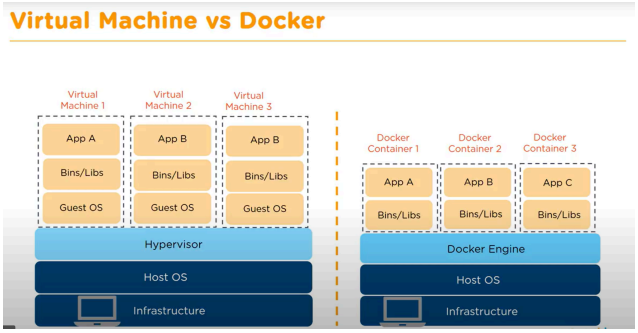
NHỮNG KIẾN THỨC ĐÃ HỌC ĐƯỢC

1. Vấn đề đặt ra: Mã nguồn không chạy được trên máy tính khác vì khác môi trường cài đặt

2. Giải pháp: Máy ảo hoặc Docker

Vậy để hiểu rõ docker là gì và tại sao việc dùng docker ngày càng phổ biến thì đầu tiên chúng ta sẽ đi so sánh Docker với một khái niệm mà đã học qua rồi là máy ảo



Dựa vào hình ta sẽ thấy máy ảo có một lớp hypervisor trong khi đó docker sẽ có 1 lớp docker engine tuy nhiên trong máy ảo còn có thêm một lớp nữa là Guest OS còn docker thì không. Từ đó tạo ra những điểm khác biệt sau:

2.1. Memory usage: VM dùng khá nhiều nhưng docker thì không

Với máy ảo, khi phân bổ bộ nhớ rồi thì dù không dùng cũng sẽ không thể phân bổ lại. Nhưng với docker, nếu còn bộ nhớ trống thì nó sẽ được phân bổ lại và sử dụng lại trên các container khác

2.2. Performance: Khi dùng nhiều máy ảo trên cùng 1 server thì hiệu suất sẽ tệ đi trong khi đó với docker thì hiệu suất sẽ luôn tốt nhờ vào kiến trúc tinh gọn được sử dụng để xây dựng docker container

2.3. Portability (tính di động): việc dùng máy ảo có thể gây ra nhiều lỗi ngược lại docker được thiết kế cho tính di động

2.4. Thời gian khởi động: thời gian khởi động cho máy ảo khá chậm so với docker, gần như ngay lập tức

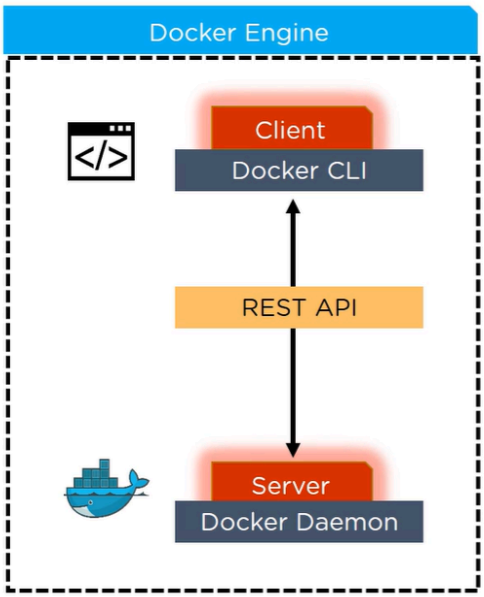
3. Vậy Docker thật sự là gì ?

Docker là một phần mềm cho phép tạo ra các container để chạy ứng dụng cùng tất cả các thứ cần thiết cho nó, giúp các lập trình viên và quản trị viên IT dễ dàng triển khai và vận hành ứng dụng mà không lo thiếu thư viện hay cấu hình.

4. Những lợi ích của Docker

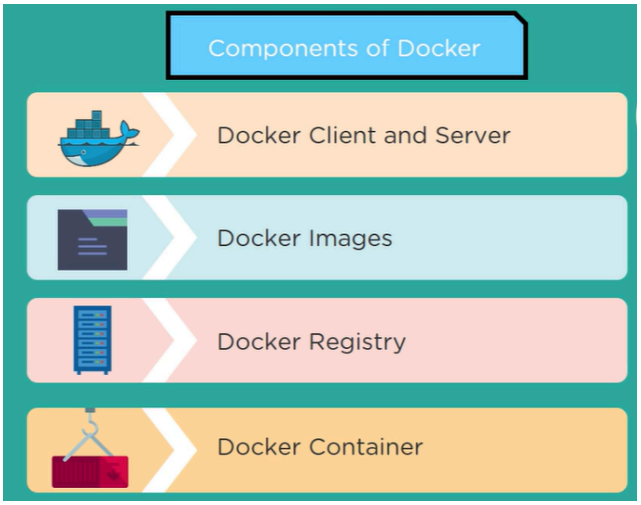
- Triển khai nhanh chóng
- Có tính di động cao
- Hiệu suất cao hơn
- Cấu hình nhanh hơn
- Khả năng mở rộng cao
- Có tính bảo mật cao

5. Docker làm việc như thế nào?

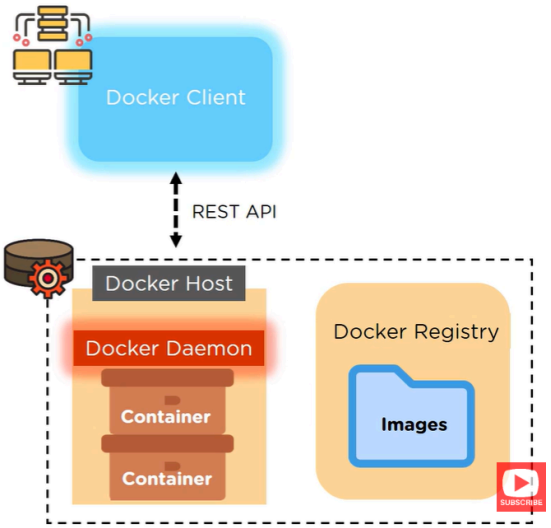


- Docker Engine gồm Docker Client và Docker Daemon, chúng giao tiếp với nhau thông qua REST API
  + Docker Client là một dịch vụ dùng để chạy lệnh.
    Lệnh này được chuyển đổi qua REST API và gửi đến Docker Daemon (server)
  + Sau đó, Docker Daemon sẽ kiểm tra yêu cầu từ client và tương tác với hệ điều hành để tạo hoặc quản lý các container

6. Các thành phần của Docker



6.1. Docker Client and Server (Máy chủ và máy khách)



6.1.1. Docker Client

- Docker Client bao gồm các lệnh CLI, được dùng để gửi đến Docker Daemon
- Docker Client sử dụng REST API để gửi lệnh đến Docker Daemon, thông qua các script hoặc các lệnh CLI trực tiếp
- Ví dụ: Khi chạy lệnh **docker pull**, Docker Client sẽ gửi lệnh này đến Docker Daemon. Daemon sẽ thực hiện các thao tác đó bằng cách tương tác với các thành phần khác như Image, Container, Registry

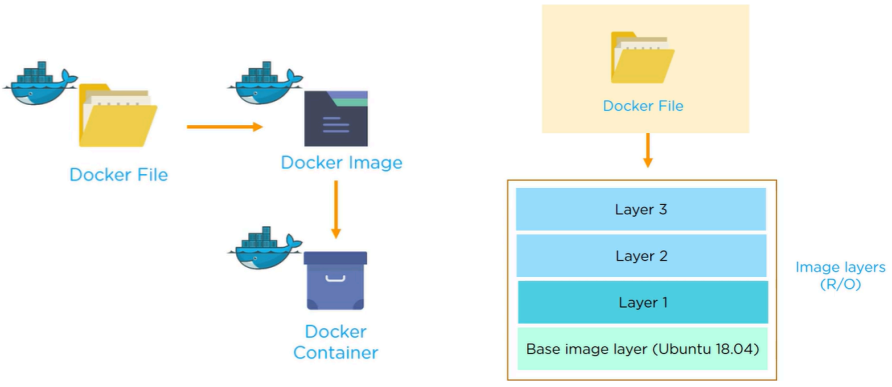
6.1.2. Docker Daemon (server)

- Docker Daemon là một máy chủ tương tác với hệ điều hành và thực hiện mọi loại dịch vụ
- Docker Daemon lắng nghe các yêu cầu REST API và thực hiện các thao tác tương ứng
- Lệnh **dockerd** được sử dụng để khởi động một Docker Daemon
- Docker Host là máy chủ chạy Docker Daemon và Docker Registry

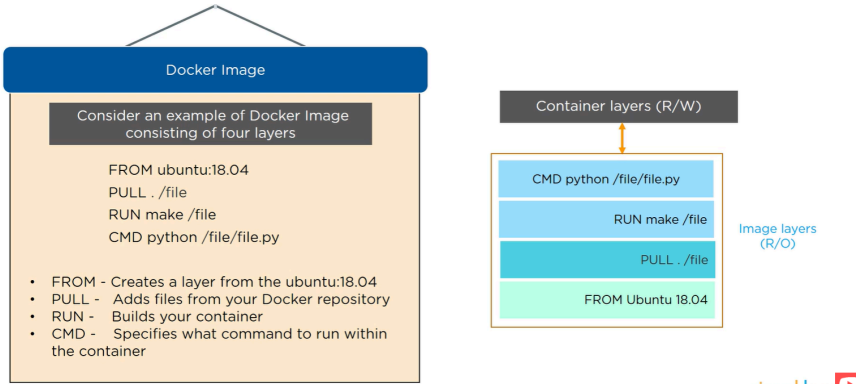
7. Docker File

- Dockerfile là một file văn bản đơn giản, chứa các hướng dẫn để xây dựng Docker Image

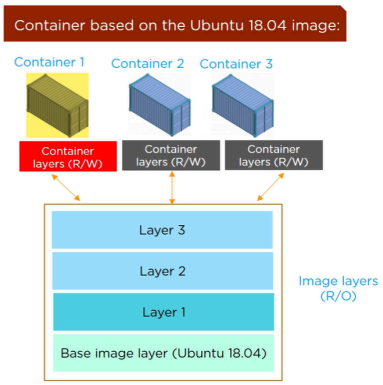
8. Docker Image



- Docker Image là một mẫu (template) chứa các hướng dẫn, được dùng để tạo Docker Containers
- Docker Image được xây dựng từ một file gọi là Dockerfile
- Docker Image được lưu trữ trên Docker Hub hoặc trong một repository (ví dụ: **registry.hub.docker.com**)
- Docker Image gồm nhiều lớp layer. Mặc định Docker Image bắt đầu với base image layer
- Mỗi lớp trong image phụ thuộc vào lớp bên dưới nó, khi lớp bên dưới có thay đổi thì những lớp bên trên cũng phải cập nhật theo
- Các lớp của image được tạo ra bằng cách thực thi từng lệnh trong Docker File, và các lớp này ở chế độ read-only
- Vd: tạo một Docker Image gồm 4 lớp layer



  + Dùng lệnh FROM để tạo lớp base với hệ điều hành Ubuntu
  + Thêm các file từ Docker repository bằng lệnh PULL
  + Chạy thư mục đó (biên dịch, cài đặt các phụ thuộc,...) hay tạo container bằng lệnh RUN
  + Chỉ định file nào sẽ chạy khi container start bằng CMD



- Mỗi khi người dùng tạo một container, một lớp mới được hình thành trên các layer của image, gọi là container layer
- Mỗi container có một lớp riêng (có thể đọc/ghi – R/W) và mọi thay đổi trong container chỉ ảnh hưởng đến lớp này
- Khi container bị xóa, lớp trên cùng (container layer) cũng sẽ bị xóa
- Khi có thay đổi trong một layer của image người dùng có thể thêm một layer mới lên trên base image tuy nhiên người dùng không thể chỉnh sửa bất kỳ layer nào đã tồn tại trong image
- Các lớp base (base layers) ở chế độ chỉ đọc (read-only)
- Các layer có thể được kết hợp trong một union file system để tạo thành một image duy nhất
- Union file system tiết kiệm bộ nhớ bằng cách tránh sao chép trùng lặp các file
- Điều này cho phép file system trông như có thể ghi (writable) nhưng không thay đổi file gốc, được gọi là copy-on-write
- Copy-on-write (CoW) = file gốc dùng chung cho nhiều container, chỉ khi cần sửa thì mới tạo bản sao. Nhờ đó: Không phải sao chép tất cả file → tiết kiệm bộ nhớ và container khởi động và chạy nhanh hơn → hiệu suất cao hơn

9. Docker Registry

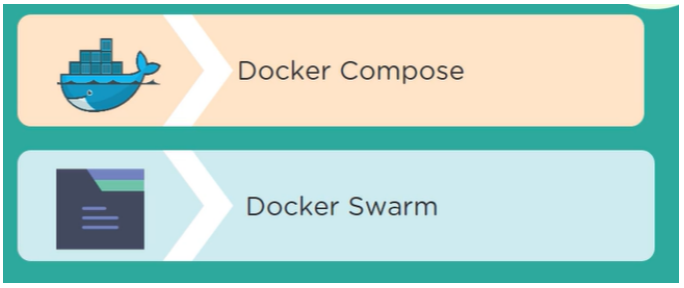
- Docker Registry có các repository công khai (public) và riêng tư (private)
- Trong Registry, lệnh **push** và **pull** được dùng để tương tác với Docker Images
- Pull → lấy (tải về) một Docker Image từ Docker Registry
- Push → đẩy (lưu trữ) một Docker Image lên Docker Registry
- Trong Docker Registry, xóa một repository là hành động không thể hoàn tác

10. Docker Container

- Docker Container là một gói thực thi (executable package) gồm ứng dụng và tất cả các phụ thuộc của nó.
- Vì nhẹ, container có thể dễ dàng triển khai và chạy trên các môi trường máy tính khác nhau, bất kể hệ điều hành hoặc cấu hình của máy chủ
- Docker Container chạy ứng dụng trong môi trường tách biệt, nhưng chia sẻ kernel của hệ điều hành với các container khác
- Ở đây, data volumes có thể được chia sẻ và tái sử dụng giữa nhiều container
- Container được xây dựng từ Docker Images

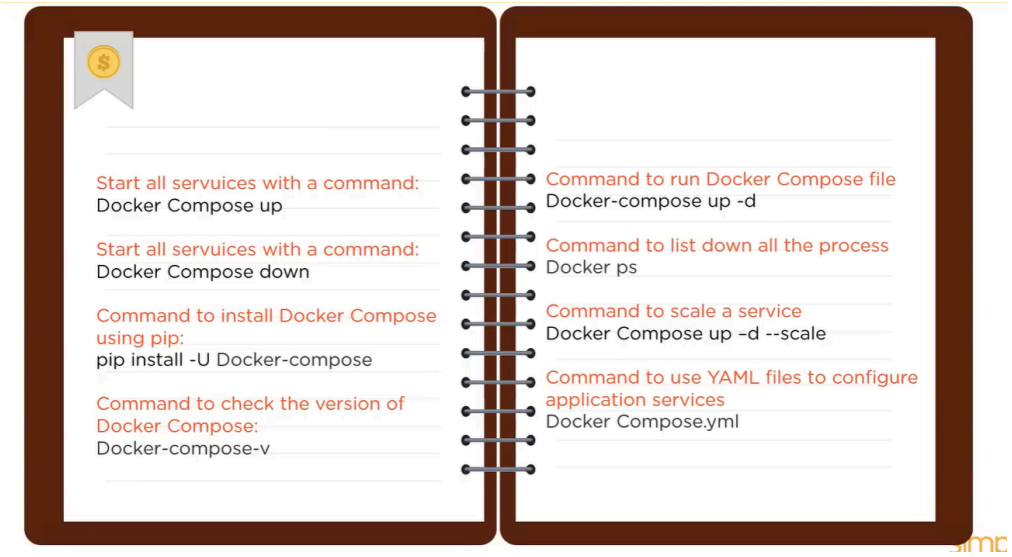
- Lệnh docker run dùng để tạo và chạy một container
- Docker Container nhẹ vì không cần một lớp hypervisor bổ sung và chạy trực tiếp trên hệ điều hành của máy chủ (host OS)

11. Những khái niệm nâng cao trong Docker



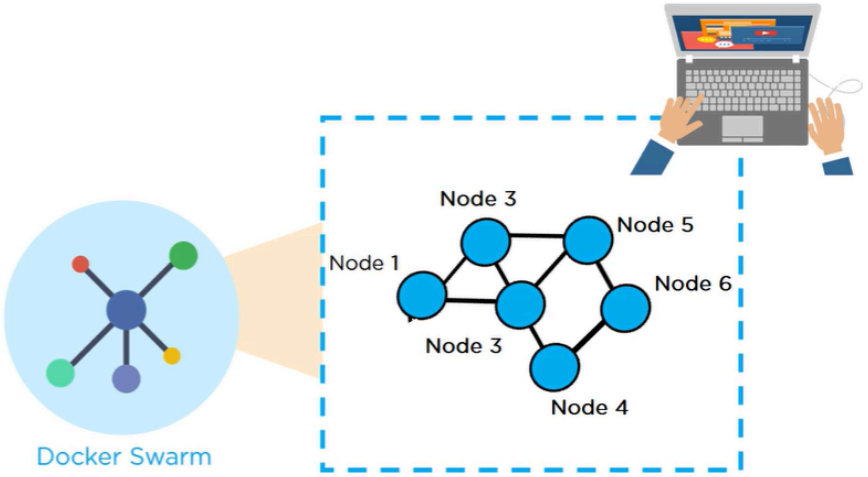
11.1. Docker Compose

- Docker Compose được sử dụng để chạy nhiều container cùng lúc
- Ở đây, mỗi container chạy riêng biệt (isolation) nhưng có thể tương tác với nhau
- Tất cả các file Docker Compose đều là file YAML
- Ví dụ: Nếu bạn có một ứng dụng cần máy chủ Apache và cơ sở dữ liệu MySQL, bạn có thể tạo một file Docker Compose để chạy cả hai container như một dịch vụ duy nhất, mà không cần phải khởi động từng container một
- Ưu điểm:
    + Triển khai trên một môi trường duy nhất
    + Cấu hình nhanh và đơn giản
    + Hiệu suất cao
    + Bảo mật
- Một số lệnh cơ bản của Docker Compose

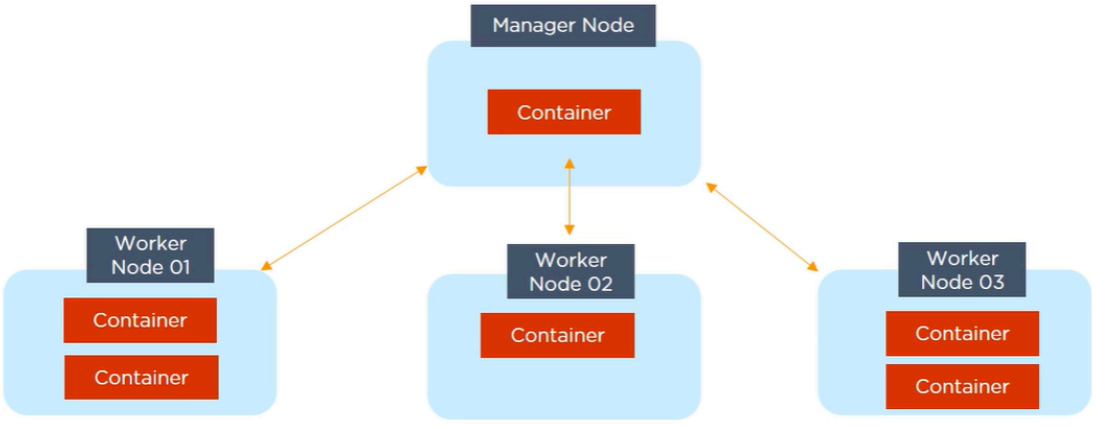


11.2. Docker Swarm

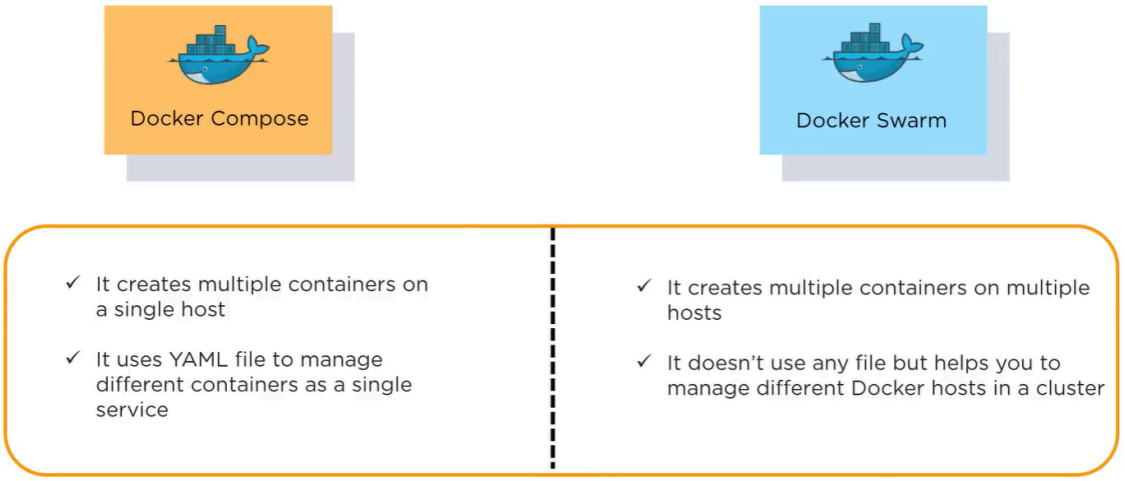
- Docker Swarm là một dịch vụ quản lý container, cho phép quản trị viên IT và lập trình viên tạo và quản lý một cụm (cluster) các nút Swarm bên trong nền tảng Docker
- Mỗi node trong Docker Swarm là một Docker Daemon, và tất cả Docker Daemon này giao tiếp với nhau thông qua Docker API



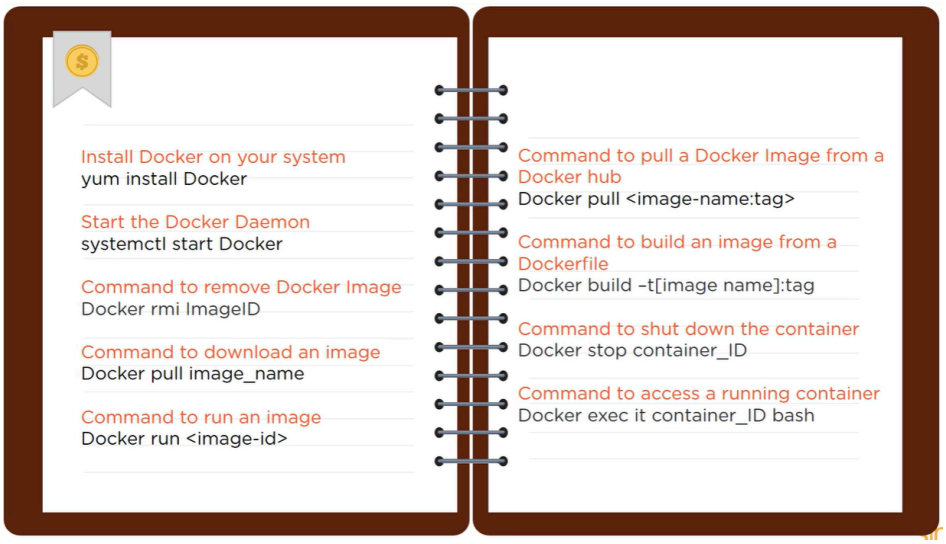
- Một Swarm bao gồm hai loại node: Manager node và Worker node
    + Manager Node chịu trách nhiệm duy trì và quản lý toàn bộ cụm (cluster)
    + Worker Node sẽ nhận và thực thi các tác vụ được gửi từ manager node
- Ưu điểm:
    + Truy cập phân tán
    + Bảo mật cao
    + Tự động cân bằng tải
    + Khả năng mở rộng cao
    + Có thể roll-back khi gặp sự cố



11.3. Phân biệt Docker Compose vs Docker Swarm



14. Một số lệnh Docker cơ bản



15. Docker Networking

- Docker networking cho phép người dùng kết nối một container Docker với nhiều mạng khác nhau tùy nhu cầu
- Docker Networks được sử dụng để cung cấp sự cô lập hoàn toàn cho các container Docker
- Ưu điểm:
    + Triển khai nhanh
    + Tính di động cao
    + Hiệu suất tốt hơn
    + Cấu hình nhanh hơn
    + Có thể mở rộng
    + Bảo mật
- Docker Driver là thành phần chịu trách nhiệm thiết lập, quản lý và vận hành mạng cho container. Bao gồm các loại như bridge, host, none, overlay, macvlan
    + **Bridge:**

Đây là mạng mặc định riêng tư (private default network) được tạo trên host

Các container được kết nối vào mạng này sẽ có địa chỉ IP nội bộ, nhờ đó giao tiếp với nhau dễ dàng

Docker server (daemon) tạo một cầu nối Ethernet ảo tên docker=0, hoạt động bằng cách tự động truyền gói dữ liệu giữa các interface mạng khác nhau

+ **Host:**

Đây là một mạng công cộng (public network)

Nó sử dụng địa chỉ IP của host và không gian cổng TCP để hiển thị các service đang chạy bên trong container

Điều này về cơ bản tắt cô lập mạng giữa host Docker và container, có nghĩa là nếu dùng driver này, người dùng sẽ không thể chạy nhiều container trên cùng một host mà sử dụng cùng cổng

+ **None:**

Với network driver này (none), các container Docker không có quyền truy cập mạng bên ngoài và không thể giao tiếp với các container khác

Tùy chọn này dùng khi người dùng muốn tắt hoàn toàn truy cập mạng của container

Nói đơn giản, none được gọi là loopback interface, có nghĩa là container chỉ có giao diện mạng nội bộ, không có kết nối ra ngoài

+ **Overlay:**

Driver này được sử dụng để tạo một mạng nội bộ riêng tư (internal private network) cho các node Docker trong Docker Swarm cluster.

Lưu ý: Docker Swarm là một dịch vụ quản lý container, cho phép các IT administrator và developer tạo và quản lý một cluster các node Swarm trên nền tảng Docker

Đây là một network driver quan trọng trong Docker networking, giúp các container độc lập giao tiếp với service trong Docker Swarm

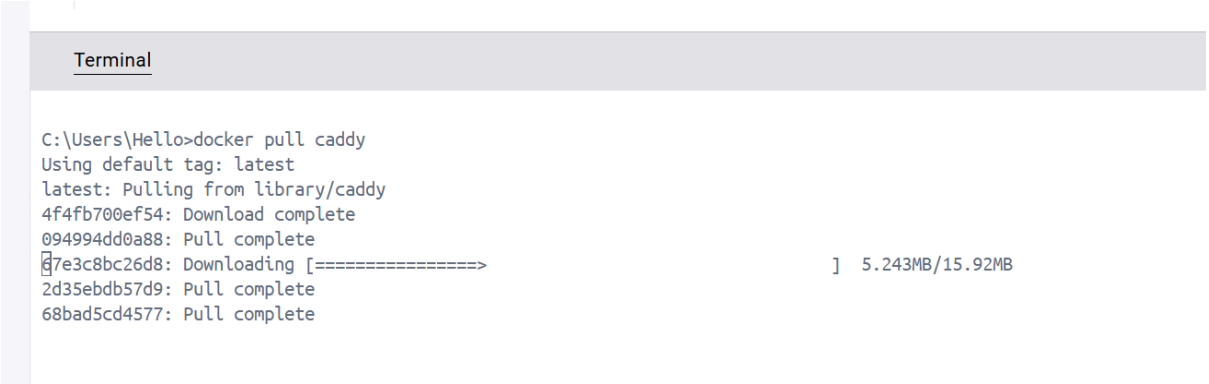
+ **Macvlan:**

Driver này đơn giản hóa quá trình giao tiếp giữa các container

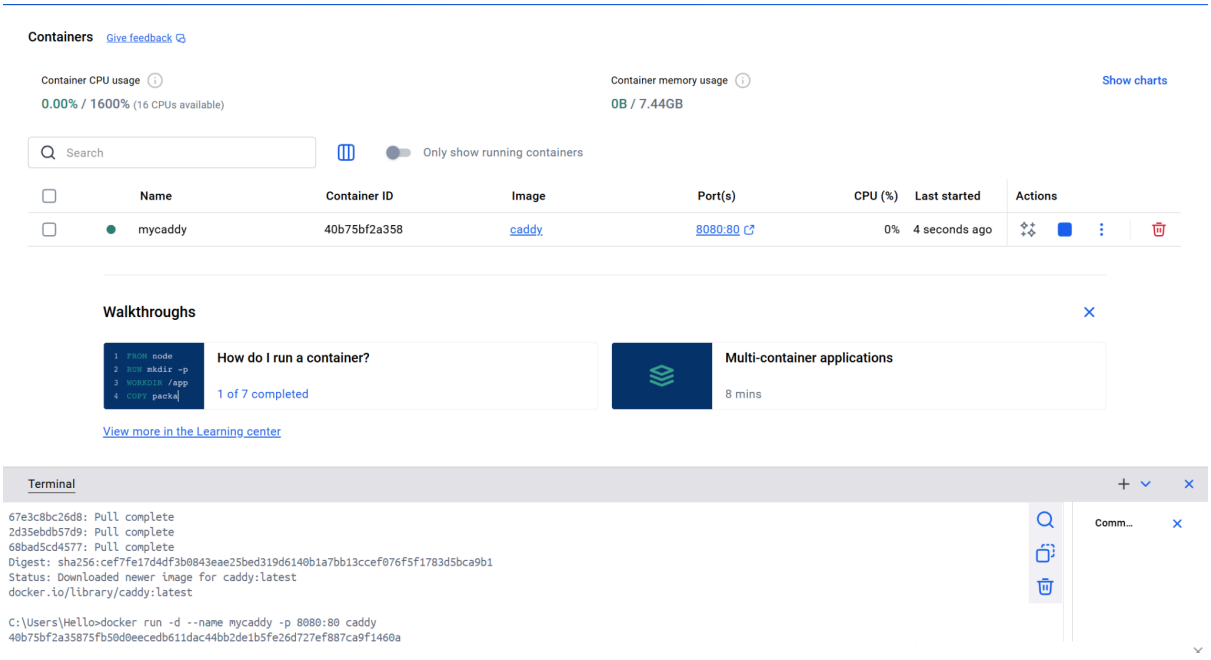
Mạng này gán một địa chỉ MAC cho container Docker. Nhờ địa chỉ MAC này, Docker server (daemon) có thể định tuyến lưu lượng mạng đến router

Driver này thích hợp khi người dùng muốn kết nối trực tiếp container với mạng vật lý, thay vì thông qua mạng của Docker host

CÁC BƯỚC THỰC HÀNH



Quá trình pull Docker image



Quá trình chạy Docker

KHÓ KHĂN/ LỖI GẶP PHẢI

Vì dùng Docker trên hệ điều hành Window nên các tài liệu hướng dẫn ban đầu khá khó kiếm

CÁC CHỦ ĐỀ MUỐN TÌM HIỂU THÊM VỀ DOCKER

Em muốn tìm hiểu thêm về Docker Networking vì em chỉ mới hiểu sơ lý thuyết chứ chưa nắm rõ được phải làm như thế nào