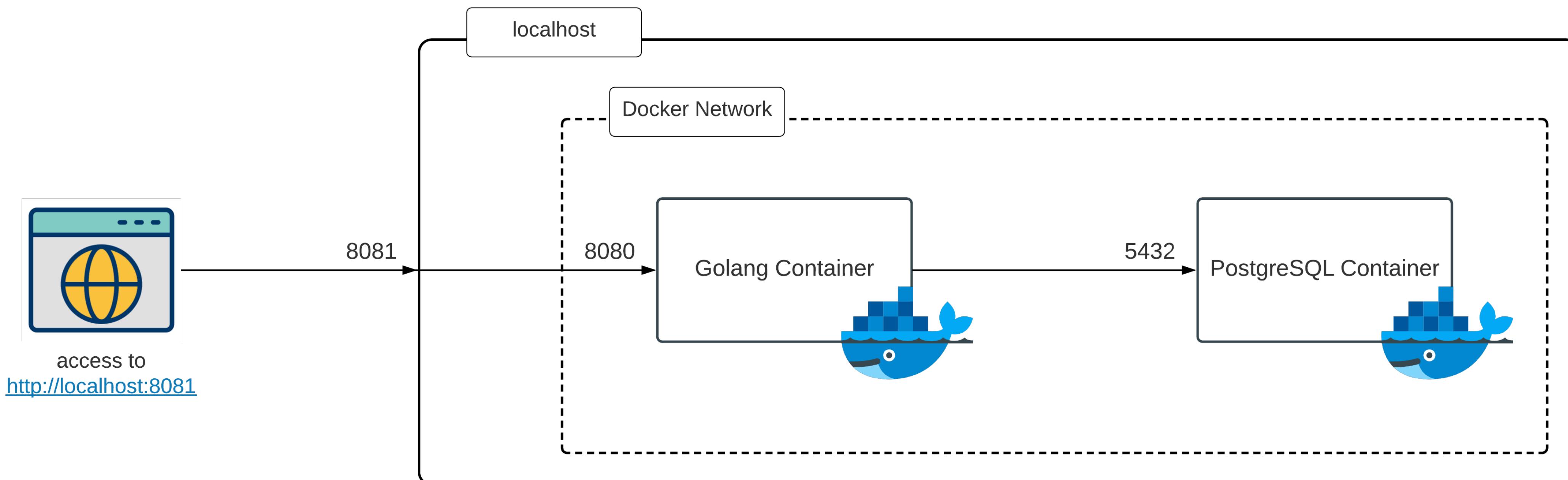
yêu cầu

- Tạo 1 docker network có tên là my-app
- Chạy 1 nginx container có tên là my-nginx, chạy ngầm và đặt trong network my-app
- Chạy 1 container sử dụng image [jwilder/whoami](#), tên container là hello-whoami, chạy ngầm và cũng đặt trong network my-app.
- docker exec vào 2 container my-nginx và hello-whoami, sau đó tiến hành cài 2 gói package ping và curl (nếu chưa có)
- Từ bên trong container my-nginx, lần lượt chạy 2 lệnh: ping hello-whoami và curl hello-whoami:8000. Kiểm tra kết quả.
- Từ bên trong container hello-whoami, lần lượt chạy 2 lệnh: ping my-nginx và curl my-nginx:80. Kiểm tra kết quả.

TASK #7

TRIỂN KHAI ỨNG DỤNG GOLANG KẾT NỐI VỚI POSTGRESQL

kiến trúc ứng dụng



yêu cầu

- Tạo một docker network có tên là todo-app
- Chạy container PostgreSQL
- Chạy container cho ứng dụng ToDo App
- Kiểm tra kết quả

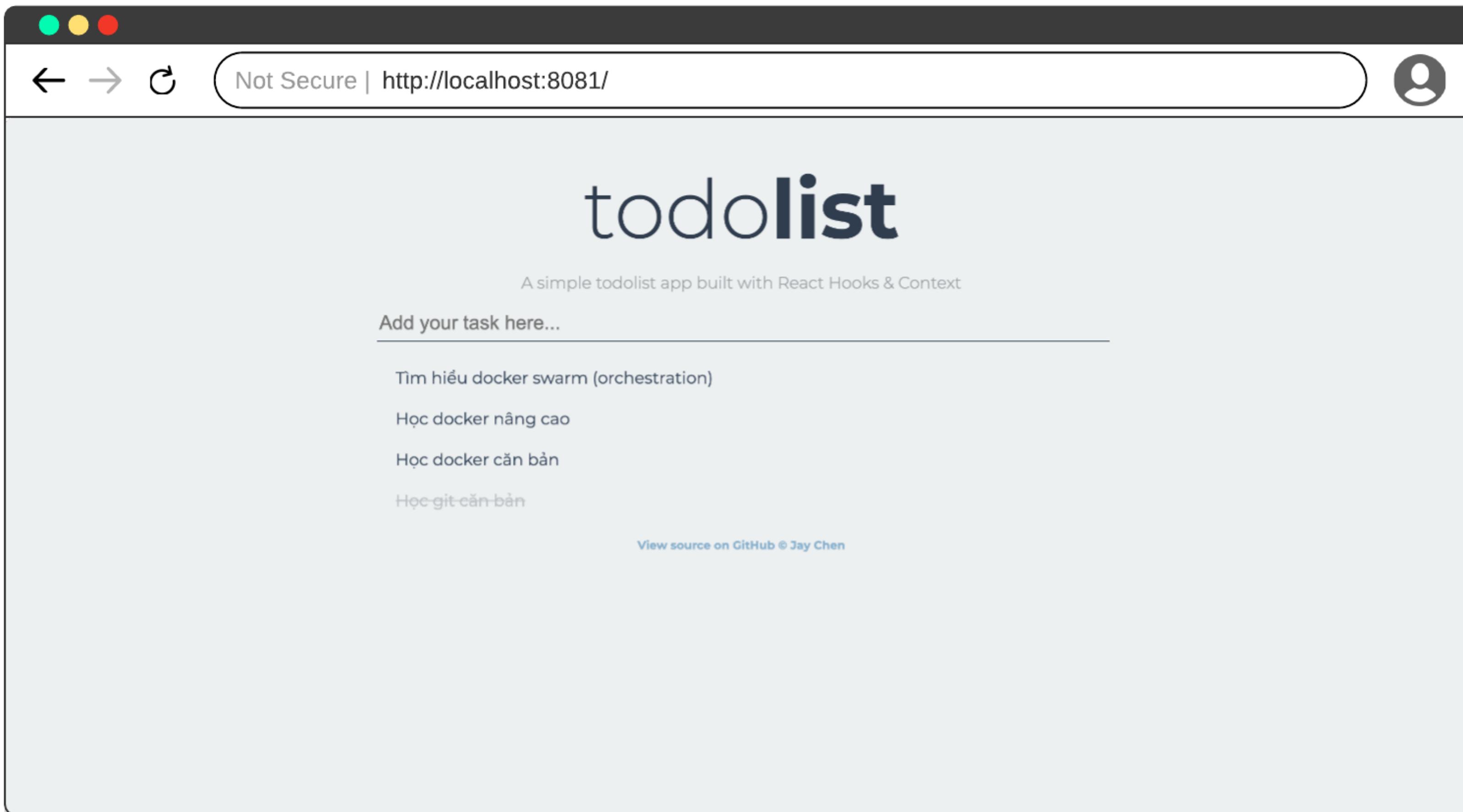
chạy container PostgreSQL

- Chạy ngầm
- Tên của container là `postgresql`
- Đặt trong network `todo-app`
- Mount thư mục `/var/lib/postgresql/data` của container ra một thư mục bất kỳ trên máy host
- Đặt giá trị cho biến môi trường `POSTGRES_USER` là `postgres`
- Đặt giá trị cho biến môi trường `POSTGRES_PASSWORD` là `root123`
- Đặt giá trị cho biến môi trường `POSTGRES_DB` là `registry`
- Sử dụng image là `thuongnn1997/todo-app-db:latest`

chạy container cho ứng dụng Golang

- Chạy ngầm
- Đặt trong network `todo-app`
- Expose cổng 8080 của container ra cổng `8081` trên máy host
- Sử dụng image là `thuongnn1997/todo-app:latest`

kiểm tra kết quả

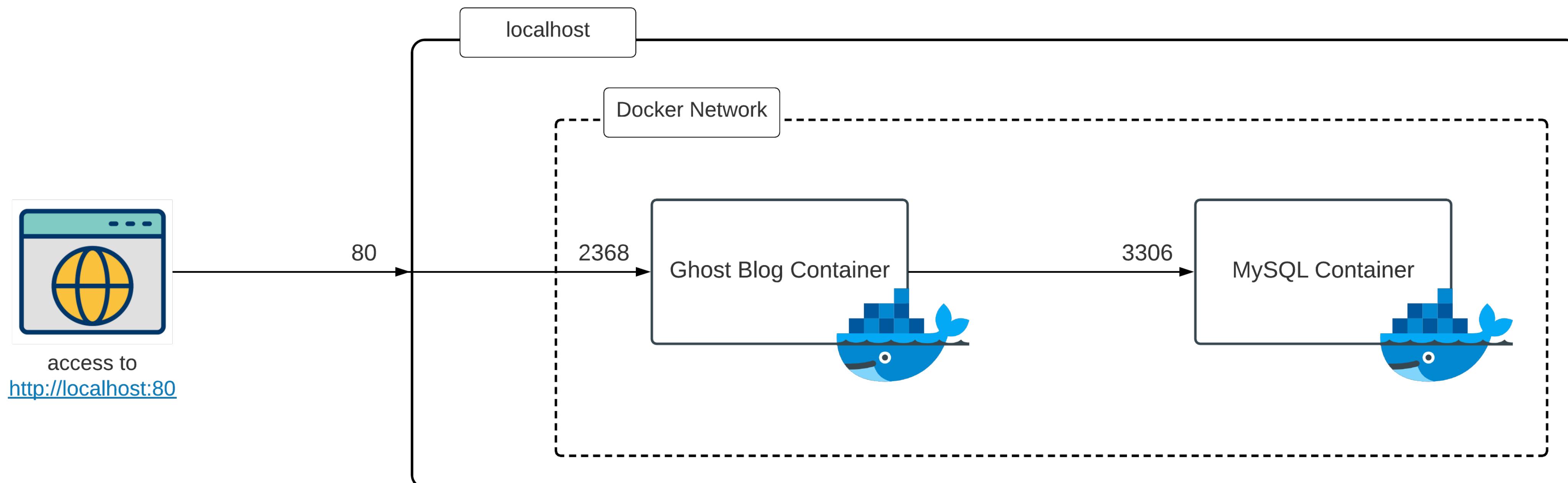


Truy cập vào địa chỉ
<http://localhost:8081>
để kiểm tra

TASK #8

TRIỂN KHAI TRANG WEB VIẾT BLOG BẰNG DOCKER

kiến trúc ứng dụng



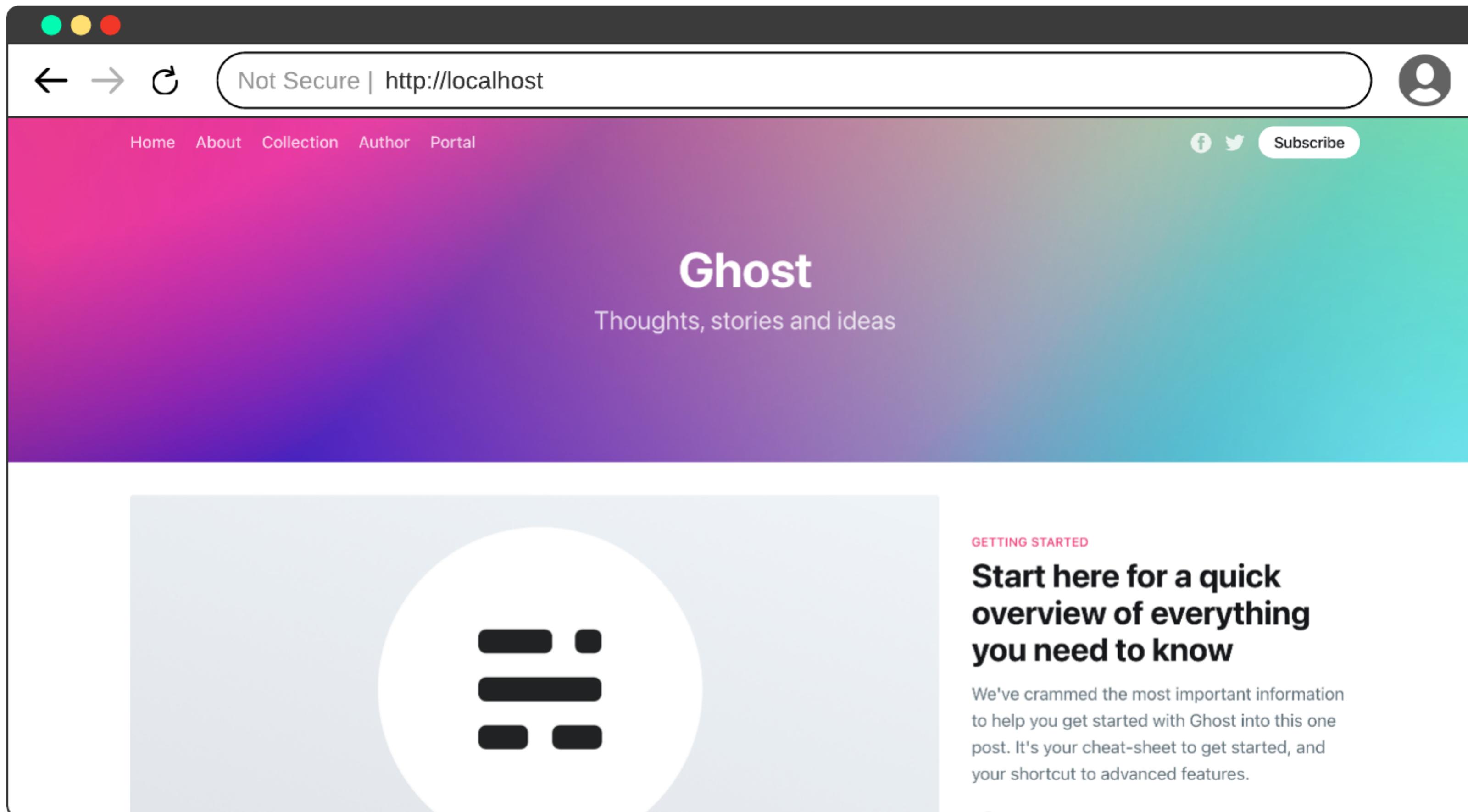
chạy container MySQL

- Chạy ngầm
- Tên của container là db
- Đặt trong network ghost-network
- Mount thư mục /var/lib/mysql của container sử dụng volume quản lý bởi docker
- Đặt giá trị cho biến môi trường MYSQL_ROOT_PASSWORD là example
- Sử dụng image là mysql:5.7

chạy container Ghost Blog

- Chạy ngầm
- Đặt trong network `ghost-network`
- Expose cổng 2368 của container ra cổng `80` trên máy host
- Thiết lập biến môi trường cho container:
 - `database_client = mysql`
 - `database_connection_host = db`
 - `database_connection_user = root`
 - `database_connection_password = example`
 - `database_connection_database = ghost`
- Sử dụng image là `ghost:alpine`

kiểm tra kết quả



Truy cập vào địa chỉ
<http://localhost> để
kiểm tra

Thanks for watching

Email: thuongnn666@gmail.com
