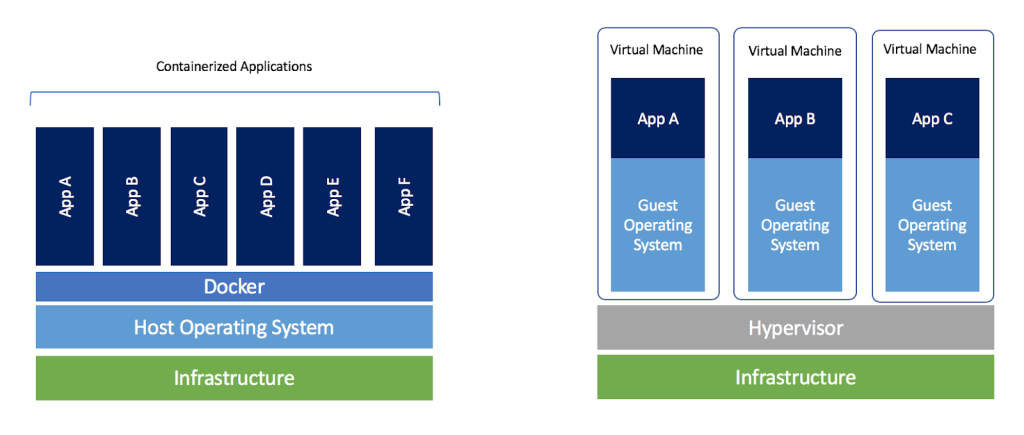
***TÀI LIỆU THAM KHẢO:***

**TÌM HIỂU CƠ BẢN VỀ CÔNG NGHỆ DOCKER**

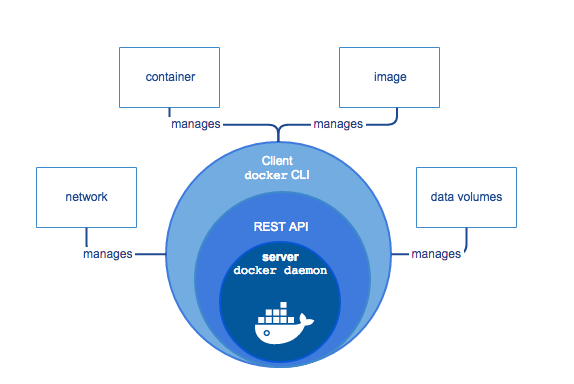
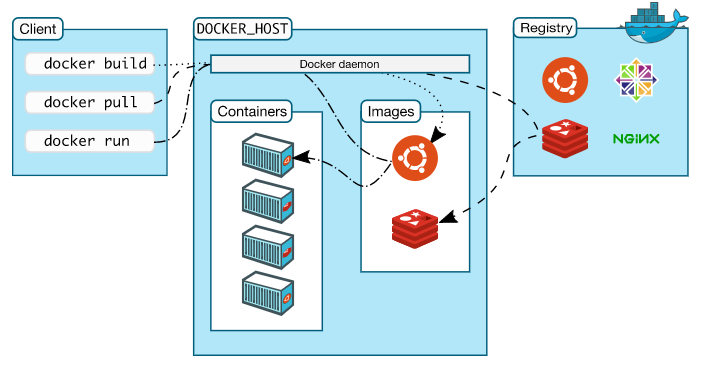
1. **Introduction**

Virtual Machine (VM - Máy ảo) là một chương trình đóng vai trò như một máy tính ảo, có đầy đủ chức năng, được dựng lên bằng cách tạo guest OS trên hOSt OS, phân hoạch độc lập (chiếm resource cố định) và tương tác trực tiếp với phần cứng của máy hOSt.

Docker cũng tương tự như VM, tạo môi trường ảo độc lập nhưng không sử dụng guest OS. Thay vào đó, Docker đóng gói các chương trình và môi trường chạy lại thành các containers, sau đó thực thi trực tiếp trên host OS, sử dụng chung phần cứng và các infrastructure khác.



1. **Docker Engine**



Docker Engine hoạt động như một client-server application và bao gồm các phần sau:

* Dockerd : Tiến trình daemon quản trị các Docker objects như image, container, network, volume
* REST API : Được Dockerd expOSe ra ngoài dùng để tương tác với users
* CLI : Giao diện tương tác command line của client, lệnh được chuyển thành REST API gửi tới server

1. **Docker concepts**

**a. Image**

Là template để tạo Container, read only, thường được tạo dựa trên 1 image khác (initial image), với một số customizations. Có thể tự build image hoặc dùng image có sẵn, pull từ các Docker Registry.

Một docker image được xây dựng từ nhiều layer xếp chồng lên nhau. Mỗi layer là một dòng lệnh trong Dockerfile và tất cả các layer trừ layer ngoài cùng là read-only. Khi khởi tạo container thì mình tạo thêm 1 writable layer lên trên các layer của image, đây được gọi là container layer, tất cả các thay đổi được tạo ra khi container chạy, như thêm/xóa/sửa file đều được add thêm vào layer này.

Để tạo Docker image thì mình tạo Dockerfile (định nghĩa các steps để tạo image và chạy). Nếu mình sửa Dockerfile và dựng lại image thì chỉ phần thay đổi của image được build lại, vì từng lệnh của Dockerfile sẽ tạo từng layer cho image. Đây chính là Docker đã tái sử dụng các lớp trung gian khi tạo image, giúp giảm việc chiếm quá nhiều disk space -> cache.

Tên của Docker image gồm 3 phần:

\* Registry address: optional, nếu muốn push lên Registry thì bắt buộc phải có

\* Name

\* Tag: optional, mặc định là latest

**Ví dụ:** 10.60.100.39:8297/saga-service:1.0 có

Registry address: 10.60.100.39:8297

Name: saga-service

Tag: 1.0

**b. Containers**

Hiểu nôm na, nếu image là class thì container là instance và chạy duy nhất một tiến trình chính. Vòng đời của tiến trình cũng gắn liền với vòng đời của container. Cần lưu ý rằng khi container bị xóa thì mọi dữ liệu sinh ra trong quá trình chạy cũng mất theo, trừ khi cấu hình trong lệnh khởi chạy container.

Container như một chiếc hộp giúp tiến trình và các resource cần thiết được đóng kín lại và biệt lập với các tiến trình khác. Để cung cấp sự biệt lập (isolation) và resource allocation, Docker sử dụng 2 công nghệ chính:

+ Linux kernel namespace: Mỗi container sẽ có các namespace riêng.

+ Kernel cgroups: Thiết lập giới hạn truy cập của container đến các resource/namespace chỉ định.

Bởi vì mỗi container có cho riêng mình một writable layer nên mặc dùng chúng dùng chung image, chúng vẫn có data của riêng mình.

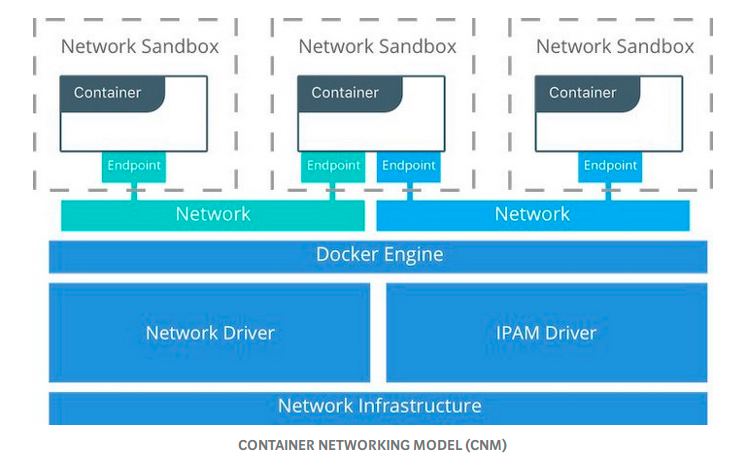
Tham khảo thêm: <https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf>

**Đọc thêm:** So sánh giữa hai phương pháp ảo hóa container và virtual machine

|  |  |
| --- | --- |
| CONTAINER | VIRTUAL MACHINE |
| Giả lập hệ điều hành | Giả lập phần cứng |
| Quản trị trên nền là hệ điều hành | Nằm giữa host OS và lớp infrastructure phía dưới |
| Định nghĩa các dependency ở application level | Định nghĩa các dependency ở OS level |
| Sử dụng trực tiếp host OS, sử dụng chung host OS | Chạy trên nền hypervisor để truy cập đến phần cứng, sử dụng guest OS riêng |
| Resource fit theo máy host | Resource bị giới hạn theo cấu hình |
| Small size on disk | Big size on disk |
| Low overhead | High overhead |
| Chạy một application | Chạy nhiều application |

**c. Volumes**

Volume là phương pháp backup dữ liệu cho container, kể cả khi vòng đời của container kết thúc thì volume vẫn được lưu lại và không bị ảnh hưởng. Lưu ý rằng mọi update dữ liệu ở volume đều sẽ được chuyển tiếp đến dữ liệu bên trong container và ngược lại.



**d. Network**

Vì các container là cô lập với nhau, Docker network cung cấp phương thức cho các container communicate. Có 3 loại network chính:

+ Bridge: liên kết nhiều container lại với nhau. Ngoài ra còn dùng để liên kết mạng bên ngoài và mạng bên trong (Docker local network) của máy host.

+ Host: gán network của container vào network của máy host.

+ Overlay: kết nối nhiều Docker Machine lại với nhau. Là mode mặc định khi chạy Docker Swarm, Docker Stack, Docker Compose.

1. **Docker commands**

|  |  |
| --- | --- |
| docker images | Lấy danh sách tất cả các image hiện có trong hệ thống |
| docker pull <tên image> | Download docker image từ registry hoặc hub xuống. |
| docker run <các cấu hình> <tên image> | Cho chạy một docker image và sẽ tạo container từ image này |
| docker start/stop/restart | Điều khiển vòng đời của container |
| docker push <tên image> | Đẩy một docker image lên registry hoặc hub |
| docker ps | Liệt kê các container đang chạy. Nếu muốn xem cả những container đã terminated, thì thêm flag -a (docker ps -a) |
| docker exec | Dùng để thực hiện command bên trong container từ CLI |
| docker inspect | Kiểm tra thông tin của các Docker objects |
| docker stack --compose-file <path to Docker compose file> <tên stack> | Tạo service stack, gần tương tự Docker Compose |
| docker build <options> <Dockerfile directory> | Dựng image từ Dockerfile |
| docker rm | Xóa container |
| docker rmi | Xóa image |

1. **Dockerfile**

Dockerfile là file text chứa các lệnh để xây dựng image với mỗi lệnh là một file layer được thêm vào image gốc.

Cấu trúc cơ bản của một Dockerfile

|  |  |
| --- | --- |
| FROM <image> | Bắt buộc. Nếu image chưa có trong máy thì docker sẽ tự động kéo về |
| COPY <source > <destination> | Copy directory hoặc file từ máy vào thư mục <destination> ở trong image |
| ADD <source> <destination> | Giống copy nhưng có thể thêm cả những remote resource |
| RUN <lệnh> | Lệnh được nêu ra ở RUN sẽ được chạy duy nhất một lần lúc build image, có thể có nhiều lệnh RUN khác nhau |
| CMD [“”,””,””,...] | Tạo lệnh mặc định hoặc tham số mặc định trong dockerfile. |
| ENTRYPOINT | Cấu hình container chạy như một executable |
| ENV <key><value> | Đặt các biến môi trường cho Dockerfile |

**Chú ý: Một số lệnh cần sử dụng absolute path.**

Ngoài ra, cần phân biệt giữa RUN vs. CMD vs. ENTRYPOINT

|  |  |  |
| --- | --- | --- |
| RUN | CMD | ENTRYPOINT |
| Cài đặt các ứng dụng và packages cần thiết | Đặt lệnh mặc định để chạy khi container khởi động (được gán sau ENTRYPOINT) | Lệnh đầu tiên để khởi chạy container |
| Thực thi lệnh trên file layer gần nhất của image. | Nếu container chạy mà không có lệnh nào được specify thì sẽ sử dụng lệnh mặc định, ngược lại thì lệnh mặc định được bỏ qua | CMD có thể là tham số bổ sung cho ENTRYPOINT |
| Chạy duy nhất một lần khi build image | Runtime operation | Runtime operation |
| Có thể có nhiều lệnh RUN trong một Dockerfile | Có thể có nhiều lệnh CMD trong một Dockerfile, tuy nhiên chỉ CMD cuối cùng được sử dụng. | Tương tự CMD |

1. **Docker swarm**

Là phương thức liên kết nhiều server chạy docker lại với nhau, thống nhất lại thành một Docker Host. Khi chạy docker ở swarm mode, các container được xử lý ở mức cao hơn là các service, có thể schedule, communicate và cấu hình phức tạp hơn bằng yaml file, quản lý tự động bởi Docker Swarm.

Các object của Docker Swarm bao gồm:

+ **Node:**

Mỗi docker machine được gọi là một node. Gồm 2 loại:

- Manager: Node chạy tiến trình quản lý các node khác và các service cần triển khai.

- Worker: Node chạy tiến trình thực thi và giám sát các container.

- Một docker machine có thể đồng thời là Manager và Worker.

+ **Service:**

Định nghĩa và cấu hình chương trình cần chạy, bao gồm các cấu hình khi chạy docker build (image, network, v.v..) và các cấu hình cho quá trình quản lý và schedule của Docker Swarm. Khái niệm hiểu gần tương tự như trong microservices.

Service gồm 2 loại:

+ Replicated service: Service triển khai gồm nhiều instance. Ví dụ: saga-service của BCCS gồm 3 instance để phục vụ người dùng.

+ Global service: Service phải có ở tất cả các node. Mỗi khi một node được thêm vào swarm, Manager node sẽ tạo mới Task thực thi Global service ở node đó.

+ **Task:** Là service được chuyển giao từ manager xuống worker để thực thi thành các container. Đây là đơn vị cho quá trình schedule của Docker Swarm.

Cổng mạng được yêu cầu để chạy Docker Swarm:

+ TCP port 2376: Dùng để chạy Docker machine, bảo mật giao tiếp của các Docker client

+ TCP port 2377: Giao tiếp giữa các node của Docker Swarm hay cluster, dùng để mở các node quản lý

+ TCP & UDP port 7946:giao tiếp giữa các node (container network discovery)

+ UDP port 4789: container ingress networking

Docker swarm share volume.

Volume được tạo trong docker swarm thông qua driver mặc định thì chỉ trong nội bộ node. Nếu cho 2 container trong cùng node đều mount từ 1 volume thì chúng sẽ share volume đó, tuy nhiên nếu các containers nằm trên các node khác nhau thì sẽ tách biệt volume với nhau.

Nếu muốn bind mount/volume giữa nhiều node khác nhau thì có thể thử các cách sau:

+ Cluster filesystem: glusterfs, ceph

+ Đổi sang dùng Kubernetes thay vì Docker swarm vì Kubernetes đã có sẵn cơ chế automated volume provisioning

(\* Provisioning: The process of setting up IT infrastructure. Can also refer to steps required to manage access to data and resources, and make them available to users and systems.)

1. **Docker-compose**

Nếu Dockerfile dùng để build image thì docker-compose dùng để build và run các containers. Docker-compose là tool dùng để định nghĩa và chạy nhiều docker services.

**8, Storage driver**

Storage drivers cho phép tạo data trong writable layer của container, tuy nhiên các file này sẽ không được persist sau khi container bị xóa, và tốc độ đọc viết đều thấp hơn native file system. Nếu muốn persist data và tăng hiệu năng thì nên dùng volume.

Container chiếm disk space vì những lí do sau:

+ Log files. Nếu container gen nhiều log và không được cấu hình log rotation thì đây sẽ là vấn đề không nhỏ.

+ Volume và bind mounts.

+ Configuration files của container (thường nhỏ).

+ Memory viết ra đĩa nếu có swapping.

+ Checkpoints.

Các folder trong /var/lib/docker là storage driver.

THAM KHẢO:

<https://medium.com/@BeNitinAgarwal/understanding-the-docker-internals-7ccb052ce9fe>

<https://www.youtube.com/watch?v=L1ie8negCjc>

<https://youtu.be/EnJ7qX9fkcU>

[https://www.Docker.com/resources/what-container](https://www.docker.com/resources/what-container)

<https://www.electronicdesign.com/technologies/dev-tools/article/21801722/whats-the-difference-between-containers-and-virtual-machines>

[https://goinbigdata.com/Docker-run-vs-cmd-vs-entrypoint/](https://goinbigdata.com/docker-run-vs-cmd-vs-entrypoint/)

[https://medium.com/@phamducquan/Docker-l%C3%A0-g%C3%AC-ki%E1%BA%BFn-th%E1%BB%A9c-c%C6%A1-b%E1%BA%A3n-v%E1%BB%81-Docker-13c6efc4aefe](https://medium.com/@phamducquan/docker-l%C3%A0-g%C3%AC-ki%E1%BA%BFn-th%E1%BB%A9c-c%C6%A1-b%E1%BA%A3n-v%E1%BB%81-docker-13c6efc4aefe)

[https://blog.container-solutions.com/understanding-volumes-Docker](https://blog.container-solutions.com/understanding-volumes-docker)

[https://docs.micrOSoft.com/en-us/dotnet/architecture/containerized-lifecycle/Docker-terminology](https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-terminology)

[https://docs.Docker.com/engine/swarm/admin\_guide/](https://docs.docker.com/engine/swarm/admin_guide/)

[https://docs.Docker.com/network/bridge/](https://docs.docker.com/network/bridge/)

(Overlay) <https://youtu.be/Xxhhdo2e-DA>

[http://www.dasblinkenlichten.com/Docker-networking-101/](http://www.dasblinkenlichten.com/docker-networking-101/)

<http://www.dasblinkenlichten.com/Docker-networking-101-user-defined-networks/>