VIỆN CÔNG NGHỆ THÔNG TIN T3H

Đào tạo chuyên sâu - Trải nghiệm thực tế

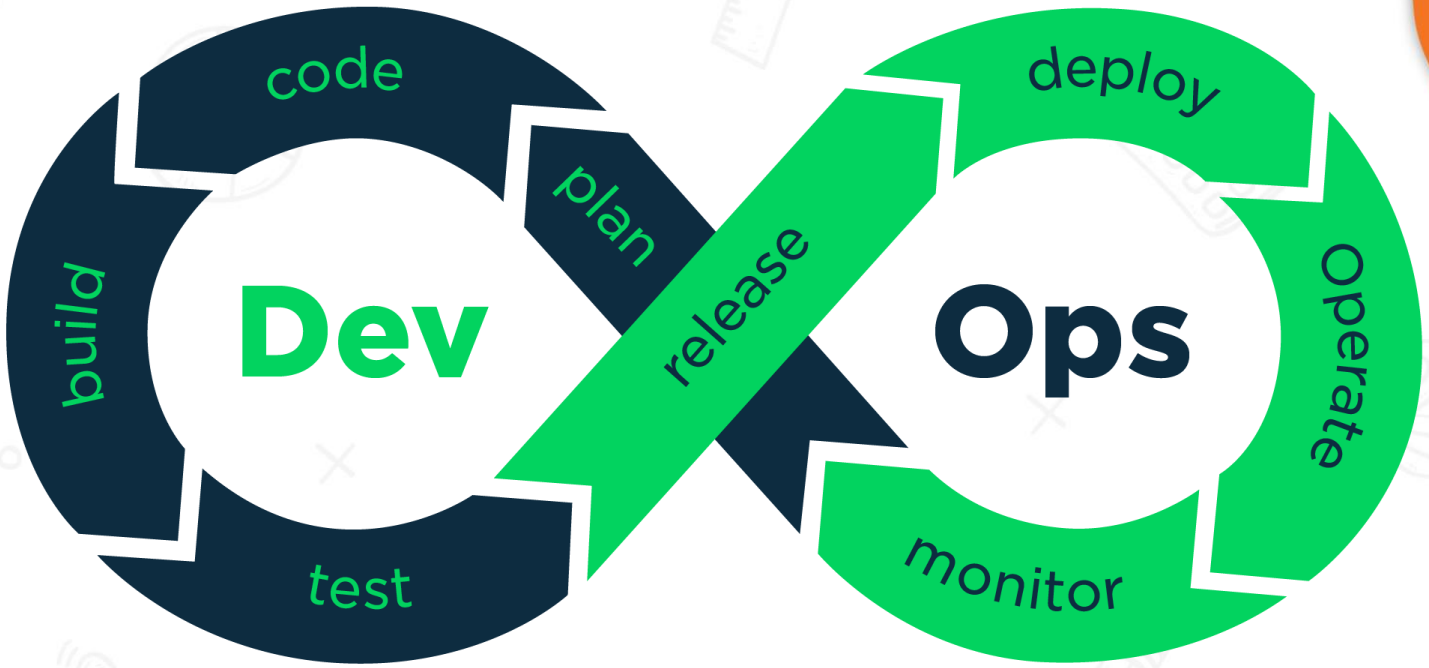
Môn học

DevOps

Giảng viên

Tan Do

0868880797



Nội dung buổi 01

1. Tổng quan về DevOps, Roadmap
2. Container
3. So sánh Container với VM
4. Docker
5. DockerHub

Tổng quan DevOps

1. Nguồn gốc ra đời DevOps
2. DevOps là gì?
3. Các công việc của DevOps?
4. Các kiến thức cơ bản của DevOps Engineer
5. Roadmap
6. Trở thành DevOps Engineer dễ hay khó?
7. Cơ hội nghề nghiệp

Nguồn gốc ra đời DevOps

Mô hình truyền thống

1. Team Developer
   - Phát triển mã nguồn và các tính năng mới
   - Kiểm thử và debug
   - Xây dựng và bảo trì
   - Tối ưu hóa hiệu suất và khả năng mở rộng
   - Tương tác với khách hàng để triển khai yêu cầu mới
2. Team Operation (System Admin)
   - Cài đặt, triển khai và duy trì hệ thống
   - Theo dõi và giám sát hiệu suất
   - Xử lý sự cố và vấn đề kỹ thuật
   - Thực hiện sao lưu và phục hồi dữ liệu
   - Đảm bảo bảo mật và tuân thủ các quy định về an toàn thông tin

Nguồn gốc ra đời DevOps

Các vấn đề thường gặp

1. Sự chậm chạp trong việc chuyển giao sản phẩm từ nhóm phát triển sang nhóm vận hành có thể làm giảm tốc độ triển khai
2. Các môi trường phát triển, thử nghiệm và sản xuất thường không đồng nhất, dẫn đến sự không chắc chắn khi chuyển giao sản phẩm từ môi trường này sang môi trường khác
3. Sự tách biệt giữa các nhóm có thể làm giảm khả năng phát hiện lỗi và xử lý sự cố một cách nhanh chóng

Mục đích sự ra đời DevOps

1. Xóa bỏ rào cản
2. Tăng cường hợp tác

DevOps là gì?

Sự kết hợp của:

- ✓các nguyên lý
- ✓triết lý
- ✓văn hóa
- ✓thực hành
- ✓quy trình
- ✓các công cụ giúp tự động hóa quá trình lập trình và chuyển giao phần mềm

giúp tăng khả năng phân phối ứng dụng và dịch vụ của một tổ chức ở tốc độ cao

Các công việc của DevOps

- **Xây dựng và duy trì hệ thống tự động hóa:** Tạo và duy trì các công cụ và quy trình tự động hóa để tối ưu hóa quá trình phát triển, kiểm thử và triển khai phần mềm
- **Triển khai liên tục (Continuous Deployment):** Thực hiện việc triển khai tự động của ứng dụng và cập nhật hạ tầng hệ thống một cách liên tục và đáng tin cậy
- **Tích hợp liên tục (Continuous Integration):** Đảm bảo rằng mã nguồn được tích hợp và kiểm thử tự động một cách liên tục, giúp phát hiện lỗi sớm và đảm bảo chất lượng mã nguồn

Các công việc của DevOps

- **Quản lý cấu hình và môi trường:** Quản lý cấu hình hệ thống và môi trường phát triển, thử nghiệm và sản xuất để đảm bảo sự nhất quán và tin cậy trong quá trình triển khai ứng dụng
- **Giám sát và phản hồi tự động:** Thiết lập và duy trì hệ thống giám sát tự động để theo dõi hiệu suất và sự hoạt động của hệ thống, và phản hồi tự động khi phát hiện sự cố
- **Hỗ trợ và hợp tác với các nhóm khác:** Làm việc chặt chẽ với các nhóm phát triển, vận hành và bảo mật để đảm bảo sự hợp tác và hiệu suất làm việc

Các kiến thức cơ bản của DevOps Engineer

- ✓**Hệ điều hành và mạng:** Hiểu biết về hệ điều hành như Linux và Windows, cùng với kiến thức về mạng máy tính, giao thức mạng (như TCP/IP), và bảo mật mạng là rất quan trọng để quản lý hạ tầng
- ✓**Công cụ và công nghệ DevOps:** Nắm vững các công cụ và công nghệ DevOps như Docker (containment), Kubernetes (orchestration), Jenkins (CI/CD), Ansible (cấu hình tự động), Terraform (Infrastructure as Code), Git (quản lý mã nguồn), các công cụ giám sát và ghi nhật ký như Prometheus và ELK Stack
- ✓**Quy trình DevOps:** Hiểu biết về các quy trình và phương pháp DevOps như Continuous Integration (CI), Continuous Deployment (CD), Infrastructure as Code (IaC), và các phương pháp và công cụ giám sát hệ thống
- ✓**Kiểm thử phần mềm:** Hiểu biết về các phương pháp kiểm thử phần mềm, bao gồm kiểm thử tự động, kiểm thử tích hợp và kiểm thử hệ thống, cũng như các công cụ kiểm thử phần mềm như Selenium

Các kiến thức cơ bản của DevOps Engineer

- ✓**Quản lý hạ tầng và dịch vụ Cloud:** Nắm vững kiến thức về các dịch vụ cloud phổ biến như IaaS, PaaS, SaaS. Hiểu biết về các dịch vụ cụ thể của các nhà cung cấp cloud hàng đầu như AWS, Azure, GCP
- ✓**Kỹ năng lập trình:** Hiểu biết về lập trình và kỹ năng viết script là rất hữu ích để tự động hóa các tác vụ và quy trình trong môi trường DevOps. Các ngôn ngữ lập trình như Python, Shell scripting, và Ruby thường được sử dụng
- ✓**Kỹ năng giao tiếp và làm việc nhóm:** Kỹ năng giao tiếp hiệu quả với các thành viên khác trong nhóm DevOps và các bộ phận khác trong tổ chức là rất quan trọng, cũng như khả năng làm việc nhóm và giải quyết vấn đề
- ✓**Tinh thần tự học và sẵn lòng học hỏi mới:** DevOps là một lĩnh vực liên tục thay đổi, vì vậy kỹ sư DevOps cần có tinh thần tự học và sẵn lòng nắm bắt và áp dụng các công nghệ và công cụ mới

Roadmap DevOps

https://roadmap.sh/devops

Trở thành DevOps Engineer dễ hay khó?

- Dễ
- Khó

Cơ hội nghề nghiệp

- Vị trí quan trọng trong tổ chức
- Kiến thức bền vững
- Càng có thâm niên càng được săn đón
- Lương rất cao

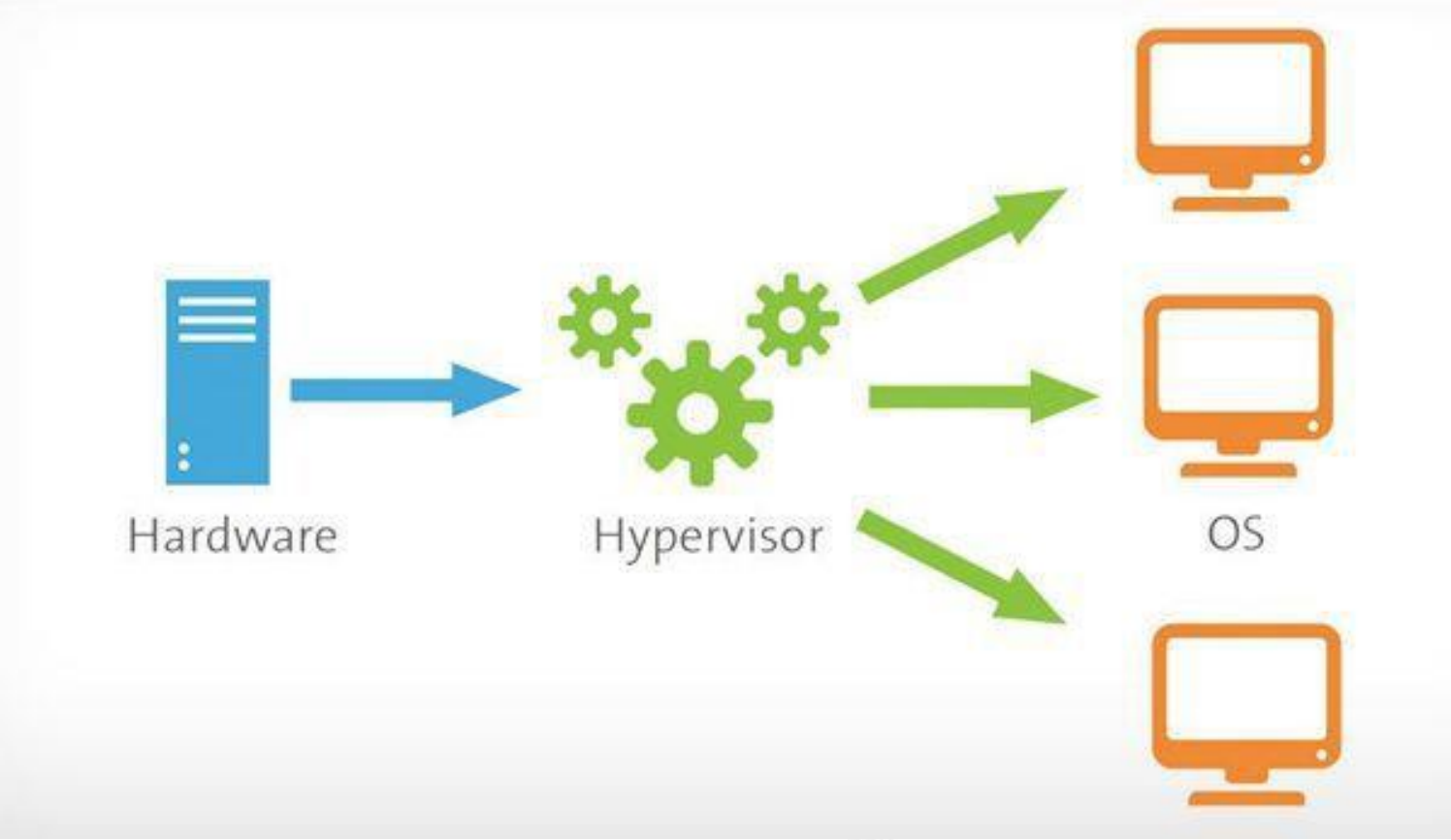
Virtual Machine là gì?

- Là một máy ảo
- Sử dụng các tài nguyên trên máy tính vật lý để khởi chạy chương trình và các ứng dụng
- Chạy hệ điều hành riêng
- Hoạt động độc lập với các máy ảo khác
- Độc lập với máy tính vật lý tạo ra nó

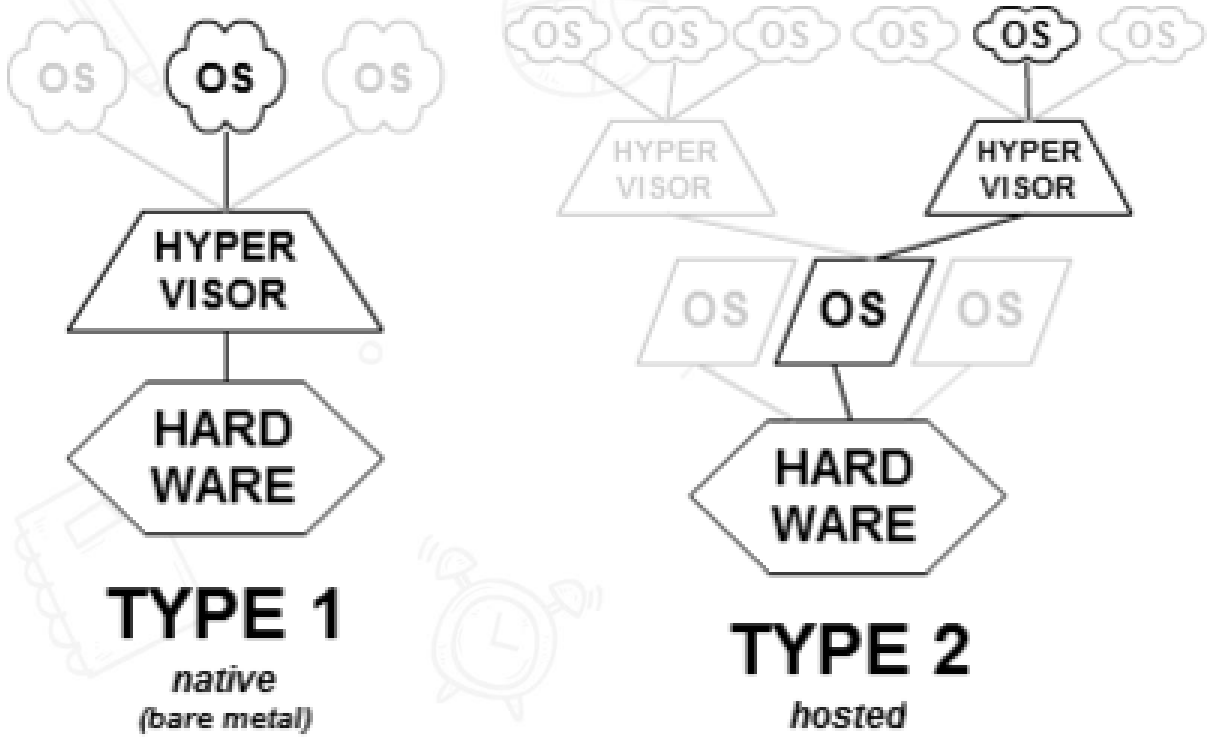
Cách hoạt động Virtual Machine

- Dựa vào công nghệ ảo hóa
- Một máy tính vật lý có thể chia sẻ hệ thống và tài nguyên cho nhiều Virtual Machine khác nhau
- Phần mềm giám sát máy ảo (Hypervisor) có nhiệm vụ quản lý phần cứng và tách máy ảo khỏi các phần cứng
- Yêu cầu của người dùng liên quan đến việc lấy thêm tài nguyên từ môi trường máy tính vật lý đều sẽ được Hypervisor phiên dịch và gửi đến máy chủ
- Có 2 loại Hypervisor: Native và Hosted
- Hypervisor phổ biến: MS Hyper-V, Apple Boot Camp,… VMware, Oracle VirtualBox,…

Cách hoạt động Virtual Machine



Cách hoạt động Virtual Machine



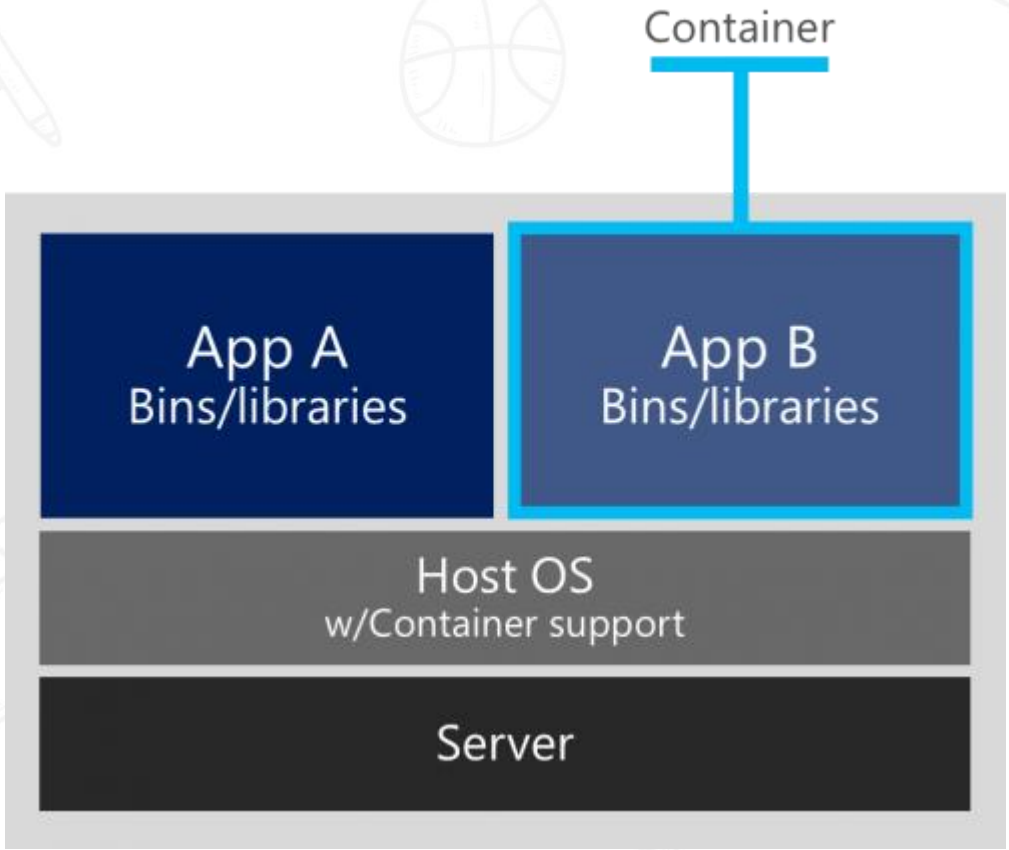
Container là gì

- Công nghệ Container, hay Container là một phương pháp ảo hóa cấp cao được sử dụng để đóng gói, triển khai và quản lý các ứng dụng và tài nguyên của chúng trong một môi trường tách biệt, độc lập với các chương trình khác
- Container chứa tất cả những gì cần thiết để chạy một ứng dụng cụ thể, bao gồm mã nguồn (source code), thư viện hệ thống (system libraries), thời gian chạy (runtime), tệp cấu hình (configuration files) và các phụ thuộc khác

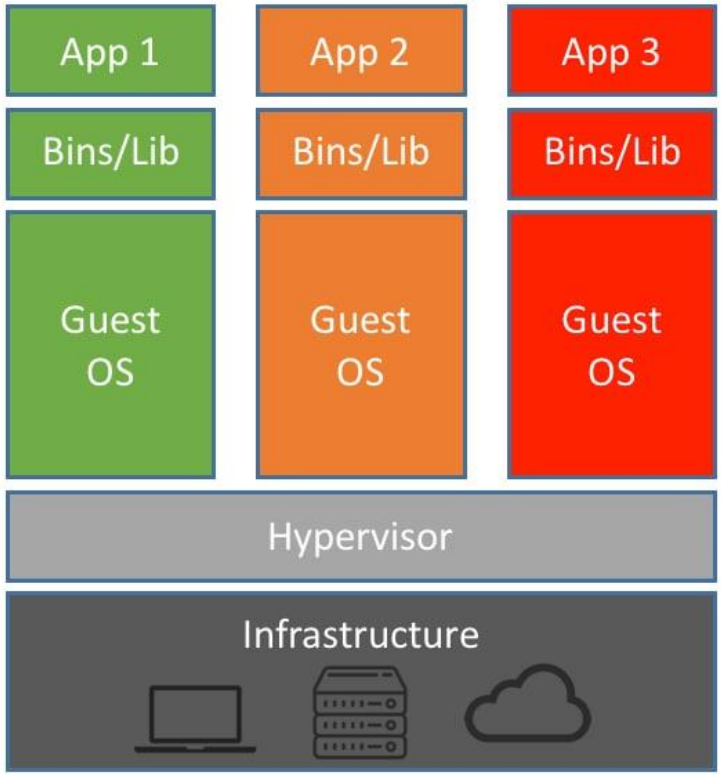
Kiến trúc Container

- Các thành phần chính:
  - Server (máy chủ vật lý hoặc máy ảo
  - Host OS (hệ điều hành cài đặt trên server)
  - Các container
- Mỗi ứng dụng (App) sẽ có những sự phụ thuộc riêng của nó bao gồm cả về phần mềm (các dịch vụ hay thư viện) lẫn cả về phần cứng (CPU, bộ nhớ, lưu trữ)
- Các ứng dụng sẽ được Container Engine (công cụ ảo hóa tinh gọn) được cài đặt trên Host OS cô lập sự phụ thuộc của các ứng dụng khác nhau bằng cách đóng gói thành các container
- Các tiến trình (process) trong một container bị cô lập với các tiến trình của các container khác trong cùng hệ thống tuy nhiên tất cả các container này đều chia sẻ kernel của Host OS

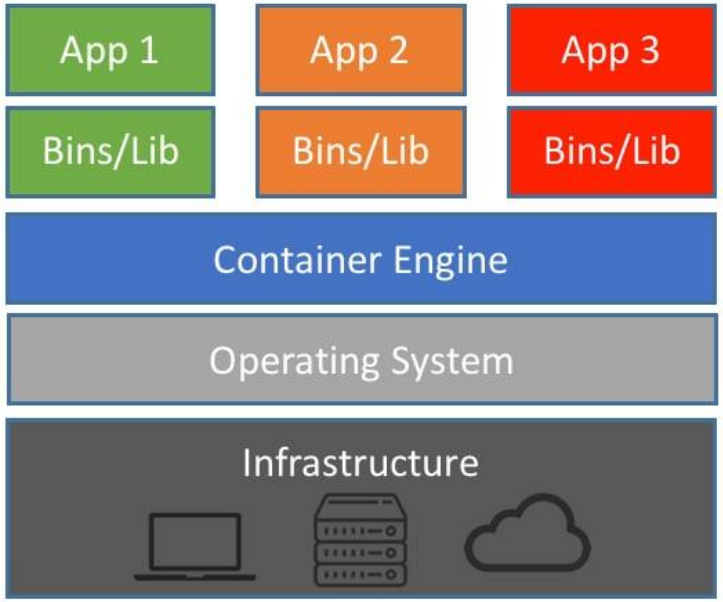
Kiến trúc Container



Virtual Machine và Container



Machine Virtualization



Containers

Lợi ích của Container

Kích thước nhỏ nhẹ: Các Container chia sẻ Kernel của máy chủ và chỉ chứa các thành phần thực sự cần thiết của hệ điều hành và các thư viện. Đồng thời, các Container thường có kích thước rất nhỏ do chúng bị giới hạn ở một chức năng duy nhất

Triển khai nhanh: Do kích thước nhỏ, các Container có thể được khởi động trong vòng vài giây, khiến chúng trở nên lý tưởng cho các ứng dụng cần liên tục di chuyển lên và xuống

CI/CD: Các Container được thiết kế để khởi động thường xuyên, vì vậy chúng có thể dễ dàng tiếp nhận các thay đổi

Tính di động: Thiết kế độc lập làm cho việc di chuyển Container giữa các máy tương đối dễ dàng miễn là đúng vị trí Kernel

Tính nhất quán: Giải pháp đóng gói ứng dụng gồm các library, source code, framework,... duy nhất 1 lần và chạy ở bất kỳ đâu

Thách thức khi sử dụng Container

Bảo mật: Mặc dù có môi trường cô lập, nhưng Container vẫn cần đảm bảo tính bảo mật, đặc biệt trong trường hợp người dùng không cẩn thận có thể tạo ra các hình ảnh Container không an toàn

Yêu cầu kết nối mạng phức tạp: Các chức năng thường được chia thành nhiều Container và cần giao tiếp với nhau. Việc có nhiều container cần liên lạc với nhau có thể phức tạp. Các hệ thống điều phối, chẳng hạn như Kubernetes, có các nhóm đa Container, giúp việc trao đổi dễ dàng hơn một chút nhưng có thể phức tạp hơn so với việc sử dụng máy ảo

Có thể đòi hỏi nhiều tác vụ hơn máy ảo: Khi sử dụng Container, sẽ cần chia ứng dụng thành các dịch vụ thành phần khác nhau do vậy sẽ khiến việc dùng Container có thể phát sinh nhiều việc hơn dùng máy áo.

So sánh Container với VM

- Tính độc lập:
  - Containers chia sẻ hạt nhân (kernel) với hệ điều hành chủ, nhưng có môi trường cô lập với các tiến trình và tài nguyên. Chúng sử dụng cơ chế hạt nhân chia sẻ và phân chia tài nguyên để đảm bảo tính cô lập giữa các Container
  - Mỗi máy ảo (VM) có hệ điều hành riêng và không chia sẻ hạt nhân với hệ điều hành chủ. Điều này đảm bảo cô lập tương đối cao giữa các máy ảo

So sánh Container với VM

- Hệ điều hành:
  - Các Container chia sẻ nhân hệ điều hành của máy chủ, yêu cầu các ứng dụng phải tương thích với hệ điều hành chủ. Hạn chế này làm cho các Container phù hợp hơn với môi trường ứng dụng đồng nhất. Nếu hầu hết các ứng dụng đều có cùng yêu cầu về hệ điều hành thì việc Container hóa sẽ là một giải pháp thiết thực hơn nhiều
  - Ảo hóa cho phép quản lý nhiều phiên bản của các hệ điều hành khác nhau, giúp nó phù hợp để chạy các ứng dụng khác nhau. Do đó, máy ảo là lựa chọn tốt nhất khi doanh nghiệp chạy nhiều ứng dụng mà mỗi ứng dụng yêu cầu hệ điều hành chuyên dụng riêng

So sánh Container với VM

- Tốc độ
  - Container tận dụng nhân hệ điều hành của máy chủ, cho phép khởi động nhanh hơn và sử dụng tài nguyên hiệu quả hơn. Khi nói đến tốc độ, Container có ưu thế hơn hẳn vì chúng được thiết kế để giảm tải thời gian chạy cho các ứng dụng phần mềm một cách đáng kể
  - Ảo hóa bao gồm việc mô phỏng phần cứng và khởi động hệ điều hành, điều này có thể dẫn đến thời gian khởi động và phân bổ tài nguyên chậm hơn

So sánh Container với VM

- Tính di động:
  - Containers đóng gói ứng dụng cùng, tạo sự đồng nhất và tiện lợi cho việc di động và triển khai. Chúng thường nhẹ nhàng hơn do không cần chạy nhiều hệ điều hành riêng nhau
  - Công nghệ máy ảo đóng gói toàn bộ hệ điều hành và ứng dụng, tạo ra môi trường tự đủ. Tích hợp và di động không được linh hoạt như trong trường hợp Container

So sánh Container với VM

- Tiết kiệm tài nguyên
  - Containers sử dụng chia sẻ tài nguyên và cơ chế cô lập để tiết kiệm tài nguyên hơn so với ảo hóa, cho phép nhiều Container chia sẻ cùng một hạt nhân
  - Mỗi máy ảo sử dụng tài nguyên riêng biệt, làm tăng tiêu thụ tài nguyên trên một hệ thống.

So sánh Container với VM

- Dễ dàng quản lý
  - Containers thường dễ dàng quản lý hơn qua các công cụ như Docker và Kubernetes. Công nghệ Container phù hợp với quy mô lớn và việc tích hợp liên kết mạnh mẽ
  - Ảo hóa có thể phức tạp hơn trong việc quản lý, đặc biệt đối với số lượng lớn các máy ảo. Tích hợp liên kết cũng có thể đòi hỏi công việc cấu hình và quản lý phức tạp hơn

Docker là gì?

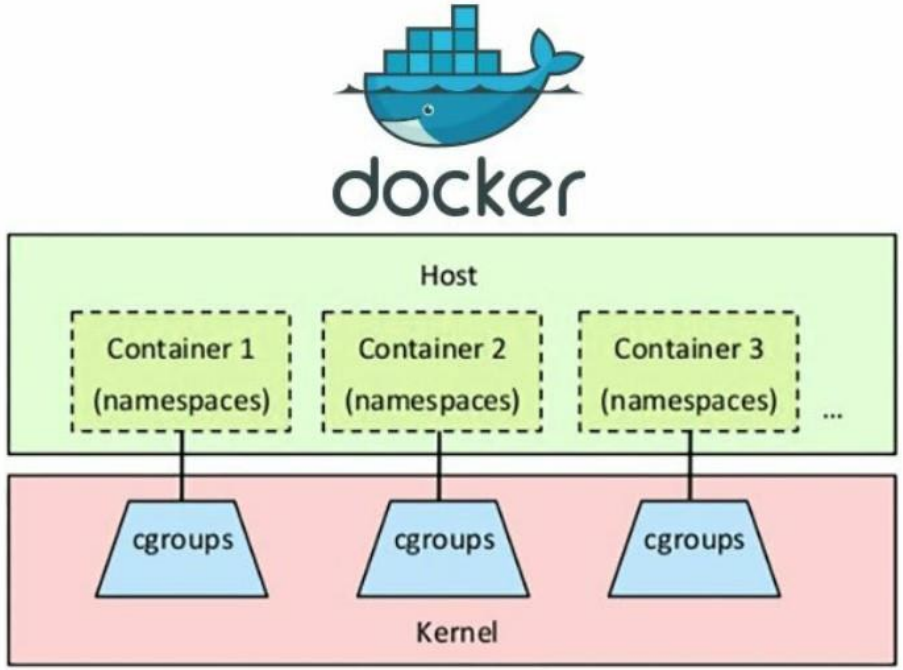
- Là nền tảng phần mềm cho phép đóng gói, triển khai và chạy ứng dụng một cách nhanh chóng
- Đóng gói phần mềm vào các đơn vị tiêu chuẩn hóa được gọi là container có mọi thứ mà phần mềm cần để chạy, trong đó có thư viện, công cụ hệ thống, mã và thời gian chạy
- Giúp nhanh chóng triển khai và thay đổi quy mô ứng dụng vào bất kỳ môi trường nào và biết chắc rằng mã sẽ chạy được

Cách thức hoạt động Docker

- Docker hoạt động bằng cách cung cấp phương thức tiêu chuẩn để chạy mã
- Docker là hệ điều hành dành cho container
- Cũng tương tự như cách máy ảo ảo hóa (loại bỏ nhu cầu quản lý trực tiếp) phần cứng máy chủ, các container sẽ ảo hóa hệ điều hành của máy chủ
- Docker được cài đặt trên từng máy chủ và cung cấp các lệnh đơn giản mà bạn có thể sử dụng để dựng, khởi động hoặc dừng container

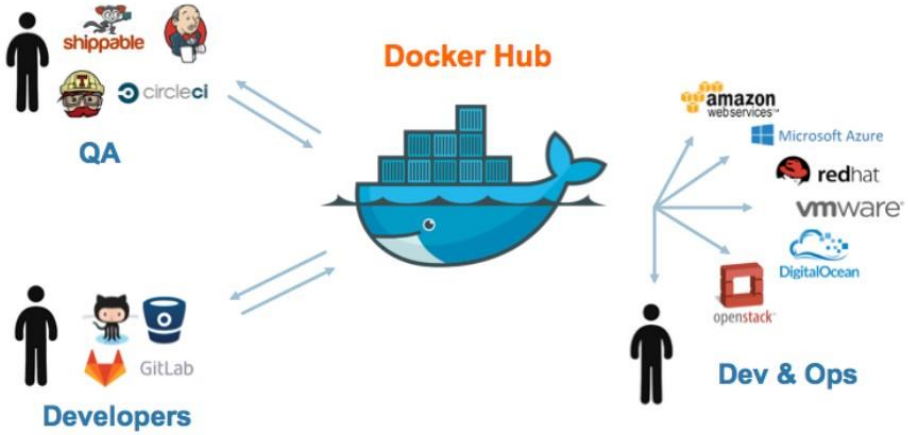
Các khái niệm cơ bản Docker

- Docker Engine
  - Là thành phần chính của Docker
  - Dùng để đóng gói ứng dụng



Các khái niệm cơ bản Docker

- Docker Hub
  - Là một “github for docker images”
  - Hàng nghìn public images được tạo ra và chia sẻ bởi cộng đồng
  - Dễ dàng tìm ra được những image theo nhu cầu

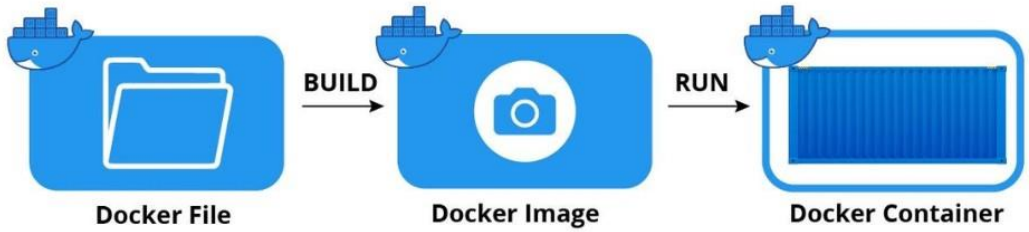


Các khái niệm cơ bản Docker

- Image
  - Chỉ một khuôn mẫu để tạo ra một cấu trúc container
  - Thông thường, image sẽ dựa trên 1 khuôn mẫu đã có sẵn với một số tùy chỉnh thêm
  - Có thể tự build riêng một image hoặc cũng có thể sử dụng những image được đăng công khai trên cộng đồng Docker Hub
  - Một image thường sẽ được build dựa theo những chỉ dẫn trên Dockerfile

Các khái niệm cơ bản Docker

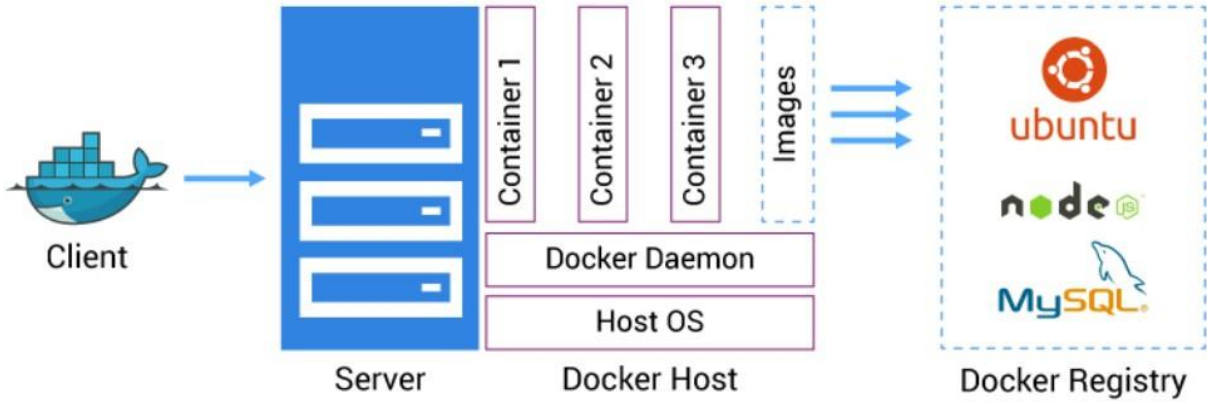
- Container
  - Là một instance của image
  - Có thể khởi tạo, bắt đầu, tạm dừng, di chuyển hoặc xóa container đi dựa trên giao diện lập trình Docker API hoặc Docker CLI



Các khái niệm cơ bản Docker

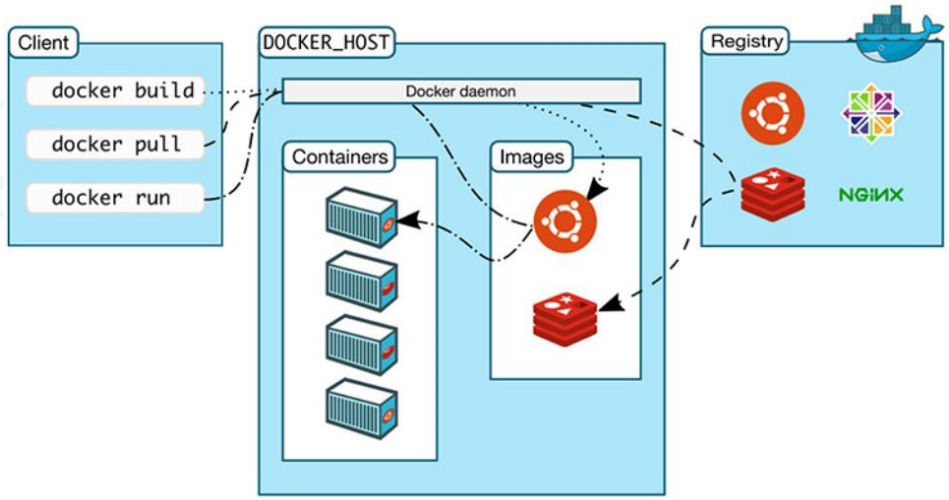
- Docker Client

  Là công cụ dành cho người dùng có thể giao tiếp được với Docker host



Các khái niệm cơ bản Docker

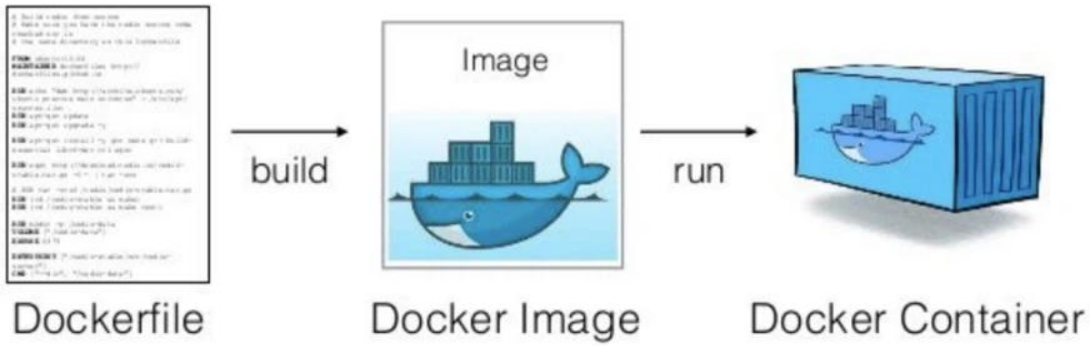
- Docker Daemon
  - Lắng nghe những yêu cầu từ trên Docker Client, từ đó sẽ quản lý được những đối tượng như là cấu trúc Container, Image, Network hay là Volumes thông qua tiêu chuẩn REST API
  - Các Docker Daemon cũng có thể giao tiếp với nhau để dễ dàng quản lý tất cả các Docker Service



Các khái niệm cơ bản Docker

- Dockerfile

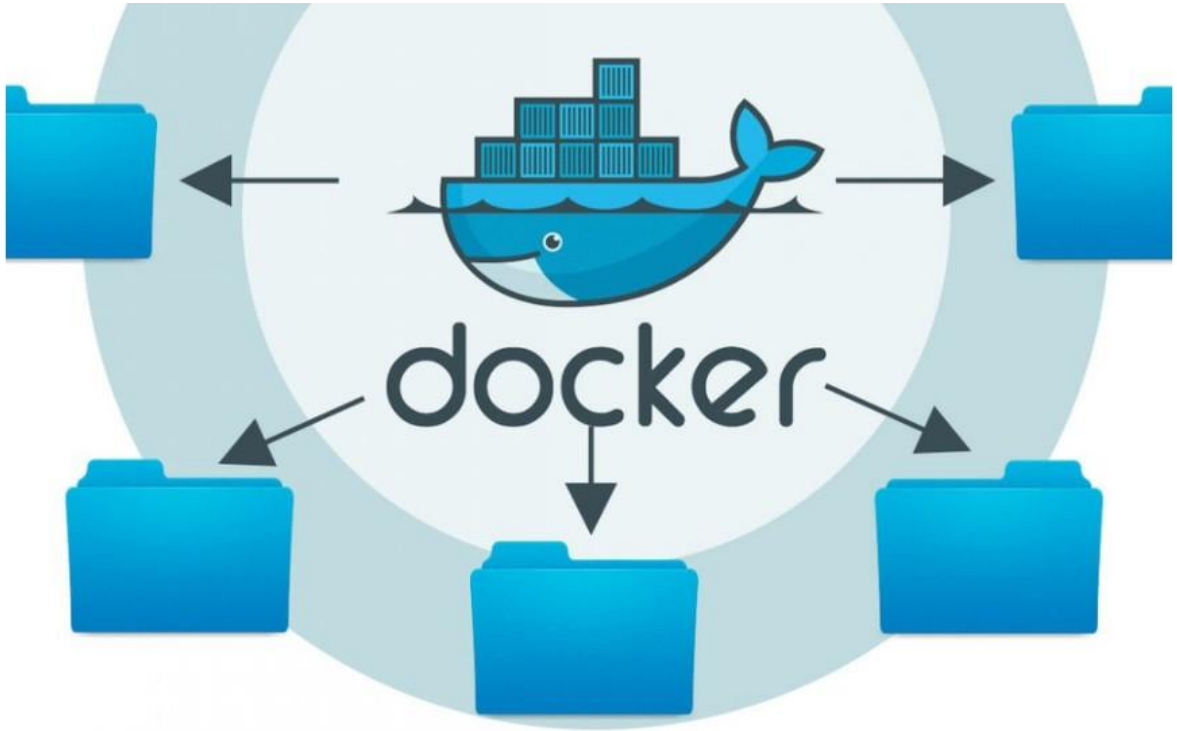
  Là một file tổng hợp bao gồm tất cả những chỉ dẫn chi tiết để xây dựng lên một image



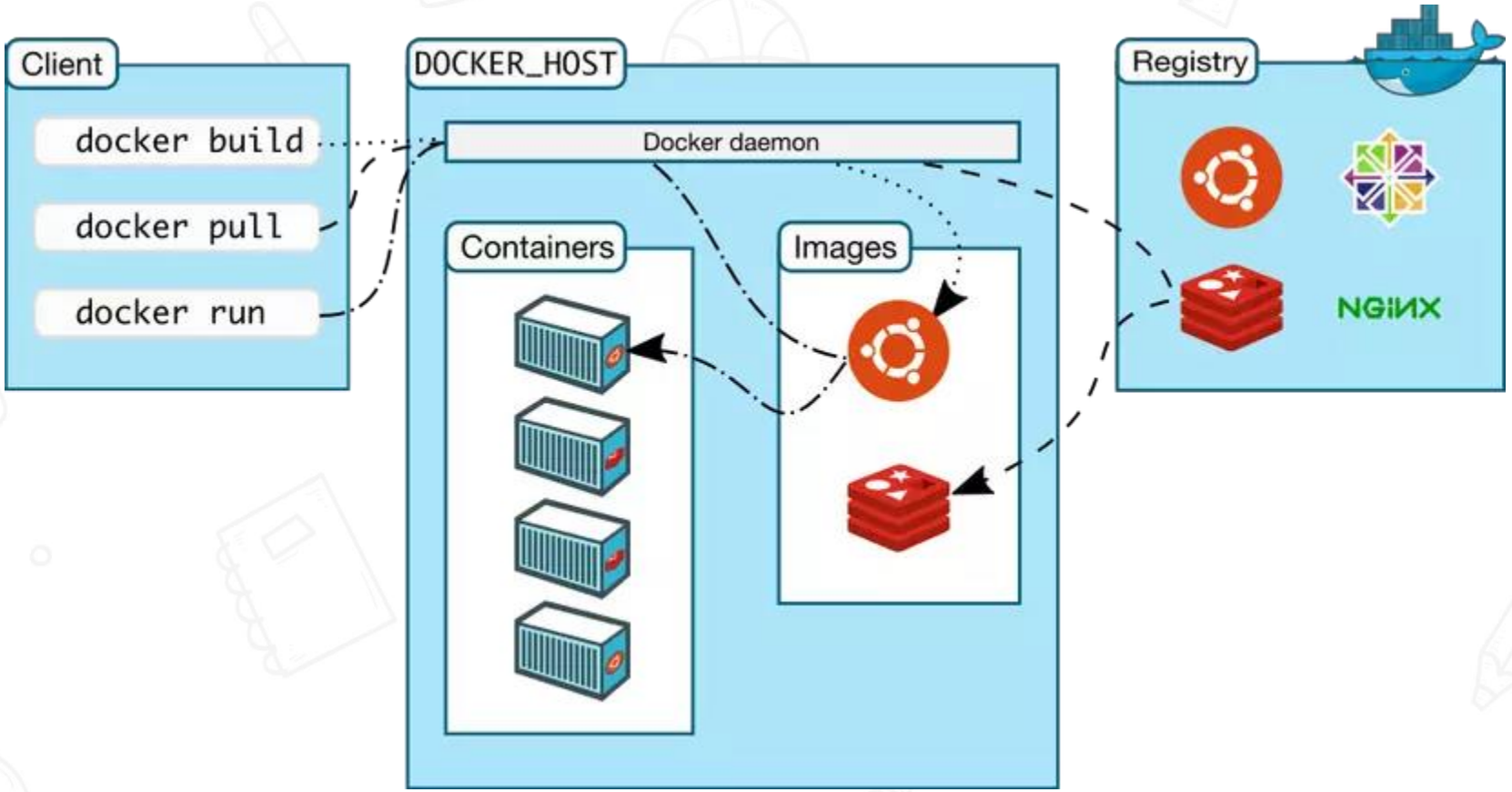
Các khái niệm cơ bản Docker

- Volumes

  Là gói dữ liệu được tạo ra khi cấu trúc container được tạo ra



Kiến trúc của Docker



Cài đặt và sử dụng Docker

- Docker Desktop
  - https://www.docker.com/products/docker-desktop/
- Git Bash
  - https://www.git-scm.com/downloads
- docker info
- docker --version
- docker images
- docker ps –a # -a: all
- docker run hello-world

Các lỗi có thể gặp Docker

- WSL (Windows Subsystem for Linux) chưa cài hoặc out of date
  - Với Windows 10 v2024 hoặc Windows 11
    - wsl --install
    - wsl --update
    - https://learn.microsoft.com/en-us/windows/wsl/install
  - Với Windows phiên bản cũ hơn
    - https://learn.microsoft.com/en-us/windows/wsl/install-manual
  - https://github.com/microsoft/WSL/issues/9157

Các lệnh cơ bản Docker

- List image/container
  - docker image/container ls # ls: list
- Delete image/container
  - docker image/container rm <tên image/container> # rm: remove
- Delete all image
  - docker image rm $(docker images –a –q) # -a: all, -q: quiet
- List all container
  - docker ps –a # ps: process status

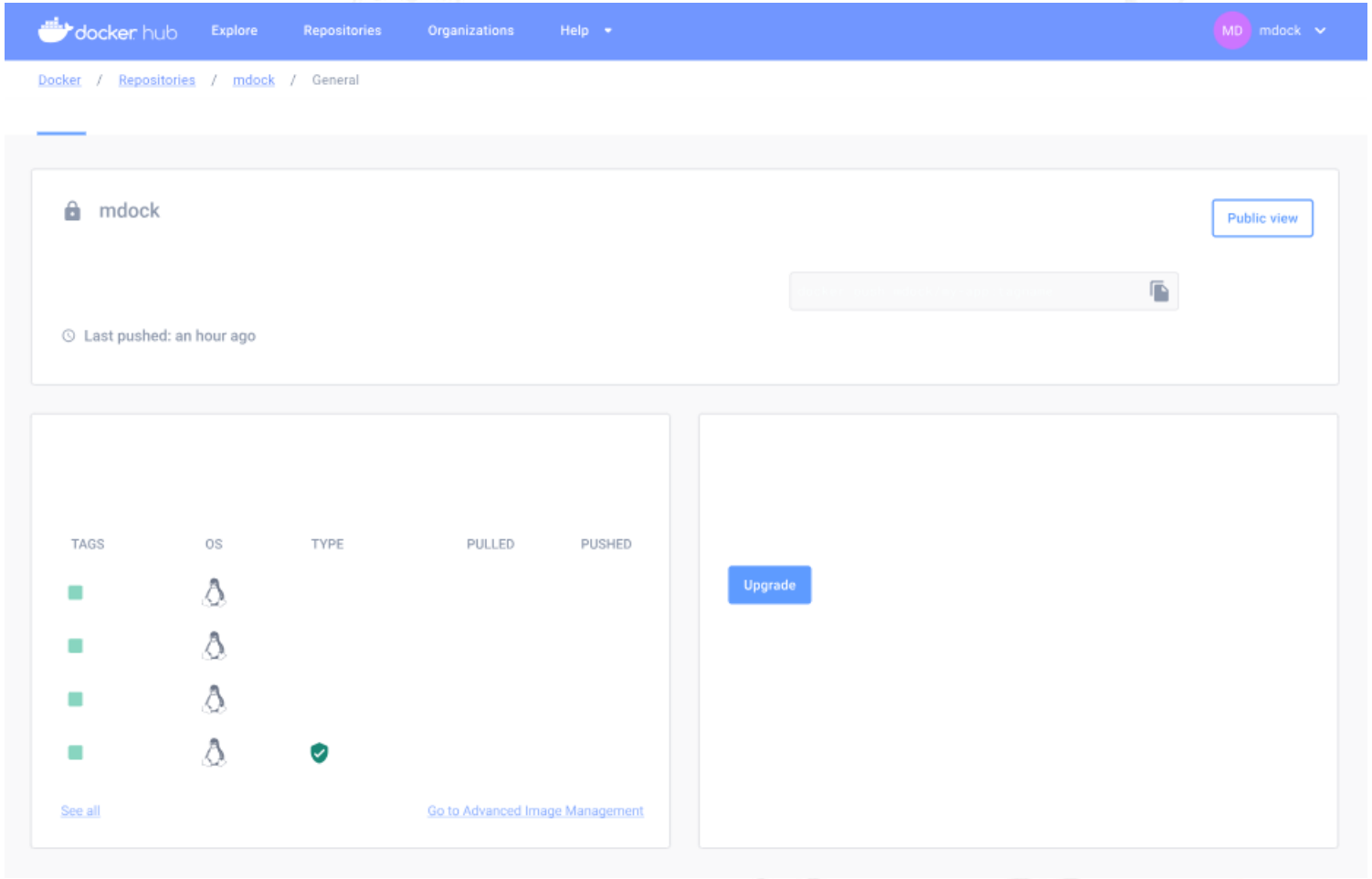
Các lệnh cơ bản Docker

- Start container:
  - docker start <tên container>
- Stop a container
  - docker stop <tên container>
- Run container từ image và thay đổi tên container:
  - docker run --name <tên container> <tên image>
- Stop all container
  - docker stop $(docker ps –a –q)# -a: all, -q: quiet
- Delete all container/image
  - docker rm $(docker ps -a -q)
  - docker image rm $(docker images –a –q)

Các lệnh cơ bản Docker

- Build image từ container
  - docker build -t <tên container> . # -t: tag
  - docker build -t <tên container>:<tên tag> .
- Tạo container chạy ngầm
  - docker run -d <tên image> # -d: detach
- Show log a container
  - docker logs <tên container>
  - docker logs --follow <tên container>
- Pull image docker hub
  - docker pull <tên image>

Tạo tài khoản, quản lý image trên DockerHub



Các lệnh cơ bản Docker Hub

- docker login --username username
- docker tag my-image username/my-repo
- docker push username/my-repo

THỰC HÀNH 1

- Cài đặt Docker Desktop
- Kiểm tra phiên bản docker
- Kiểm tra số lượng image và container hiện có
- Kéo image nginx mới nhất từ Docker Hub về
- Run 2 server nginx trên 2 cổng 81 và 82 với tên web1 và web2
- Kiểm tra số lượng image và container hiện có
- Stop server web1, web2
- Xóa container web2
- Xóa toàn bộ container/image

THỰC HÀNH 2

- Kiểm tra số lượng image và container hiện có
- Kéo image nginx/apache/iis mới nhất từ Docker Hub về
- Run 3 server trên ứng 3 cổng 81, 82, 82 với tên w1, w2 và w3
- Kiểm tra số lượng image và container hiện có
- Stop server w1, w2
- Start server w1
- Xóa toàn bộ container/image

T3H
IT-Institute

VIỆN CÔNG NGHỆ THÔNG TIN T3H

Đào tạo chuyên sâu - Trải nghiệm thực tế

THANK YOU

www.t3h.edu.vn