TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CNTT & TT

NIÊN LUẬN CƠ SỞ NGÀNH TRUYỀN THÔNG VÀ MẠNG MÁY TÍNH

Giảng viên:

Sinh viên: Lê Hoàng Giang

MSSV:

Đề tài:

MỤC LỤC

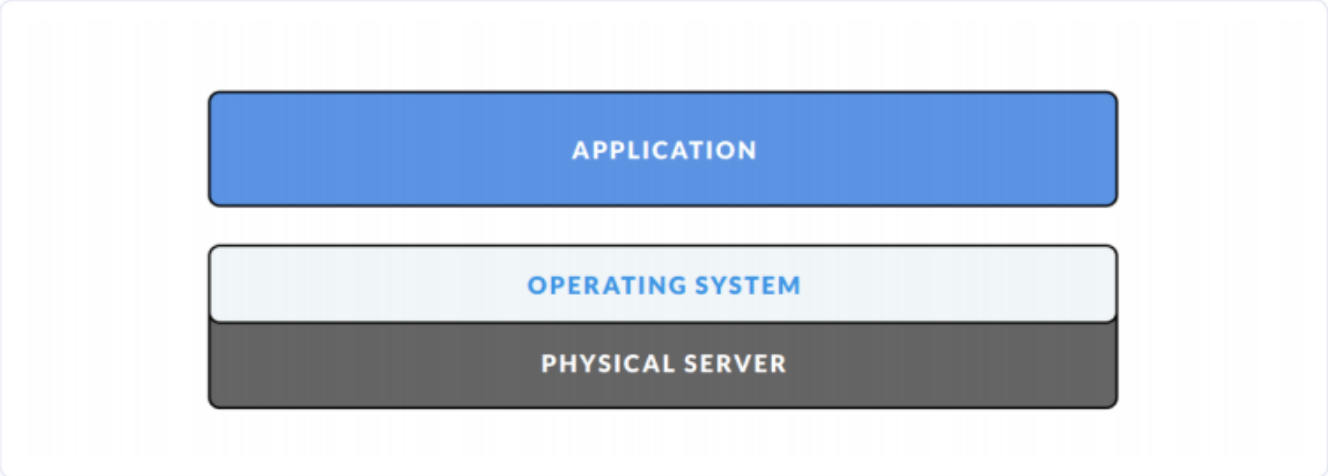
Phần 1: Sự ra đời của Docker

I. Giới thiệu

1. Sự ra đời của Docker

1.1. Theo dòng lịch sử của sự phát triển về công nghệ và máy tính ta có những mô hình máy tính như sau:

Buổi đầu một máy tính được cấu tạo theo mô hình thường là *Máy Chủ Vật Lý + Hệ Điều Hành (OS) + Các Ứng Dụng (Application)*



Tuy nhiên, trong quá trình phát triển của công nghệ và yêu cầu về sự tối ưu hóa hiệu suất của tài nguyên máy tính nên đã xuất hiện các hạn chế như sau:

- Một máy tính có một ổ cứng lớn, Ram lớn cũng không thể cũng không thể tận dụng hết được tài nguyên và ưu thế sẵn có bởi vì mỗi máy chủ như thế chỉ cài được một *Hệ Điều Hành (OS).*
- Với yêu cầu sử dụng ngày càng tăng thì với mỗi máy chủ chỉ chạy duy nhất một *Hệ Điều Hành (OS)* sẽ không thể tăng được hiệu suất làm việc.

Chính vì thế *Công Nghệ Ảo Hóa (vitualization)* ra đời.

1.2. *Vitualization* là một công nghệ mà cho phép trên cùng một máy chủ vật lý ta có thể tạo được nhiều *OS* cùng lúc giúp cho việc tận dụng được tài nguyên tốt hơn xử lý được nhiều công việc hơn và tăng hiệu suất làm việc của máy tính.

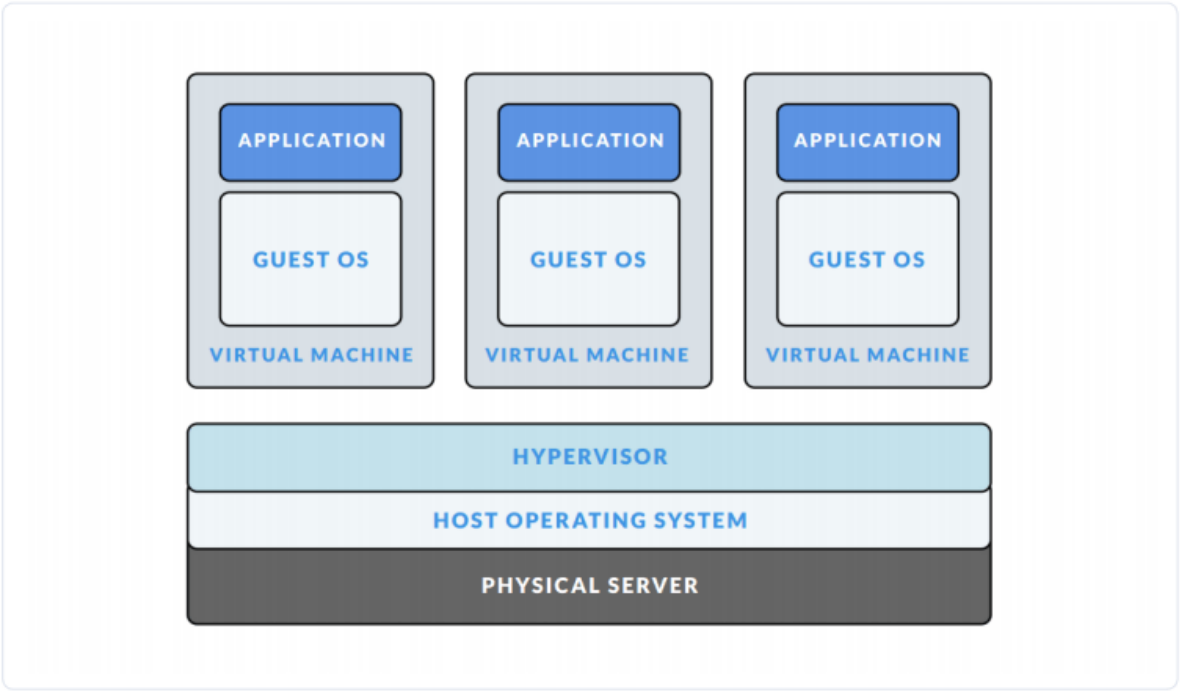
Nhưng công nghệ mới lại nảy sinh một số vấn đề tiếp theo:

- *Về quản lí hệ thống*: việc cập nhật và quản lí các version cho từng ứng dụng rất khó khăn, không thể đồng bộ trong cập nhật vì mỗi application có thể có những phiên bản version riêng biệt.
- *Về tài nguyên*: Khi bạn bật một máy ảo lên thì phải cung cấp "cứng" dung lượng bộ nhớ cũng như ram cho máy ảo đó. Điều này dẫn đến nếu bật nhiều

máy ảo cùng lúc và có một số máy không dùng để làm gì thì đúng là rất lãng phí tài nguyên.

- *Về thời gian*: Việc khởi động cũng như shutdown máy tốn khá nhiều thời gian có khi lên đến vài phút.

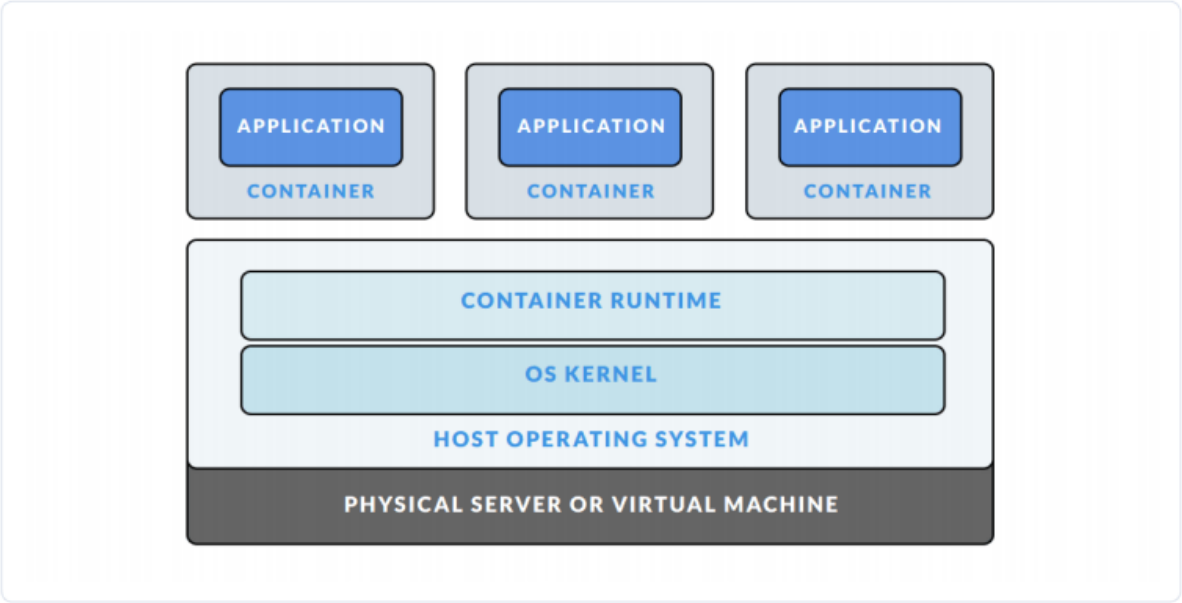
Điều này dẫn đến sự ra đời của *Công Nghệ containerlization.*



1.3 Với công nghệ *Containerlization* trên một máy chủ chúng ta có thể tạo ra được nhiều máy ảo con (giống như công nghệ ảo hóa *Vitualization)*, tuy nhiên nó được cải tiến hoàn thiện hơn là các máy ảo con này (*Guess OS)* sử dụng chung phần nhân của máy mẹ (*Host OS)* đồng thời các máy con sẽ cùng chia sẻ với nhau nguồn tài nguyên từ máy mẹ.

- Với sự cải tiến mới này việc phân chia tài nguyên của máy mẹ cho các máy con được tối ưu hóa, không cần dùng sẽ không cấp khi nào cần thì cấp và cần bao nhiêu sẽ đáp ứng bấy nhiêu.

- Ngoài ra điểm vượt trội của công nghệ *Containerlization* chính là việc nó sử dụng các *container.*



1.4. *Container* là một công cụ tạo môi trường được "đóng gói" trên máy tính mà không làm ảnh hưởng tới hiện tại trong máy, nói cách khác môi trường do *container* tạo ra chạy độc lập so với máy chủ.

II. Sự ra đời của Docker

1. 3/2013, dạng mã nguồn mở đầu tiên của *Docker* được phát hành bởi một công ty, công ty này sau này đổi tên thành *Docker* (www.docker.com)
2. *Docker* khởi đầu được xây dựng trên nền tảng *Linux* đây chính là đặc thù của Docker vì nó cần can thiệp trực tiếp vào nhân (*Kernel)* mà *Linux* lại là mã nguồn mở, trong khi đó *Kernel* của Windows chứa những thứ không public.
3. Tuy nhiên tới thời điểm hiện tại *Docker* hoàn toàn có thể sử dụng trên Windows, Mac dựa trên một máy ảo *Linux*.
4. Cho tới thời điểm hiện tại Docker có 2 phiên bản:

- *CE (Community Edition)*: Phiên bản dành cho nhà phát triển, nhóm nhỏ, lập trình viên.
- *EE (Enterprise Edition)* : Phiên bản dành cho doanh nghiệp.

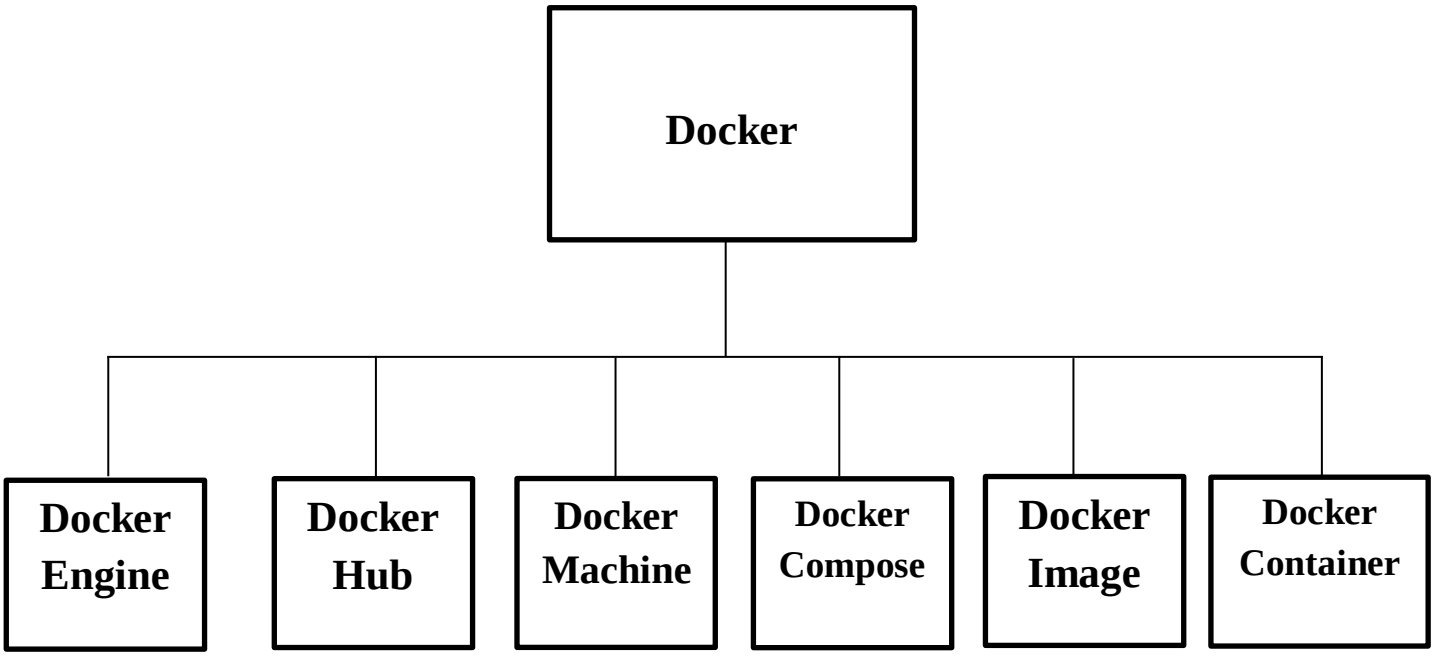
Phần 2: Thành phần, chức năng và cơ chế hoạt động

I. Thành phần

1. Sơ đồ

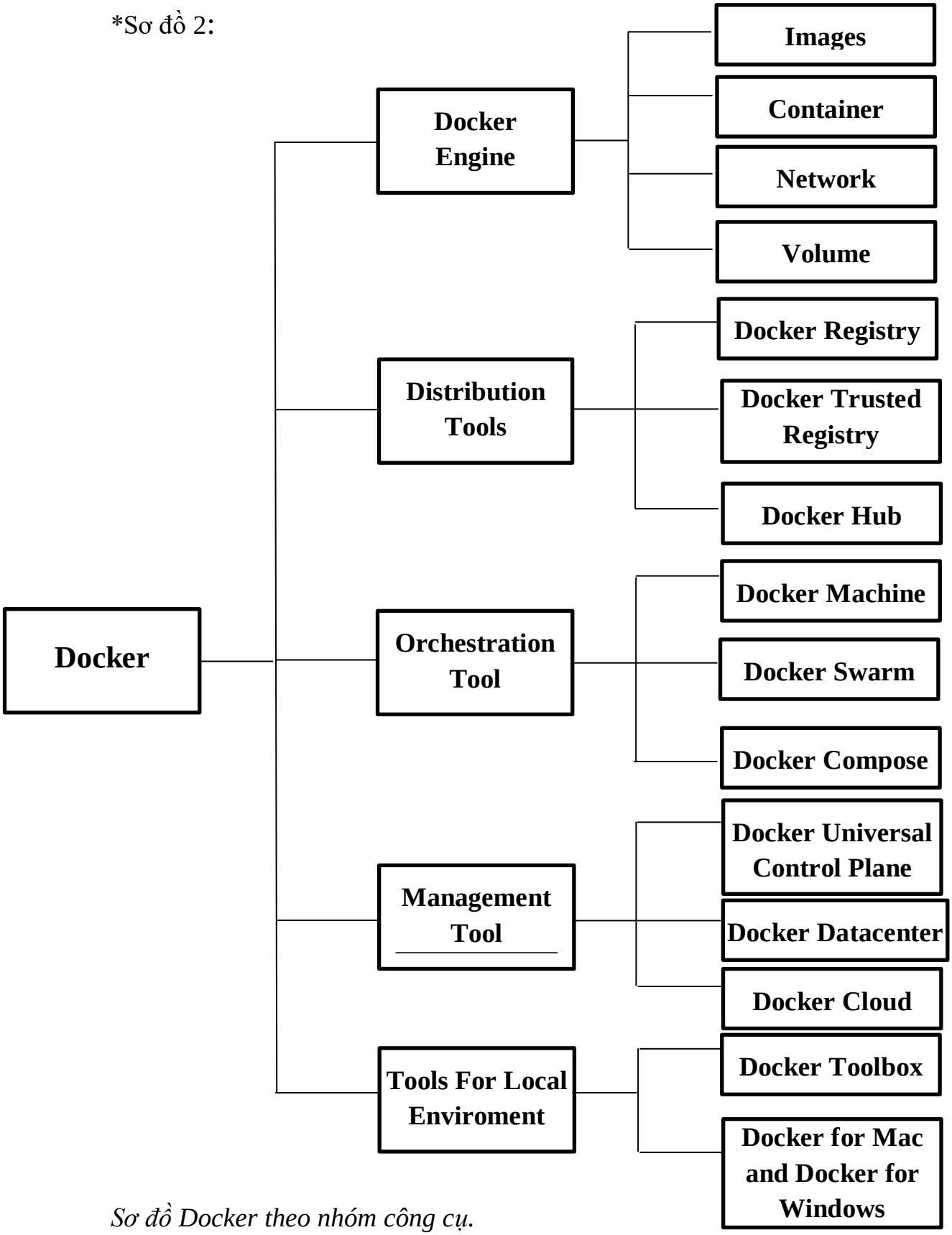
Sơ đồ cấu tạo của Docker có thể biểu diễn bằng hai cách sau đây:

*Sơ đồ 1:



Sơ đồ cấu tạo Docker theo thành phần

*Sơ đồ 2:



Sơ đồ Docker theo nhóm công cụ.

Trong bài viết sẽ phân tích chức năng các thành phần của Docker thông qua sơ đồ 2.

II. Chức năng

1. Docker Engine

1.1. Là công cụ *Client – Server* được hỗ trợ công nghệ container để xử lí các nhiệm vụ và quy trình công việc liên quan đến việc xây dựng các ứng dụng dựa trên vùng chứa (container).

1.2. *Engine* tạo ra một quy trình deamon phía máy chủ lưu trữ *images, containers, networks* và *storage volumes.*

1.3. Quy trình *deamon* chạy trên các máy host, người dùng sẽ tương tác với *deamon* với quy trình *deamon* thông qua Docker Client (*giao diện người dùng của Docker*).

1.4. Docker Client sẽ giao tiếp với Docker Engine thông qua một *RESTful API* để thực thi một câu lệnh nào đó.

1.5. Docker Engine còn có thể cho phép load third-party plugins để mở rộng chức năng khác khi cần thiết.

1.6. *Images, Container, Network, Volumes* là 4 đối tượng của *Engine,* mỗi đối tượng trên đều được xác định bằng ID và 4 đối tượng này phối hợp với nhau để vận hành hệ thống giúp cho người dùng có thể xây dựng (*Build),* chuyển (*Ship),* chạy (*Run*) một ứng dụng (*application)* nào đó ở bất cứ đâu.

- *Image:*

+ Được tạo ra bởi Docker Engine, là nền tảng của *Container* được dùng để chạy các Docker Container. Nói theo cách lập trình hướng đối tượng thì Docker Images là Lớp (*Class)* còn các *Container* là thực thể (*entity)* nằm trong lớp đó.

+ Docker Image là một dạng tập hợp các tệp của ứng dụng, lưu và sử dụng lại các files và settings được sử dụng trong các container, các Docker Image sẽ không bị thay đổi khi di chuyển.

+ Một Image bao gồm *Hệ Điều Hành (Windows, CentOS, Ubuntu,…)* và các môi trường lập trình được cài đặt sẵn *(httpd, mysqld, nginx, python, git,…)*.

+ Image có thể được lưu trữ ở *local* hoặc trên một *registry*

- *Container*

+*Container* là một đơn vị phần mềm tiêu chuẩn, được đóng gói mã và tất cả các phụ thuộc của nó để ứng dụng chạy nhanh và tin cậy trên các môi trường khác nhau.

+Là một dạng *Runtime instance* của các Docker image, dùng để làm môi trường chạy ứng dụng, nói dễ hiểu nó như một máy ảo về mặt chức năng xuất hiện khi mình khởi chạy các image.

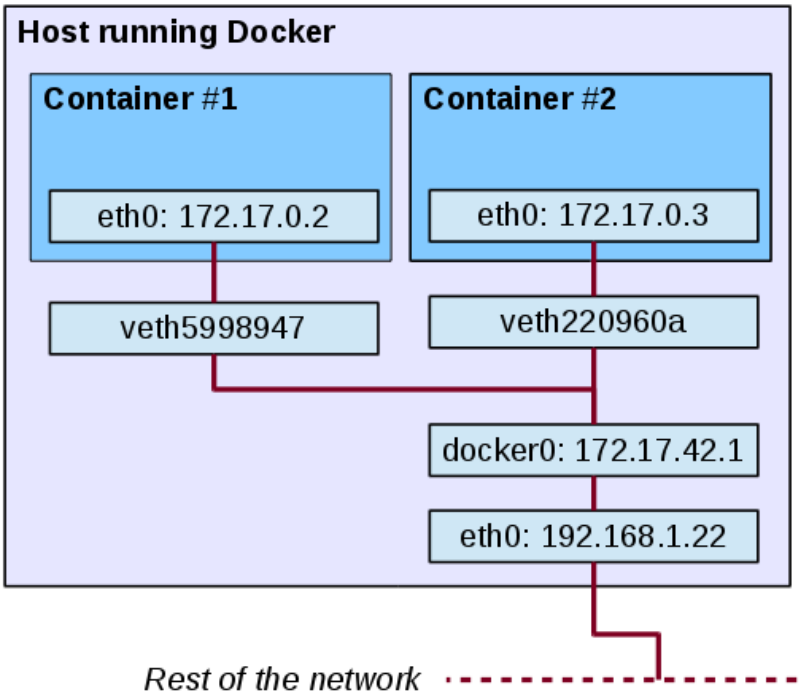
+ *Container* có tốc độ khởi chạy nhanh hơn máy ảo (*virtual machine)* có thể giúp người dùng chạy 4,5 container cùng một lúc mà không sợ nghẽn hệ điều hành (treo máy).

+ *Container* lưu trữ mọi thứ người dùng cần để chạy một ứng dụng, *container* có thể có các trạng thái *run, started, stopped, moved* và *deleted.*

- *Network:*

+Network của Docker được quản lí thông qua một *virtual bridge* gọi là *Docker0* nhằm tạo ra một môi trường mạng độc lập tách biệt với môi trường khác.

+ Cung cấp một *private network* mà chỉ tồn tại giữa *container* và *host.*



- *Volume:*

+ Được thiết kế để lưu trữ các dữ liệu độc lập với vòng đời của *container.*

+ Volume là cơ chế để duy trì dữ liệu được tạo ra bởi các *container.*Volume có thể hoạt động trong nhiều môi trường *container, network, Windows,…*

+ Cho phép lưu trữ khối lượng trên máy chủ từ xa hoặc từ các nhà cung cấp đám mây, để mã hóa nội dung của khối lượng hoặc thêm chức năng khác

+ Volume sử dụng tuyên truyền liên kết *private.*

2. Distribution tools

- *Docker Registry:*

+ *Registry* là một công cụ phân phối image, giúp lưu trữ và quản lí các image.

+*Registry* là một ứng dụng phía máy chủ không trạng thái, có khả năng mở rộng cao, kiểm soát chặt chẽ các image đang lưu trữ.

- *Docker Trusted Registry:*

+Là một công cụ trả phí có giao diện quản lí và cung cấp một số tính năng bảo mật.

+ Có khả năng lưu trữ hình ảnh gần hơn với người dùng để giảm lượng băng thông, được sử dụng khi pulling Docker Image.

+ Clean up các biểu hiện hoặc lớp không được ước tính.

- *Docker Hub:*

+ Là nơi chứa và chia sẻ các files images, xác thực người dùng

+ Xây dựng hình ảnh tự động và các công cụ dòng công việc như kích hoạt xây dựng và móc web.

+ Tích hợp với *GitHub* và *Bitbuckett.*

+ *Docker Client* có thể sử dụng Docker Hub nếu không có *Registry* nào được cấu hình, nói cho dễ hiểu Hub như một thư viện chứa các Image Offical của các phần mềm (*software)* như *nginx, mongodb, mysql,…*

3. Orchestration Tool

- *Docker Machine:*

+ Là một *Privisioning tools* tạo ra các Docker Engine trên máy chủ của bạn hoặc trên các dịch vụ *cloud* Docker Machine sẽ tạo các máy ảo và cài Docker Engine lên chúng và cuối cùng nó sẽ cấu hình Docker Client để giao tiếp với Docker Engine một cách bảo mật.

+ Docker Machine có thể cấu hình các deploy,cài đặt và chạy Docker trên Mac hoặc Windows, cung cấp và quản lí máy chủ từ xa.

- *Docker Swarm:*

+ Docker Swarm giúp người dùng tạo ra một Docker phân cụm (*Clustering Docker)* và tập hợp các Docker Engine nhằm hướng người dùng nhìn các Docker Engine như một *Virtual Docker Engine* duy nhất. Đồng thời các *tool* đều có khả năng giao tiếp với *Swarm* nếu như có thể giao tiếp với Docker Engine theo chuẩn của *Docker API,* một cụm *Swarm* được cấu hình và triển khai thông qua Docker Machine.

- *Docker Compose:*

+ Là một công cụ điều phối (*orchestration tool)* tạo ra multi-container hay tạo một *Single – Network* cho các ứng dụng của người dùng và các container có thể truy cập lẫn nhau thông qua mạng vừa tạo.

4. Management Tools

- *Docker Universal Control Plane (UCP)*

+ Là giải pháp quản lí cụm cấp doanh nghiệp từ Docker. Bạn cài đặt nó tại chỗ hoặc trong đám mây riêng ảo của mình và nó giúp bạn quản lí cụm Docker và các ứng dụng thông qua một giao diện duy nhất.

- *Docker Datacenter AWS Quickstart:*

+ Là trung tâm dữ liệu lưu giữ các trình quản lí và công cụ trả phí.

+ Sử dụng các mẫu CloudFormation và các mẫu dựng sẵn trên Azure Marketplace để giúp triển khai môi trường Dock CaaS doanh nghiệp trên cơ sở hạ tầng đám mây công cộng dễ dàng hơn.

- *Docker Cloud:*

+ Cung cấp dịch vụ đăng kí được lưu trữ với các phương tiện xây dựng và thử nghiệm cho hình ảnh ứng dụng Dockerized; các công cụ để giúp bạn thiết lập và quản lí cơ sở hạ tầng máy chủ; và các tính năng vòng đời ứng dụng để tự động hóa các dịch vụ triển khai (và triển khai lại) được tạo từ hình ảnh.

5. Tools For Local Enviroment.

- *Docker Toolbox:*

+ Do Engine dùng một số feature của kernel Linux nên ta sẽ không thể chạy Docker Engine natively trên Windows hoặc BSD được, vì vậy trên các máy chủ Windows hoặc Mac ta sẽ phải cần một máy ảo với phiên bản Linux theo các bước *Docker Client -> Vitual Box (VM Linux) -> Docker Engine* thay vì trực tiếp từ *Docker Client -> Docker Engine* như trên Linux.

- *Docker for Mac and Docker for Windows*

+ Trên máy chủ Mac thì Docker Engine sẽ chạy trên xhyve Virtual Machine (VM), xhyve là một giải pháp ảo hóa *lightweight* trên OSX, và distribution chạy trên nền ảo hóa này là Alpine Linux, một distribution vô cùng nhỏ ngọn.

+ Trên máy chủ Windows thì Docker Engine cũng sẽ trên một máy ảo Alpine Linux, trên công nghệ ảo hóa của Windows là *Hyper-V* .

III. Cơ chế hoạt động của Docker

1. Kernel (nhân)

- *Kernel* chạy trực tiếp trên phần cứng và đảm nhận rất nhiều nhiệm vụ khác nhau:

+ Phản hồi các thông điệp từ phần cứng như 1 ổ đĩa mới được kết nối hoặc 1 gói tin từ mạng được chuyển đến,…

+ Khởi động và đặt lịch các chương trình, quyết định tiến trình nào chạy lúc nào.

+ Quản lí hệ thống các tác vụ.

+ Truyền tin giữa các chương trình.

+ Phân chia tài nguyên, bộ nhớ, CPT, Network,…

- Vì vậy *Docker* tạo *Container* bằng cách chỉnh thiết lập của *kernel.*

2. Docker là chương trình quản lí Kernel

- Docker được viết bằng ngôn ngữ *GO (golang* là một ngôn ngữ dành cho hệ thống)

- Docker quản lí các đặc tính của *kernel* và dùng những đặc tính đó để tạo ra định nghĩa về *container* và *image.*

+ Docker sử dụng "*cgroup"* (*control group)* để nhóm các tiến trình lại với nhau và bao lấy các tiens trình cùng nhóm trong 1 khoảng không gian ảo riêng. Chính vì điều này mà các container không thể can thiệp lẫn nhau tạo nên sự tối ưu khi hoạt động của các container.

+ Docker sử dụng "*namespace" (namespace* là một chức năng của Kernel Linux) để chia tách các tầng network.namespace có từng địa chỉ cho từng container và các địa chỉ chưa thuộc container.

+ Docker sử dụng cơ chế "*copy-on-write"* và một số hướng tiếp cận khác để tạo nên định nghĩa về image, nghĩa là bạn có thể chạy các thứ khác ngay bên trên image này mà không làm thay đổi cấu trúc cũng như cấu hình mặc định của nó.

- Tuy nguyên tắc hoạt động không hề mới nhưng docker mang lại một hướng tiếp cận đơn giản hơn, dễ sử dụng hơn và có những lợi điểm khắc phục được một số hạn chế trước đó nên Docker thực sự đã tạo ra một cuộc cách mạng.

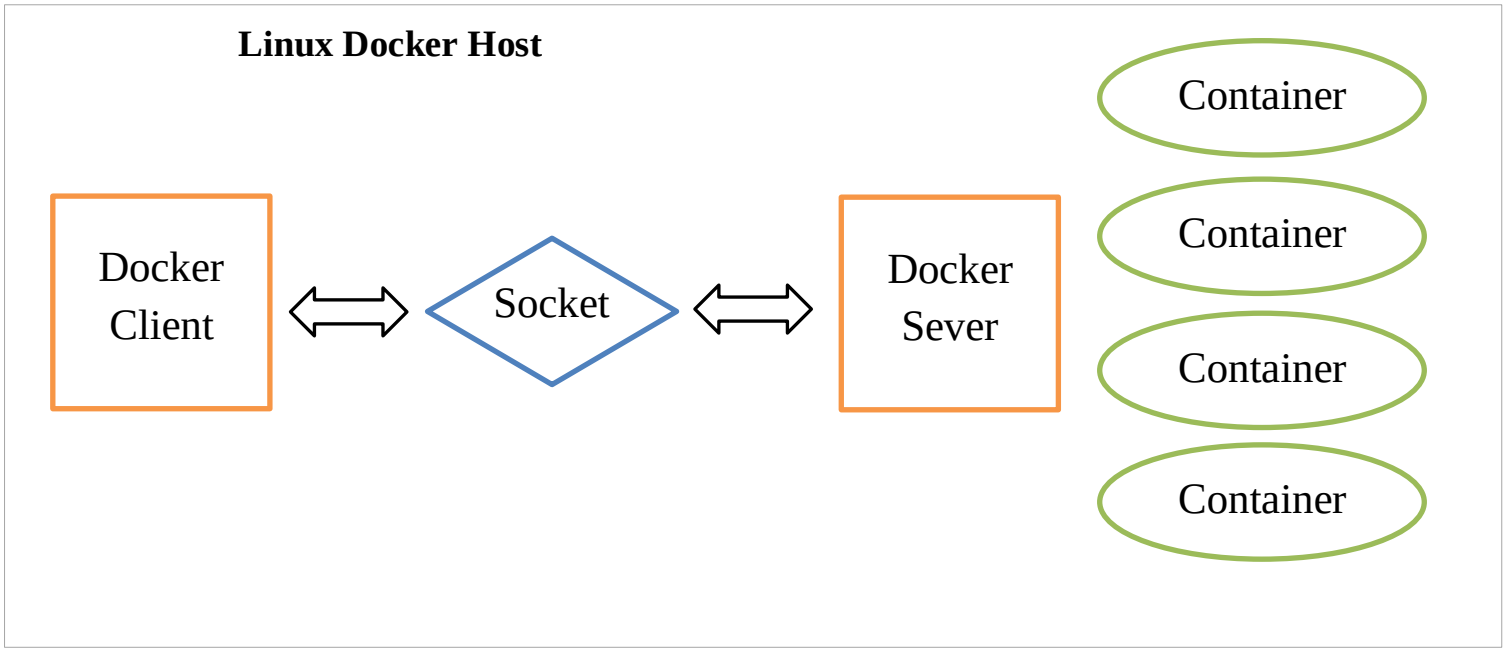
3. Socket điều khiển của Docker.

- Docker được chia thành 2 phần lớn là *Client* và *Sever.*

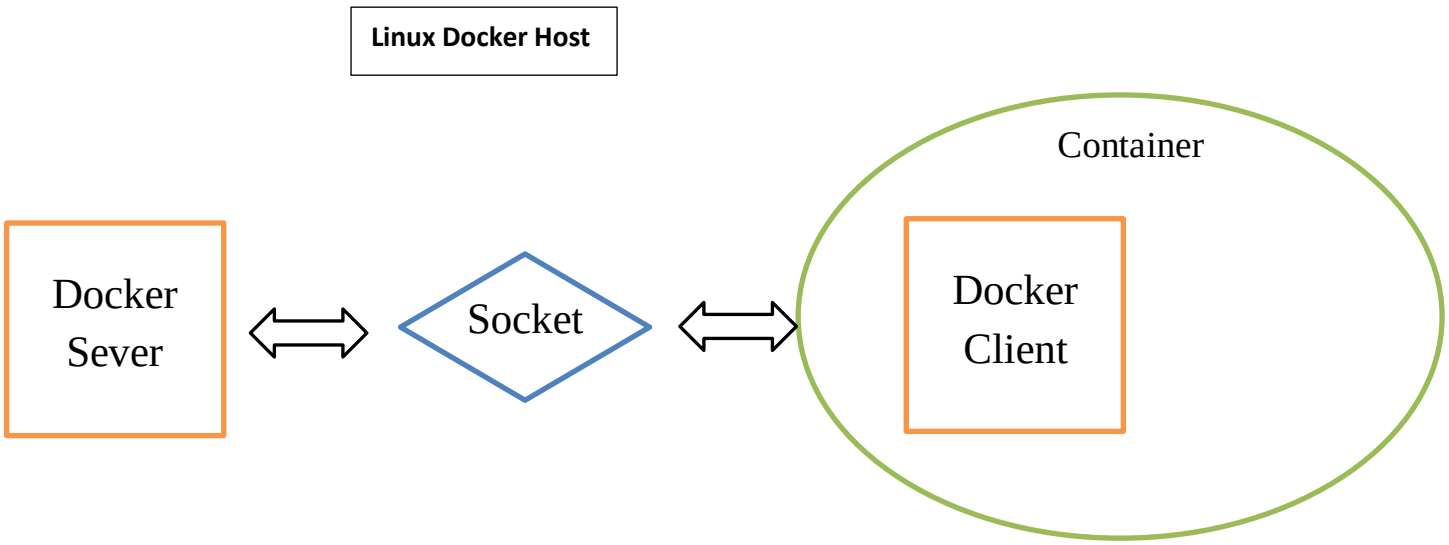
- *Client – Sever* giao tiếp với nhau qua *socket.* Network Socket ở máy mà Client dang chạy có thể giao tiếp với Sever ở máy khác như trên đám mây (*Cloud*)

hoặc giao tiếp trực tiếp với Sever trên cùng 1 máy với Client hoặc Sever chạy trên một máy ảo.

- Client sẽ gửi thông điệp đến Docker Sever để thực hiện các yêu cầu như tạo mới một container, chạy một container có sẵn, dừng một container hoặc xóa một container,… Bằng cách thực hiện chuỗi thực thi *Build -> Push -> Pull, Run* (chuỗi hoạt động này sẽ được phân tích rõ hơn ở phần sau).
- Khi Client và Sever ở cùng một máy chúng có thể kết nối với nhau thông qua một file đặc biệt gọi là *socket* và nhờ cùng giao tiếp thông qua một file duy nhất nên chúng có thể dễ dàng chia sẻ giữa *Host* và *Container*, điều này đồng nghĩa với việc có thể chạy Client ở bên trong Docker (điều này sẽ được minh họa rõ hơn qua hai *userkey* dưới đây).

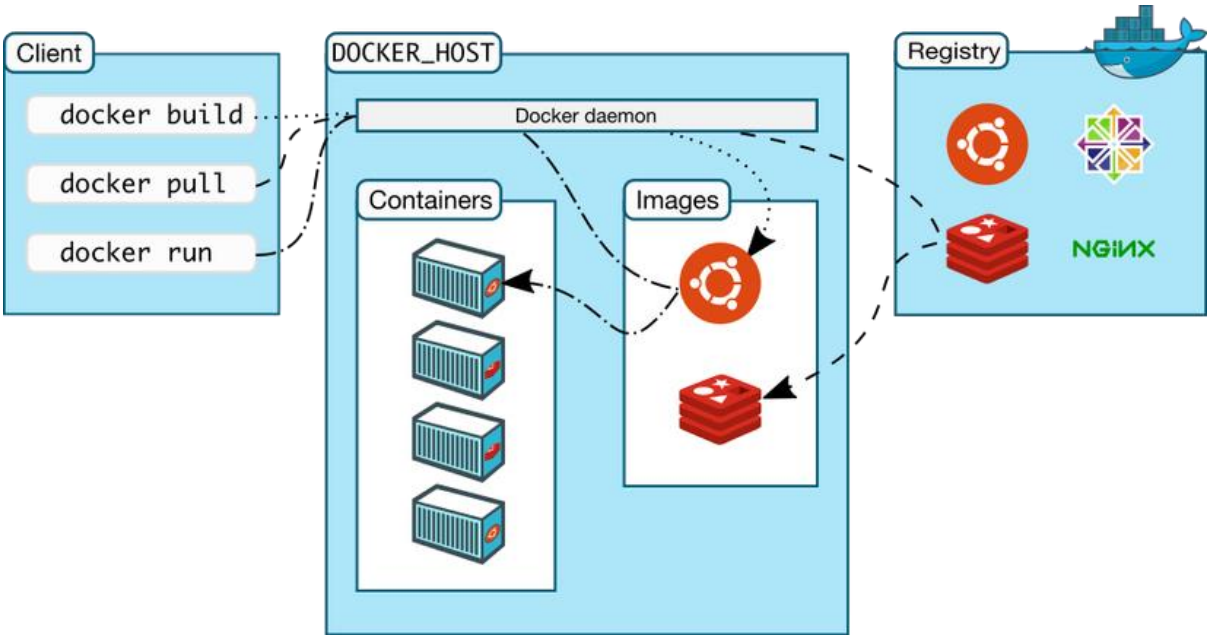


+ Ở user key này trong một *Linux Docker Host* người dùng có chương trình Docker Client kết nối với Socket, Docker Client tiến hành gửi lệnh yêu cầu thông qua socket này đến Docker Sever, sau khi nhận được thông điệp từ docket Docker Sever tiến hành xử lí theo yêu cầu, ví dụ như tạo, chạy, xóa một container nào đó.



+ Ở userkey này người dùng có thể chạy Docker Client ngay bên trong contianer cùng chia sẻ socket trong container điều này cho phép gửi cùng một thông điệp đi qua cùng một socket đến Docker Sever để thực hiện các công việc như bình thường.

4. Kiến trúc của Docker

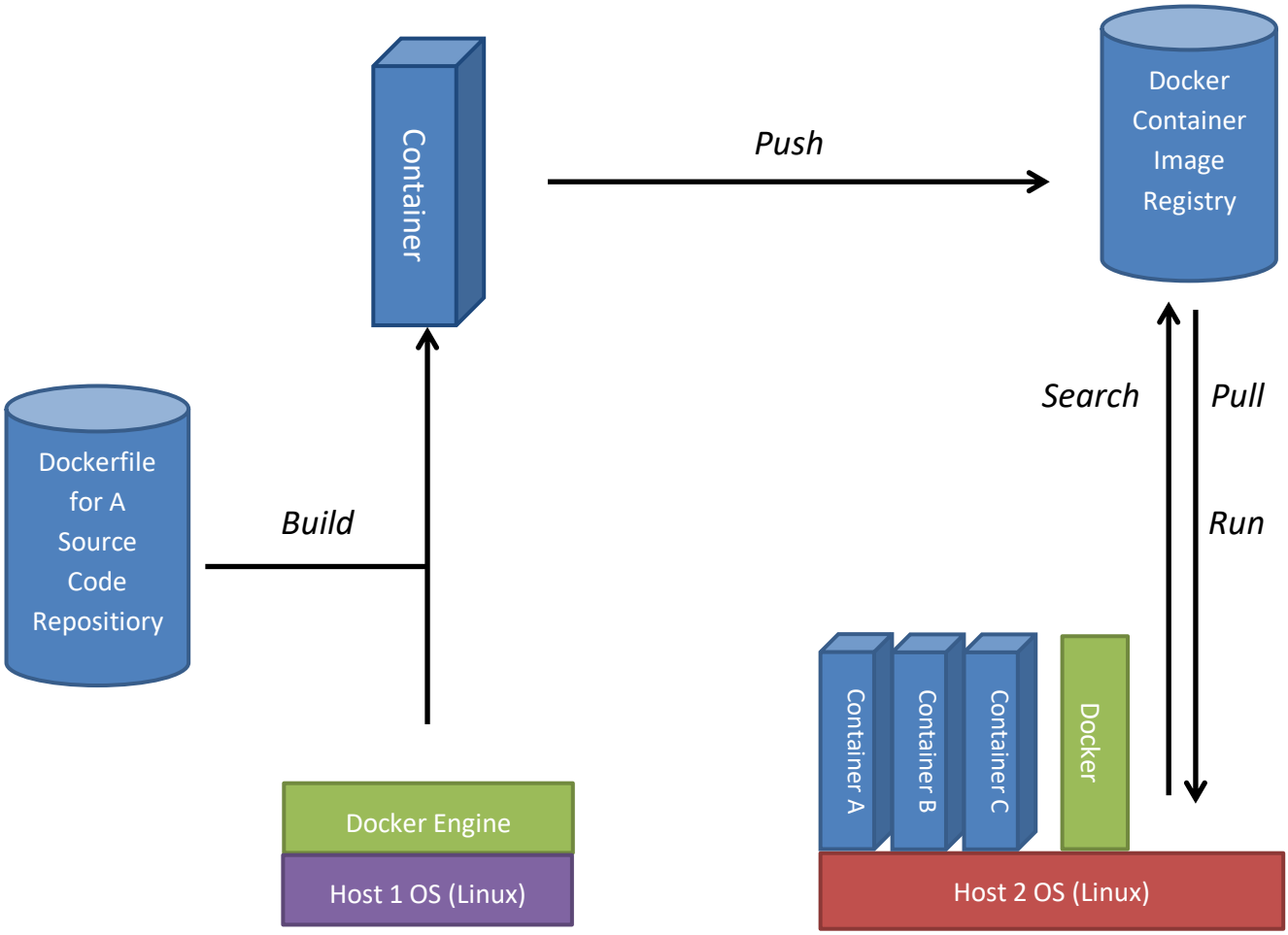


- Docker sử dụng kiến trúc *Client – Sever*. Docker Client sẽ nói liên lạc với các Docker Daemon, các Docker Daemon sẽ thực hiện các tác vụ build, run và distribuing các Docker container. Cả Docker Client và Docker daemon có thể

chạy trên cùng một máy, hoặc có thể kết nối theo kiểu Docker Client điều khiển các Docker Daemon như hình phía trên.

- Docker client và daemon giao tiếp với nhau thông qua socket hoặc RESTful API vì Docker daemon chạy trên Host nên người dùng sẽ không tương tác trực tiếp với daemon mà thông qua Docker Client (Docker Client có thể chạy trên cùng Host hoặc khác Host với Docker daemon).

5. Quy trình '*Build –> Push –> Pull, Run*' trong Docker



- Mô hình trên là một hệ thống Docker được thực thi qua ba bước chính: *Build –> Push –> Pull, Run.* Nguyên lí hoạt động của ba bước này như sau:

+ *Bước 1 – Build*: Người dùng sẽ tạo ra một Dockerfile (đây là nơi chứa các câu lệnh mà chúng ta muốn thực thi). Dockerfile sẽ được build tại một máy tính đã cài đặt Docker Image, sau khi build người dùng sẽ có được một container (container này chứa toàn bộ thư viện và ứng dụng của người dùng).

+ *Bước 2 – Push:* Sau khi có Container hệ thống sẽ push Container này lên đám

mây (Cloud) và lưu trữ ở đó, việc push này có thể thực hiện thông qua môi trường mạng Internet.

+ *Bước 3 – Pull, Run:* Sau đó nếu người dùng muốn sử dụng container mà mình vừa push lên đám mây thì thực hiện pull container đó về máy sau đó thực hiện run container này (đối với các Client khác nếu muốn sử dụng container này thì phải cài đặt Docker Image trước).

Phần 3: Cài đặt và sử dụng

I. Cài đặt

1. Kiểm tra Curl và Proxy

- Chúng ta sẽ cài đặt Docker trên Ubuntu phiên bản 18.04
- Chúng ta sẽ cài đặt Docker với bản CE trên Ubuntu phiên bản 18.04 thông qua Repository
- Đăng nhập vào User
- Kiểm tra cài đặt của Curl và Proxy bằng câu lệnh sau

$ which curl

Nếu Curl chưa được cài đặt thì ta cài đặt bằng câu lệnh sau:

$ sudo apt-get update

$ sudo apt-get install curl

- Tải phiên bản mới nhất của Docker bằng câu lệnh sau:

$ curl – sSL https://get.docker.com/ | sh

Nếu hệ thống yêu cầu quyền admin hãy thực thi câu lệnh trên bằng lệnh:

sudo.

Trường hợp nếu bị lỗi proxy thì thực hiện câu lệnh sau đây trước khi thực hiện câu lệnh trên:

$ curl -sSL https://get.docker.com/gpg | sudo apt-key add –

- Update the apt package index

sudo apt-get update

- Cài đặt các gói

*$ sudo apt-get install *
*apt-transport-https *
*ca-certificates *
*curl *
software-properties-common

- Thêm khóa chính thức của Docker

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
pub 4096R/0EBFCD88 2017-02-22
Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid Docker Release (CE deb) <docker@docker.com>
sub 4096R/F273FCD8 2017-02-22

- Thiết lập kho lưu trữ bằng câu lệnh sau:

*sudo add-apt-repository *
*"deb [arch=amd64] https://download.docker.com/linux/ubuntu *
*$(lsb_release -cs) *
stable"

2. Cài đặt Docker CE

- Cập nhật gói atp

$ sudo apt-get update

- Cài đặt phiên bản mới nhất của Docker CE

$ sudo apt-get install docker –ce

- Sau khi cài đặt, xác minh Docker CE đã cài đặt thành công bằng cách: Chạy image hello word

$ sudo docker run hello-word

3. Ví dụ minh họa

- Pull một imgae có tên nginx từ Docker Hub

sudo docker pull nginx

- Tạo mới container bằng cách chạy image nginx

sudo docker run –i –t –nginx

- Kiểm tra image nginx bằng câu lệnh:

docker image

II. Một số câu lệnh trong Docker

1. Pull một image từ Docker Hub

sudo docker pull image_name

2. Tạo mới container bằng cách chạy image, kèm theo các tùy chọn:

sudo docker run -v <forder_in_computer>:<forder_in_container> -p <port_in_computer>:<port_in_container> -it <image_name> /bin/bash

3. Chạy một image

sudo docker run –i –t –image_name

Ví dụ: *sudo docker pull nginx*

sudo docker run –i –t inginx

4. Hiện thị thông tin Kernel Ubuntu

uname -a

5. Liệt kê các image hiện có

docker image

6. Xóa một image

docker rmi {image_name}

7. Liệt kê các container đang chạy

docker ps

8. Liệt kê các container đã tắt

docker ps –a

9. Xóa một container

docker rm –f {container_name}

10. Chạy một container

docker start {new_container_name}