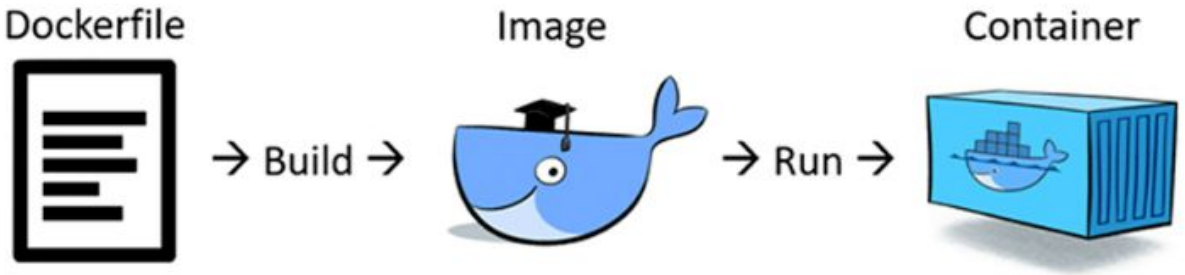
Docker



1 - Docker Installation?

Window:

- Install Docker Desktop on Windows
- [Video] - Docker Tutorial for Beginners - Install Docker on Windows 10

Mac:

- Install Docker Desktop on Mac
- [Video] - Docker Tutorial for Beginners - Install Docker Desktop on Mac

Linux

- Install Docker on Linux
- [Video] - Docker Installation On Ubuntu? | Linux

2 - What is container?

Một số lý do sử dụng docker:

- **Consistency**: Docker cho phép các nhà phát triển đóng gói ứng dụng và tất cả các phụ thuộc, thư viện và cấu hình vào một đơn vị duy nhất gọi là container.

- **Portability**: Container Docker có trọng lượng nhẹ và có thể chạy trên bất kỳ nền tảng nào hỗ trợ Docker, bất kể cơ sở hạ tầng cơ bản.

- **Resource Efficiency**: So với các máy ảo truyền thống, các container Docker chia sẻ kernel hệ điều hành máy chủ, giúp tiết kiệm tài nguyên hệ thống

- **Isolation**: Các container cung cấp “process-level isolation”, có nghĩa là mỗi container chạy trong môi trường cô lập riêng của nó

2 - What is container?

Một số lý do sử dụng docker:

- **Scalability**: Docker giúp dễ dàng mở rộng ứng dụng. Vì các container nhẹ nên có thể nhanh chóng mở rộng hoặc thu hẹp ứng dụng dựa theo nhu cầu.

- **Version Control and Rollbacks**: Docker images, là "các bản thiết kế" (blueprints) cho các container, có thể kiểm soát các version. Điều này cho phép dễ dàng phục hồi về version trước của một ứng dụng nếu gặp bất kỳ vấn đề nào trong quá trình triển khai hoặc cập nhật.

- **Continuous Integration and Continuous Deployment (CI/CD)**: Docker đóng vai trò quan trọng trong quy trình phát triển phần mềm hiện nay. Nó cho phép các nhà phát triển xây dựng, kiểm tra và triển khai ứng dụng một cách nhanh chóng và nhất quán.

2 - What is container?

Một số lý do sử dụng docker:

- **Dependency Management**: Docker đơn giản hóa việc quản lý các “requirements” của ứng dụng.

- **Microservices Architecture**: Docker thường được sử dụng để triển khai các ứng dụng theo kiến trúc microservices. Mỗi microservices có thể được container hóa một cách độc lập, thúc đẩy tính mô đun, dễ bảo trì và dễ cập nhật

- **Community and Ecosystem**: Docker có một cộng đồng lớn và hoạt động tích cực, đóng góp vào một hệ sinh thái phong phú đa dạng Docker images đã được xây dựng trước, giúp dễ dàng cài đặt và sử dụng các phần mềm và dịch vụ khác nhau trong các container.

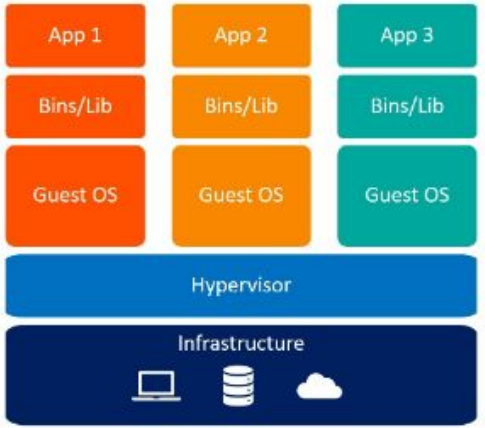
2 - What is container?

Trước khi có Docker?

- Virtual Machines (VM): Máy ảo chạy một phiên bản của một số Hệ điều hành và phần cứng của máy chủ được ảo hóa bởi Bare-metal hypervisor (một phần mềm chịu trách nhiệm phân bổ tài nguyên cho các máy ảo).

Máy ảo hiện không còn được sử dụng thịnh hành:

- Tốn thời gian khởi tạo và khởi động máy ảo
- Hầu hết các OS đều tính bằng GBs
- Cần phải được cài đặt từ đầu
- Không thể chia sẻ VM instance
- Lãng phí tài nguyên phần cứng



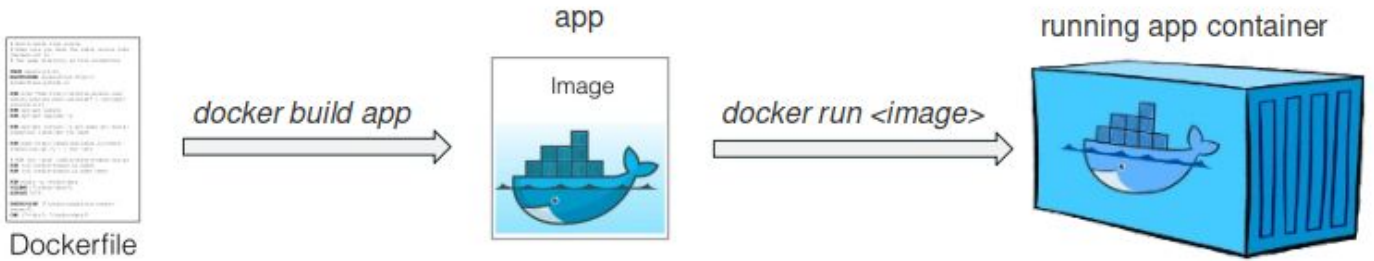
Virtual Machines

2 - What is container?

Docker cơ bản là một phần mềm giúp nhà phát triển tạo và quản lý containers.

Một số thành phần của docker

- **Dockerfile**: Docker file là một file cơ bản hướng dẫn docker cách xây dựng một docker image.
- **Docker image**: Chứa thông tin về phần mềm mà container sẽ chạy và cho biết flow chạy như thế nào. Docker image được build từ Dockerfile.
- **Docker container**: Docker container lấy thông số từ Docker image và thực thi chúng để tạo môi trường cho ứng dụng.



3 - Dockerfile: Instructions and Examples

FROM NAME[:TAG|@DIGEST]

From - cho docker engine biết image nào sẽ được sử dụng
Ví dụ:

- FROM ubuntu:latest
- FROM redis:community-7.0.0-beta

3 - Dockerfile: Instructions and Examples

- **RUN** <COMMAND> (shell form)
- **RUN** ["EXECUTABLE", "PARAM1", "PARAM2"] (exec form)

RUN - thực thi một lệnh docker container's shell. Mặc định là /bin/sh.
Ví dụ:

- RUN apt update
- RUN ["npm", "install"]

Lưu ý: chỉ sử dụng một trong 2 cách trên, trong docker file không nên sử dụng cả 2 cách

3 - Dockerfile: Instructions and Examples

EXPOSE <PORT> [<PORT>/PROTOCOL]

EXPOSE - chỉ định port cho docker container.
Ví dụ:

- EXPOSE 5000
- EXPOSE 5522/udp

Lưu ý: Điều này không có nghĩa là người dùng có thể truy cập vào port này

3 - Dockerfile: Instructions and Examples

- **CMD** <COMMAND> (shell form)
- **CMD** ["EXECUTABLE", "PARAM1", "PARAM2"] (exec form)
- **CMD** ["PARAM1", "PARAM2"] (entrypoint default args)

CMD - chỉ định lệnh mặc định mà container sẽ thực thi khi được khởi chạy. Lệnh này chỉ được định nghĩa duy nhất một lần trong Dockerfile và thường đặt ở cuối cùng của file.

Ví dụ:

- CMD ["python3", "vehicles_detection_api.py"]

3 - Dockerfile: Instructions and Examples

```
FROM nvidia/cuda:10.2-cudnn7-devel-ubuntu18.04

RUN apt-get update

RUN apt-get install -y git \
    software-properties-common \
    && apt-get clean && rm -rf /tmp/* /var/tmp/*

RUN add-apt-repository ppa:deadsnakes/ppa && \
    apt update && \
    apt install python3.6 -y && \
    apt install python3-distutils -y && \
    apt install python3.6-dev -y && \
    apt install build-essential -y && \
    apt-get install python3-pip -y && \
    apt update && apt install -y libsm6 libxext6 && \
    apt-get install -y libxrender-dev && \
    apt install libgl1-mesa-glx -y

RUN apt install pkg-config
RUN DEBIAN_FRONTEND="noninteractive" apt-get -y install tzdata
RUN apt-get install -y libopencv-dev

COPY . /Vehicle_detection
```

```
RUN cd Vehicle_detection && \
        sed -i "s/OPENCV=0/OPENCV=1/" Makefile &&\
        sed -i "s/GPU=0/GPU=1/" Makefile &&\
        sed -i "s/CUDNN=0/CUDNN=1/" Makefile &&\
        sed -i "s/CUDNN_HALF=0/CUDNN_HALF=1/" Makefile &&\
        sed -i "s/LIBSO=0/LIBSO=1/" Makefile &&\
        make &&\
    python3 -m pip install -U pip &&\
    # fix bug can not install skbuild
    python3 -m pip install -U setuptools &&\
    pip3 install -r requirements.txt &&\
    gdown --id 1tOtF9vLSR0vjBwSo0ZkBUf3Zz9JnUEHd &&\
    gdown --id 1NL_ZojTjOIhw4O_ZdXD1aGgi-B_Ud9BP &&\
    gdown --id 17wjBkIIpbEo-JBK1IWVADLVOQjO7-qyc &&\
    gdown --id 14DO1-GPLZIiLXI0H-zYD05CE6II4e4XW

WORKDIR /Vehicle_detection

EXPOSE 7003

CMD ["python3", "vehicles_detection_api.py"]
```

4 - Docker Images

Image sẽ được khởi tạo theo các bước:

- Dockerfile sẽ parsed thông tin theo từng dòng
- Với mỗi command, Docker-engine sẽ build 1 separate layer.
- Số lượng layer sẽ bằng với số lượng command được chỉ định trong dockerfile
- Mỗi khi Image được build lại, Docker sẽ kéo các layer cũ và chỉ build những layer đã được chỉnh sửa.
- Những trường hợp layer cũ bị thay đổi là những layer này nằm bên dưới layer đã được thay đổi

4 - Docker Images

- docker image ls [OPTIONS]
- docker images [OPTIONS]

List all Docker Image
Ví dụ:

- docker image ls

4 - Docker Images

- docker image build [OPTIONS] PATH
- docker build [OPTIONS] PATH

Build Docker Image
Ví dụ:

- docker image build .
- docker image build -t my_image:latest

-t chỉ định tên và tag của image

4 - Docker Images

- docker image rm [OPTIONS] IMAGE [IMAGES..]
- docker rmi [OPTIONS] IMAGE [IMAGES..]

Delete Docker Image

Ví dụ:

- docker image rm -f nginx:1.1
- docker image rm Redis:latest python:slim node:old

4 - Docker Images

- docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
- docker pull [OPTIONS] NAME[:TAG|@DIGEST]

Pull Image from Docker hub
Ví dụ:

- docker image pull postgres:latest

4 - Docker Images

```
docker run [OPTIONS] IMAGE_NAME[:TAG|@DIGEST] [COMMAND] [ARGS...]
```

Run Docker
Ví dụ:

- docker run ubuntu:latest
- docker run -it -p 5000:80 -e "MODE=DEBUG" my_mongo_image

```
-v VOLUME_NAME:CONTAINER_PATH
```

-v chỉ định mount 1 volume vào bên trong docker container
Ví dụ:

- docker run -v vol2:/code/app alpine_image:jimjam

volume_name là tên đường dẫn chứa thư mục muốn mount vào container