Devops on AWS for beginner

Instructor – Linh Nguyen

Engineering Consultant, AWS Cloud Solution Architect

Thao tác cơ bản với Docker

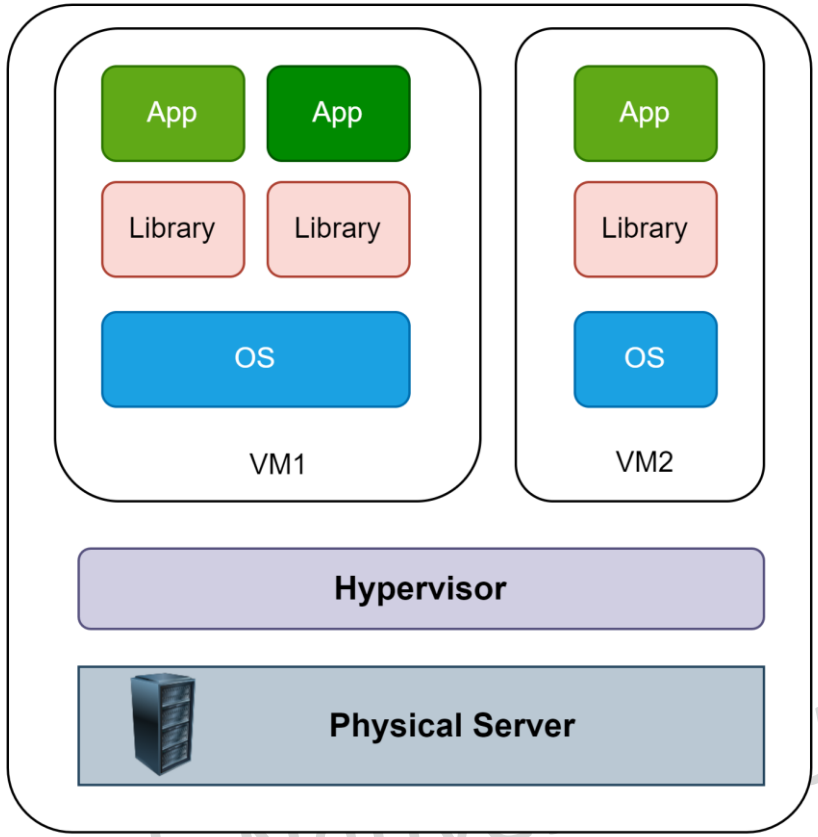
“Không có việc gì khó, chỉ sợ không biết làm!”

Target

- Nắm được các khái niệm cơ bản về Container & Docker.
- Cài đặt môi trường làm việc với Docker trên máy local hoặc Cloud9
- Biết cách thao tác với Image, Container, Docker File
- Thực hành Docker & Docker Compose qua các bài lab

Container là gì? tại sao lại cần Container?

Cùng xét một mô hình về việc deployment sử dụng máy ảo (VM) hay các EC2 server.



Virtual Machine Architect

Container là gì? tại sao lại cần Container?

Giả sử một ngày đẹp trời bạn được sếp giao nhiệm vụ install một software mới lên server production của công ty, bạn tìm hiểu rất kỹ càng những thư viện, runtime cần thiết của software này và đã tải về đầy đủ.

Đến ngày thực hiện, việc đầu tiên là bạn sẽ backup server (đương nhiên rồi ☺)

Bạn cài đặt một thư viện Jxxx version 20, tuy nhiên trong quá trình cài đặt Jxxx nói rằng không thể tồn tại 2 version song song trên cùng một máy và bạn phải gỡ version cũ hơn ra trước. Bạn gỡ version cũ là Jxxx 19 ra khỏi máy và cài Jxxx 20 vào (mới hơn tất nhiên là tốt hơn).

Sau khi hoàn thành thao tác cài đặt software, bạn thông báo với sếp là đã xong việc.

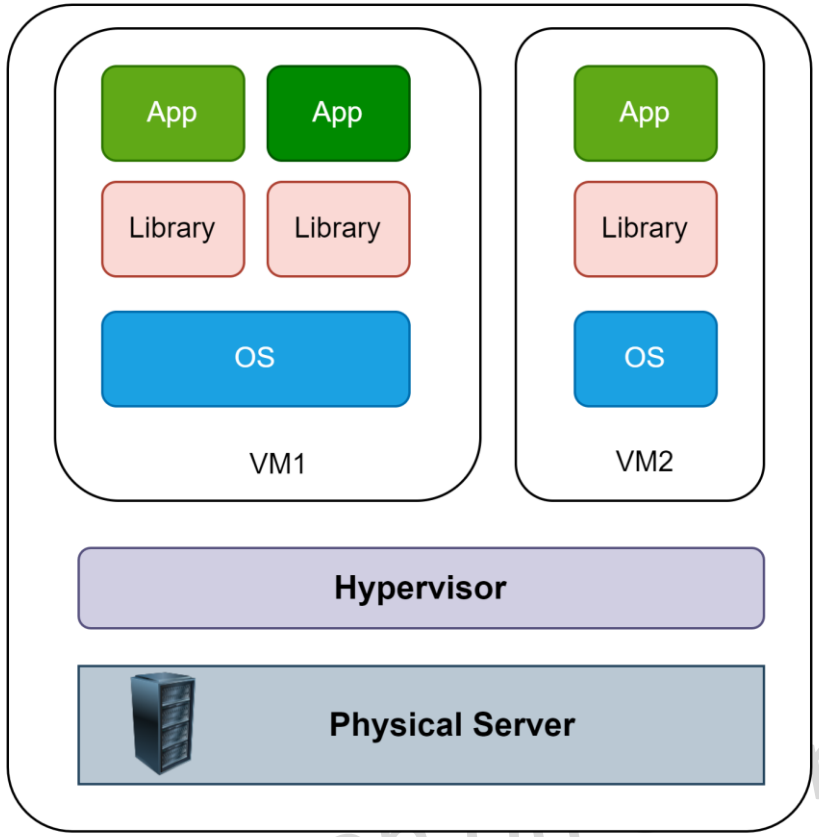
30 phút sau, khách hàng gọi điện và claim về việc một số software trước giờ vẫn sử dụng bình thường thì hôm nay không khởi động lên được và văng lỗi thiếu thư viện.

Và sau đó.... ... ☹

Container là gì? tại sao lại cần Container?

Problem khi sử dụng VM

- Xung đột version giữa Library, Binary, Runtime cùng cài trên các VM.
- Không có khả năng độc lập môi trường giữa các ứng dụng.
- Mất thời gian trong việc triển khai (chuẩn bị OS, cài các library cần thiết, setup môi trường,vv)
- Khó đảm bảo tính nhất quán của ứng dụng được triển khai (ứng dụng work OK trên một môi trường nhưng không chắc sang môi trường khác chạy bình thường).

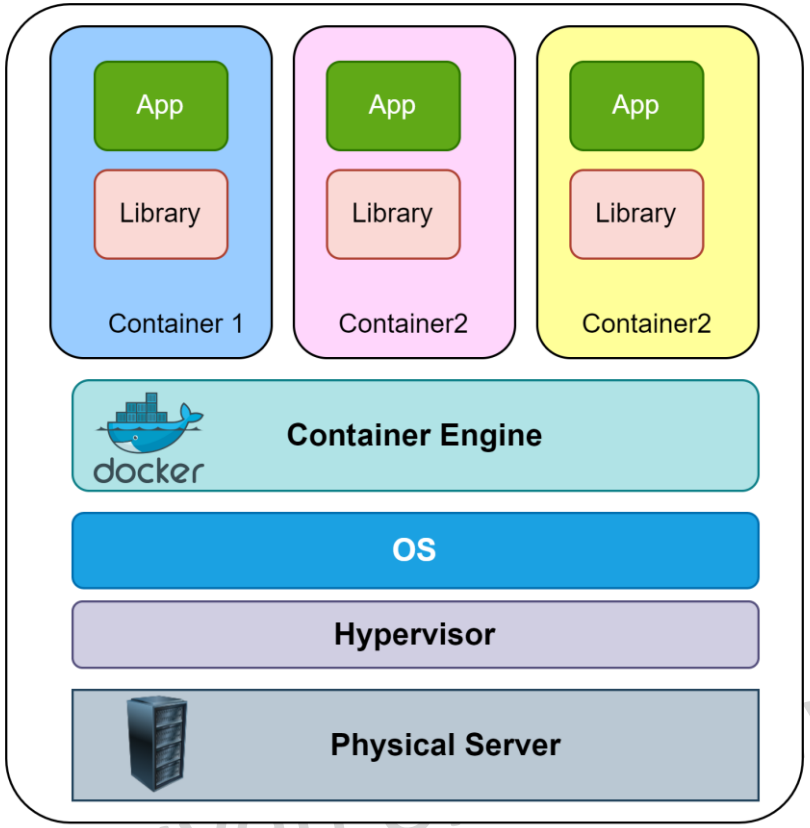


Virtual Machine Architect

Container là gì? tại sao lại cần Container?

Container ra đời để giải quyết những vấn đề trên, bằng cách:

- Đóng gói ứng dụng cùng với những thứ cần thiết để chạy được ứng dụng đó thành một image có thể run ở bất cứ đâu có hỗ trợ container.
- Cung cấp môi trường & cơ chế cấp phát tài nguyên để image đó có thể run được.
- Cung cấp cơ chế & công cụ cho phép các nhà phát triển đóng gói, lưu trữ, phân phối và triển khai ứng dụng một cách thuận tiện.



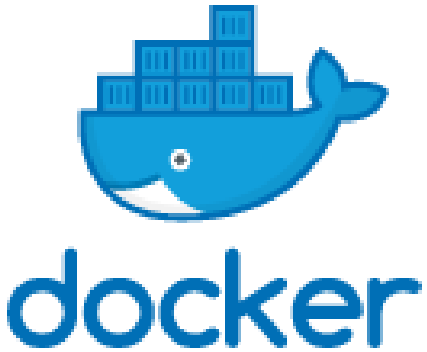
Containers Architect

Lợi ích của việc sử dụng Container

1. **Độc lập với môi trường**: Containers cung cấp một cách để đóng gói ứng dụng và tất cả các dependencies của nó, bao gồm OS, libraries, tools. Nó cho phép ứng dụng chạy một cách độc lập và nhất quán trên bất kỳ môi trường nào.
2. **Đơn giản hóa quy trình triển khai**: tính nhất quán, tốc độ, thuận tiện là những gì Containerlize mang lại khi so sánh với mô hình truyền thống.
3. **Quản lý tài nguyên hiệu quả**: triển khai ứng dụng bằng Container cho phép bạn chia sẻ và sử dụng tài nguyên của hệ thống một cách hiệu quả. Bằng cách chạy nhiều container trên cùng một máy chủ vật lý hoặc máy ảo, bạn có thể tận dụng tối đa khả năng tính toán và tài nguyên của hệ thống.
4. **Linh hoạt và mở rộng**: Containers cho phép bạn dễ dàng mở rộng ứng dụng theo nhu cầu. Bằng cách scale horizontal, ứng dụng có thể mở rộng để đáp ứng workload. Ngoài ra, việc triển khai nhiều version của một ứng dụng cùng lúc cũng trở nên dễ dàng.

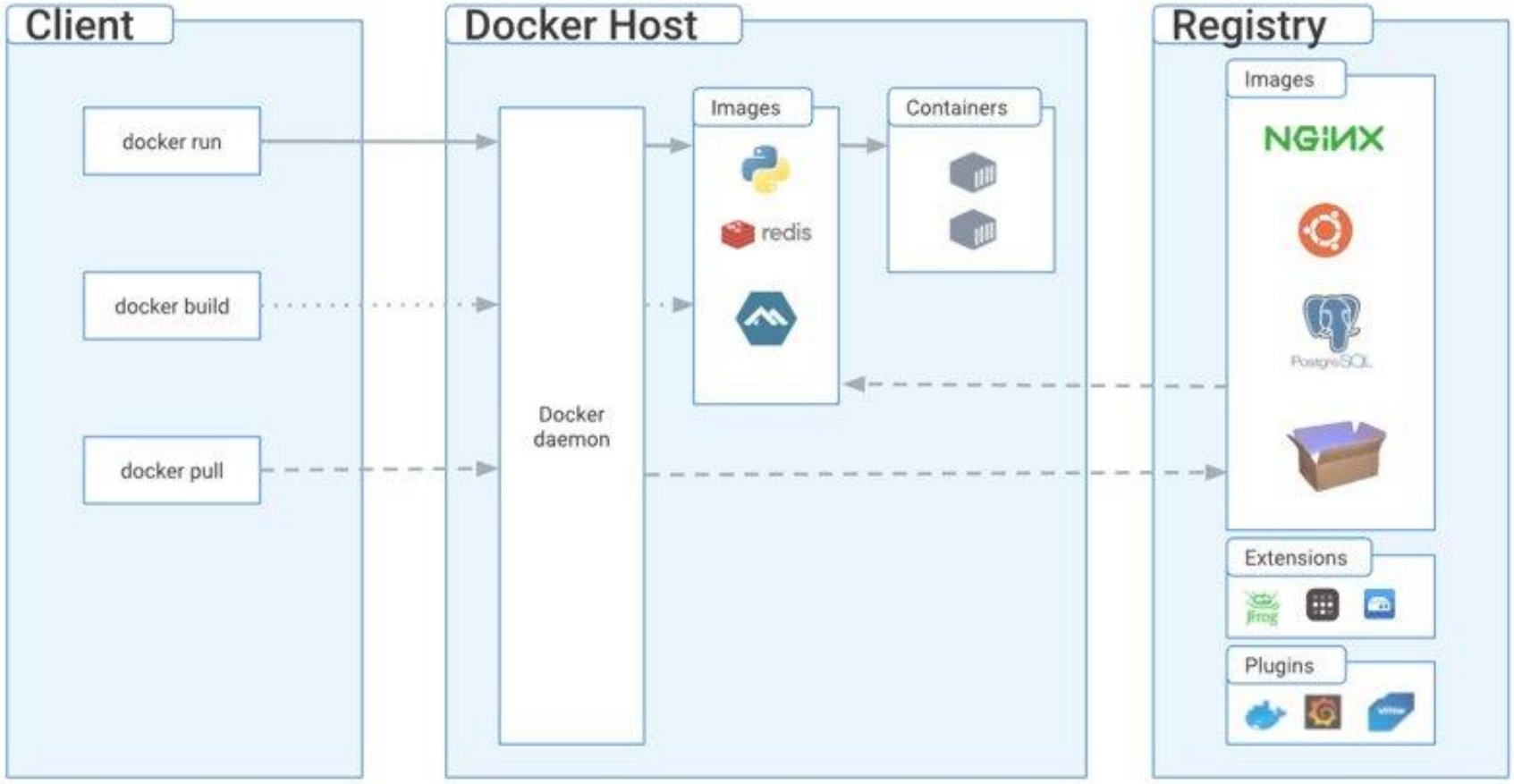
Docker là gì?

Docker là nền tảng phần mềm cho phép bạn dựng, kiểm thử và triển khai ứng dụng một cách nhanh chóng. Docker 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, code, runtime. Bằng cách sử dụng Docker, bạn có thể nhanh chóng triển khai và thay đổi quy mô ứng dụng vào bất kỳ môi trường nào và biết chắc rằng code của bạn sẽ chạy được (build one – run anywhere).



*Lưu ý: Docker không phải là nền tảng duy nhất cho phép triển khai ứng dụng dưới dạng container.

Các thành phần cơ bản của Docker



Các thành phần cơ bản của Docker

- **Docker daemon**: là nơi quản lý các thành phần của Docker như image, contain er, volume, network.
  Docker daemon nhận API từ Client để thực thi các nhiệm vụ.
- **Docker Client**: Cung cấp phương thức để tương tác với Docker daemon.
- **Docker registry**: nơi lưu trữ các docker
  image. Mặc định docker sẽ connect tới docker registry là Docker hub.

Khi bạn cài Docker Desktop, Daemon và Client sẽ cùng nằm trên máy tính của bạn.

Các bước cơ bản để xây dựng ứng dụng docker

1. **Chuẩn bị Dockerfile**: Dockerfile mô tả các bước cần thiết để tạo ra môi trường container chứa ứng dụng của bạn.
2. **Build Docker Image**: Sử dụng lệnh **docker build** để xây dựng Docker Image từ Dockerfile.
3. **Kiểm tra Docker Image**: Sử dụng lệnh **docker images** để kiểm tra danh sách các ảnh Docker có sẵn trên máy tính của bạn. Đảm bảo rằng Docker Image của ứng dụng của bạn đã được xây dựng thành công và xuất hiện trong danh sách.
4. **Chạy Docker Container**: Sử dụng lệnh **docker run** để chạy một container từ Docker Image.
5. **Kiểm tra ứng dụng**: Truy cập ứng dụng của bạn thông qua địa chỉ IP hoặc tên miền cùng với port đã chỉ định.

Trong thực tế khi triển khai lên môi trường Cloud sẽ bao gồm nhiều bước phức tạp hơn, sẽ được trình bày sau.

Cài đặt Docker & Docker compose trên máy local (Windows)

Tham khảo guide sau:

https://docs.docker.com/desktop/install/windows-install/

Check version:

```
>docker --version
Docker version 24.0.6, build ed223bc
>docker compose version
Docker Compose version v2.22.0-desktop.2
```

Cài đặt Docker & Docker compose trên máy local (MacOS)

Tham khảo guide sau:

https://docs.docker.com/desktop/install/mac-install/

Check version:

```
>docker --version
Docker version 24.0.6, build ed223bc
>docker compose version
Docker Compose version v2.22.0-desktop.2
```

Cài đặt Docker & Docker compose trên AWS Cloud9

Tạo một môi trường làm việc sd Cloud9

Tham khảo guide sau:

https://docker.awsworkshop.io/10_prerequisites/dockerconfig.html

***Một số ưu điểm của Cloud9**

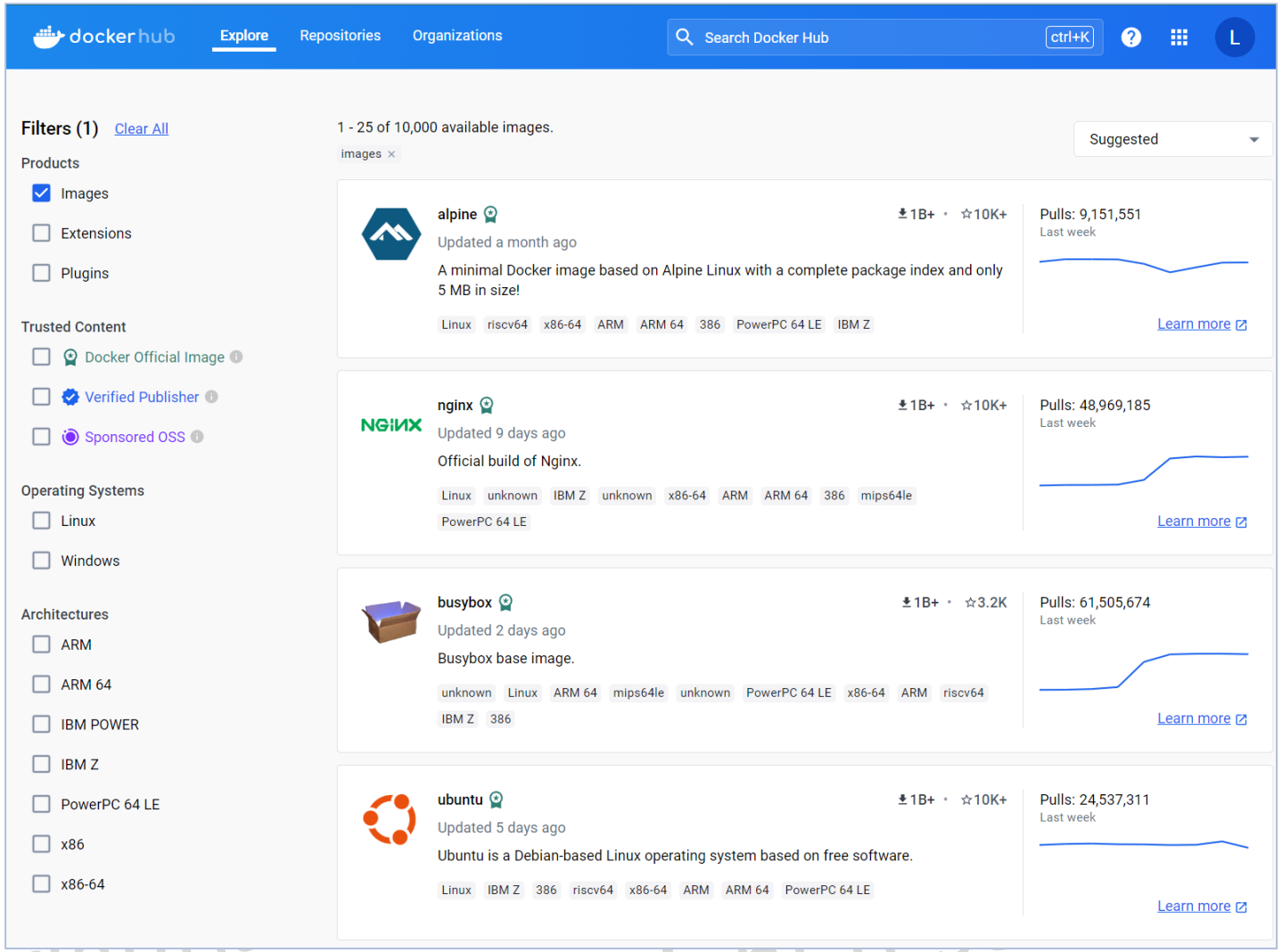
- Tương thích native với Linux
- Không bị chặn bởi rule mạng công ty (vd khi pull package bị chặn).
- Cấu hình linh hoạt, chi phí hợp lý, auto-idle khi không sử dụng.
- Bảo mật cao thông qua cơ chế phân quyền IAM.

Đăng ký account trên DockerHub

Các bạn sử dụng link sau để đăng ký account DockerHub

https://hub.docker.com/signup

Giao diện sau khi login vào Dockerhub như hình bên =>



Lab 1: Một số thao tác cơ bản với Docker.

Thực hành làm quen với Docker qua các thao tác sau

- Pull một image từ docker registry (Docker Hub)
- List image
- List all container
- List all container - All status
- Start/Stop/Restart một container
- Delete một container
- Delete một image
- SH vào một container đang chạy

Giới thiệu Dockerfile

- Khai báo Base image: Dockerfile thường bắt đầu bằng một chỉ thị **FROM** để chỉ định base image mà new Docker image sẽ dựa trên. Ví dụ: **FROM ubuntu:latest, FROM httpd vv.**
- Sao chép các tệp và thư mục: Sử dụng chỉ thị **COPY** hoặc **ADD**, bạn có thể sao chép các tệp và thư mục từ máy chủ nơi Dockerfile được chạy vào bên trong Docker image
  Ví dụ: **COPY app.py /app**, **COPY . /usr/local/apache2/htdocs/**
- Chỉ thị **RUN**. bạn có thể thực thi các lệnh bên trong quá trình build image vd để cài đặt phần mềm, update package hoặc thực hiện các tác vụ khác.
  Ví dụ: **RUN apt-get update && apt-get install -y python**.
- Thiết lập biến môi trường: Bằng cách sử dụng chỉ thị **ENV**, bạn có thể định nghĩa các biến môi trường cho Docker image. Ví dụ: **ENV LOGLEVEL=DEBUG**.
- Mở port: Bằng cách sử dụng chỉ thị **EXPOSE**, bạn có thể xác định các port mà ứng dụng trong hình ảnh Docker sẽ lắng nghe. Ví dụ: **EXPOSE** 80.
- Chạy ứng dụng: Bằng cách sử dụng chỉ thị **CMD** bạn có thể chỉ định lệnh mà Docker sẽ chạy khi khởi động một container từ image.
  Ví dụ: **CMD ["python", "/app/app.py"]**

```
1  FROM httpd
2
3  COPY index.html /usr/local/apache2/htdocs/
4
5  RUN chown www-data:www-data /usr/local/
   apache2/htdocs/index.html
6  ENV LOGLEVEL=DEBUG
7  EXPOSE 80
8  CMD ["httpd-foreground"]
9
```

Giới thiệu Dockerfile

Phân biệt chỉ thị COPY và chỉ thị ADD

COPY và ADD đều hỗ trợ copy file/folder.

ADD hỗ trợ giải nén trong quá trình copy, copy từ URL trên Internet.

COPY không hỗ trợ giải nén hoặc hỗ trợ URL trên Internet.

```
1  FROM httpd
2
3  COPY index.html /usr/local/apache2/htdocs/
4
5  RUN chown www-data:www-data /usr/local/
   apache2/htdocs/index.html
6  ENV LOGLEVEL=DEBUG
7  EXPOSE 80
8  CMD ["httpd-foreground"]
9
```

Ngoài chỉ thị CMD, còn 1 chỉ thị khác là ENTRYPOINT cũng được sử dụng để định nghĩa command sẽ chạy khi Container start, tuy nhiên có sự khác biệt:

- ENTRYPOINT định nghĩa process sẽ được thực thi bên trong container
- CMD định nghĩa default arguments cung cấp cho entrypoint process.

Thông thường ENTRYPOINT sẽ định nghĩa path tới process sẽ được thực thi, CMD định nghĩa param (nếu có).

Mặc định không định nghĩa ENTRYPOINT, docker sẽ hiểu entry point là **/bin/sh -c**

*Tham khảo thêm tại: https://spacelift.io/blog/docker-entrypoint-vs-cmd

Lab 2: Build & Run 1 container đơn giản sd Dockerfile.

Yêu cầu:

Thiết kế Dockerfile sử dụng image **httpd**, sd chỉ thị **COPY** để copy code **html** từ workspace vào trong container, start server và expose ra port 80.

Build image, run container từ image, mapping với 8080 của host.

Test việc truy cập thông qua browser.

Stop container, xoá container đã tạo.

Xoá image đã tạo.

Lab 3: Phân biệt CMD và ENTRYPOINT

1. Sử dụng câu lệnh **docker inspect <image-name>** để xem ENTRYPOINT default.
2. Thực hiện build một số docker image với thông số như bảng sau (FROM alpine).

No	Image name	ENTRYPOINT	CMD	docker run
1	alpine-1	ENTRYPOINT ["ls"]	N/A	không chỉ định param
2	alpine-1	ENTRYPOINT ["ls"]	N/A	chỉ định param “-alh”
3	alpine-2	N/A	CMD ["ls"]	không chỉ định param
4	alpine-2	N/A	CMD ["ls"]	chỉ định param “-alh”
5	alpine-3	ENTRYPOINT ["ls"]	CMD ["-alh"]	không chỉ định param
6	alpine-3	ENTRYPOINT ["ls"]	CMD ["-alh"]	-p --full-time

Multi stages Dockerfile

Tại sao cần sử dụng Multi stages Dockerfile?

- Giảm kích thước của image cần build: trong quá trình build cần cài rất nhiều công cụ, nếu để nguyên như vậy sẽ làm tăng size của final image.
- Tăng hiệu suất: sử dụng các base image sẵn có cho quá trình build thay vì tự cài từng tools.
- Tách biệt stage trong quá trình phát triển.

Multi stages Dockerfile

Sample multi-stage dockerfile.

- Stage #1: sử dụng image node14 để build một project Nodejs thành dạng static file.
- Stage #2: sử dụng image nginx để build image, copy static content output từ stage #1.

```
# Stage 1: Build
FROM node:14 AS build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci
COPY . .
RUN npm run build

# Stage 2: Production image
FROM nginx:latest
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Lab 4: Multi stages Dockerfile

Thực hành build 1 Nodejs App sau đó đóng gói thành 1 static web trên **httpd** sử dụng 2 stages:

- Stage #1: sử dụng **Nodejs** image để build ra static file
- Stage #2: Sử dụng **httpd** image, copy toàn bộ output từ stage #1 tạo thành image **custom-httpd** hoàn chỉnh.

Build image, kiểm tra kết quả.

Chạy container từ image **httpd** đã build (**image thứ #2**).

Giới thiệu Docker Compose

Docker Compose là một công cụ dùng để định nghĩa và chạy các ứng dụng multi-container trong Docker. Nó cho phép bạn định nghĩa các service, network, volumes và các cấu hình khác trong một **file YAML** duy nhất. Sau đó, bạn có thể sử dụng Docker Compose để tạo và quản lý các container dựa trên file YAML đó. Điều này giúp đơn giản hóa việc triển khai và quản lý các ứng dụng phức tạp với nhiều container trong môi trường Docker.

**Trong thực tế Docker-compose được các developer sử dụng để start môi trường tại máy local một cách nhanh chóng cũng như apply thay đổi tức thì. Tuy nhiên trên Cloud sẽ thường không sử dụng Docker-compose để deploy.*

Giới thiệu Docker Compose

Cú pháp của 1 docker-compose file cơ bản:

- **service**: đơn vị quản lý của docker compose.
- **build**: chỉ thị build, sử dụng Dockerfile nào để build ra Docker image.
- **ports**: mapping giữa hostport:container-port.
- **volume**: mapping giữa thư mục của máy host và container.
- **environment**: khai báo overwrite variable.
- **volumes**: định nghĩa các persistent volume cho service.

```yaml
version: '3'
services:
  web:
    build: .
    ports:
      - 8080:80
    volumes:
      - ./app:/var/www/html
    depends_on:
      - db
  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=mydb
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

Lab5 – DockerCompose - Simple

Tạo 1 file docker-compose.yaml để start một Nodejs services.

Thử truy cập website nodejs

Lab6 – DockerCompose - Multi

Tạo 1 file docker-compose.yaml để start các service sau:

- nodejs
- redis-cache
- My-SQL

Yêu cầu: tạo volume riêng cho MySQL.

Thử truy cập website nodejs

Có thể truy cập MySQL thông qua hostport để kiểm tra.

Lab7 – DockerCompose – Load Balancer

Sử dụng lại resource của bài lab 6

Docker cheat sheet

Cách để tra cứu nhanh các câu lệnh Docker.

https://collabnix.com/docker-cheatsheet/

Nguồn tra cứu document:

https://docs.docker.com/manuals/

Giới thiệu Elastic Container Registry

Elastic Container Registry (ECR) là một dịch vụ của AWS cung cấp khả năng quản lý và lưu trữ các Docker Image. ECR là một Registry dựa trên cloud computing, được thiết kế đặc biệt để làm việc với các container hóa và môi trường chạy container của AWS.

- **Registry**: đơn vị quản lý của ECR (giống một repository). Thông thường một Registry sẽ chỉ lưu image của một ứng dụng.
- **Image**: Tương tự Docker Image. Các Image trên Registry cần đánh tag để quản lý.

Amazon Elastic Container Registry (Amazon ECR)

Registry

Image

Giới thiệu Elastic Container Service

Elastic Container Service (ECS) là một dịch vụ quản lý container do Amazon Web Services (AWS) cung cấp. Nó cho phép bạn chạy và quản lý các ứng dụng container trên nền tảng AWS một cách dễ dàng và linh hoạt.

Amazon Elastic Container Service (Amazon ECS)

Giới thiệu Elastic Container Service

Các tính năng của ECS

Amazon Elastic Container Service (Amazon ECS)

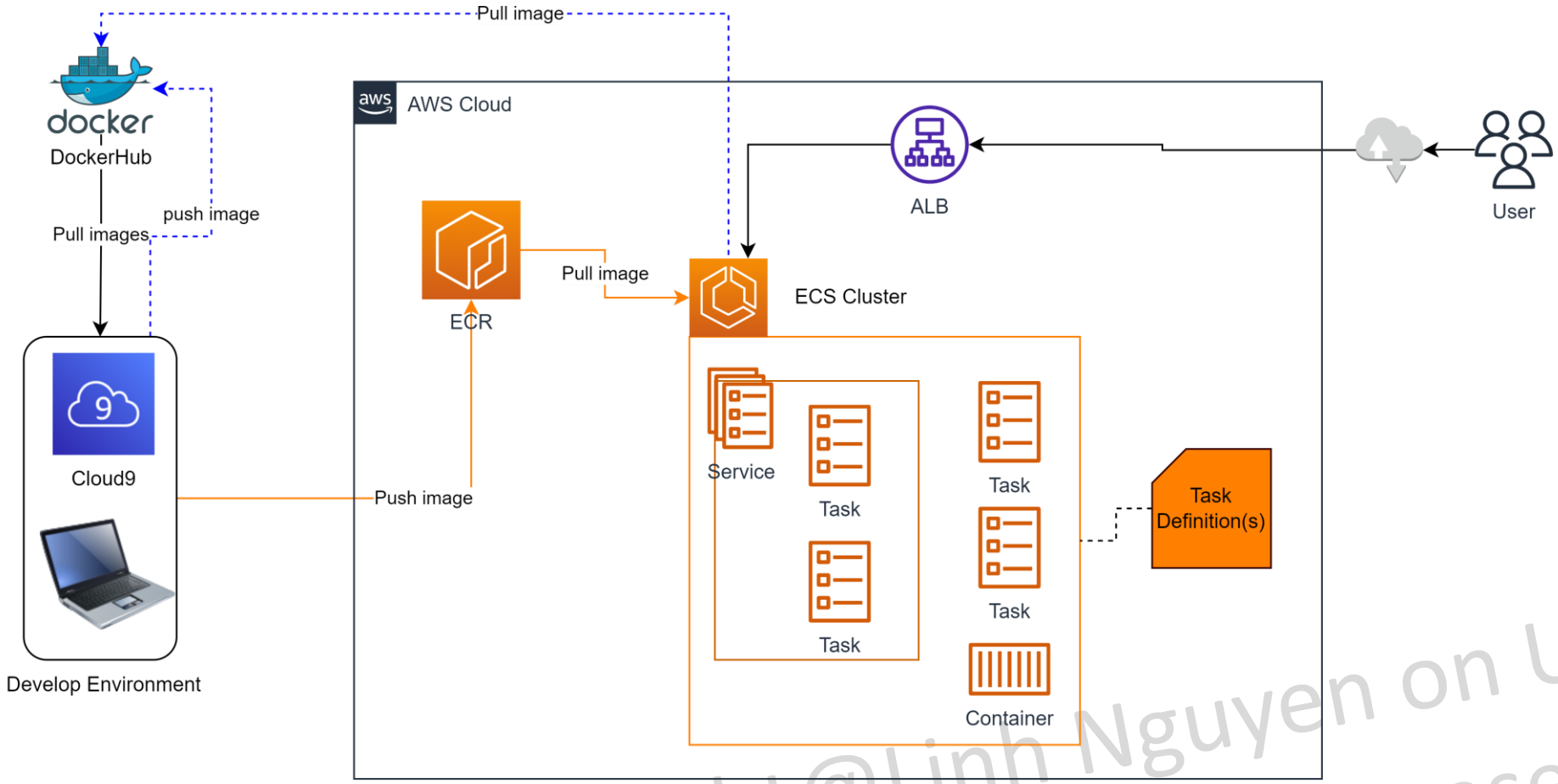
1. Quản lý đơn giản.
2. Tích hợp với công cụ container: ECS tích hợp tốt với Docker, cho phép bạn chạy các container Docker trực tiếp trên nền tảng AWS mà không cần thay đổi mã nguồn hoặc cấu hình.
3. Mở rộng linh hoạt: Kết hợp với AutoScaling, ECS cho phép scale in-out linh hoạt dựa trên nhu cầu workload.
4. Tích hợp với các dịch vụ AWS khác: ECS tích hợp tốt với các dịch vụ AWS khác như Elastic Load Balancer (ELB), Elastic Container Registry (ECR), IAM, CloudWatch và nhiều dịch vụ khác.
5. Sự linh hoạt về kiến trúc: ECS hỗ trợ hai kiểu triển khai: EC2 Launch Type và Fargate Launch Type. Giúp bạn dễ dàng lựa chọn dựa theo nhu cầu cũng như khả năng customize của đội dự án.

Giới thiệu Elastic Container Service

Các thành phần của ECS

- Cluster: Đơn vị lớn nhất của ECS, có nhiệm vụ cung cấp tài nguyên cần thiết (EC2, Fargate) để chạy ứng dụng.
- Task: Một đơn vị được cấp phát tài nguyên (CPU,RAM) trong mode Fargate. Một task có thể chứa 1 hoặc nhiều container.
- Service: Một nhóm các Task có chung nhiệm vụ được expose ra bên ngoài hoặc nội bộ cluster.
- Container: tương tự Docker Container, một runable image.
- ECS Connect Service: Cung cấp cơ chế service-to-service communication
- Task Definition (không có trong hình): chỉ dẫn để ECS biết phải tạo một task như thế nào.

Sample hệ thống triển khai trên AWS sử dụng ECS, ECR



Lưu ý: các tool như Jenkins, các service phục vụ CICD deployment chưa thể hiện trong hình.

Lab - Build, Push image lên ECR, sử dụng Image để run task trên ECS.

Mời các bạn cùng xem lại các bài lab ở level Basic liên quan tới ECS, ECR.

- Mục đích để các bạn ôn lại kiến thức cũng như các concept về Docker, ECS, ECR.
- Cụ thể về việc deploy lên ECS sử dụng CICD sẽ được trình bày trong các chương CICD sử dụng Jenkins, CodePipeline, Github Action.

Tổng kết & Clear resources

- Terminate EC2 instance, Elastic IP, Volume, Snapshot.
- Stop hoặc xoá môi trường Cloud9..
- Delete hết các ECS service, ECS Cluster.
- Xoá ECR Repository.
- Xoá Application Load Balancer.

Thanks you and see you in the next chapter!