Understanding of Cloud Computing, Docker and Container

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy/Docker-Container

Contents

- Cloud Computing Concepts
- Virtualization Concepts
- Container Concepts
- Docker Concepts

실습 환경

- Docker on Ubuntu 20.04 LTS at Oracle VirtualBox
- Docker for Windows + WSL2 on Windows 10
- Docker on Ubuntu 20.04 LTS at AWS EC2
- SSH Client Tools : PuTTY, Xshell etc.
- MySQL Workbench CE
- Google Chrome or Mozilla Firefox

References

- TecMint Docker
- [가장 빨리 만나는 도커] 원고
- Docker 간단 개념 / Docker를 이용한 MSA 기반의 Spring Boot 프로젝트
- An Introduction to Docker
- Qemu를 이용한 가상화 기초
- 리눅스 컨테이너란?
- Docker와 Linux 컨테이너 기술들
- Docker와 Container의 탄생과 설명, 차이점
- 컨테이너란 무엇인가
- Ralf Yang Github
- Ubuntu LXC
- Dial D for Docker
- 왜 도커인가? 도커와 리눅스 컨테이너의 이해
- Docker란?

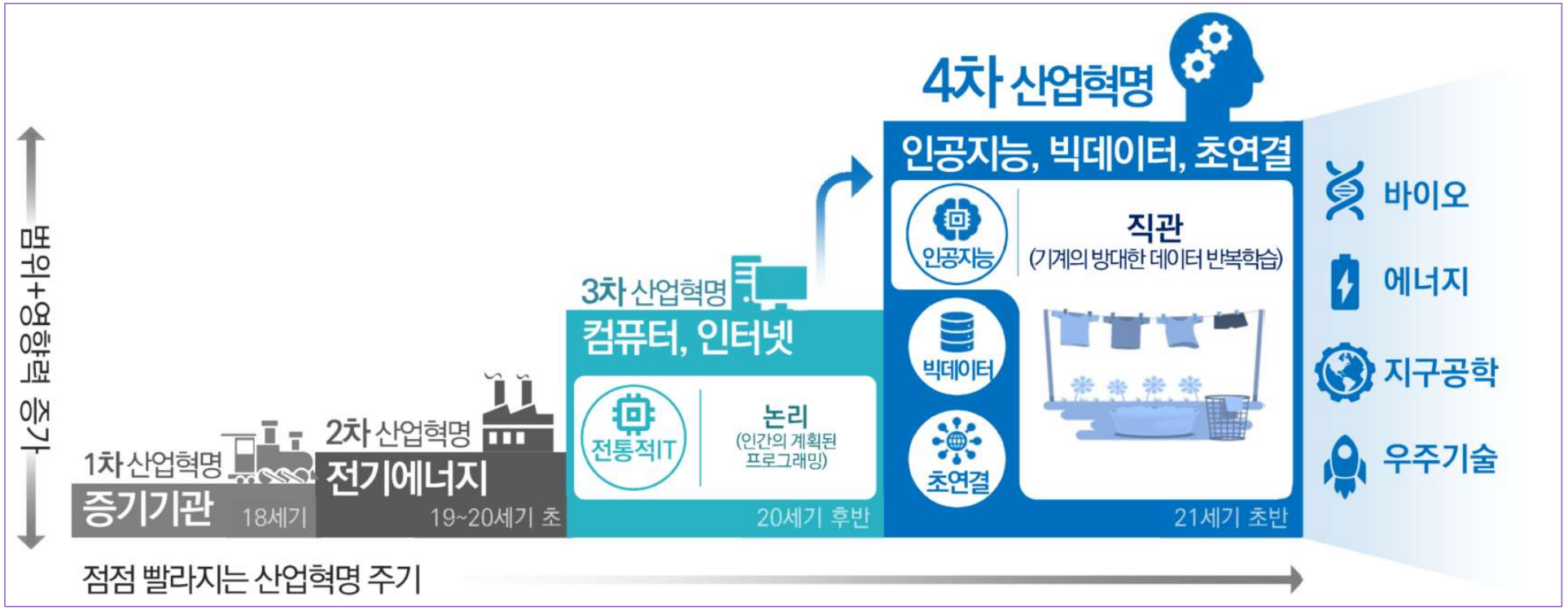
References (Cont.)

- 생활코딩 [Docker 입문]
- 44BITS
- 도커 컴포즈를 활용하여 완벽한 개발 환경 구성하기
- Glorious-Coding Docker 강좌
- 따라하면서 배우는 도커 강좌
- 인프런 [입문에서 실무까지:DevOps의 이해 및 Docker Hands-on]
- 인프런 [2021년] 제발 도커 씁시다!
- 인프런 [초보를 위한 도커 안내서]
- 인프런 [도커 쓸 땐 필수! 도커 컴포즈]

Cloud Computing Concepts

Now is…

4th Industrial Revolution



4차산업혁명위원회 홈페이지(https://d2b38vv9jledd1.cloudfront.net/home.pdf)

Now is…(Cont.)

4th Industrial Revolution

지능정보기술
(AI, IoT, Cloud Computing, Mobile 등)

+

기존 산업 및 서비스

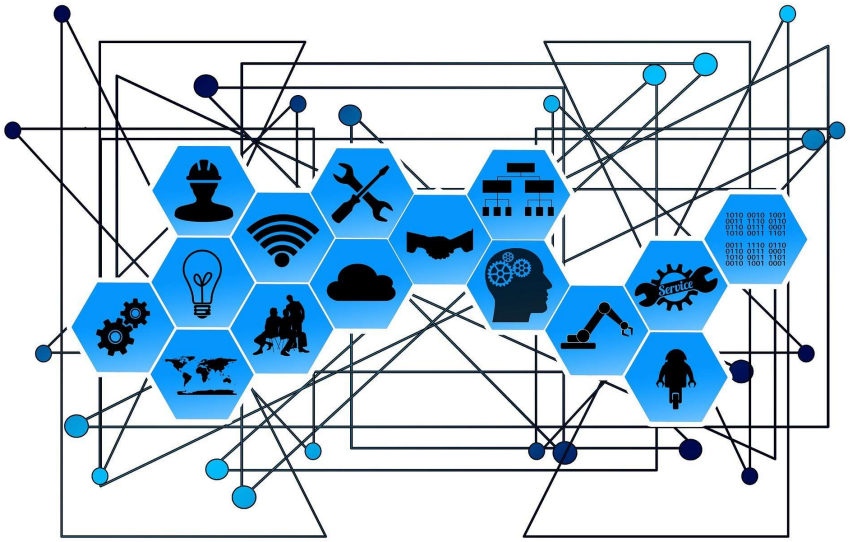
+

新 기술
(3D Printing, Robotics, Bio/Nano Tech 등)

실세계 모든 제품 및 서비스의
네트워크 연결, 사물 지능화

Now is…(Cont.)

4th Industrial Revolution



- **I** oT
- **C** loud Computing
- **B** ig Data
- **M** achine Learning(AI)

New Technology Trends

2007년	2008년	2009년
Multicore Processors	Multicore	Cloud Computing
Web 2.0	Web Platforms	Web-Oriented Architectures
Compute Utilies	User Interface	Servers-Beyond Blades
Information Access	Web Mashups	Enterprise Mashups
Ubiquitous Computing	Social Software	Social Software and Social Networking
Grid Computing	Tera-Architecture	Specialized Systems
Open Source	Power and Green IT	Green IT
Network Convergence	Networked Virtual World	Unified Communications
Virtualization	Video	Virtualization
Water Cooling	Semantics	Business Intelligence

http://www.nextdaily.co.kr/news/article.html?id=20081110800135

2010
Cloud Computing
Advanced Analytics
Client Computing
IT for Green
Reshaping the Data Center
Social Computing
Security-Active Monitoring
Flash Memory
Virtualization for Availability
Mobile Applications

	2011년	2012년	2013년	2014년	2015년	2016년
1	클라우드 컴퓨팅	미디어 태블릿 그 이후	모바일 대전	다양한 모바일 기기 관리	컴퓨팅 에브리웨어	디바이스메시 The Device Mesh
2	모바일 앱과 미디어 태블릿	모바일 중심 애플리케이션과 인터페이스	모바일 앱&HTML5	모바일앱과 애플리케이션	사물인터넷	주변 사용자 경험 Ambient User Experience
3	소셜 커뮤니케이션 및 협업	상황인식과 소셜이 결합된 사용자 경험	퍼스널 클라우드	만물인터넷	3D프린팅	3D프린팅 소재 3D-Printing Materials
4	비디오	사물인터넷	사물인터넷	하이브리드 클라우드와 서비스브로커로서의 IT	보편화된 첨단분석	만물정보 Information of Everything: IoE
5	차세대 분석	앱스토어와 마켓 플레이스	하이브리드IT& 클라우드 컴퓨팅	클라우드/클라이언트 아키텍처	콘텍스트 리치 시스템	진보된 기계 학습 Advanced Machine Learning
6	소셜 분석	차세대 분석	전략적 빅데이터	퍼스널 클라우드의 시대	스마트 머신	자율 지능형 기기 Autonomous Agents & Things
7	상황인식 컴퓨팅	빅데이터	실용 분석	소프트웨어 정의	클라우드/클라이언트 컴퓨팅	능동형 보안 아키텍처 Adaptive Security Architecture
8	스토리지급 메모리	인메모리 컴퓨팅	인메모리 컴퓨팅	웹스케일IT	소프트웨어 정의 애플리케이션과 인프라	진화된 시스템 아키텍처 Advanced System Architecture
9	유비쿼터스 컴퓨팅	저전력 서버	통합 생태계	스마트 머신	웹스케일IT	매시앱 및 서비스아키텍처 Mash app and service Architecture
10	패브릭 기반 컴퓨팅 및 인프라스트럭처	클라우드 컴퓨팅	엔터프라이즈 앱 스토어	3D 프린팅	위험기반 보안과 자가 방어	사물인터넷 아키텍처와 플랫폼 IoT Architecture & Platform

http://blog.naver.com/PostView.nhn?blogId=sysoitdotcom&logNo=220508531433&parentCategoryNo=&categoryNo=66&viewDate=&isShowPopularPosts=true&from=search

What is Cloud Computing?

영어사전 cloud 고급검색

전체 | 단어·숙어 | 뜻풀이 | 예문 | 유의어·반의어 | 영영사전

연관검색어 cloudy clouds crowd cheap 구름 church sky coin rain clock cross
cold moon clothes 클라우드 earth climb confuse darken mist

단어·숙어 921

cloud 미국·영국 [klaʊd] 영국식 ★★

1. 명사 구름 (→storm cloud, thundercloud)
2. 명사 (먼지연기메뚜기 떼 등이 구름같이) 자욱한 것
3. 동사 (기억력판단력 등을) 흐리다

What is Cloud Computing? (Cont.)

영어사전 cloud 고급검색

전체 | 단어·숙어 | 뜻풀이 | 유의어 | 영영사전

연관검색어 cloudy clouds crowd 구름 church sky coin rain clock cross
cold moon clothes 클라우드 confuse darken mist

단어·숙어 921

cloud 미국·영국 [klaʊd]

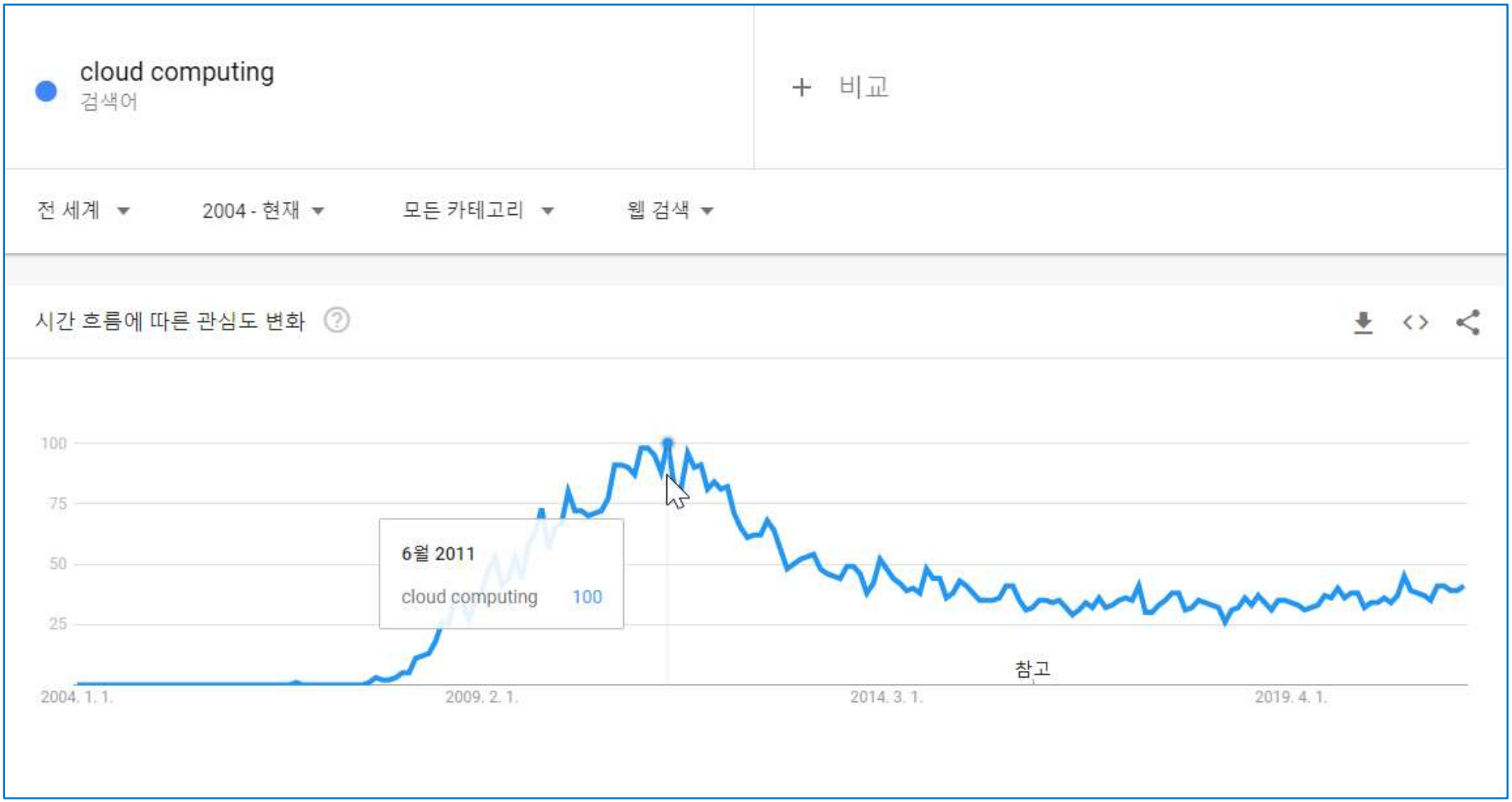
1. 명사 구름 (→storm cloud, thundercloud)
2. 명사 (먼지연기메뚜기 떼 등이 구름같이) 자욱한 것
3. 동사 (기억력판단력 등을) 흐리다

What is Cloud Computing? (Cont.)

정확하게 말하면...

클라우드가 아니라 **클라우드 컴퓨팅**입니다.

What is Cloud Computing? (Cont.)



https://trends.google.com/trends/explore?date=all&q=cloud%20computing

What is Cloud Computing? (Cont.)

Gartner

인터넷 기술을 활용하여 다수의 고객들로부터 높은 수준의 확장성을 가진 자원들을 서비스로 제공받는 컴퓨팅의 한 형태

FORRESTER®

표준화된 IT 기반 기능들이 IP 네트워크를 통해 제공되며, 언제나 접근이 허용되고 수요의 변화에 따라 가변적이며 사용량이나 광고에 기반한 과금 모형을 제공하는 웹 또는 프로그램적인 인터페이스를 제공하는 컴퓨팅

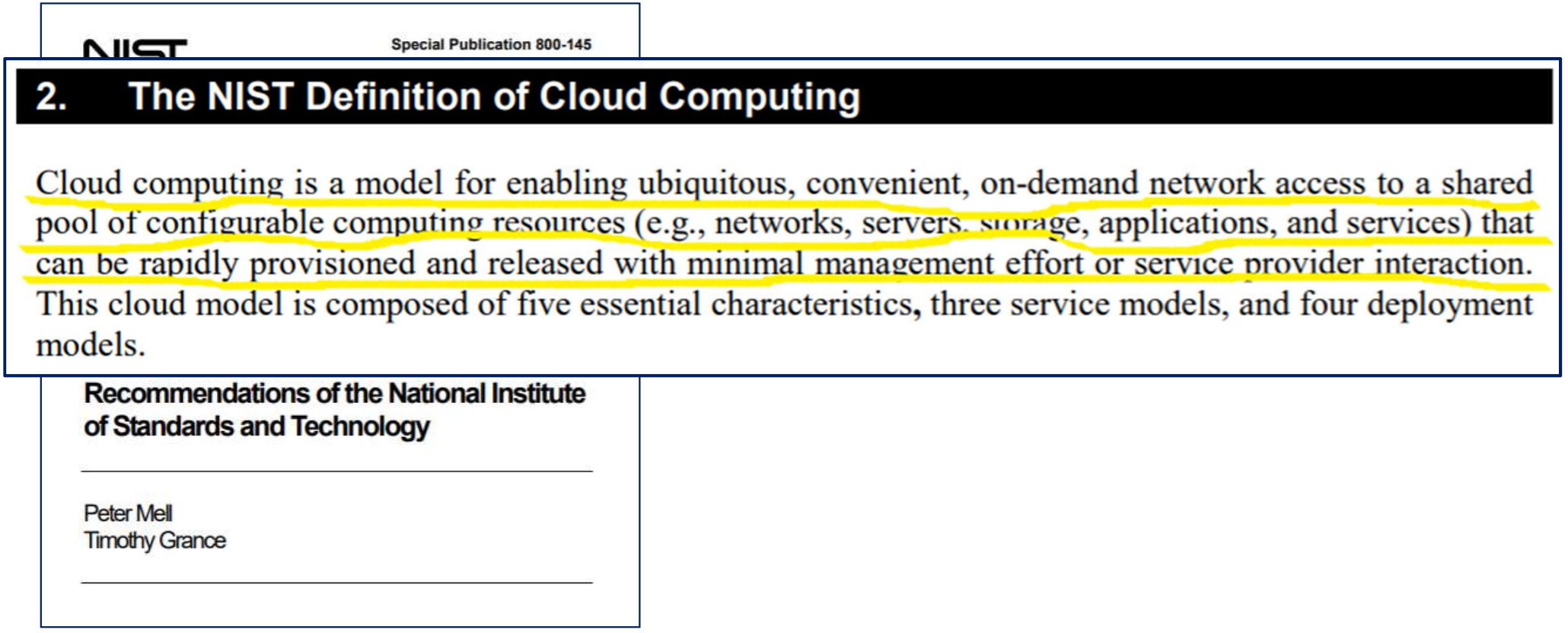
WIKIPEDIA

인터넷에 기반한 개발과 컴퓨터 기술의 활용을 뜻하는 것으로 인터넷을 통해서 동적으로 규모화 가능한 가상적 자원들이 제공되는 컴퓨팅

IBM

웹 기반 어플리케이션을 활용하여 대용량 데이터베이스를 인터넷 가상 공간에서 분산처리하고 이 데이터를 PC, 휴대 전화, 노트북 PC, PDA 등 다양한 단말기에서 불러오거나 가공할 수 있게 하는 환경

What is Cloud Computing? (Cont.)



NIST Special Publication 800-145

2. The NIST Definition of Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

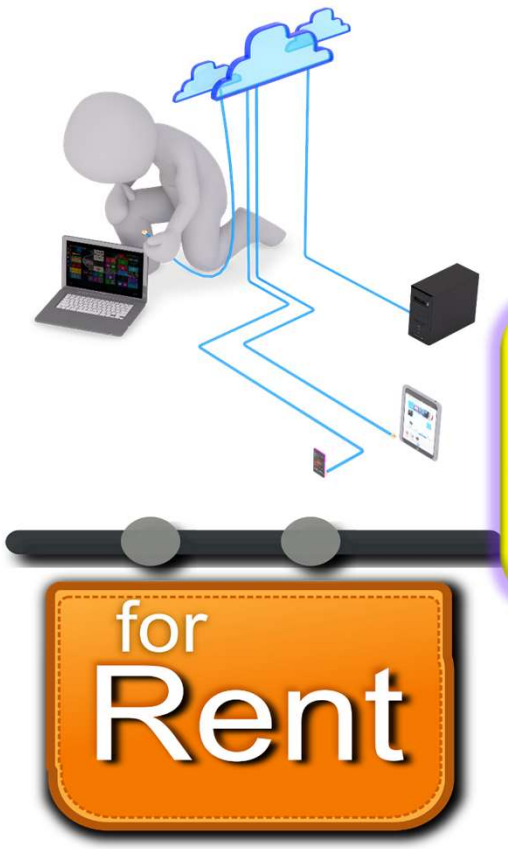
Recommendations of the National Institute of Standards and Technology

Peter Mell
Timothy Grance

What is Cloud Computing? (Cont.)

- **IT 자원을**
- 필요한 만큼 **빌려서** 사용하고
- 서비스 부하에 따라서 **실시간** 확장성을 지원받으며,
- **사용한 만큼 지불**하는 컴퓨팅 패러다임

What is Cloud Computing? (Cont.)

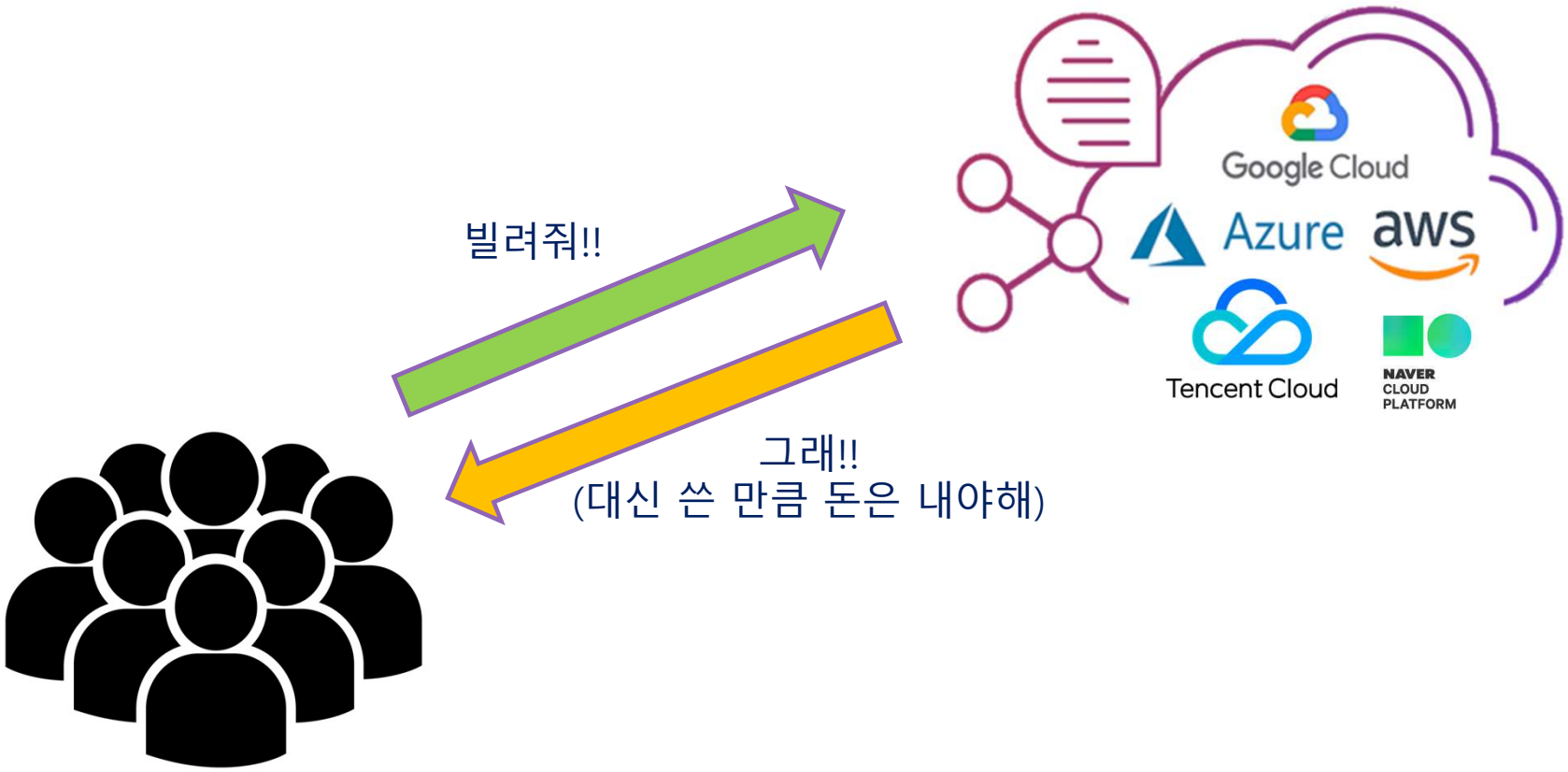


- **IT 자원을**
- 필요한 만큼 **빌려서** 사용하고
- 서비스 부하에 따라서 **실시간** 확장성을

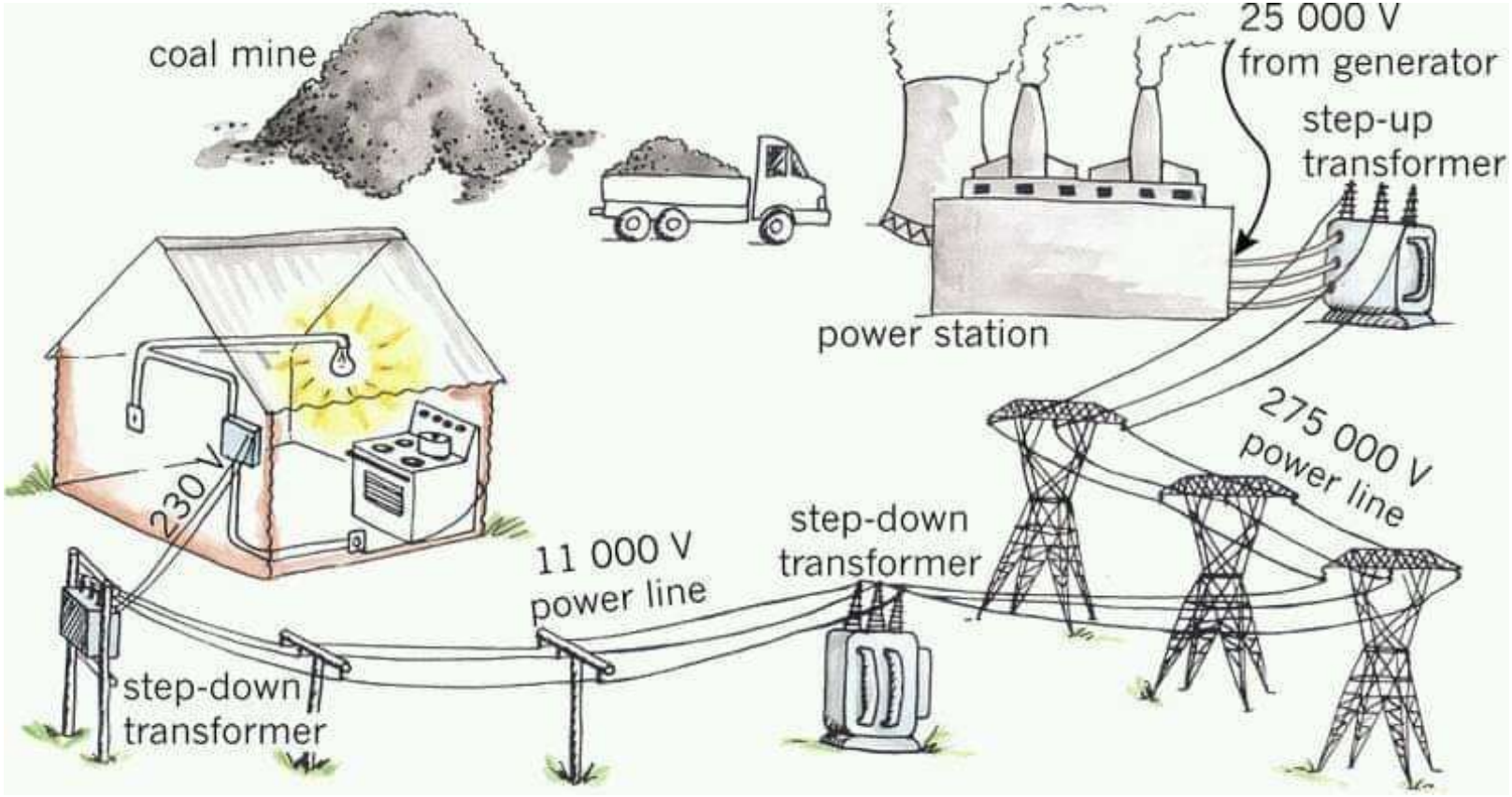
IT 자원의 렌탈샵

컴퓨팅 패러다임

What is Cloud Computing? (Cont.)



What is Cloud Computing? (Cont.)

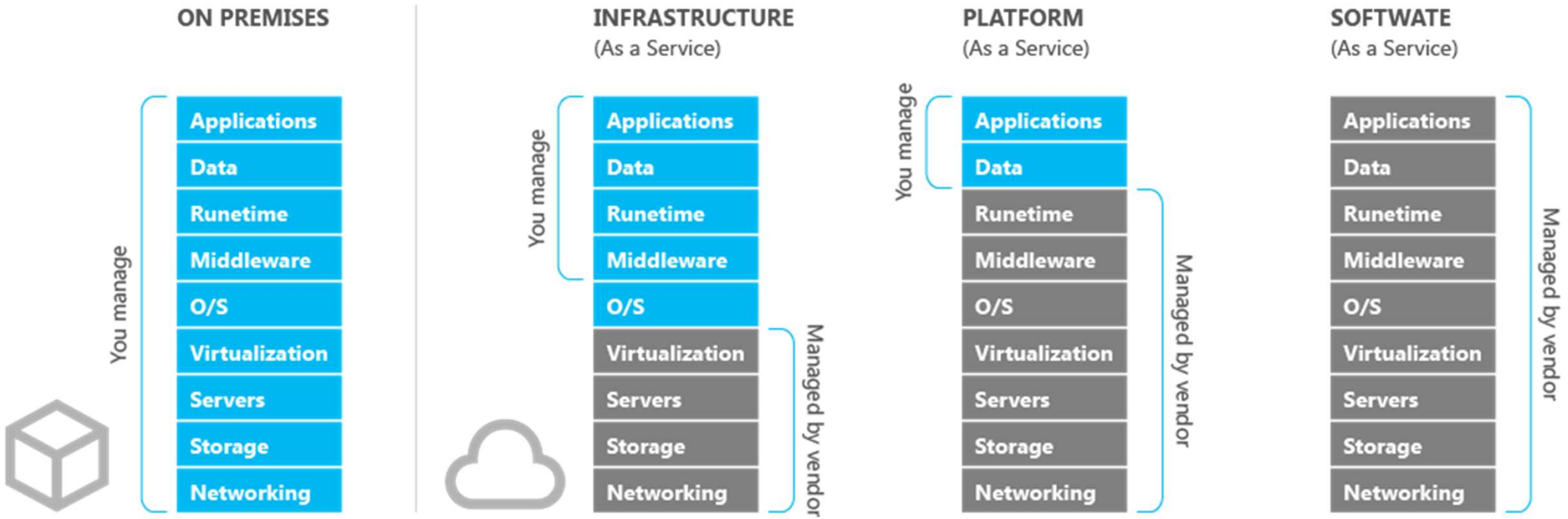


https://www.reddit.com/r/ElectricalEngineering/comments/9xd2j6/electrical_power_generation_transmission/

What is Cloud Computing? (Cont.)

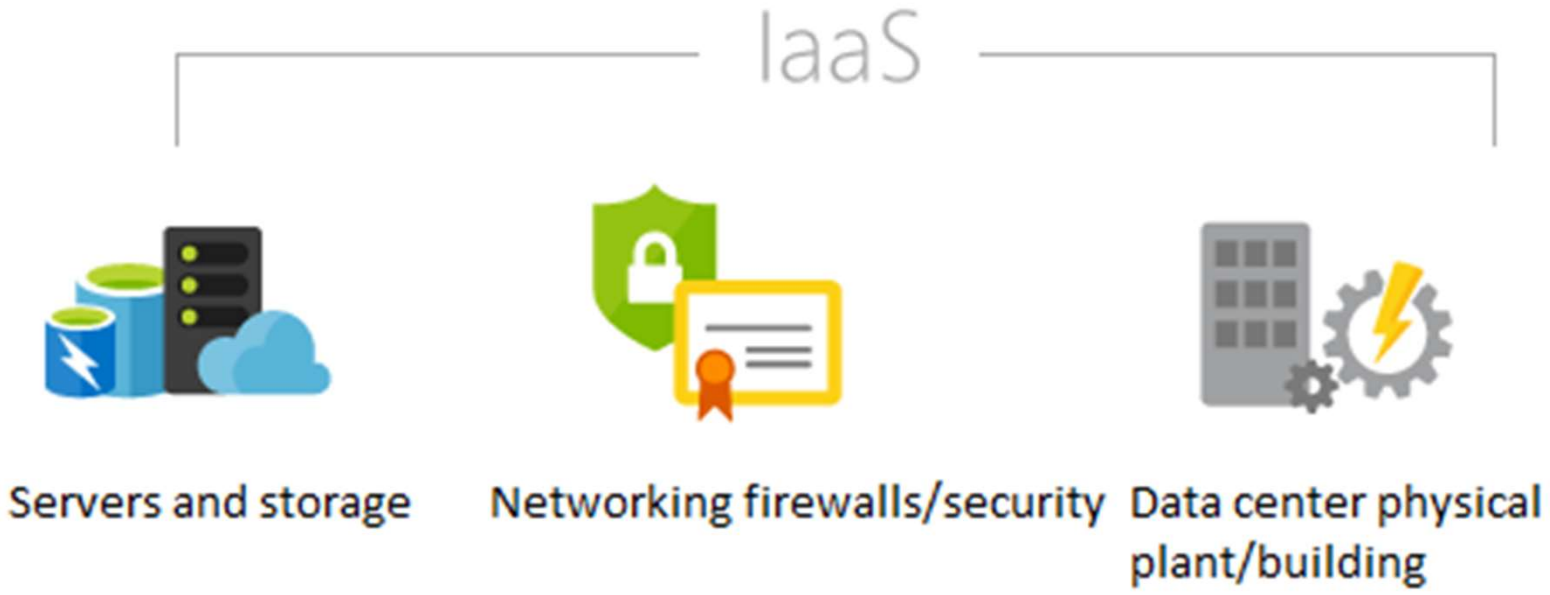
- Essential Characteristics
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
  - Measured service

What is Cloud Computing? (Cont.)



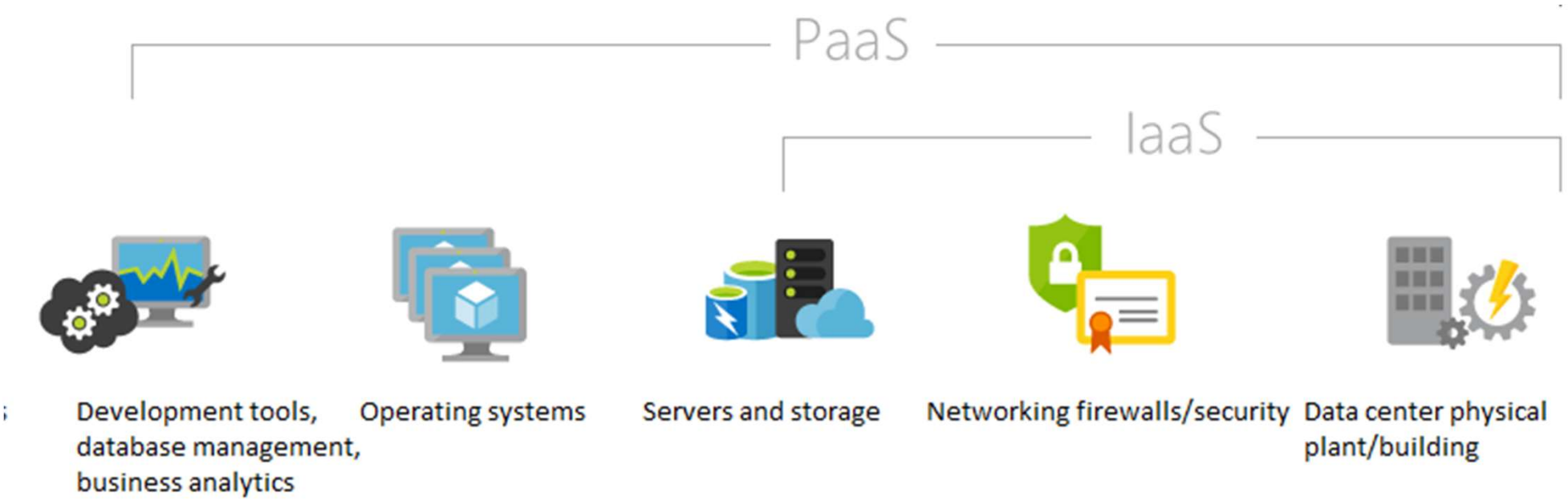
https://velog.io/@aonee/%ED%81%B4%EB%9D%BC%EC%9A%B0%EB%93%9C%EB%9E%80-nzrij240

What is Cloud Computing? (Cont.)



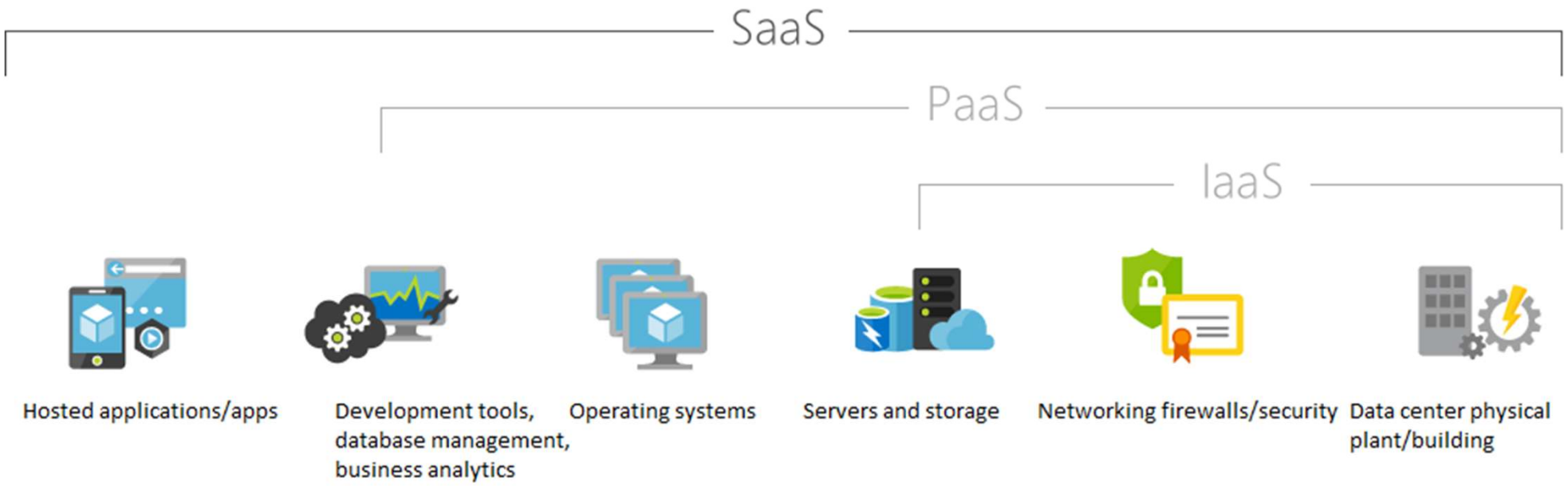
https://cic.gsa.gov/solutions/iaas

What is Cloud Computing? (Cont.)



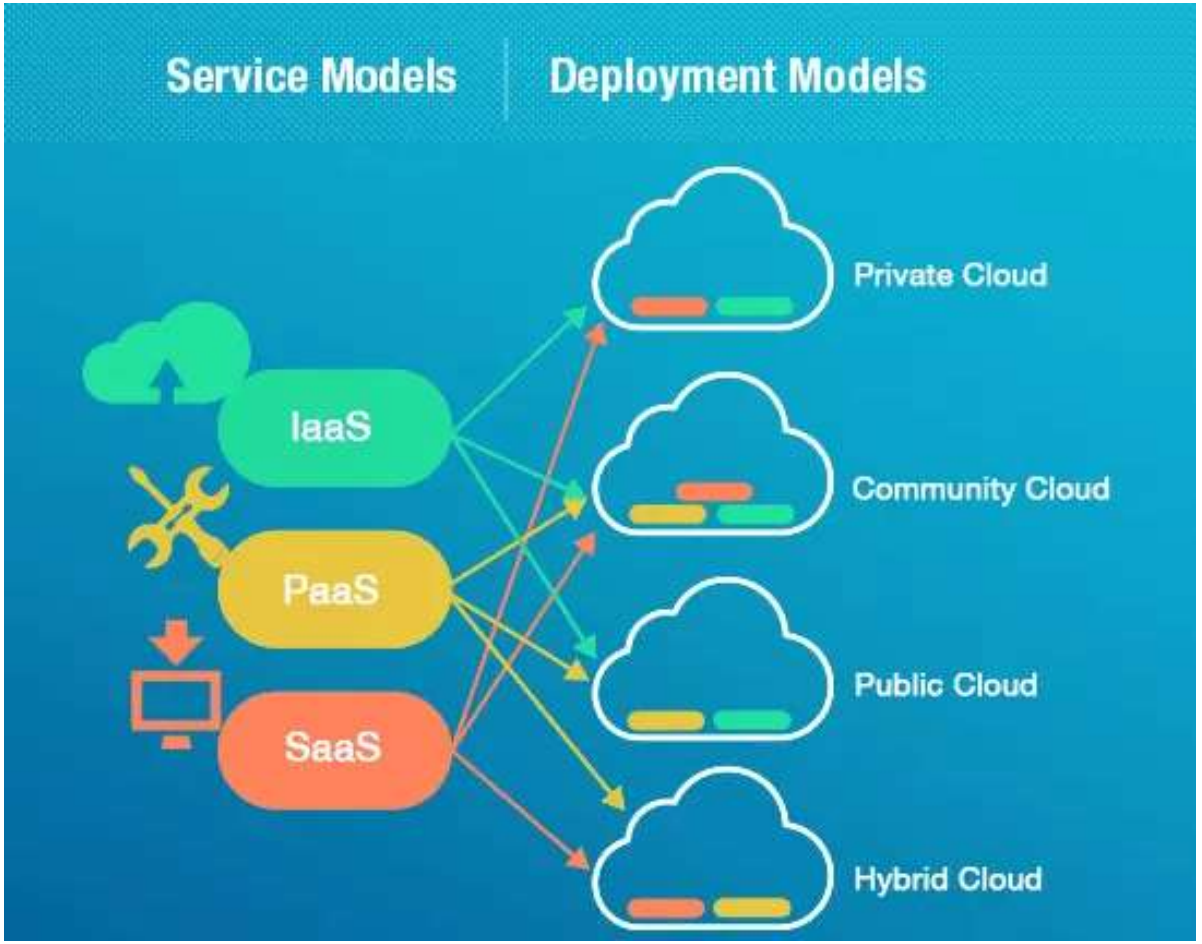
https://cic.gsa.gov/solutions/paas

What is Cloud Computing? (Cont.)



https://cic.gsa.gov/solutions/saas

What is Cloud Computing? (Cont.)



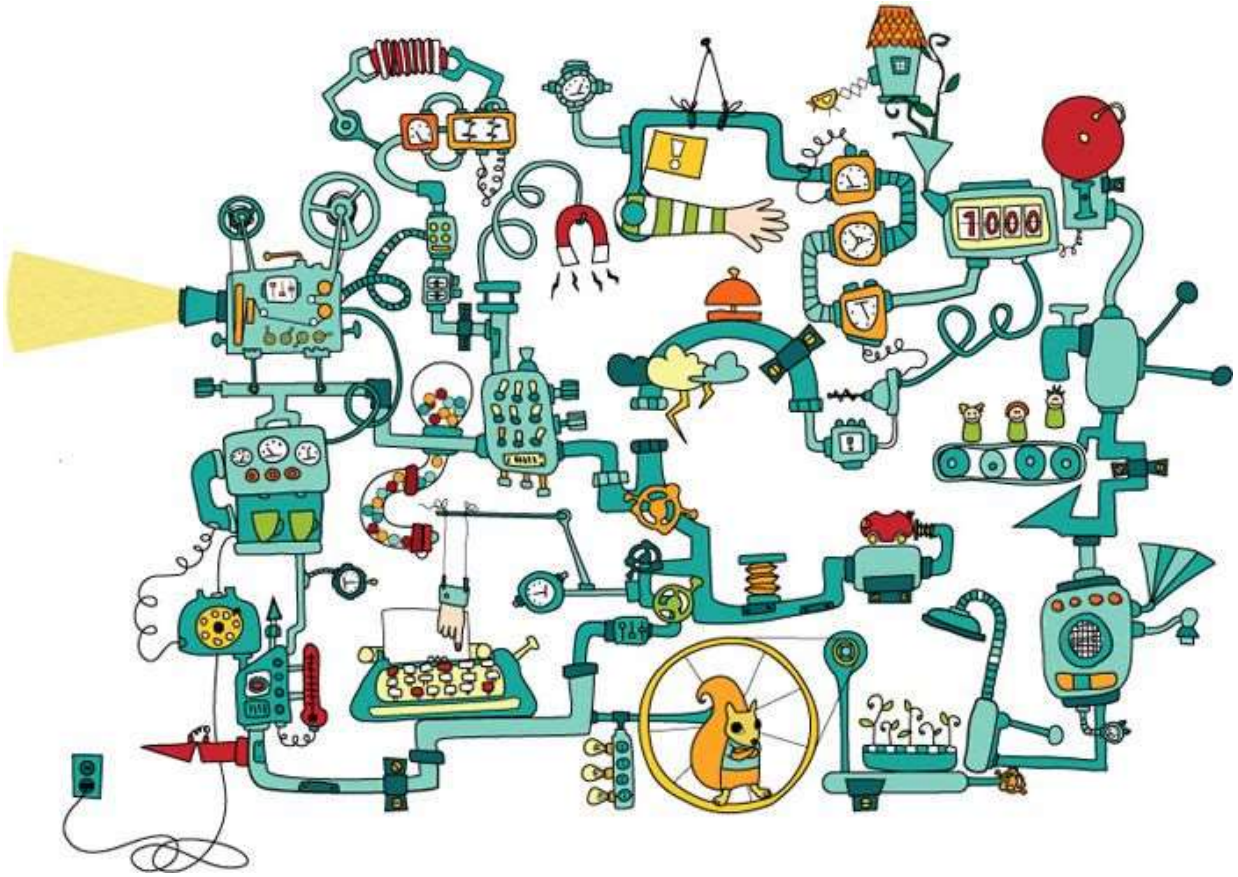
https://www.rishabhsoft.com/blog/basics-of-cloud-computing-deployment-and-service-models

Lab. AWS EC2
가상 머신 생성

Virtualization Concepts

서버를 관리한다는 것은...

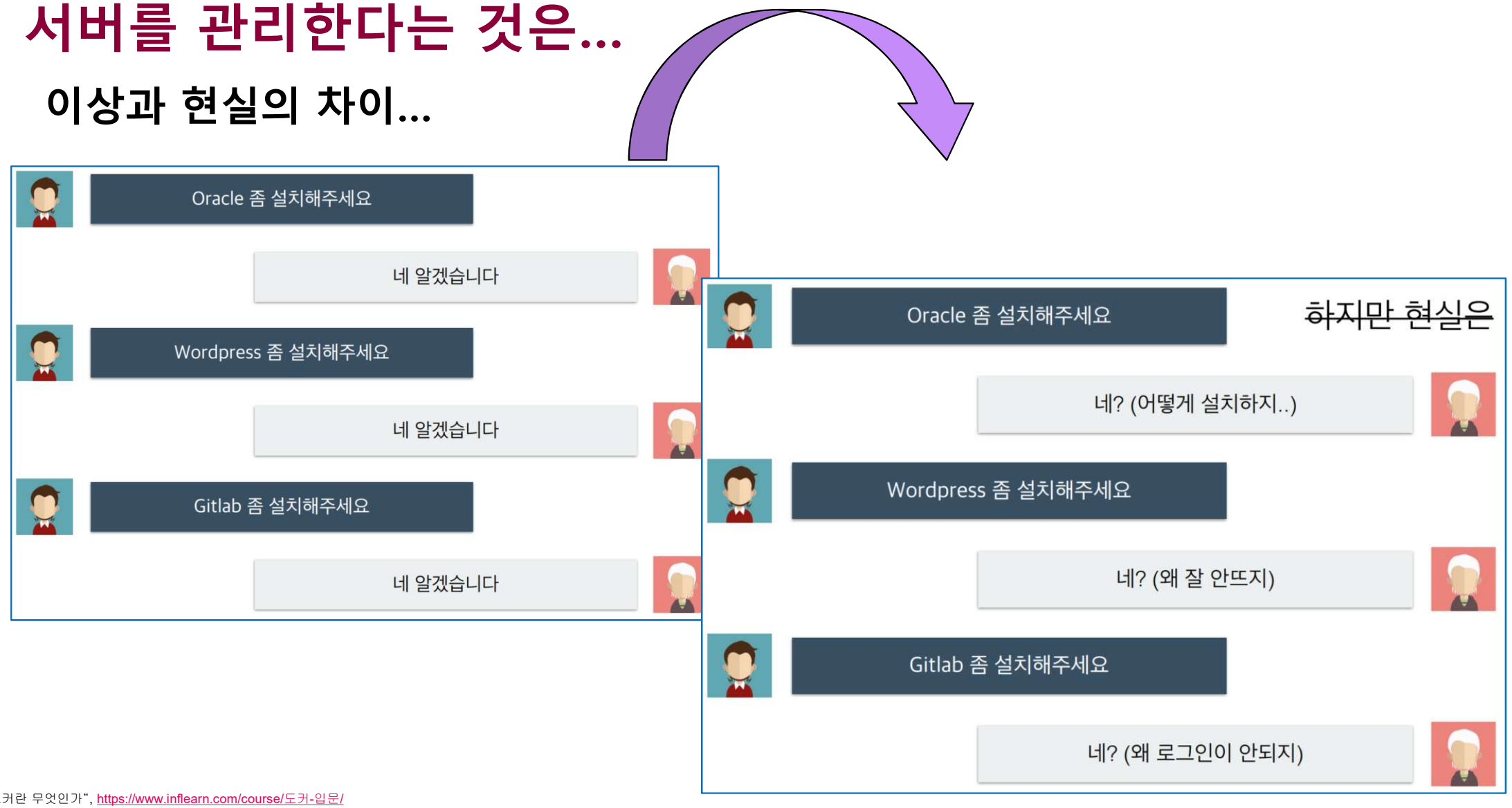
너무 너무 복잡 복잡하다.



https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

서버를 관리한다는 것은...

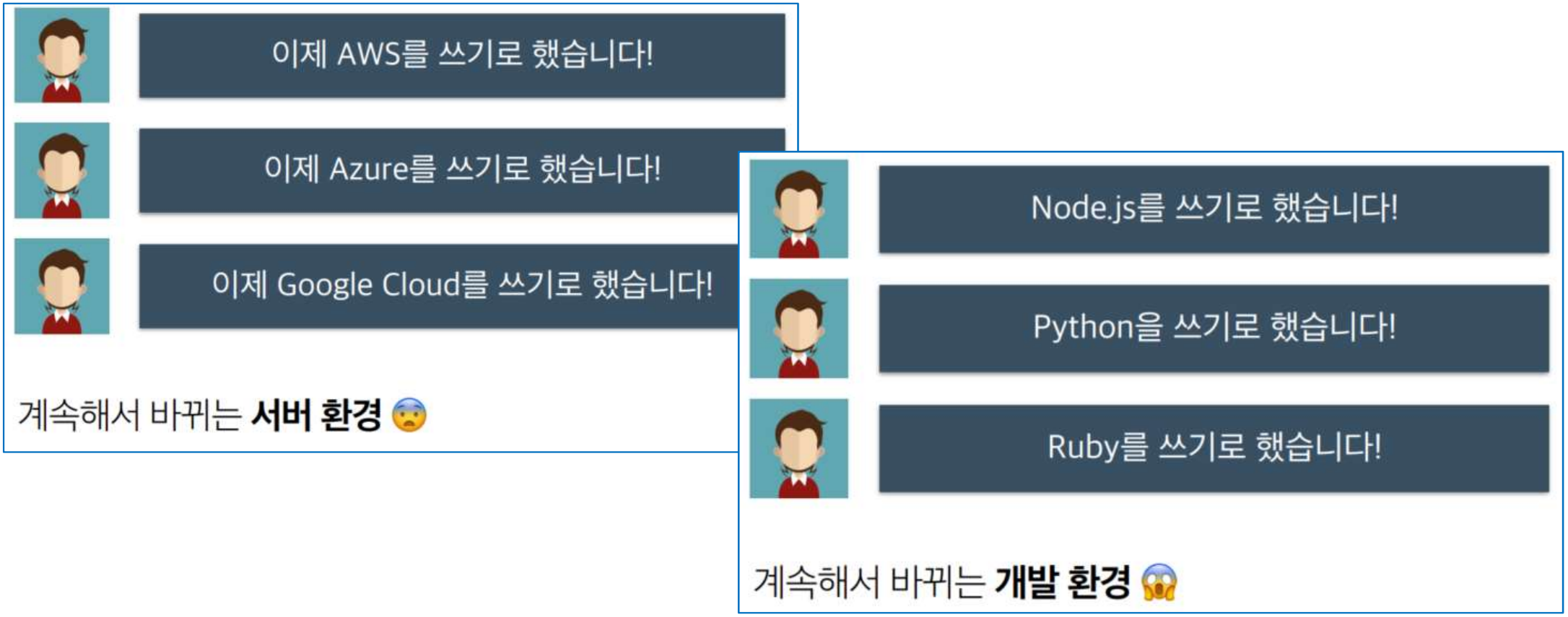
이상과 현실의 차이...



"도커란 무엇인가", https://www.inflearn.com/course/도커-입문/

서버를 관리한다는 것은...

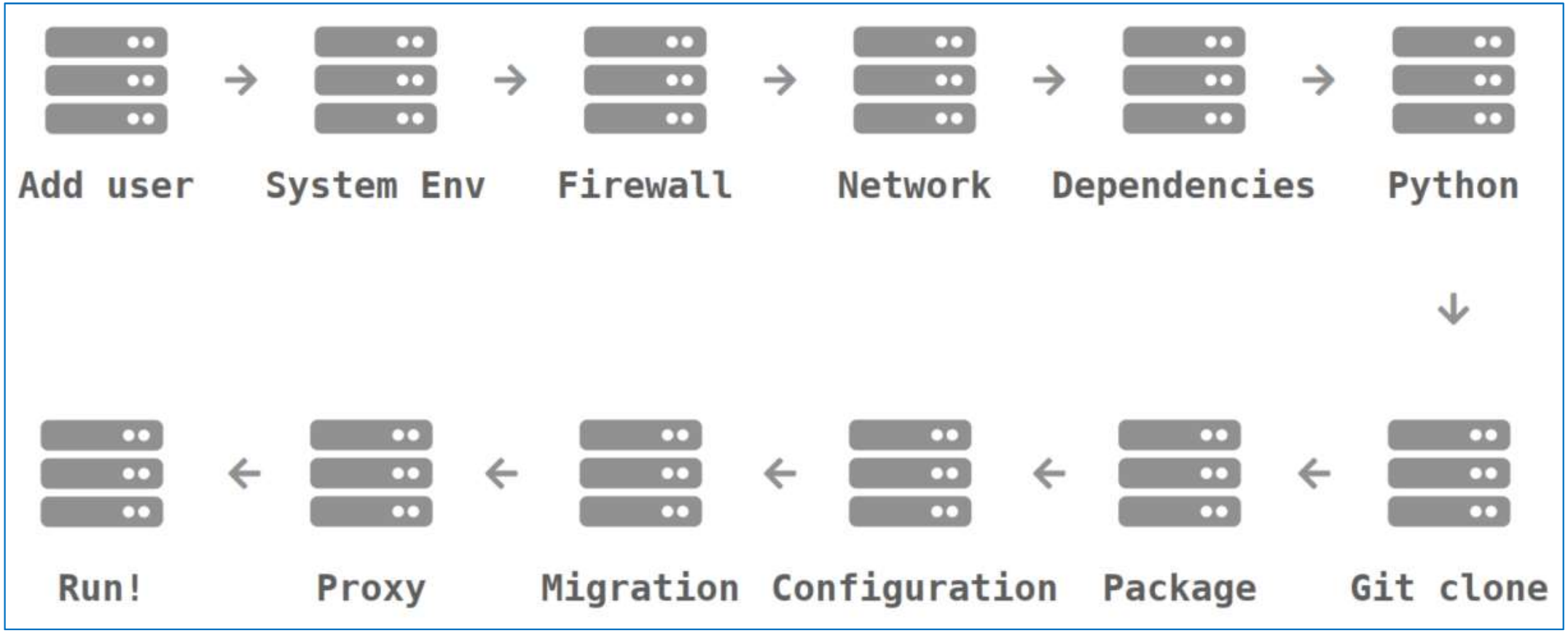
이상과 현실의 차이...



"도커란 무엇인가", https://www.inflearn.com/course/도커-입문/

서버를 관리한다는 것은...

징검다리를 건너는 것...

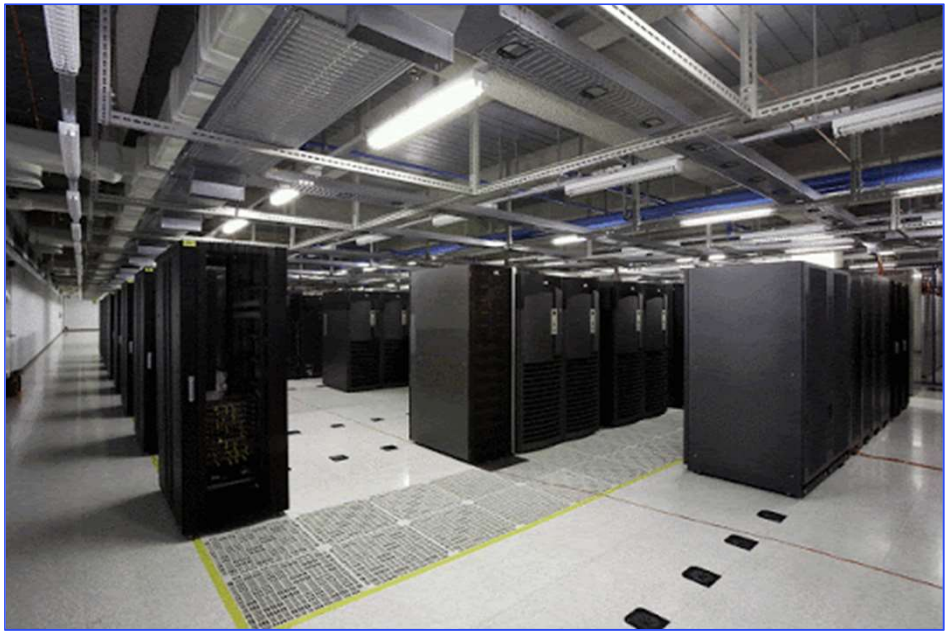


"도커란 무엇인가", https://www.inflearn.com/course/도커-입문/

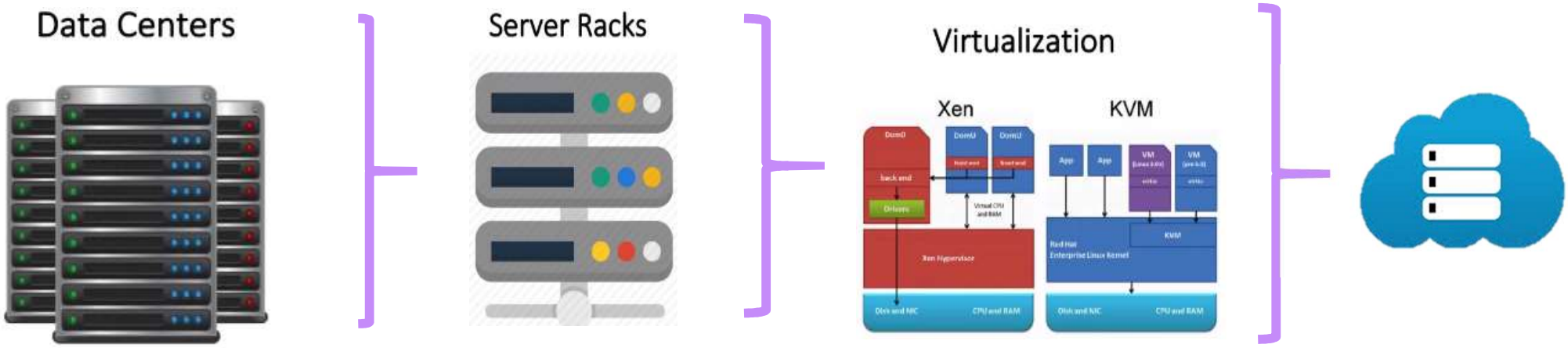
Virtualization

Cloud Computing Service Architecture

Virtualization (Cont.)

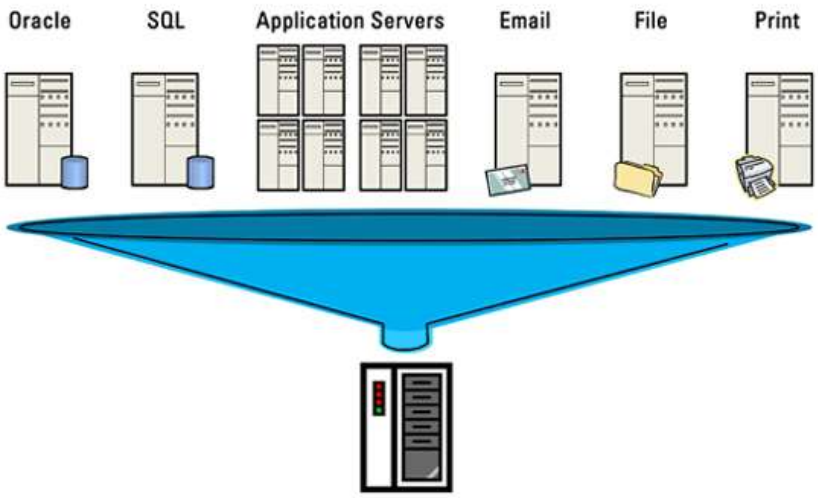


Virtualization (Cont.)



Virtualization (Cont.)

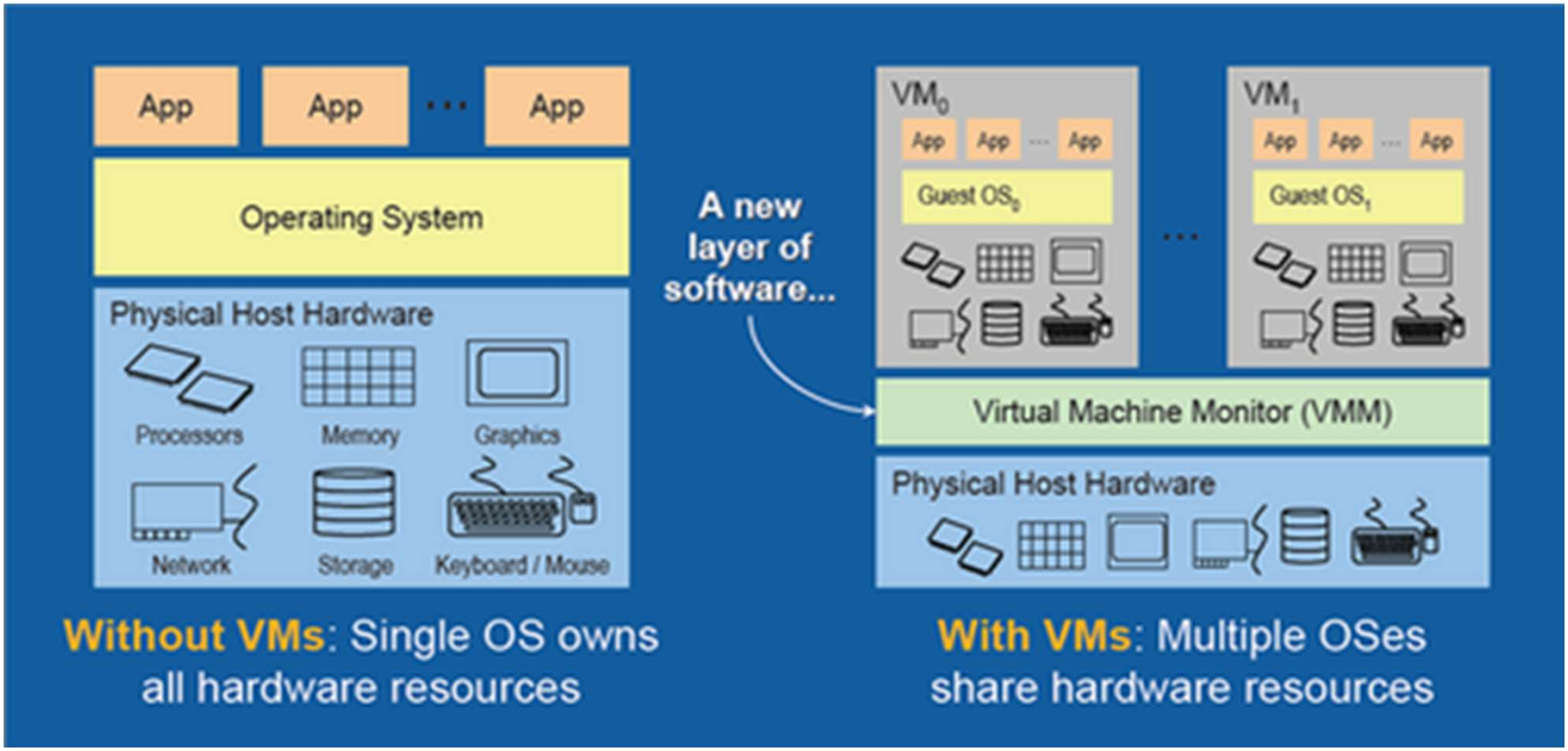
What is Virtualization?



- V12n
- 물리적인 하드웨어 장치**를**
- 논리적인 객체**로**
- 추상**화하는 기술**
- Is the act of creating a virtual(rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

https://www.energystar.gov/products/low_carbon_it_campaign/12_ways_save_energy_data_center/server_virtualization

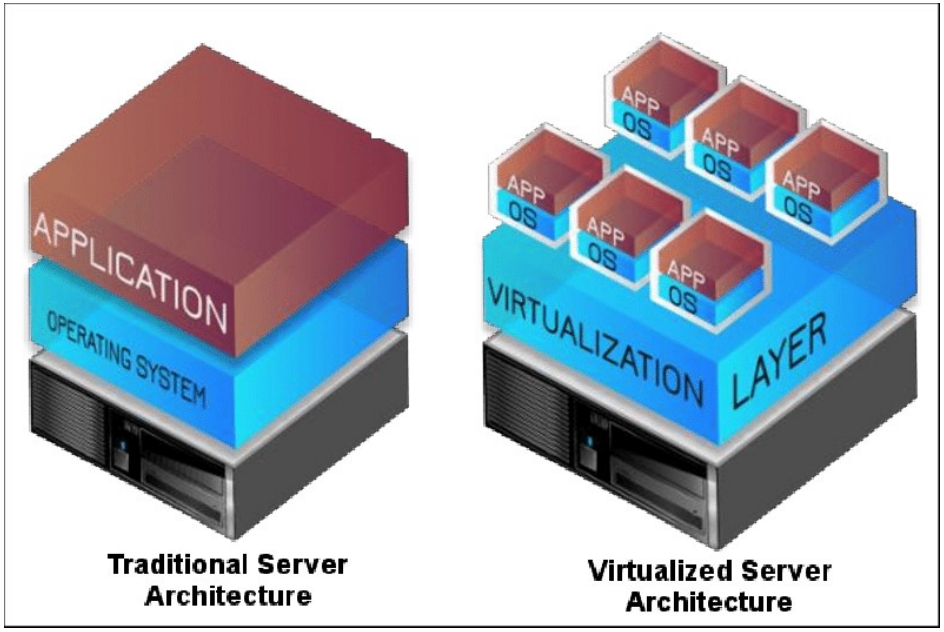
Virtualization (Cont.)



https://www.networkcomputing.com/guide-server-virtualization

Virtualization (Cont.)

Server Virtualization

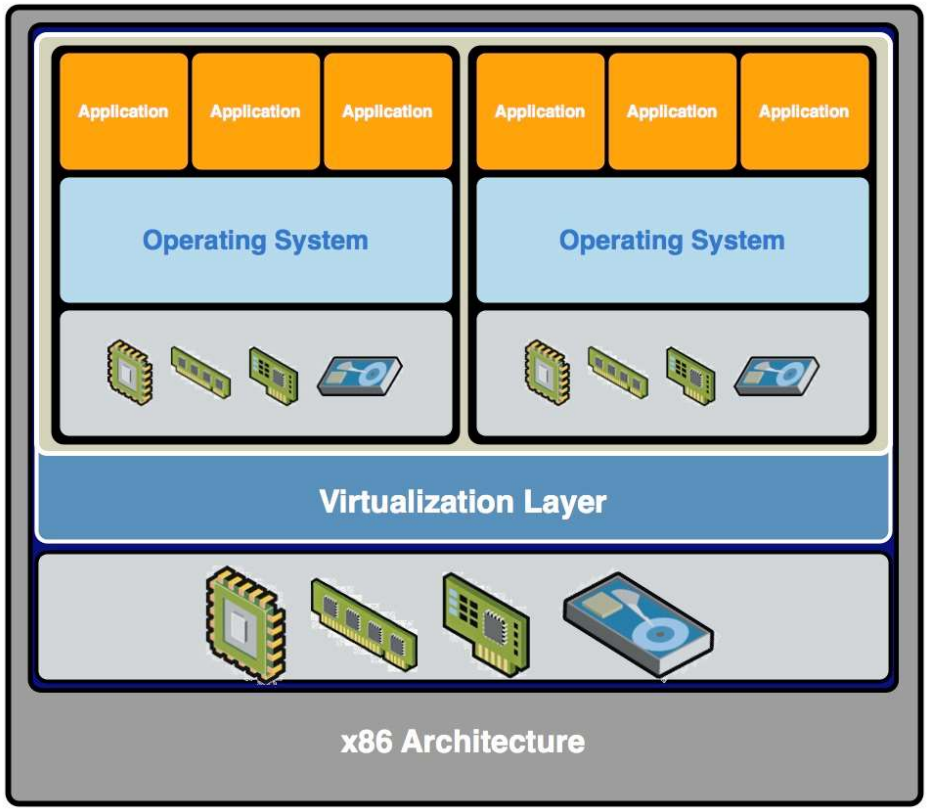


- The traditional one workload, one box approach means that most servers run at a low ***utilization rate*** – the fraction of total computing resources engaged in useful work.
- A 2012 New York Times article cited two sources that estimated the average server utilization rate to be ***6 to 12%***.
- Another study stated that the one workload, one box approach leads to ***90% of all x86 servers*** running at ***less than 10% utilization*** with a typical server running at ***less than 5% utilization***.

https://www.researchgate.net/figure/Comparison-between-Traditional-Server-and-Virtualized-Server_fig3_324692035

Virtualization (Cont.)

VM-Based Virtualization

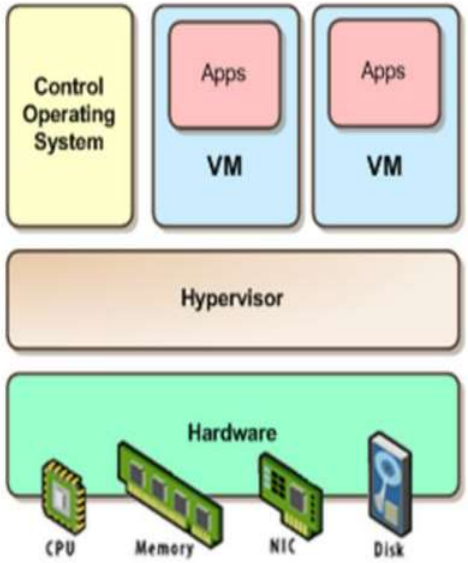


- Virtualization Layer
  - **Host** Operating Systems
  - Virtualization Software
  - Hypervisor
  - Type I vs Type II
- Virtual Machine
  - Virtualized Hardware
    - CPU, Memory, Storage, Network Resources
  - **Guest** Operating Systems

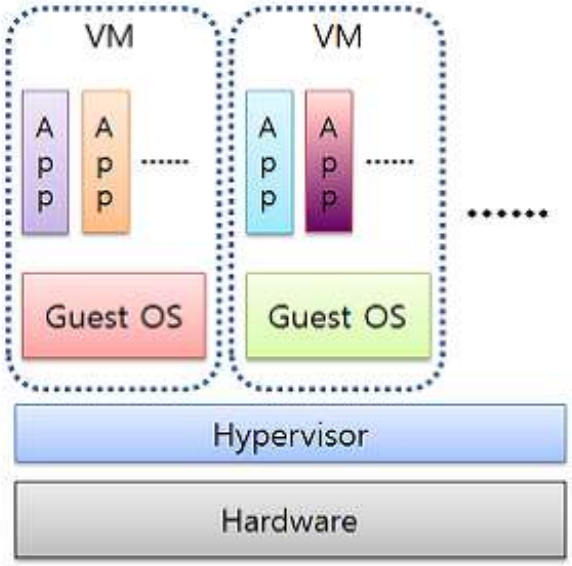
https://www.hybridtp.ie/what-is-virtualization-what-are-its-benefits-and-how-can-i-get-the-solution/

Virtualization (Cont.)

Hypervisor Virtualization



https://slidesplayer.org/slide/11113210/

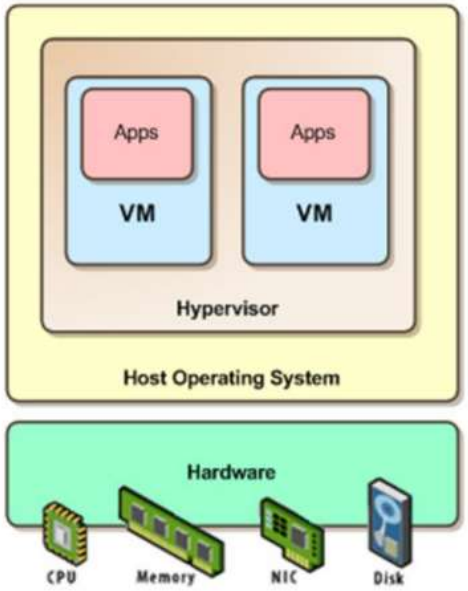


https://secmem.tistory.com/308

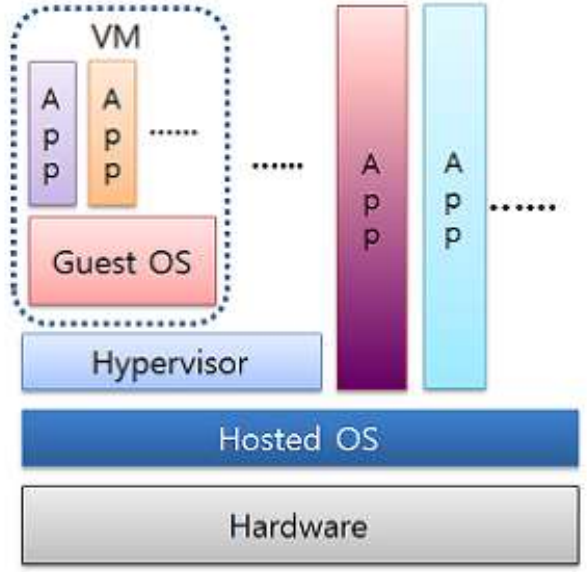
- Type I Virtualization
- 물리적인 하드웨어 위에 Hypervisor
- ≡ Bare-Metal 방식
- Citrix's Xen
- VMWare's ESX Server
- Linux's KVM

Virtualization (Cont.)

Host Virtualization



https://slidesplayer.org/slide/11113210/



https://secmem.tistory.com/308

- Type II Virtualization
- 운영체제 위에 Hypervisor를 올림
- 간단한 방법으로 보다 쉽게 가상화 가능
- Oracle VirtualBox
- VMWare VMWare Workstation
- Microsoft VirtualPC

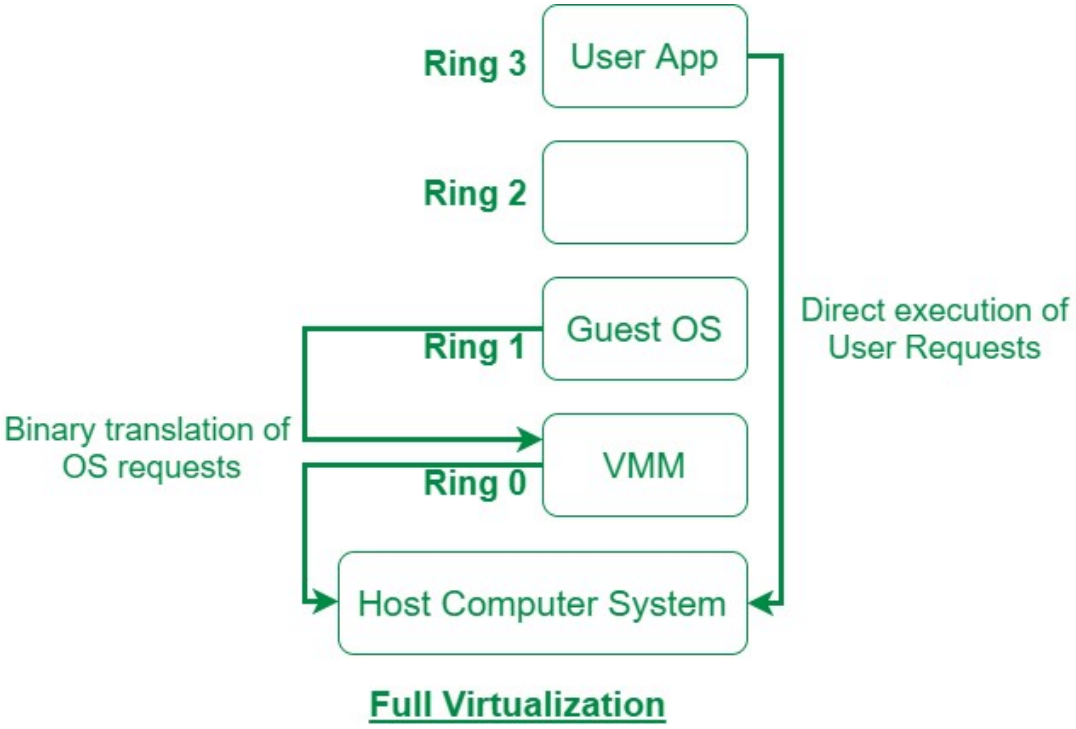
Virtualization (Cont.)

Criteria	Type 1 hypervisor	Type 2 hypervisor
AKA	Bare-metal or Native	Hosted
Definition	Runs directly on the system with VMs running on them	Runs on a conventional Operating System
Virtualization	Hardware Virtualization	OS Virtualization
Operation	Guest OS and applications run on the hypervisor	Runs as an application on the host OS
Scalability	Better Scalability	Not so much, because of its reliance on the underlying OS.
Setup/Installation	Simple, as long as you have the necessary hardware support	Lot simpler setup, as you already have an Operating System.
System Independence	Has direct access to hardware along with virtual machines it hosts	Are not allowed to directly access the host hardware and its resources
Speed	Faster	Slower because of the system's dependency
Performance	Higher-performance as there's no middle layer	Comparatively has reduced performance rate as it runs with extra overhead
Security	More Secure	Less Secure, as any problem in the base operating system affects the entire system including the protected Hypervisor
Examples	• VMware ESXi • Microsoft Hyper-V • Citrix XenServer	• VMware Workstation Player • Microsoft Virtual PC • Sun's VirtualBox

https://www.hitechnectar.com/blogs/hypervisor-type-1-vs-type-2/

Virtualization (Cont.)

Types of Virtualization – Full Virtualization



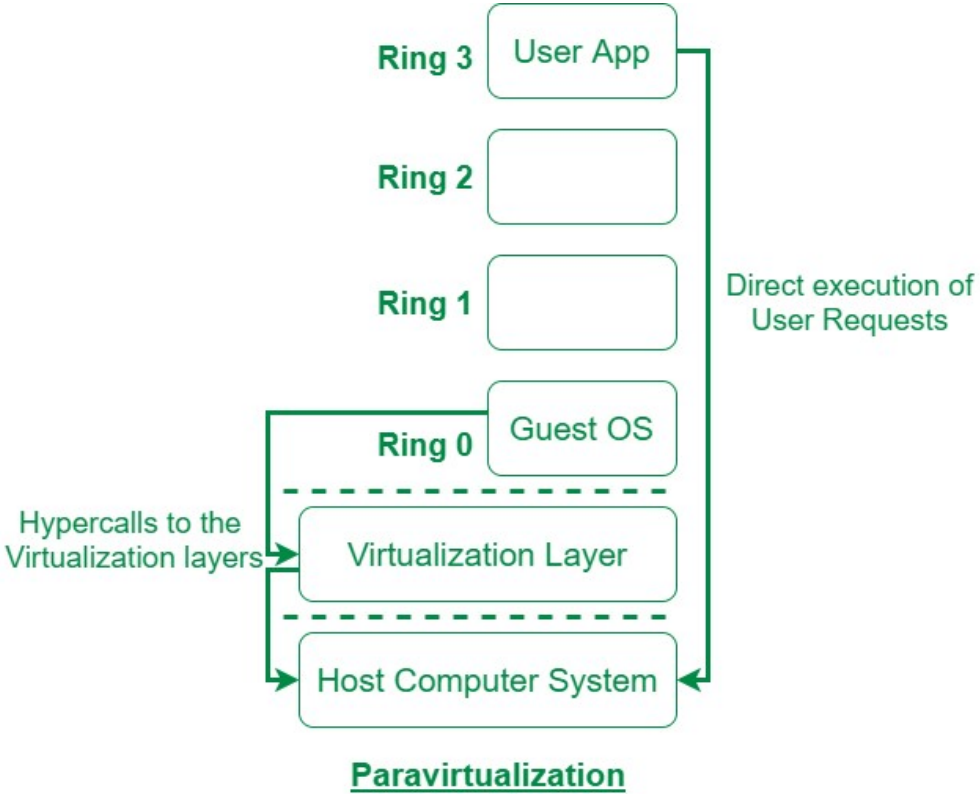
Full Virtualization

- Allows multiple guest operating systems to execute on a host operating system independently.
- Is less secure.
- Is more slow.
- Is lower performance.
- Is more portable and compatible.
- Microsoft, Parallels, and VirtualBox.

https://www.geeksforgeeks.org/difference-between-full-virtualization-and-paravirtualization/

Virtualization (Cont.)

Types of Virtualization – Para Virtualization

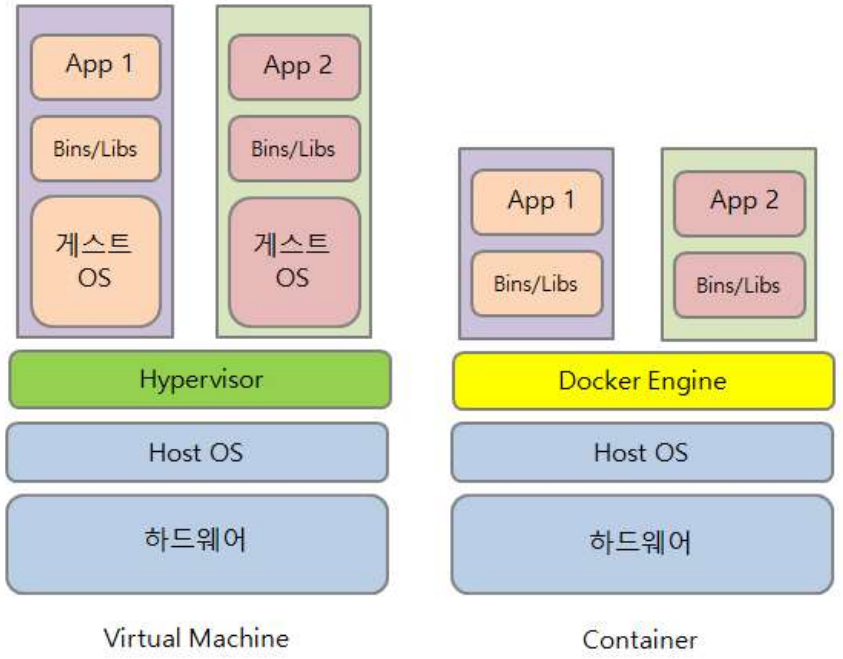


- Allows multiple guest operating systems to run on host operating systems while communicating with the hypervisor to improve performance.
- Is more secure.
- Is more fast.
- Is higher performance.
- Is less portable and compatible.
- VMware, Xen.

https://www.geeksforgeeks.org/difference-between-full-virtualization-and-paravirtualization/

Virtualization (Cont.)

Container Virtualization



https://darksoulstory.tistory.com/447

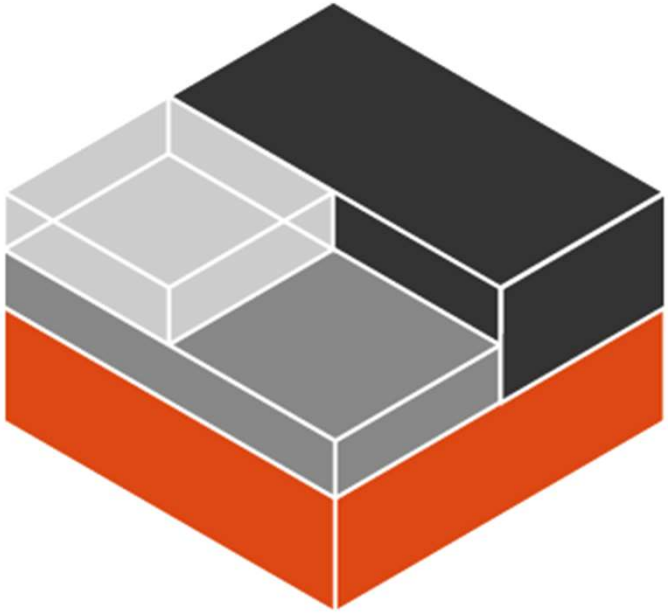
- 이전 방식과 달리
- Guest OS를 두지 않고
- Host OS의 커널을 그대로 사용하면서
- Linux Container를 사용하여
- Host OS와 다른 부분만 패키징하고
- Host의 리소스를 공유하여
- 기존 가상머신 보다 리소스를 효율적으로 사용할 수 있는 가상화 환경

Lab. VirtualBox에 Ubuntu Server 설치 및 환경설정

Container Concepts

Container

What is LXC?

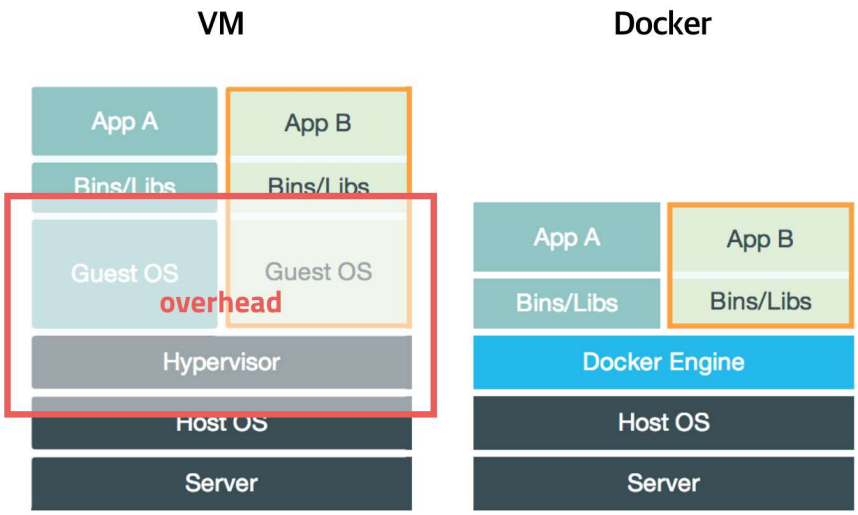


https://linuxcontainers.org/

- Is the well known set of tools, templates, library and language bindings.
- Is an **operating-system-level virtualization method** for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel.
- Features
  - 가상 머신과 달리 Kernel을 공유하는 방식
  - 실행 속도가 빠르고 성능상의 손실이 거의 없음
  - 실행된 Process는 Kernel을 공유하지만, Linux Namespaces, 컨트롤 그룹(Cgroup:Control Group, Process 격리방식), 루트 디렉토리 격리(chroot) 등의 kernel 기능을 활용해서 격리되어 실행
  - Host에서는 Process로 인식되지만, Container 관점에서는 독립적인 환경을 갖춘 가상 머신으로 인식
  - No Booting
  - Memory 적재

Container (Cont.)

What is LXC?

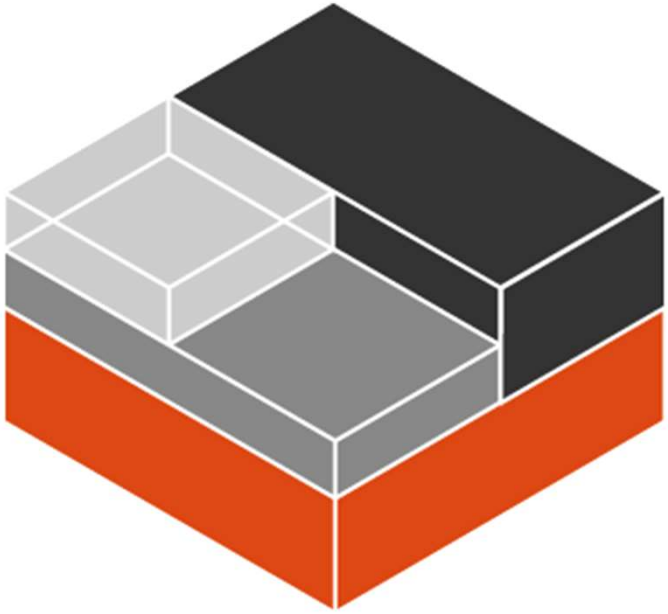


- 운영체제 수준의 가상화
  - Guest OS 관리 없음.
- 빠른 속도와 효율성
  - 하드웨어 에뮬레이션 불필요
- 높은 이식성
  - Linux Kernel을 사용하고 같은 Container 런타임 사용시 실행환경을 공유하고 손쉽게 재현가능
- Stateless
  - 독립적인 프로세스이기 때문에 다른 Container에게 영향을 주지 않음.

https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

Container (Cont.)

What is LXC?



- The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.
- Components
  - The liblxc library
  - Several language bindings for the API:
    - Python3
    - lua
    - Go
    - Ruby
    - Haskell
  - A set of standard tools to control the containers
  - Distribution container templates

https://linuxcontainers.org/

Container (Cont.)

LXC and Docker (early versions < v1.11)

- Early versions of Docker used LXC as the container execution driver
- LXC was made optional in Docker v0.9
- LXC support was dropped in Docker v1.10

https://www.indiamart.com/proddetail/40-feet-high-cube-container-6566323748.html

Docker Concepts

Docker

What is Docker?

- By Docker, we may be referring to :
  - Docker Inc., the company
  - Docker the container runtime and orchestration technology
  - Docker the open source project(now called ***Moby***)

Docker (Cont.)

History

- The company was founded as ***dotCloud*** in 2008.
- On 2013 March, **`Pycon Conference`** at Santa Clara, ***Solomon Hykes***가 The future of Linux Containers 세션 발표
- On October 29, 2013, ***dotCloud*** was renamed ***Docker***.
- On 2014 June, Docker 1.0 발표
- On August 4, 2014 the ***dotCloud*** technology and brand was sold to ***cloudControl***.

Docker (Cont.)

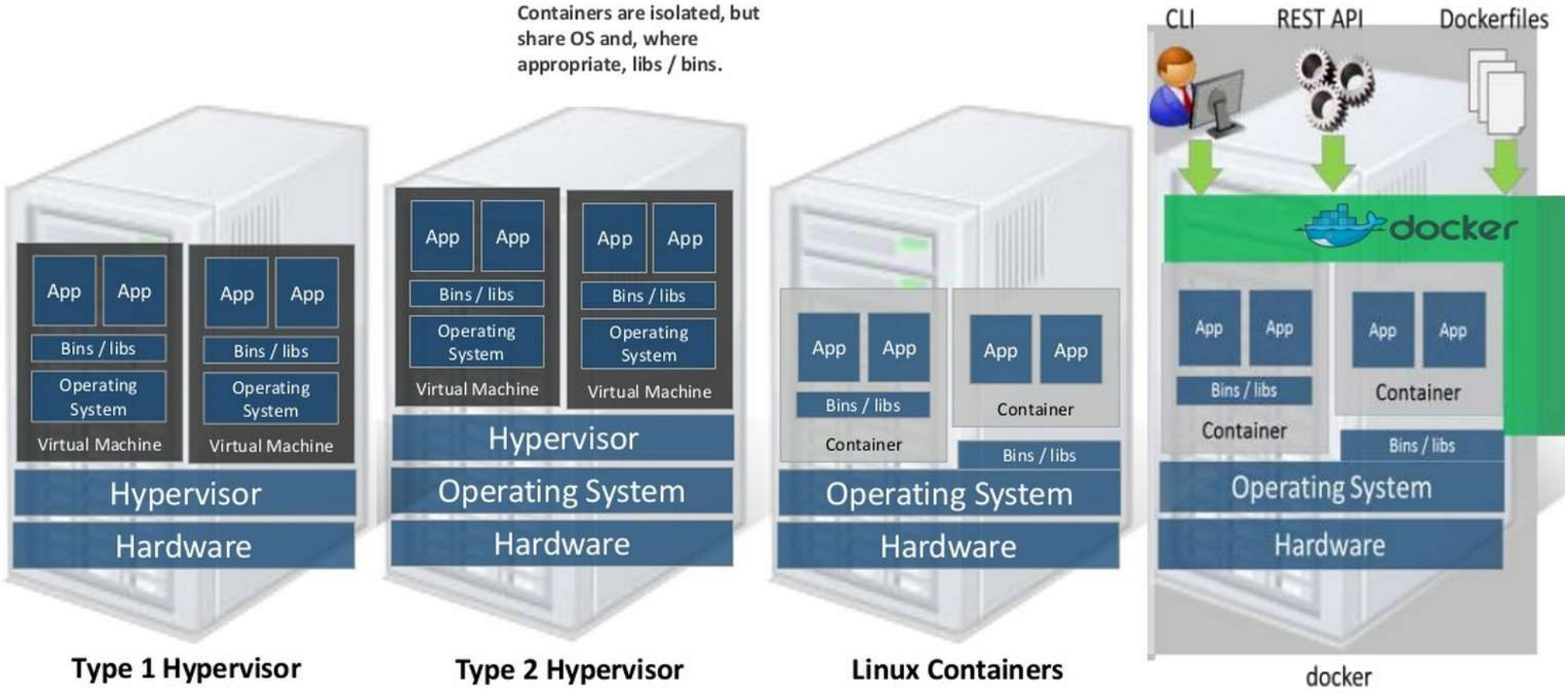
Docker (Cont.)

Docker (software)

- Open-Source Virtualization Platform based on ***Container***.
- Is a set of platform as a service (PaaS) products that use ***OS-level virtualization*** to deliver software in packages called containers.
- The service has both free and premium tiers.
- The software that hosts the containers is called Docker Engine.
- It was first started in 2013 and is developed by **`Docker, Inc.`**
- Runs on Linux and Windows.

Docker (Cont.)

Docker (software)



https://www.enterpriseai.news/wp-content/uploads/2014/08/ibm-various-virtualization.jpg

https://www.enterpriseai.news/2014/08/18/ibm-techies-pit-docker-kvm-bare-metal/

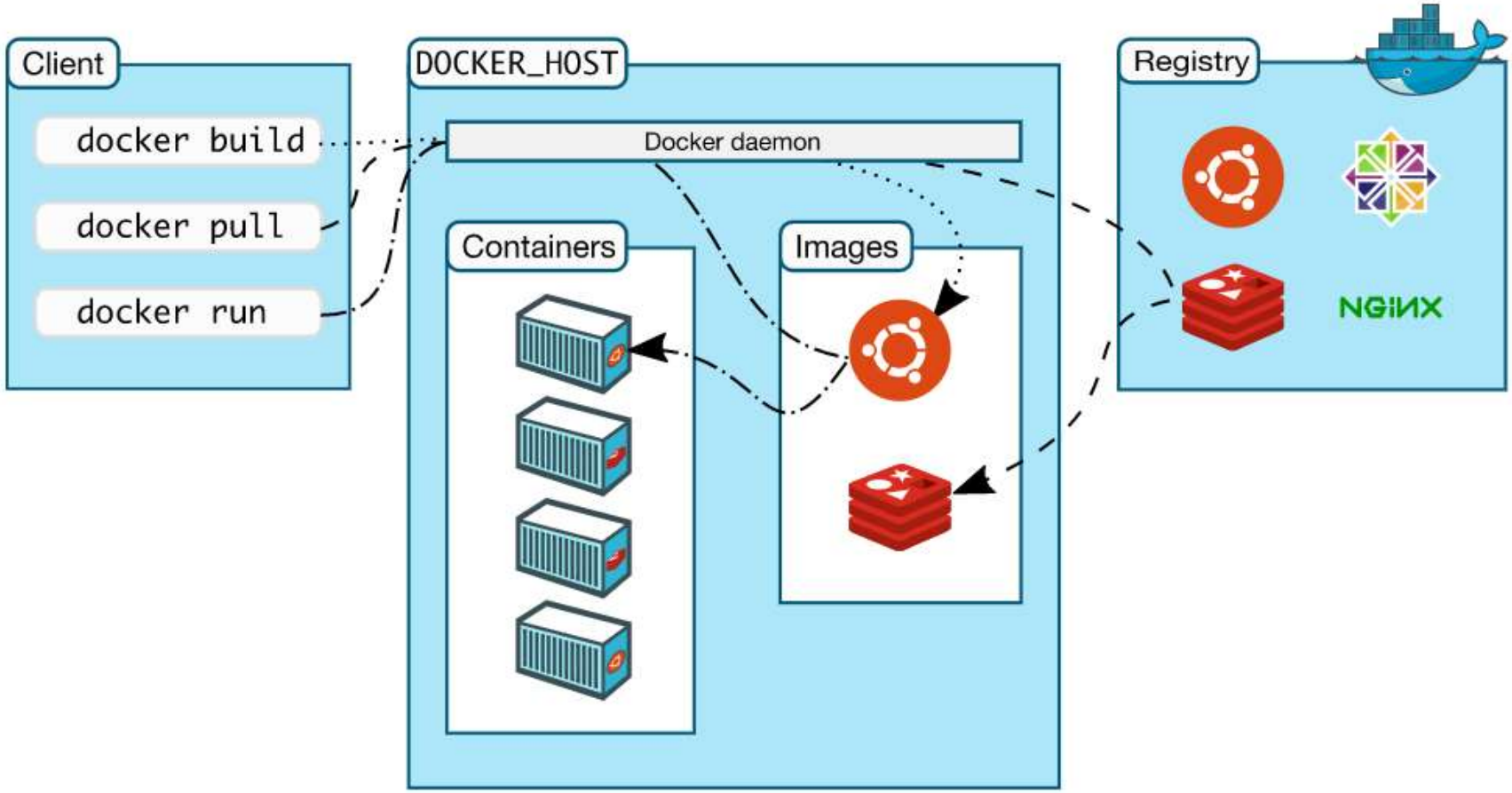
Docker (Cont.)

Moby Project

- https://mobyproject.org/
- Originally it was open-source Docker project.
  - https://github.com/docker/docker
- Officially renamed as the Moby project.
  - DockerCon 2017 in Austin, TX
  - https://github.com/moby/moby
- Upstream for Docker
- Most of the project and its tools are written in Golang

Docker (Cont.)

Docker Architecture



Docker (Cont.)

Docker Components

- Docker Software
  - Docker daemon, called **`dockerd`**
  - Docker client, called docker
- Docker Objects
  - Docker containers (running instance of images)
  - Docker images
  - Swarm, multi-node Docker daemon coordination service
- Docker Registries
  - A Docker registry is a repository for Docker images.
  - Docker clients connect to registries to download(**`pull`**) images for use or upload(**`push`**) images that they have built.
  - Registries can be public or private.
  - Docker Hub (**`docker.io`**) is the default registry where Docker looks for images.

Docker (Cont.)

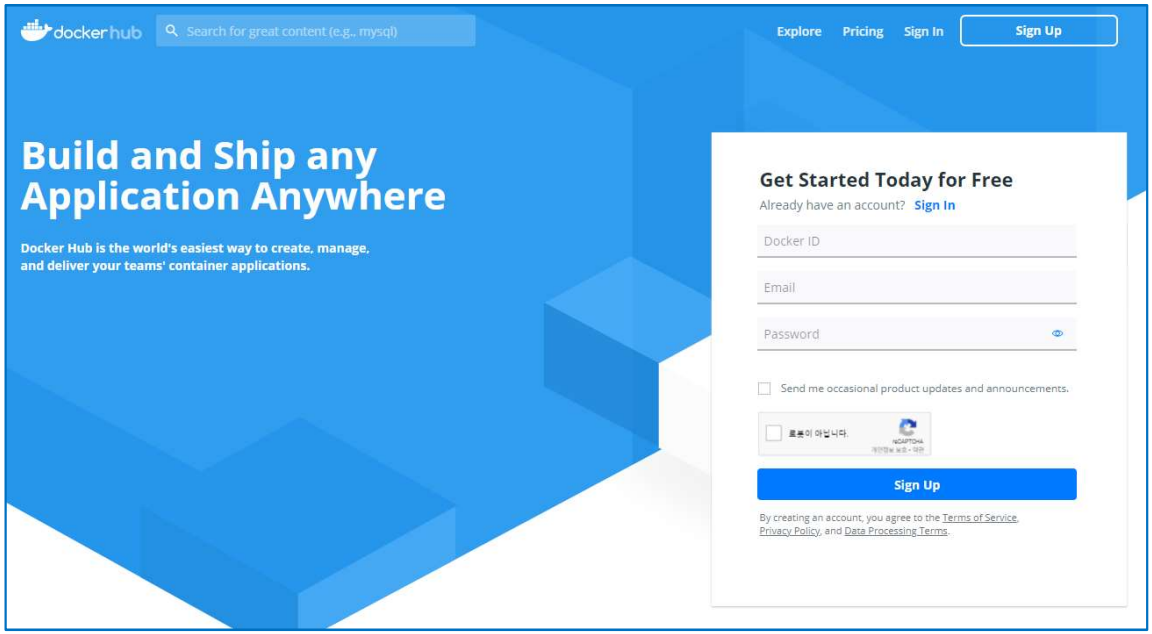
Docker Tools

- Docker compose
  - A client-side tool for defining and running multi-container Docker applications.
  - It uses ***YAML*** files to configure the application's services and performs the creation and start-up process of all the containers with a single command.
- Docker Swarm (v1.12+)
  - Provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single virtual Docker engine.

Docker (Cont.)

Docker Hub

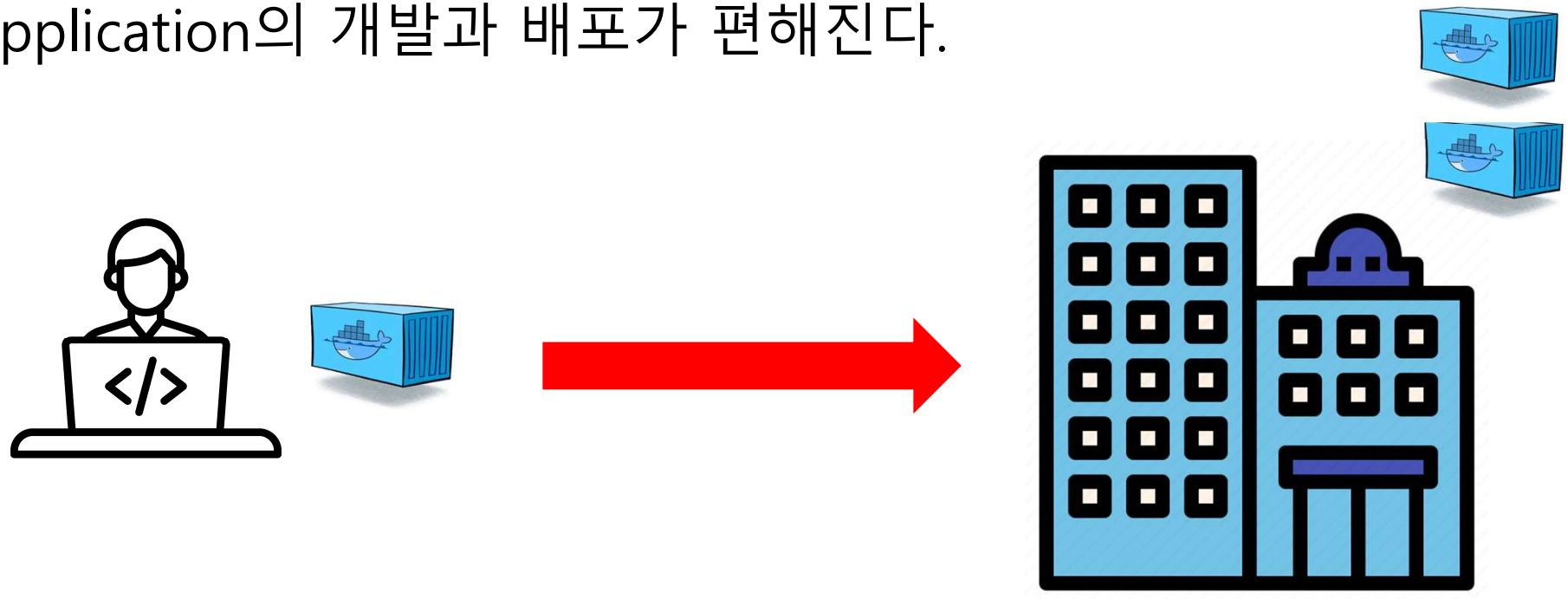
- https://hub.docker.com/
- Default public repository
- Can search image by image name.
- Can apply filters such as:
  - Docker Certified
  - Verified Publisher
  - Official Images
  - OS / Architecture
  - App Categories



Docker (Cont.)

Docker 를 사용하는 이유

- Application의 개발과 배포가 편해진다.

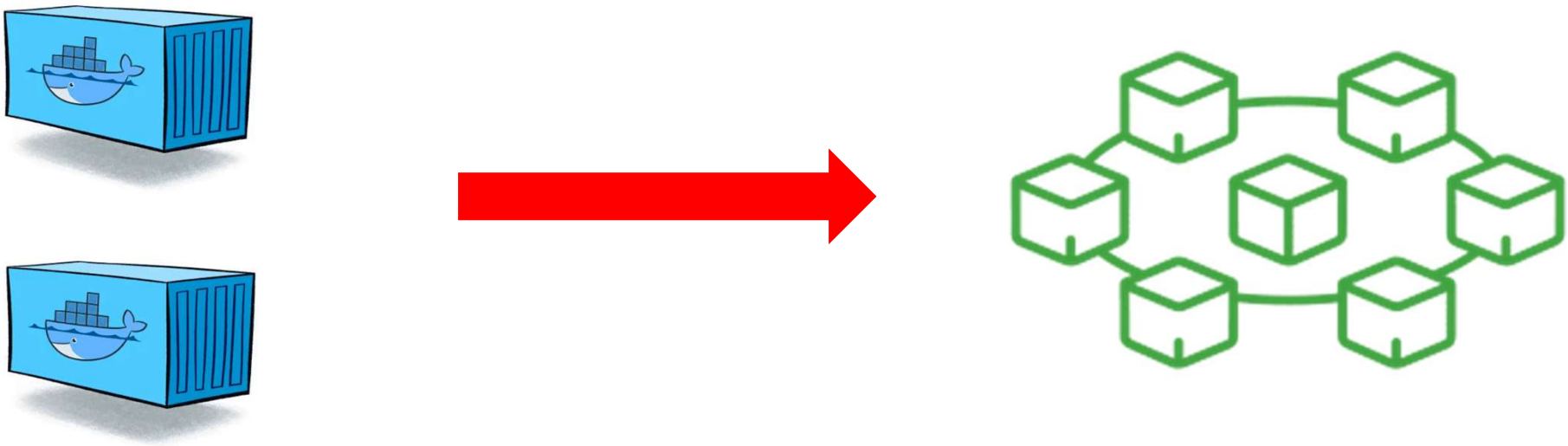


Ref 왜 굳이 도커(컨테이너)를 써야 하나요? - 컨테이너를 사용해야 하는 이유 | 44BITS

Docker (Cont.)

Docker 를 사용하는 이유

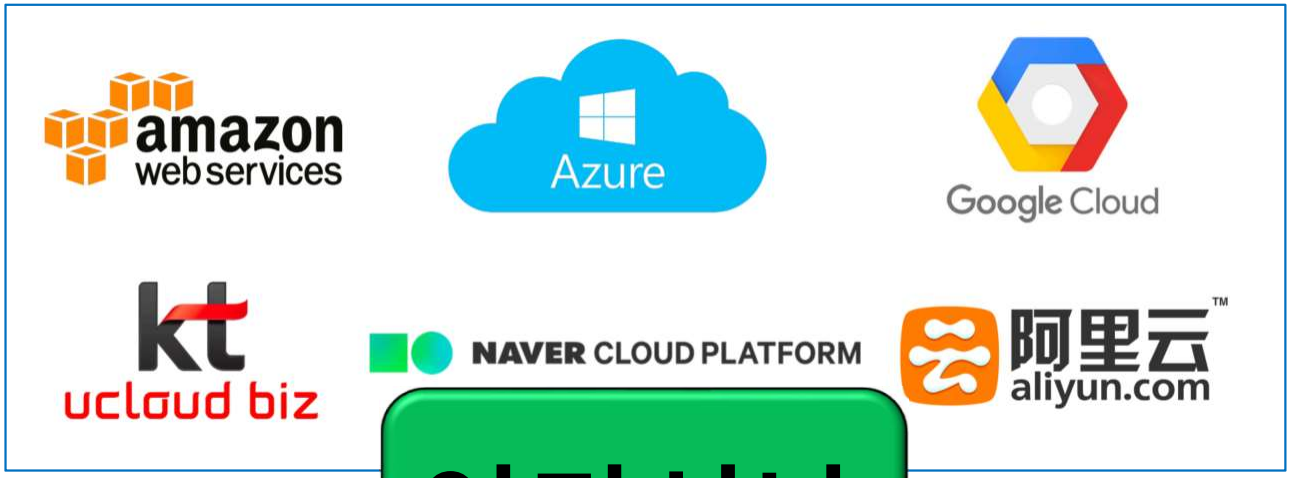
- 여러 Application의 독립성과 확장성이 높아진다.

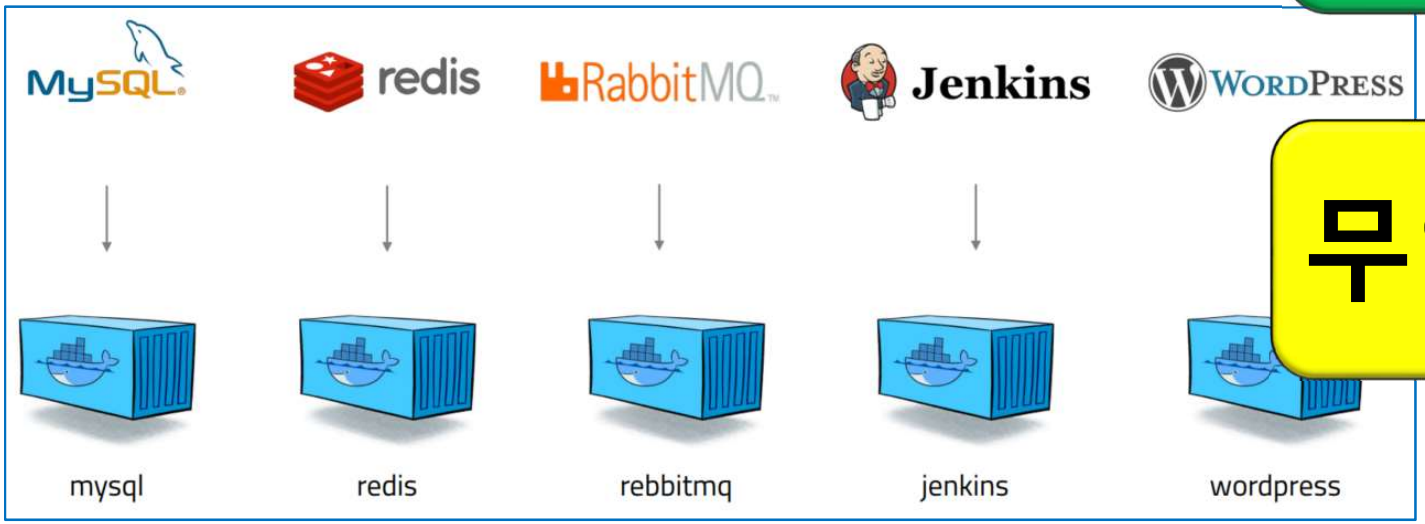


Docker (Cont.)

Docker 를 사용하는 이유

- 서버 관리가 너무 쉬워진다.



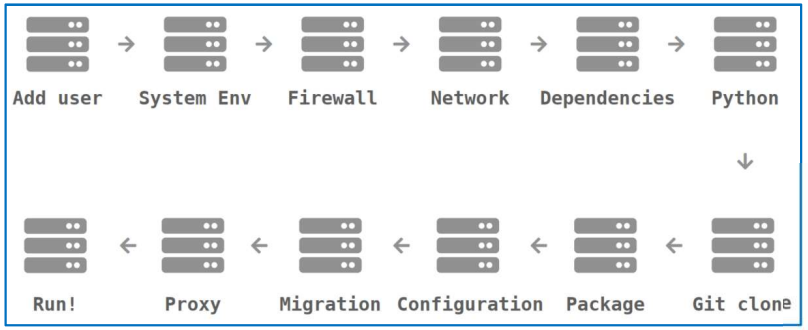


무엇이든

"도커란 무엇인가", https://www.inflearn.com/course/도커-입문/

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

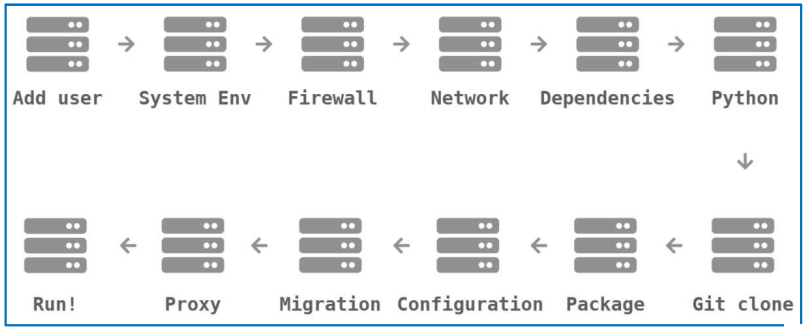
1번째 방법은...



문서화를 잘하자.

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

2번째 방법은...



CHEF

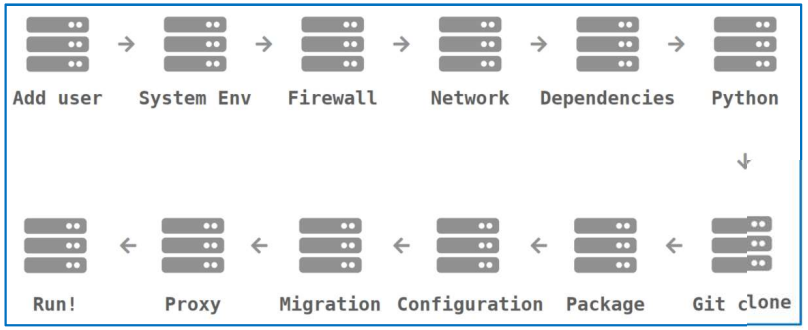
puppet labs

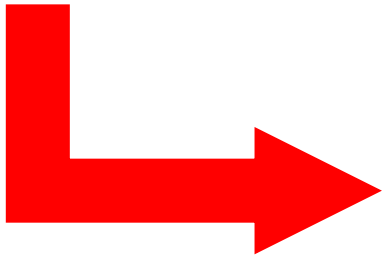
ANSIBLE

툴을 잘 쓰자.

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

3번째 방법은...

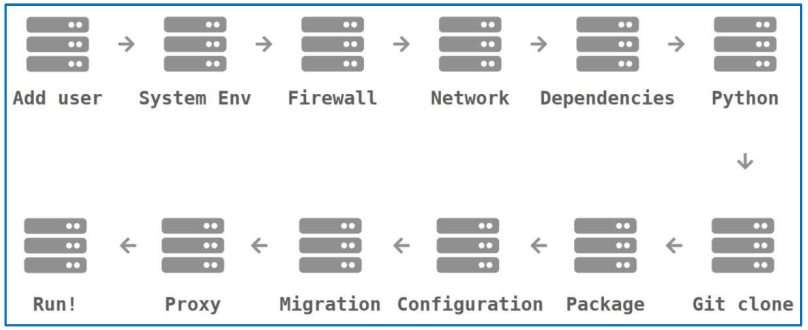


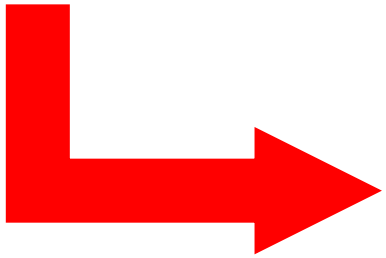


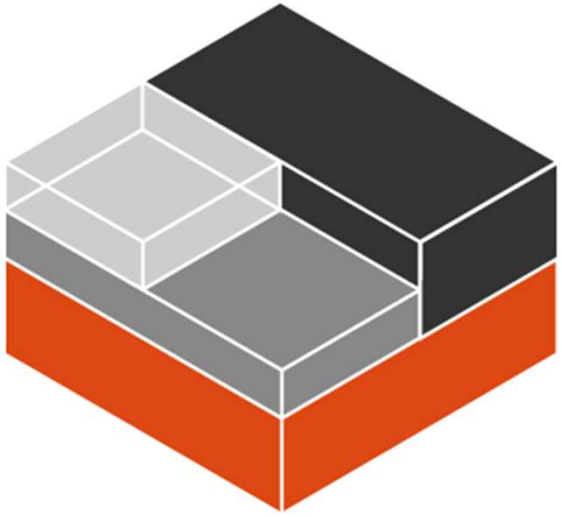
가상화

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

4번째 방법은...



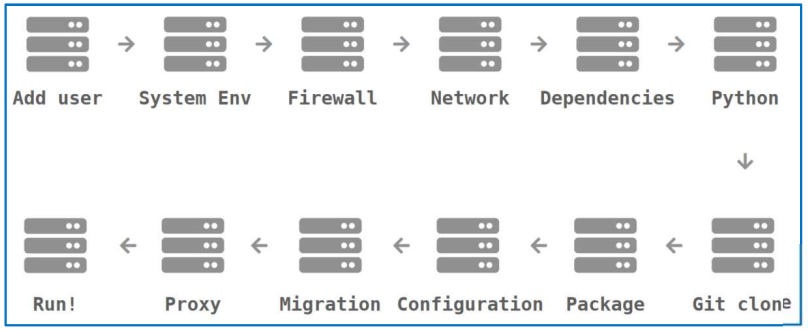


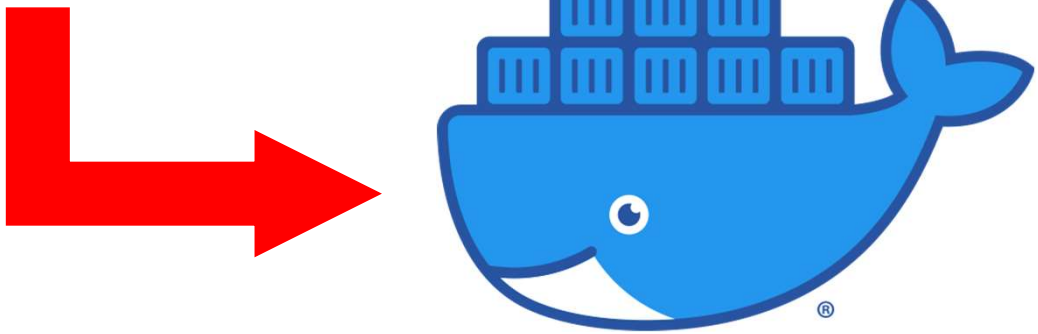


리눅스 기능을 이용한 빠르고 효율적인 서버 관리

다시 처음으로 돌아가서, 서버를 관리한다는 것은...

5번째 방법은...



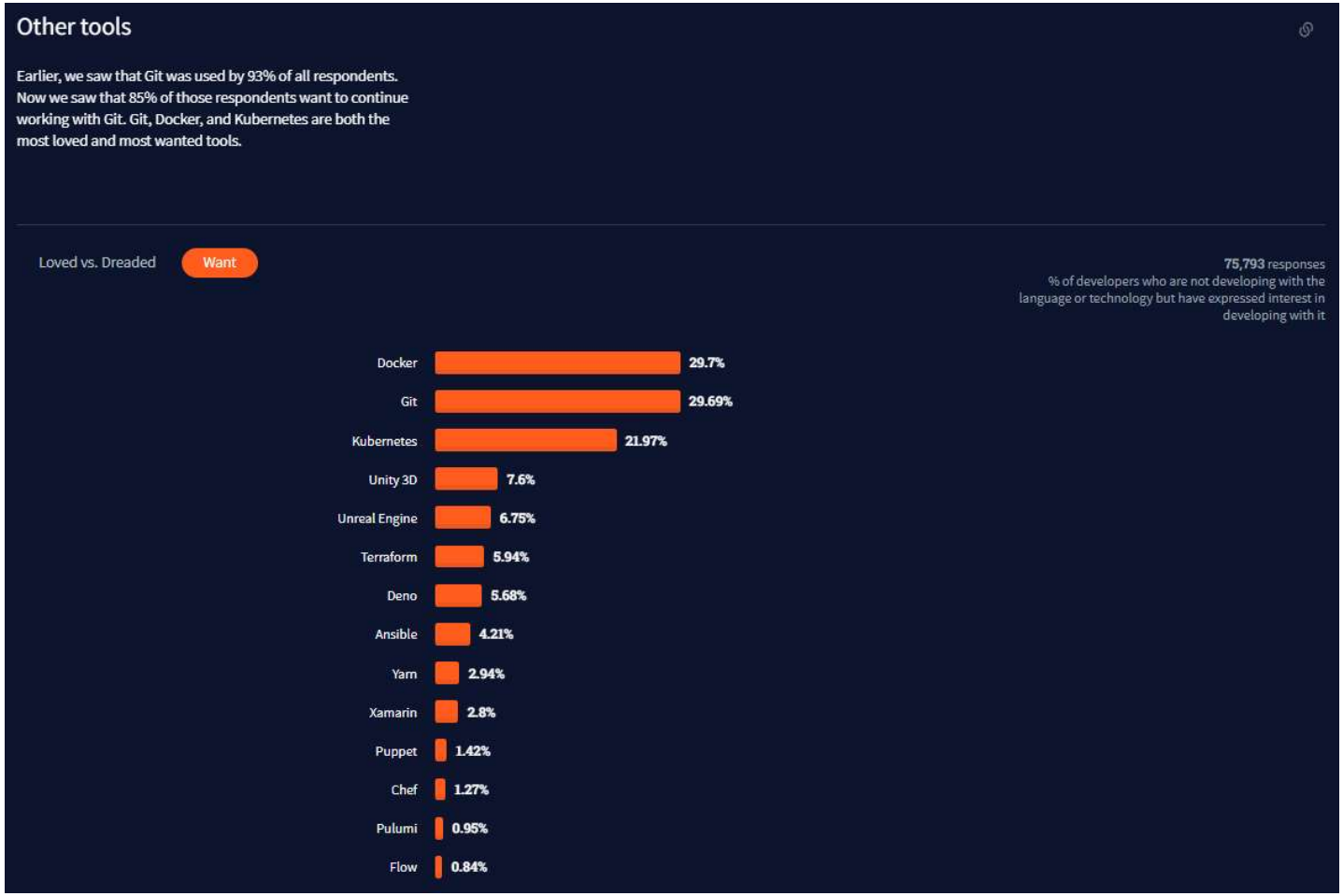


어렵고 복잡한 기능을 사용하기 쉽게!!!

Docker (Cont.)

Docker 를 사용하는 이유

- 시대적 대세이다.

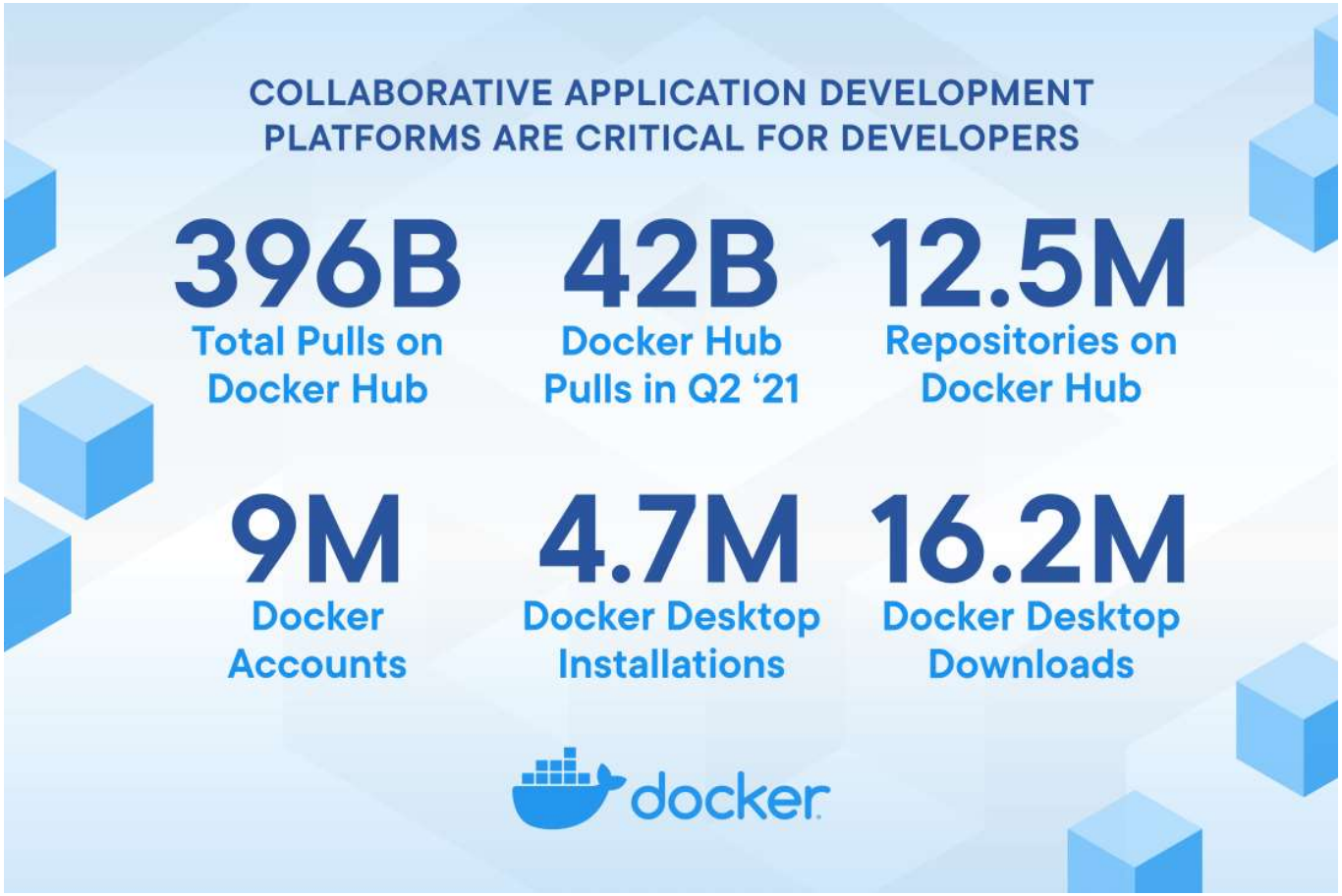


https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-tools-tech-want

Docker (Cont.)

Docker 를 사용하는 이유

- 시대적 대세이다.

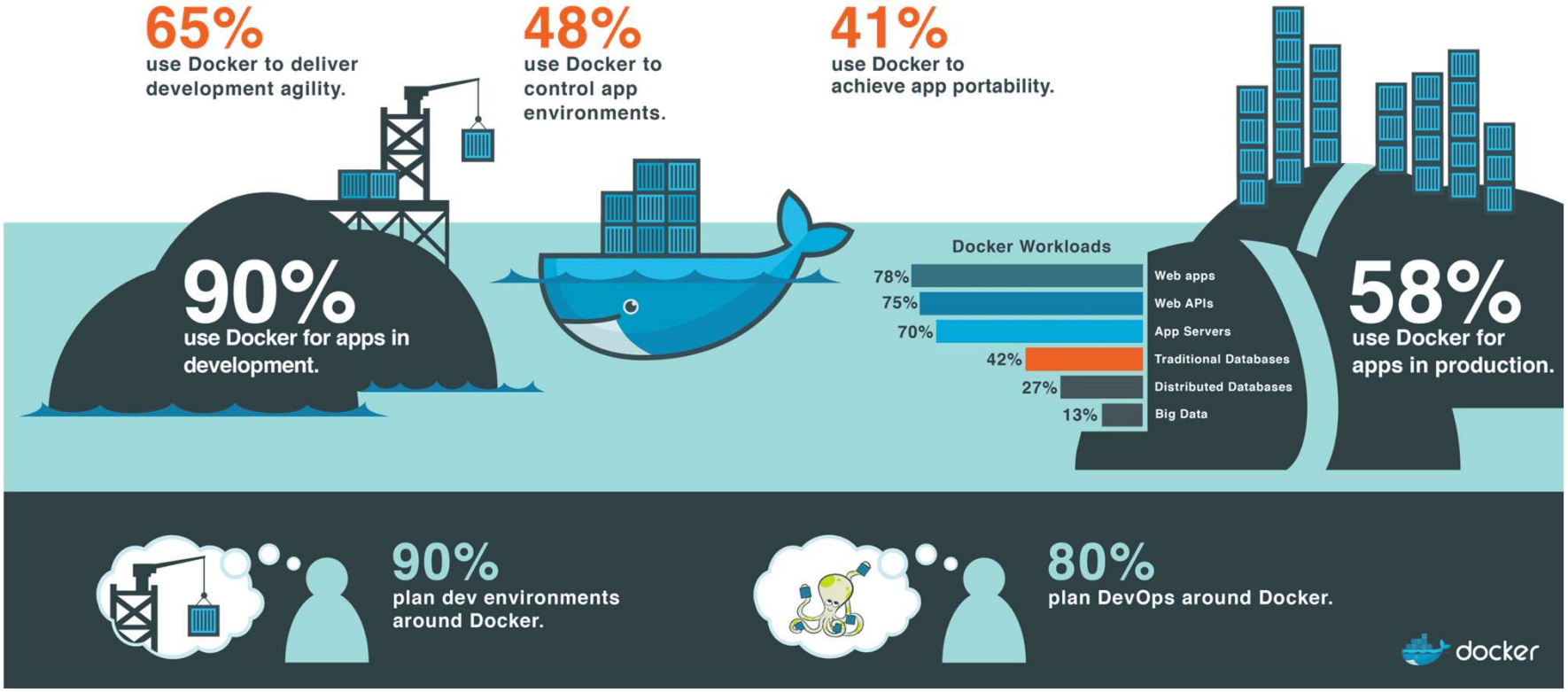


https://www.docker.com/blog/docker-index-shows-surging-momentum-in-developer-community-activity-again/

Docker (Cont.)

Docker 를 사용하는 이유

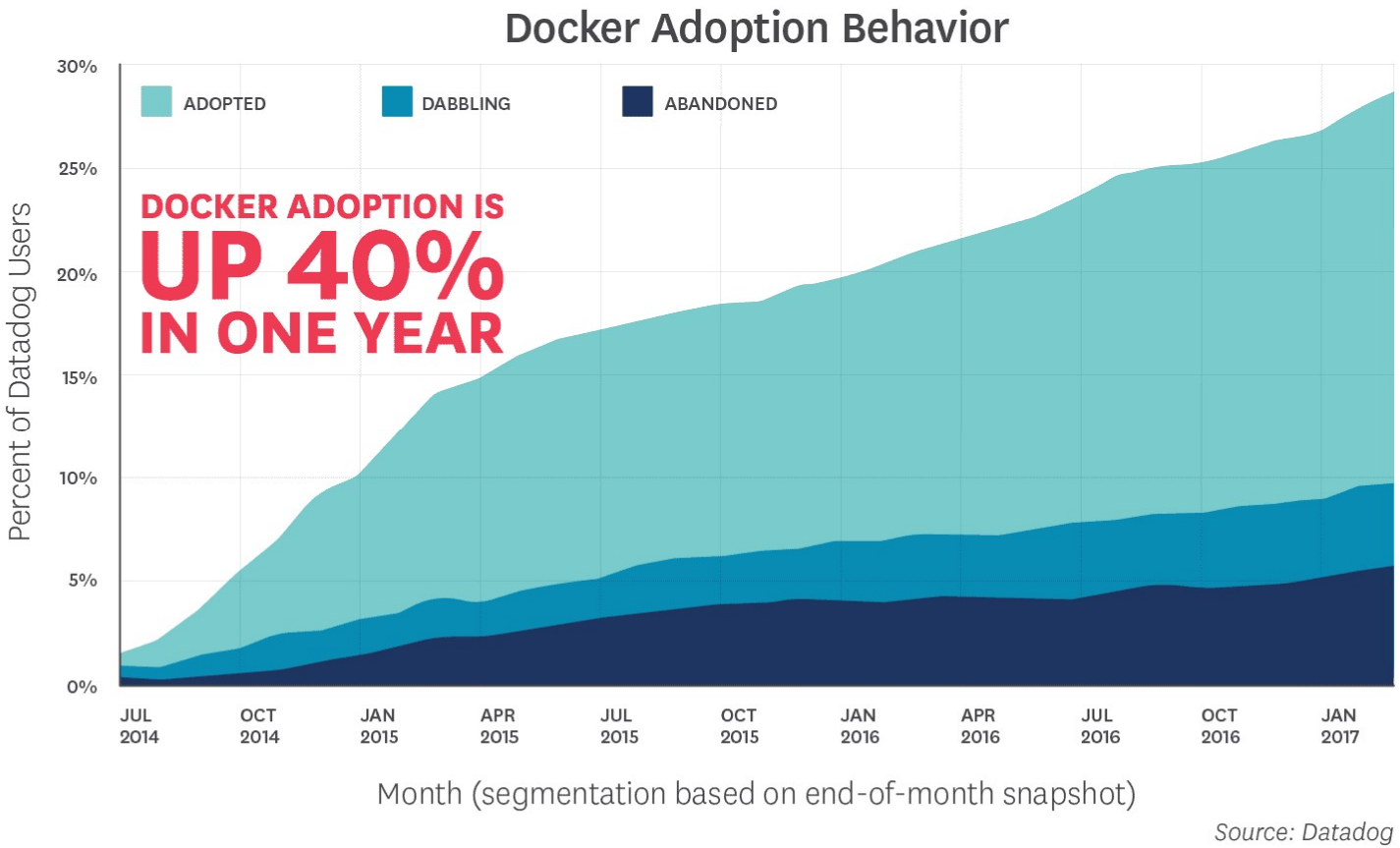
- 시대적 대세이다.



Docker (Cont.)

Docker 를 사용하는 이유

- 시대적 대세이다.



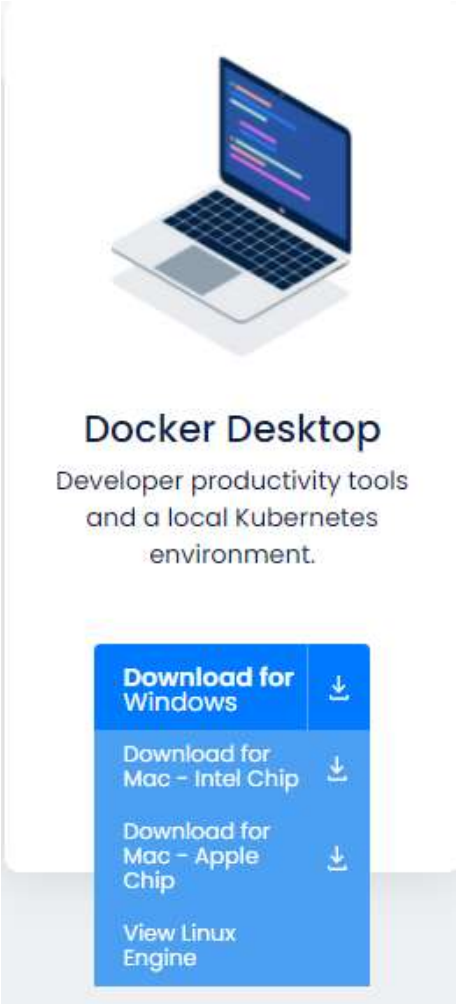
Source: Datadog

Docker Installation

Docker Installation

macOS or Windows

- Docker for Mac / Docker for Windows
- Docker는 기본적으로 Linux를 지원하기 때문에 macOS와 Windows에 설치되는 Docker는 가상머신에 설치됨.
  - macOS → xhyve
  - Windows → Hyper-V
- Windows는 Windows WSL2를 이용



Docker Installation

macOS or Windows

```
C:₩Users₩MZC01-HENRY>docker version
Client:
 Cloud integration: 1.0.17
 Version:           20.10.8
 API version:       1.41
 Go version:        go1.16.6
 Git commit:        3967b7d
 Built:             Fri Jul 30 19:58:50 2021
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.8
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.6
  Git commit:       75249d8
  Built:            Fri Jul 30 19:52:31 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.9
  GitCommit:        e25210fe30a0a703442421b0f60afac609f950a3
 runc:
  Version:          1.0.1
  GitCommit:        v1.0.1-0-g4144b63
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Docker Installation

Linux

```
docker-ubuntu@ubuntu-server:~$ sudo docker version
Client: Docker Engine - Community
 Version:           20.10.9
 API version:       1.41
 Go version:        go1.16.8
 Git commit:        c2ea9bc
 Built:             Mon Oct  4 16:08:29 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.9
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.8
  Git commit:       79ea9d3
  Built:            Mon Oct  4 16:06:37 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.11
  GitCommit:        5b46e404f6b9f661a205e28d59c982d3634148f8
 runc:
  Version:          1.0.2
  GitCommit:        v1.0.2-0-g52b36a2
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
docker-ubuntu@ubuntu-server:~$
```

Lab. Docker Installation

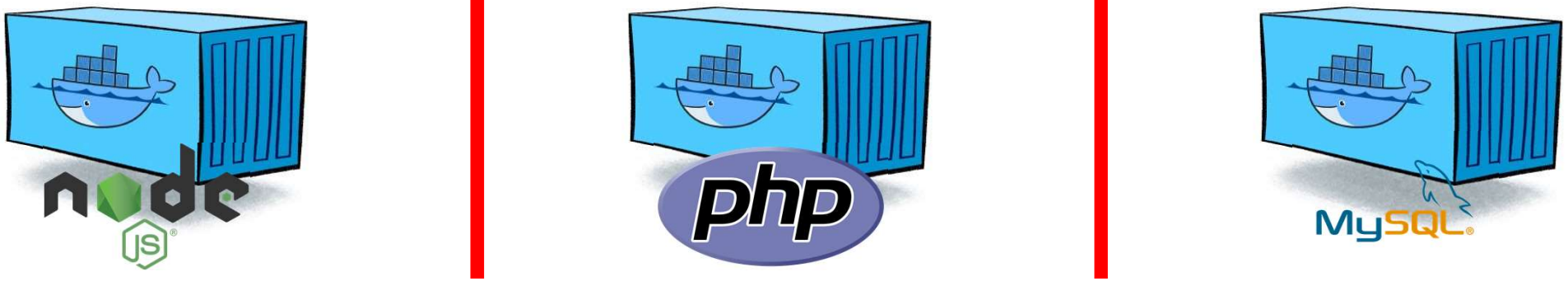
Lab. Installation WSL2 on Windows 10
Lab. Installation Docker on Windows 10 WSL2
Lab. Installation Docker on Ubuntu 20.04 LTS
Lab. Installation Docker on AWS EC2

Ref to https://docs.docker.com/engine/install/

Docker Container Overview

Docker Container

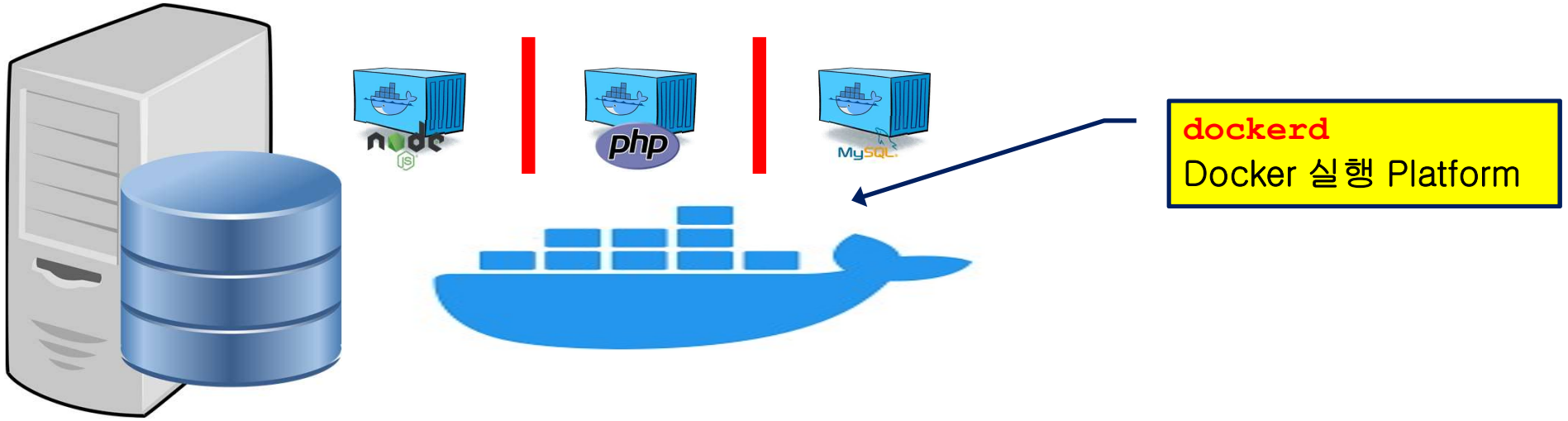
Container vs Container Image



- Container는 하나의 Application Process이다.
- 완벽하게 Memory 상에서 Isolate

Docker Container (Cont.)

Container vs Container Image



- Container는 하나의 Application Process이다.
- 완벽하게 Memory 상에서 Isolate
- Host입장에서 Docker Container는 하나의 Process이다.

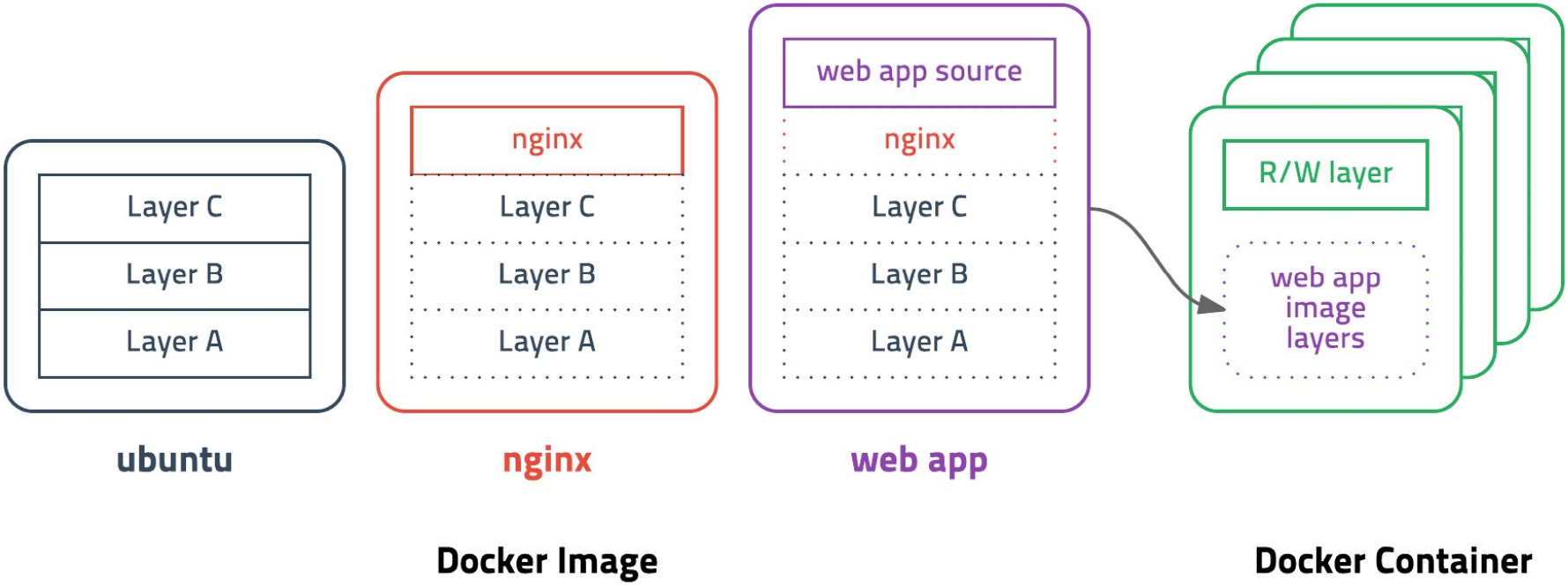
Docker Container (Cont.)

Container vs Container Image

- Container Image
  - Docker는 ***Layered File System*** 기반이다.
  - **aufs**, **btrfs**, **Overlayfs**
  - 적층형 이미지 → 효율적 저장소 관리/Network bandwidth 절감
  - Image는 Process가 실행되는 File들의 집합(환경)
  - Process는 환경(파일)을 변경할 수 있음.
  - 이 환경을 저장해서 새로운 이미지를 만든다.
  - 종류
    - Read Only
    - Writable

Docker Container (Cont.)

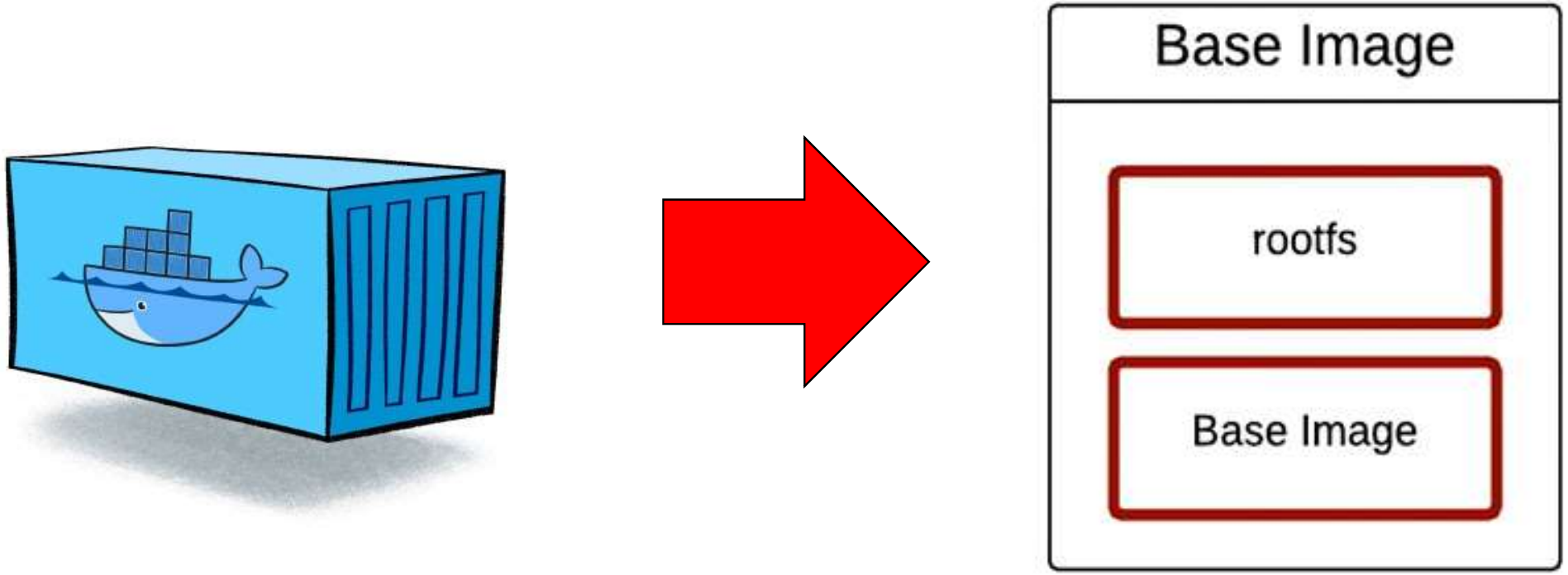
Container vs Container Image



https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

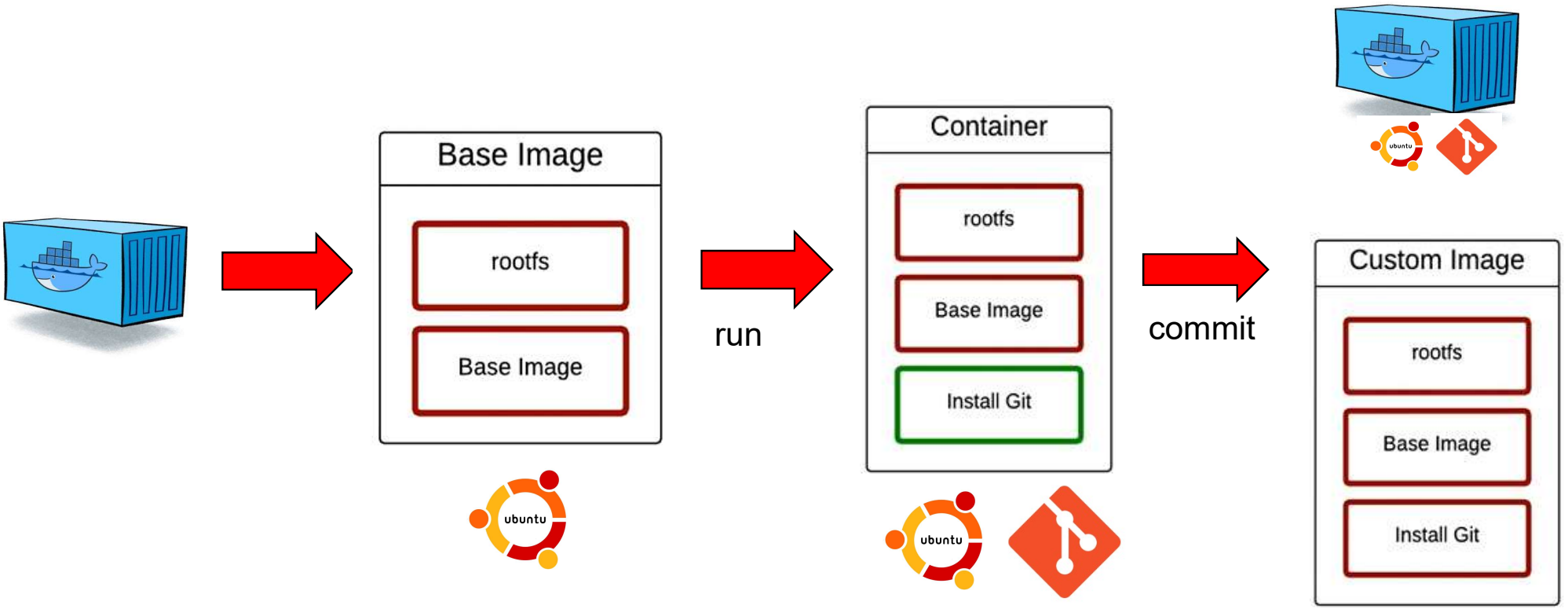
Docker Container (Cont.)

Container vs Container Image



Docker Container (Cont.)

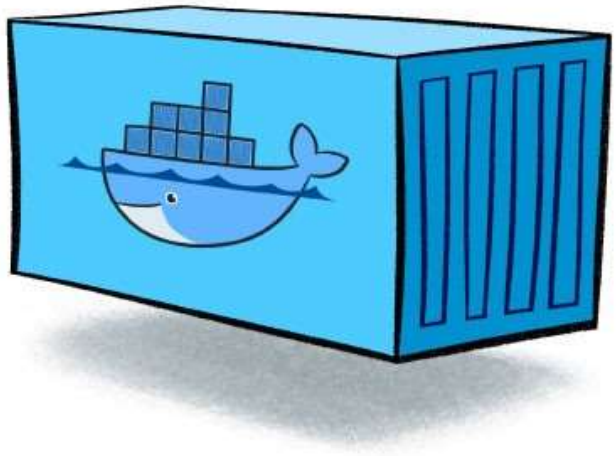
Container vs Container Image

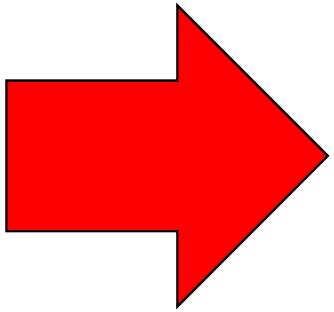


Lab1. ubuntu Base Image + git = Custom Image

Docker Container (Cont.)

Container vs Container Image





$ run node hello.js

hello.js

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.end();
});
server.on('clientError', (err, socket) => {
  socket.end('HTTP/1.1 400 Bad Request\r\n\r\n');
});
server.listen(8000);
```

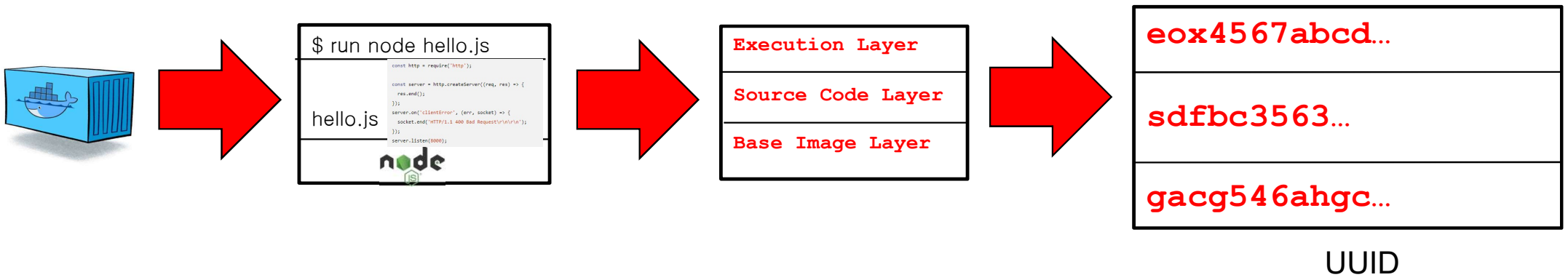
node

Docker Container (Cont.)

Container vs Container Image

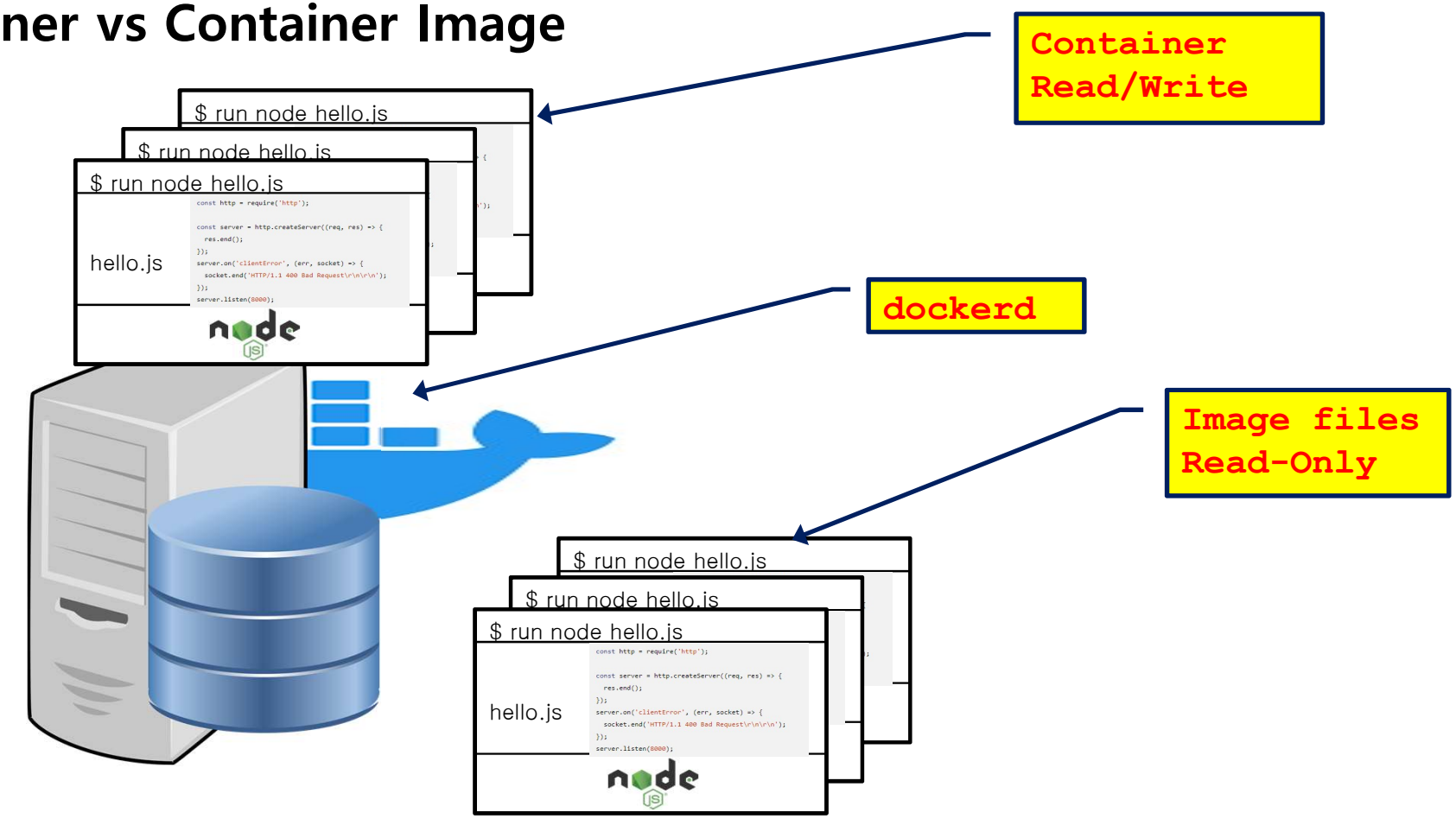
Docker Container (Cont.)

Container vs Container Image



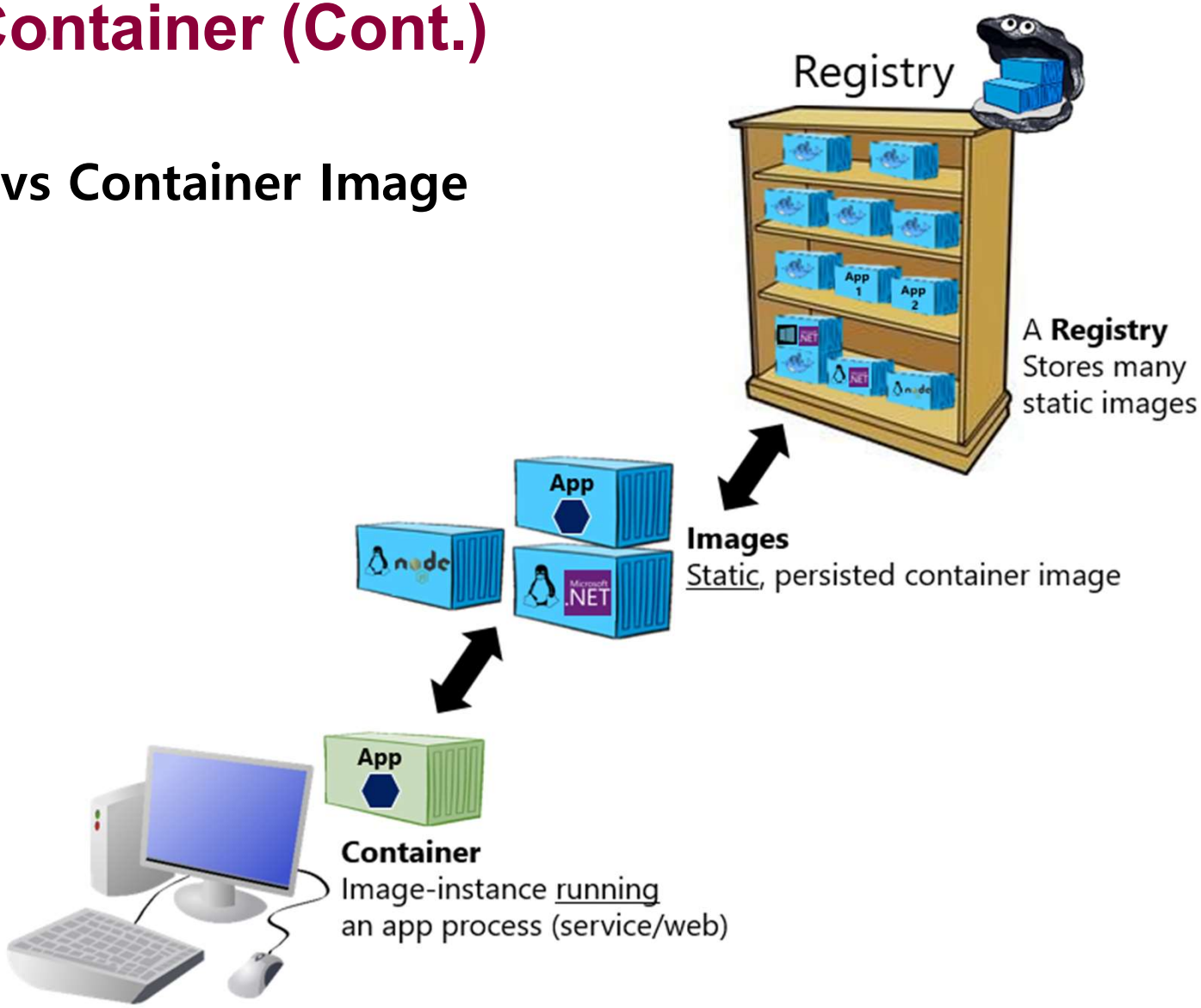
Docker Container (Cont.)

Container vs Container Image



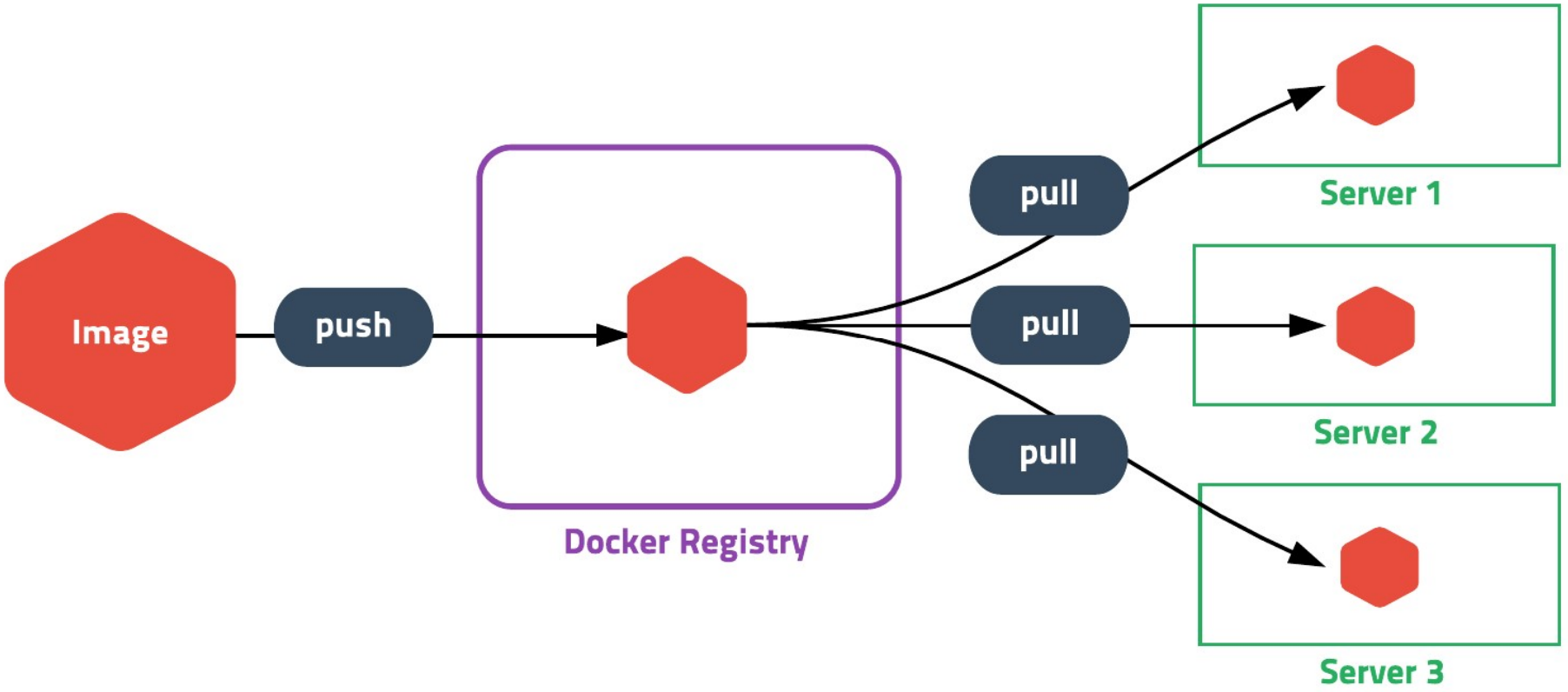
Docker Container (Cont.)

Container vs Container Image



Docker Container (Cont.)

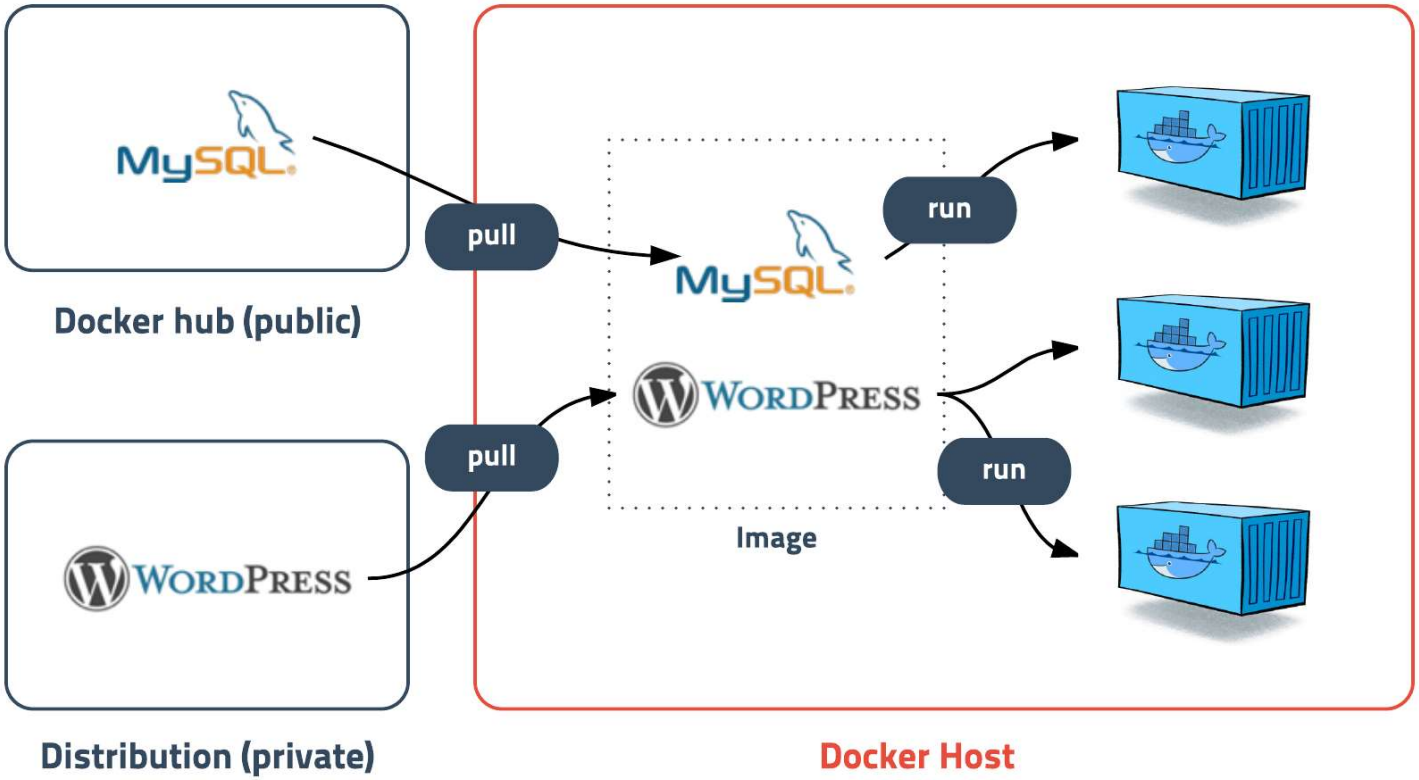
Container vs Container Image



https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

Docker Container (Cont.)

Container vs Container Image

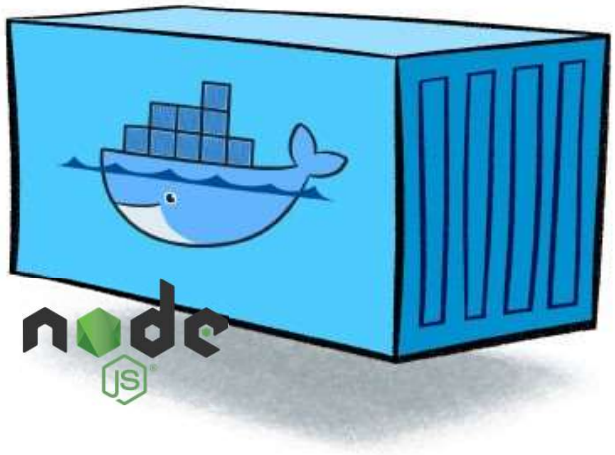


https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

Docker Container (Cont.)

Terminology

- Docker Host(Linux Kernel)
- Docker Daemon : `systemctl start docker`
- Docker Client Command : `docker`
- Docker Hub
- Container Image
- Container



Lab2. Docker Containers

Docker Container 생성하기

무엇을 Container로 만들 것인가?

$ run node hello.js

hello.js

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.end();
});
server.on('clientError', (err, socket) => {
  socket.end('HTTP/1.1 400 Bad Request\r\n\r\n');
});
server.listen(8000);
```

- 개발한 Application(실행 File)과 운영환경이 모두 들어있는 독립된 공간
- 개발한 프로그램과 실행환경 모두 Container로 만든다.

Docker Container 생성하기

Container Image Build 하기

- docker build –t {ImageName : ImageTag} {Build Context}

```
$ docker build -t sample:1 .
```

- 현재 Directory의 Dockerfile로 Build함.
  - **-f** <Dockerfile 위치> : 다른 위치의 Dockerfile 파일 사용 가능
  - **-t** : Docker Image 이름 지정
  - {Namespace}/{ImageName}:{Tag}
- 마지막에는 Build Context 위치 지정
- 현재 Directory를 의미하는 **.** 을 주로 사용
- 필요한 경우 다른 Director 지정 가능

Docker Container 생성하기

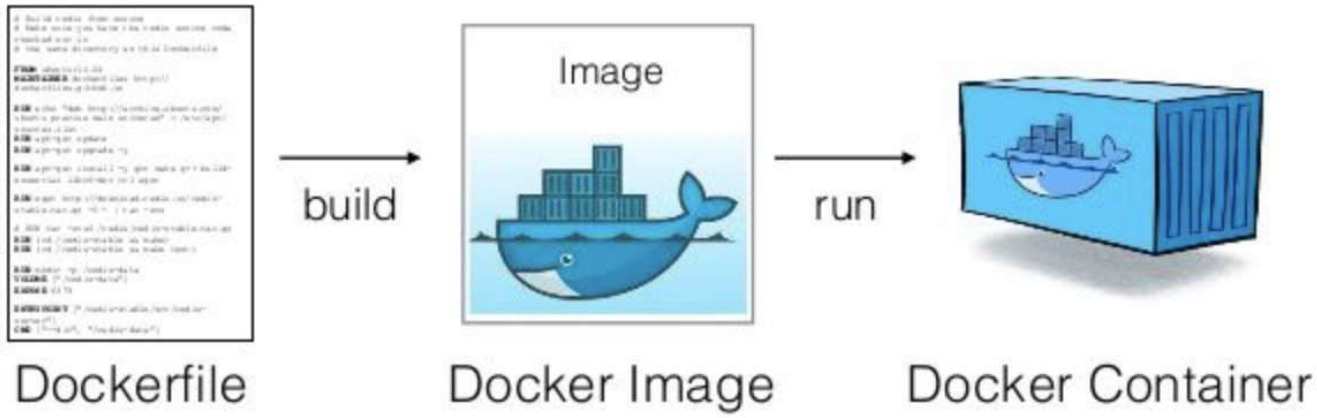
Container Image Build 하기

- .dockerignore
  - `.gitignore`와 비슷한 역할
  - Docker Build Context에서 지정된 패턴의 파일 무시
  - `.git`이나 민감한 정보를 제외하는 용도로 주로 사용
  - `.git`이나 Asset Directory만 제외시켜도 Build 속도가 개선됨.
  - Image Build시에 사용하는 파일은 제외하면 안됨.

Docker Container 생성하기 (Cont.)

Dockerfile

- 작업 지시서
- Dockerfile을 이용해서 Container를 Build한다.



https://itnext.io/intro-to-docker-part-1-5b1162c81735

Docker Container 생성하기 (Cont.)

Dockerfile

- 쉽고 간단하고 명확한 구문을 가진 Text File
- Top-Down 방식으로 해석
- Container Image를 생성할 수 있는 고유의 지시어를 가짐
- 대소문자 구별 하지 않지만, 가독성을 위해 사용 권장

```
FROM busybox
ENV FOO=/bar
WORKDIR ${FOO}   # WORKDIR /bar
ADD . $FOO       # ADD . /bar
COPY \$FOO /quux # COPY $FOO /quux
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

- **#** : Comment
- **FROM**
  - Container의 Base Image(운영환경)
  - FROM ubuntu:latest
  - FROM node:12
  - FROM python:3
- **MAINTAINER → LABEL**
  - Container Metadata 정보

```
FROM [--platform=<platform>] <image> [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

- **RUN**
  - Container Build를 위해 Base Image에서 실행할 Command
  - RUN apt-get update
  - RUN npm install

```
RUN ["/bin/bash", "-c", "echo hello"]
```

- **COPY**
  - Container Build할 때 Host의 File을 Container 안으로 복사
  - COPY index.html /var/www/html/
  - COPY ./app /usr/src/app

```
COPY hom?.txt /mydir/
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

- **ADD**
  - Container Build할 때 Host의 File(tar, url 포함)을 Container 안으로 복사

```
ADD hom?.txt /mydir/
```

- **WORKDIR**
  - Container Build할 때 명령이 실행될 작업 Directory 설정
  - WORKDIR /app

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

```
ENV MY_NAME="John Doe"
ENV MY_DOG=Rex\ The\ Dog
ENV MY_CAT=fluffy
```

- **ENV**
  - 환경 변수 지정
- **USER**
  - 명령 및 Container 실행할 때, 적용할 User 설정

```
FROM microsoft/windowsservercore
# Create Windows user in the container
RUN net user /add patrick
# Set it for subsequent commands
USER patrick
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

- **VOLUME**
  - File 또는 Directory를 Container의 Directory로 Mount
- **EXPOSE**
  - Container 동작할 때, 외부에서 사용할 Port 지정
  - EXPOSE 8000

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

Docker Container 생성하기 (Cont.)

Dockerfile Syntax

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

- **CMD**
  - Container 동작할 때, 자동으로 실행할 Service 나 Script 지정
  - CMD [“node”, “app.js”]
  - CMD node app.js
- **ENTRYPOINT**
  - **CMD**과 함께 사용하면서 Command 지정시 사용
  - Runtime시에 최종적으로 실행되는 명령어

```
ENTRYPOINT ["executable", "param1", "param2"]
```

Docker Container 생성하기 (Cont.)

Dockerfile 사용하기

- Dockerfile 예제보기 → DockerHub

```
$ mkdir build
$ cd build
$ vi hello.js
```

```
var http = require('http');

var server = http.createServer();

server.addListener('request', function (request, response) {
    console.log('requested...');
    response.writeHead(200, {'Content-Type' : 'text/plain'});
    response.write('Hello nodejs');
    response.end();
});

server.addListener('connection', function(socket){
    console.log('connected...');
});

server.listen(8888);
```

```
$ docker build -t hellojs:latest .
```

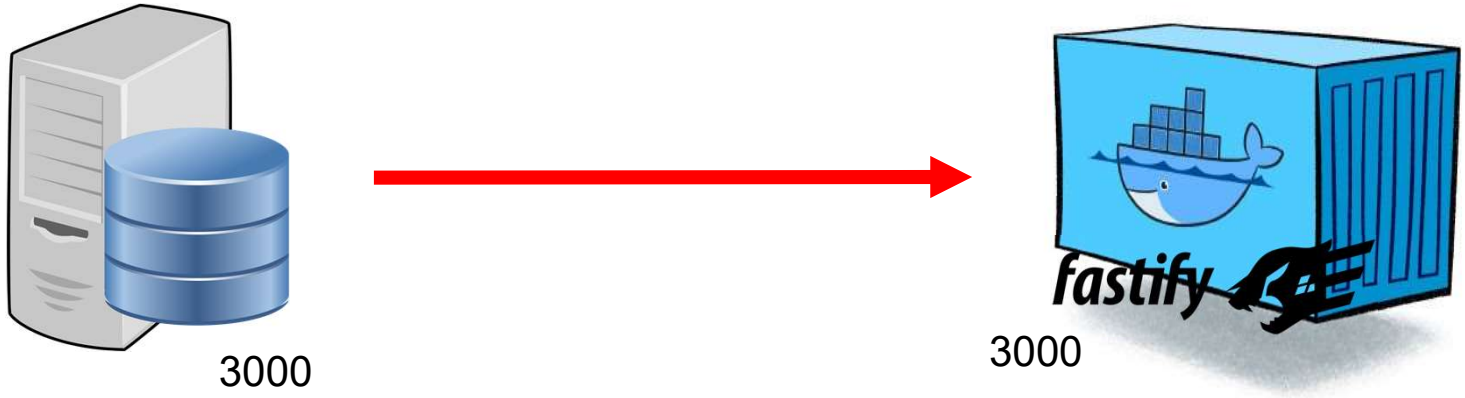
```
FROM node:12
COPY hello.js /
CMD [“node”, “/hello.js”]
```

Docker Container 생성하기 (Cont.)

실습 : Nodejs Web Application

- https://www.fastify.io/docs/latest/Getting-Started/
- Requirements
  - Source File 복사 : COPY 명령어 사용
  - node_modules 제외 : .dockerignore 사용

node_modules/*



Docker Container 생성하기 (Cont.)

실습 : Nodejs Web Application

- app.js

```
// Require the framework and instantiate it
const fastify = require('fastify')({
  logger: true
})

// Declare a route
fastify.get('/', function (request, reply) {
  reply.send({ hello: 'world' })
})

// Run the server!
fastify.listen(3000, function (err, address) {
  if (err) {
    fastify.log.error(err)
    process.exit(1)
  }
  fastify.log.info(`server listening on ${address}`)
})
```

Docker Container 생성하기 (Cont.)

실습 : Nodejs Web Application

- Dockerfile

```
# 1. nodejs 설치
FROM        ubuntu:20.04
RUN           apt update
RUN           DEBIAN_FRONTEND=noninteractive apt-get -y install
nodejs npm

# 2. Source File 복사
COPY          .  /usr/src/app

# 3. Nodejs Packages 설치
WORKDIR  /usr/src/app
RUN             npm install

# 4. Web Server 실행
EXPOSE     3000
CMD             node app.js
```

Docker Container 생성하기 (Cont.)

실습 : Nodejs Web Application

- .dockerignore

```
node_modules/*
```

- Image Build

```
$ docker build -t myweb .
```

- Container Run

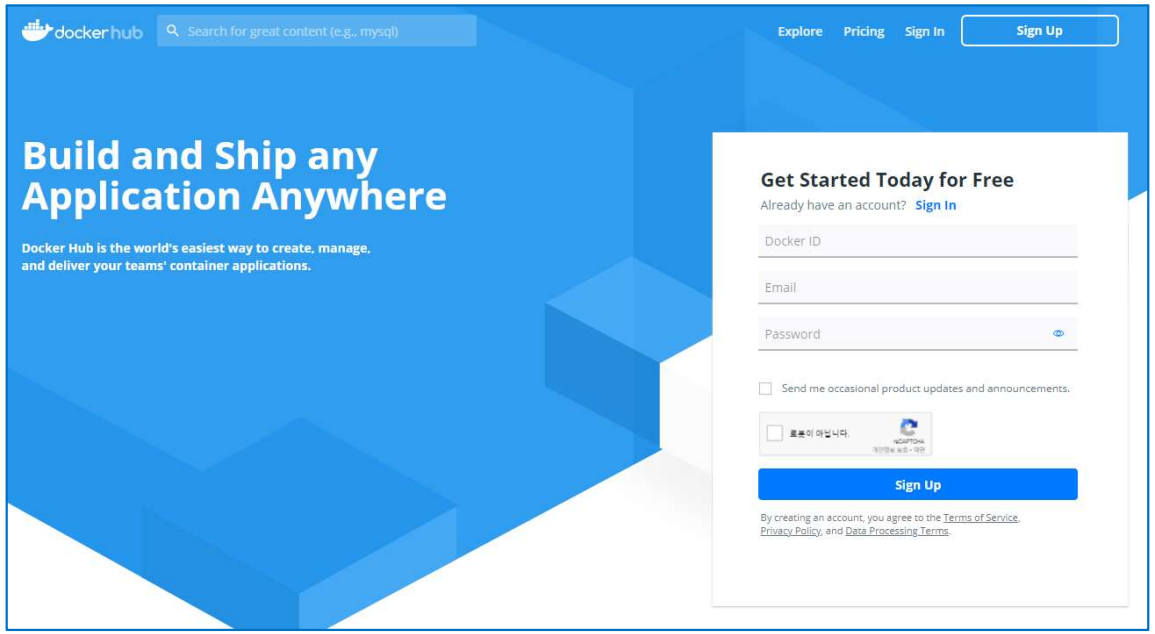
```
$ docker run --rm -d -p 3000:3000 myweb
```

Lab3. Nodejs Web Application Container

Docker Container 배포하기

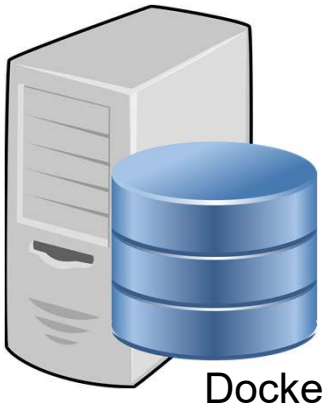
Container Image 저장소

- Docker Hub
- https://hub.docker.com/
- 회원 가입 필요
  - 무료 Push 시 Public Registry
  - Private Registry는 Image 한 개까지만 가능
  - Private Registry에 유료로 push 시 제한없음.

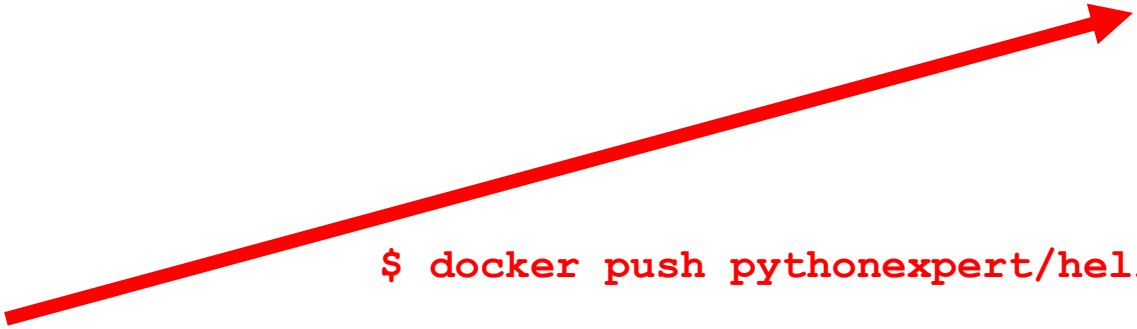


Docker Container 배포하기 (Cont.)

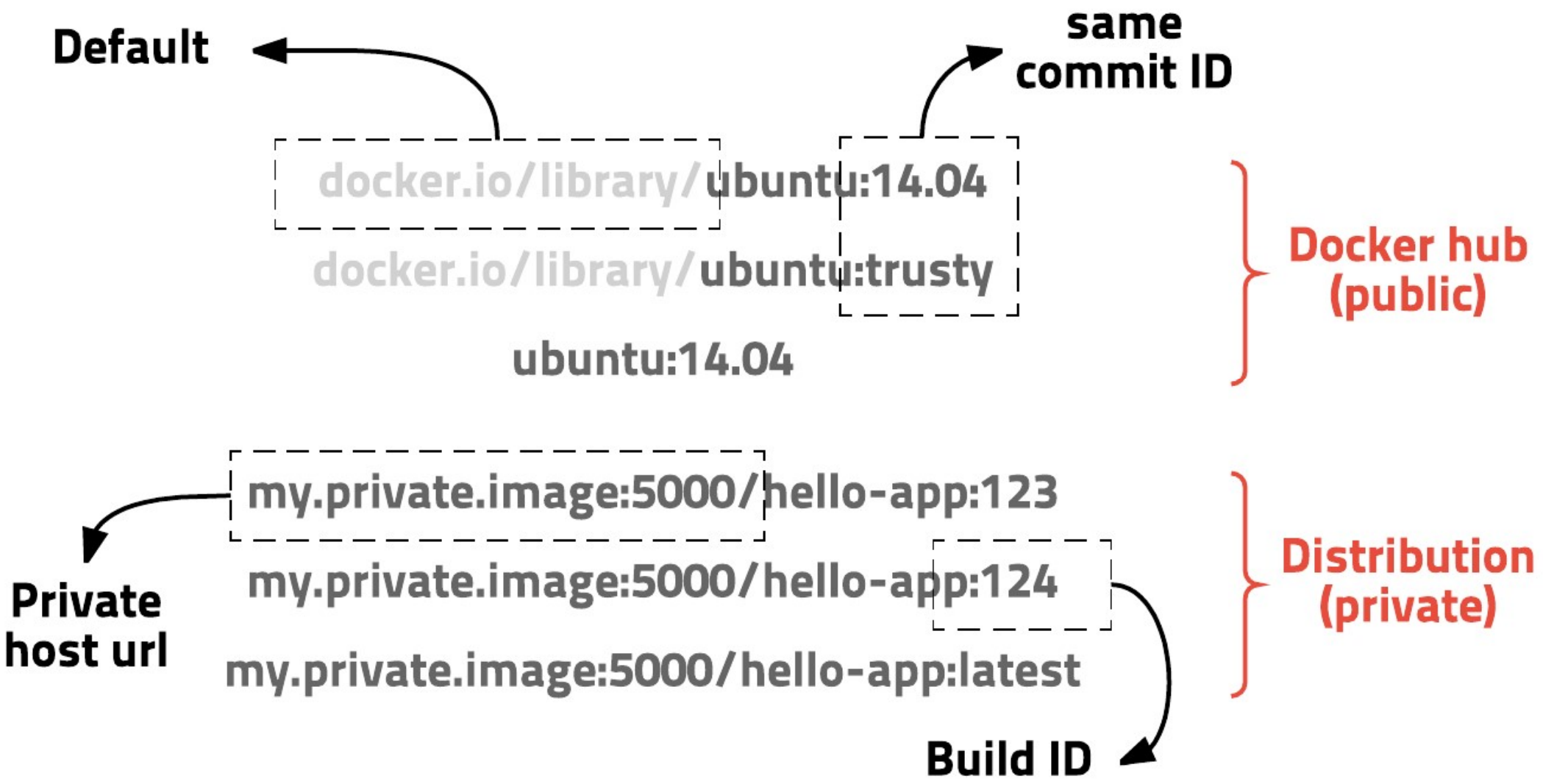

```
$ docker run -dp 80:80 pythonexpert/hellojs:latest
```



Docker Host



```
$ docker push pythonexpert/hellojs:latest
```



https://subicura.com/2017/01/19/docker-guide-for-beginners-1.html

Lab4. Docker Container 생성 및 배포

Docker CLI

Docker CLI

Use the Docker command line

- To list available commands, either run **`docker`** with no parameters or execute **`docker help`** :

```
$ docker
Usage: docker [OPTIONS] COMMAND [ARG...]
       docker [ --help | -v | --version ]

A self-sufficient runtime for containers.

Options:
      --config string      Location of client config files (default "/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context
  -D, --debug              Enable debug mode
      --help               Print usage
  -H, --host value         Daemon socket(s) to connect to (default [])
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Commands:
    attach    Attach to a running container
    # […]
```

Docker CLI (Cont.)

Base Commands

- attach, build, builder, checkpoint, commit, config, container, context, cp, create, diff, events, exec, export, history, image, images, import, info, inspect, kill, load, login, logout, logs, manifest, network, node, pause, plugin, port, ps, pull, push, rename, restart, rm, rmi, run, save, search, secret, service, stack, start, stats, stop, swarm, system, tag, top, trust, unpause, update, version, volume, wait

Docker CLI (Cont.)

$ docker run

- Run a command in a new container.
- Usage

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

- Options
  - `-d` : detached mode(백그라운드 모드)
  - `-p` : Host와 Container의 Port 연결
  - `-v` : Host와 Container의 Directory 연결
  - `-e` : Container내에서 사용할 환경변수 설정
  - `--name` : Container 이름 설정
  - `--rm` : Process 종료시 Container 자동제거
  - `-it` : `-i` & `-t` 사용, Terminal 입력을 위한 옵션
  - `--network` : Network 연결

Docker CLI (Cont.)여기에 수식을 입력하십시오.

$ docker run

```
$ docker run ubuntu:20.04
$ docker run --rm -it ubuntu:20.04 /bin/bash
$ docker run --rm -it centos:8 /bin/sh
$ docker run --rm -p 5678:5678 hashicorp/http-echo \
   -text="Hello World"
$ docker run -d --rm -p 1234:6379 redis
```

Docker CLI (Cont.)

$ docker run

```
$ docker run -d -p 3306:3306 \
  > -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
  > --name mysql \
  > mysql:5.7
$ docker exec -it mysql mysql
$ docker run -d -p 8080:80 \
  > -e WORDPRESS_DB_HOST=host.docker.internal \
  > -e WORDPRESS_DB_NAME=wp \
  > -e WORDPRESS_DB_USER=wp \
  > -e WORDPRESS_DB_PASSWORD=wp \
  > wordpress
```

Docker CLI (Cont.)

$ docker exec

- Run a command in a running container.
- Usage

```
$ docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

- Options
  - `-d` : detached mode(백그라운드 모드)
  - `-i` : interactive
  - `-t` : Allocate a pseudo-TTY, Terminal
  - `-e` : Container내에서 사용할 환경변수 설정
  - `-w` : Container내에서 Working directory

Docker CLI (Cont.)

$ docker ps

- List containers.
- Usage

  ```
  $ docker ps [OPTIONS]
  ```

- Options
  - `-a` : Show all containers

Docker CLI (Cont.)

$ docker stop

- Stop one or more running containers.
- Usage

```
$ docker stop [OPTIONS] CONTAINER [CONTAINER…]
```

Docker CLI (Cont.)

$ docker start

- Start one or more stopped containers.
- Usage

```
$ docker start [OPTIONS] CONTAINER [CONTAINER…]
```

Docker CLI (Cont.)

$ docker attach

- Attach local standard input, output, and error streams to a running container.
- Usage

```
$ docker attach [OPTIONS] CONTAINER
```

Docker CLI (Cont.)

$ docker rm

- Remove one or more containers.
- Usage

```
$ docker rm [OPTIONS] CONTAINER [CONTAINER…]
```

- Options
  - `-f` : Force the removal of a running container

Docker CLI (Cont.)

$ docker logs

- Fetch the logs of a container.
- Usage

```
$ docker logs [OPTIONS] CONTAINER
```

- Options
  - `-f` : Follow log output
  - `-tail` : Number of lines to show from the end of the logs

Docker CLI (Cont.)

$ docker images

- List images.
- Usage

```
$ docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Docker CLI (Cont.)

$ docker commit

- Create a new image from a container's changes.
- Usage

```
$ docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

Docker CLI (Cont.)

$ docker tag

- Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE.
- Usage

```
$ docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

Docker CLI (Cont.)

$ docker pull

- Pull an image or a repository from a registry.
- Usage

```
$ docker pull [OPTIONS] NAME[:TAG]
```

Docker CLI (Cont.)

$ docker rmi

- Remove one or more images.
- 단, 실행중인 Image는 삭제되지 않음.
- Usage

```
$ docker rmi [OPTIONS] IMAGE [IMAGE…]
```

- Options
  - `-f` : Force removal of the image

Docker CLI (Cont.)

$ docker login

- Log in to a Docker registry.
- Usage

  ```
  $ docker login [OPTIONS] [SERVER]
  ```

- Options
  - **`-p`**, **`--password`** : Password
  - **`-u`**, **`--username`** : Username

Docker CLI (Cont.)

$ docker push

- Push an image or a repository to a registry.
- Usage

```
$ docker push [OPTIONS] NAME[:TAG]
```

Docker CLI (Cont.)

$ docker network create

- Create a network.
- Docker Container 間 이름으로 통신할 수 있는 가상 네트워크 생성.
- Usage

```
$ docker network create [OPTIONS] NETWORK
```

Docker CLI (Cont.)

$ docker network connect

- Connect a container to a network.
- 기존에 생성된 Container에 Network를 추가한다.
- Usage

```
$ docker network connect [OPTIONS] NETWORK CONTAINER
```

Docker CLI (Cont.)

$ docker run -v

- Bind mount a volume.
- Volume Mount 명령 옵션.
- Usage

```
$ docker run -v ${PWD}/data:/data/db mongodb
```

Docker CLI (Cont.)

$ docker save

- Save one or more images to a tar archive.
- Usage

```
$ docker save [OPTIONS] IMAGE [IMAGE...]
```

- Options
  - `--output`, `-o` : Write to a file, instead of STDOUT

Docker CLI (Cont.)

$ docker load

- Load an image from a tar archive or STDIN.
- Usage

```
$ docker load [OPTIONS]
```

- Options
  - **`--input`**, **`-i`** : Read from tar archive file, instead of STDIN
  - **`--quiet`**, **`-q`** : Suppress the load output

Docker CLI (Cont.)

$ docker cp

- Copy files/folders between a container and the local filesystem.
- Usage

```
$ docker cp [OPTIONS] CONTAINER : SRC_PATH DEST_PATH
```

- Options
  - `--archive`, `-a` : Archive mode

Lab5. Docker CLI

실습

Nginx를 이용한 정적 페이지 서버 만들기

- Ref : https://hub.docker.com/_/nginx/
- Docker Image : nginx:latest
- Port : 80
- index.html 경로 : /usr/share/nginx/html
- index.html

```
<html>
  <head>
    <title>도커 이미지 예제</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  </head>
  <body>
    <h1>Nginx 서버를 도커 이미지로 만들었습니다.</h1>
  </body>
</html>
```

실습

Node.js 기반의 웹 서비스 빌드하기

- Docker Image : node:12-alpine
- Port : 8080
- Working Directory : /app
- Node 실행 : node /app/server.js
- Build한 이미지 이름 : hellonode
- Port Binding : 60000:8080
- server.js

```
const http = require('http');
const os = require('os');

const port = process.env.PORT || 8080;

process.on('SIGINT', function() {
  console.log('shutting down...');
  process.exit(1);
});

var handleRequest = function(request, response) {
  console.log(`Received request for URL: ${request.url}`);
  response.writeHead(200);
  response.end(`Hello, World!\nHostname: ${os.hostname()}\n`);
};

var www = http.createServer(handleRequest);
www.listen(port, () => {
  console.log(`server listening on port ${port}`);
});
```

Docker Container Lifecycle

Docker Container Lifecycle

Container Image를 어떻게 사용하는가?

- Image 검색하기 **docker search**

```
$ sudo docker search nginx
```

- Image 다운로드하기 **docker pull**

```
$ sudo docker pull nginx:1.14
```

- 다운로드 받은 Image 목록 출력하기 **docker images**

```
$ sudo docker images
```

- 다운로드 받은 Image 상세 보기 **docker inspect**

```
$ sudo docker inspect nginx:1.14
```

- Image 삭제하기 **docker rmi**

```
$ sudo docker rmi nginx:1.14
```

Docker Container Lifecycle (Cont.)

Container 실행 Lifecycle

- Container 생성 **docker create**

```
$ sudo docker create --name webserver -p 80:80 nginx:1.14
```

- Container 실행 **docker start**

```
$ sudo docker start webserver
```

- Container 생성 및 실행 **docker run**

```
$ sudo docker run -d -name webserver nginx:1.14
```

- 실행 중인 Container 목록 확인 **docker ps**
- 동작중인 Container 중지 **docker stop**

```
$ sudo docker stop webserver
```

- Container 삭제 **docker rm**

```
$ sudo docker rm webserver
```

Docker Container Lifecycle (Cont.)

동작중인 Container를 관리하는 명령어

- 실행중인 Container 목록 확인 **`docker ps`**
- Foreground로 실행중인 Container에 연결 **`docker attach`**
- 동작중인 Container에 새 명령어 추가 실행 **`docker exec`**

```
$ sudo docker exec -it webserver /bin/bash
```

- Container에서 동작하는 프로세스 확인 **`docker top`**

```
$ sudo docker top webserver
```

- 동작중인 Container가 생성한 log 보기 **`docker logs`**

```
$ sudo docker logs -f webserver
```

Lab6. Docker Container Lifecycle

Docker Container Registry

Docker Container Registry

Container Registry?

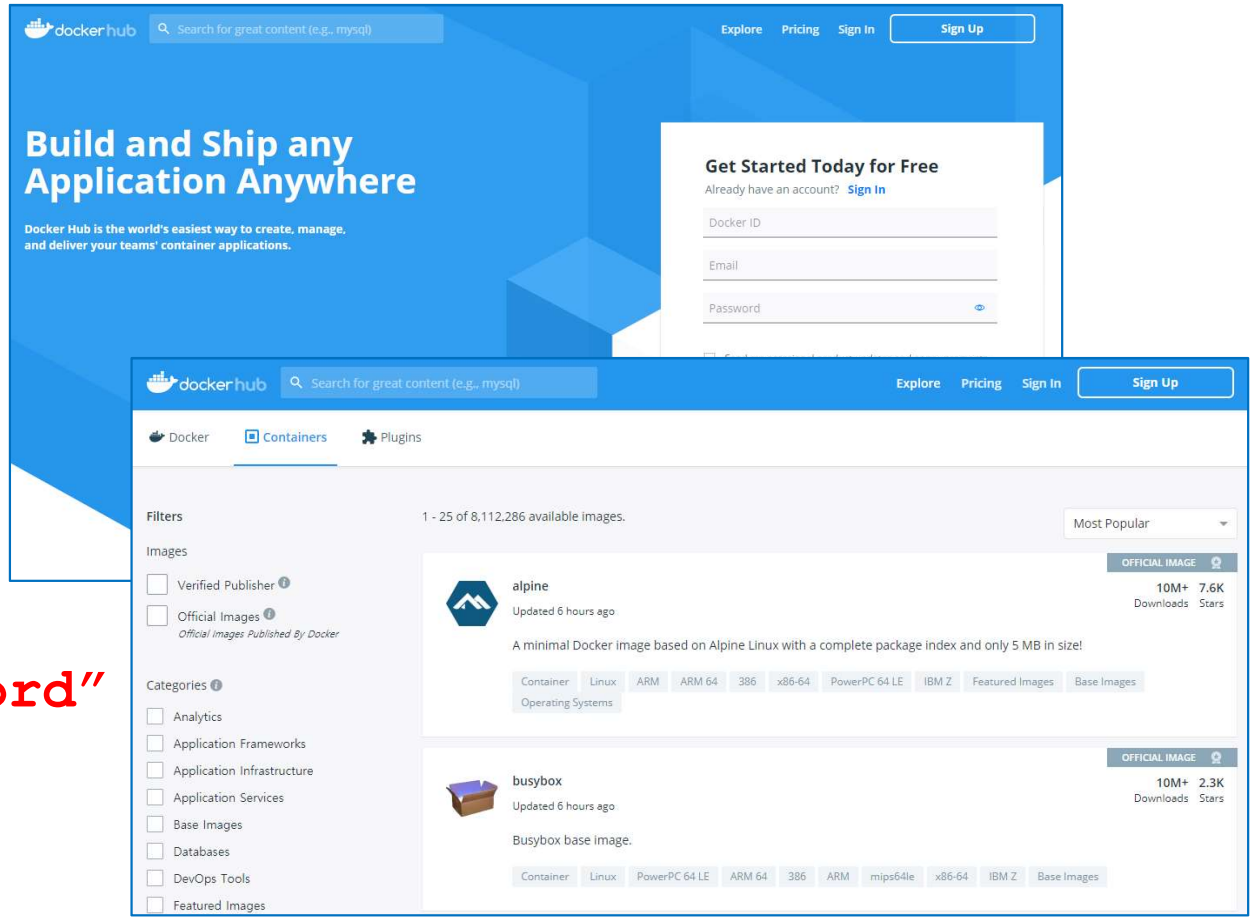
- Container Image를 저장하는 저장소
- Docker Hub
  - https://hub.docker.com/
- Private Registry
  - 社內의 Container Image 저장소

Docker Container Registry (Cont.)

Docker Hub(Registry)

- https://hub.docker.com/
- Image 종류
  - Official Images
  - Verified Publisher
  - Etc
- Image 검색

```
$ docker search "Keyword"
```



Docker Container Registry (Cont.)

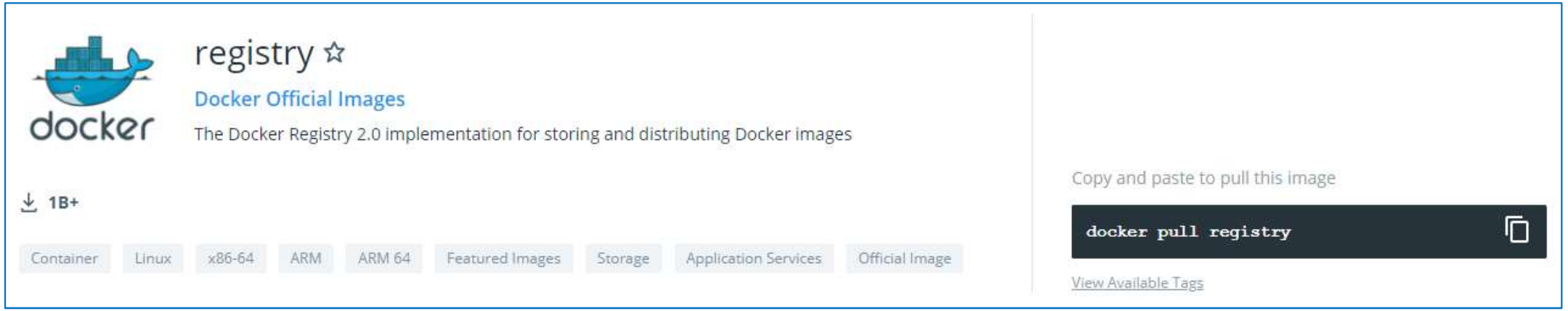
Docker Hub(Registry)

- Official Images
  - docker.com이 직접 관리
  - 누구나 사용가능
- Verified Publisher
  - Vendor가 제공하는 Image
- Categories
  - 카테고리별 검색
  - 예)Databases > mysql
- Repositories > Create Repository
  - Private Repository 생성시 1개 까지만 무료, 그 이상은 추가 비용 발생

Docker Container Registry (Cont.)

Private Repository 구축하기

- **`registry`** container를 이용해서 Private Container 운영
- **`registry`**라는 컨테이너를 운영하는 Registry
- Private Registry는 제한 없음.



Docker Container Registry (Cont.)

Private Repository 구축하기

- This image contains an implementation of the Docker Registry HTTP API V2 for use with Docker 1.6+registry.
- Run a local registry : Quick Version

```
$ docker run -d -p 5000:5000 --restart always \
  --name registry registry:2
```

Docker Container Registry (Cont.)

Private Repository 구축하기

- Image Repository
- 반드시 `hostname:port`(80 생략가능) 형식을 사용할 것

```
localhost : 5000/ubuntu:18.04
docker.example.com:5000/ubuntu:18.04
```

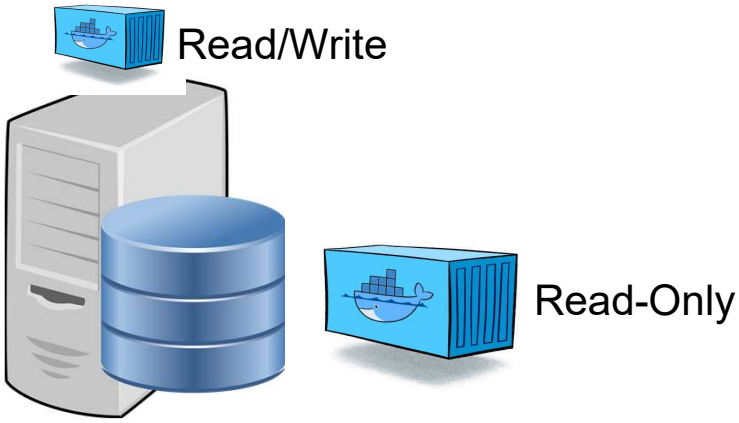
Lab7. Docker Registry

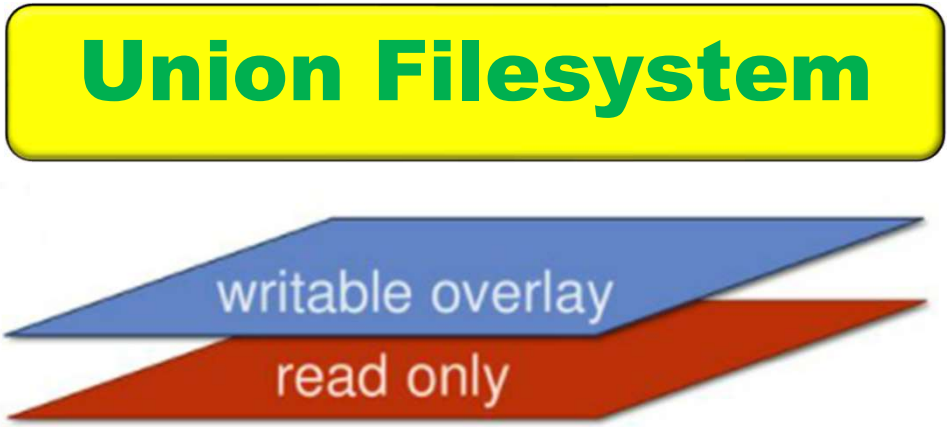
Docker Container Storage

Docker Container Storage

Container Volume

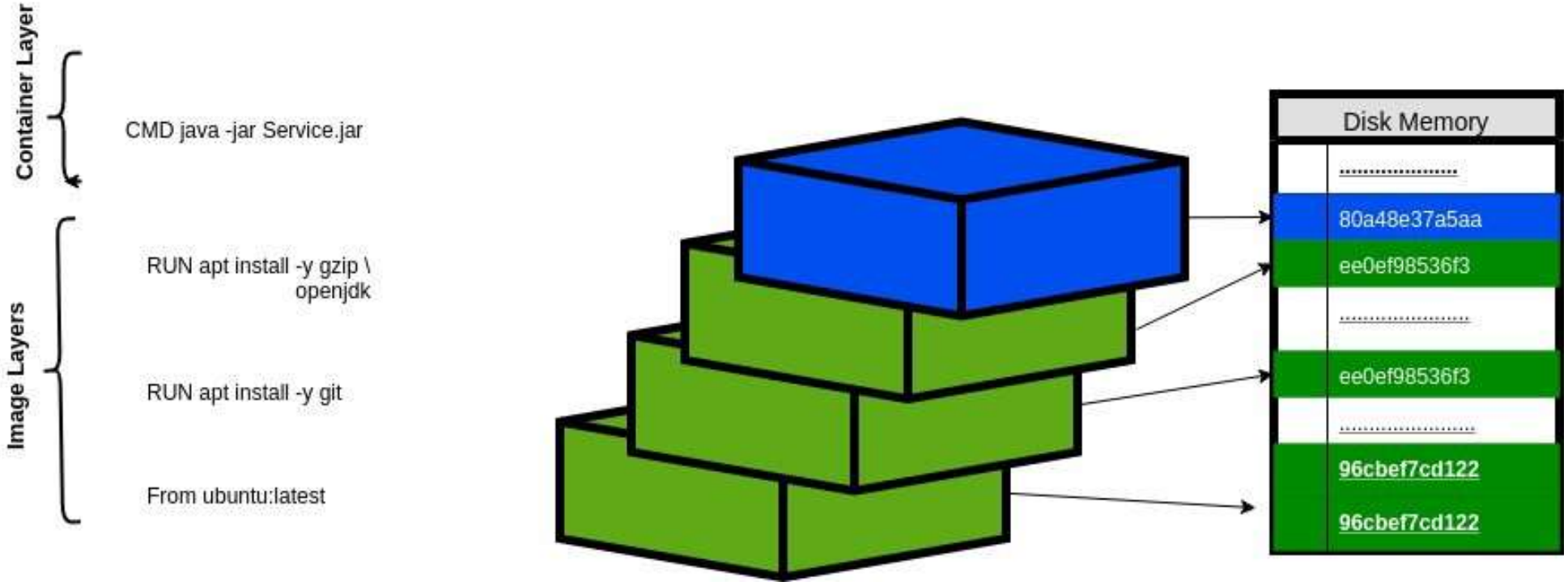
- Container Image는 Read-Only이다.
- Container에 저장(추가)되는 데이터는 별도의 Read-Write Layer에 저장됨.
- Container가 만들어주는 데이터를 영구적으로 보존





Docker Container Storage (Cont.)

Union FS



https://medium.com/@knoldus/unionfs-a-file-system-of-a-container-2136cd11a779

Docker Container Storage (Cont.)

Union FS

- Is a filesystem service for Linux, FreeBSD and NetBSD which implements a union mount for other file systems.
- It allows files and directories of separate file systems, known as branches, to be **transparently overlaid**, forming a single coherent file system.
- Contents of directories which have the same path within the merged branches will be seen together in a single merged directory, within the new, virtual filesystem.

Docker Container Storage (Cont.)

Union FS

- When mounting branches, the **priority** of one branch over the other is specified. So when both branches contain a file with the same name, one gets priority over the other.
- The different **branches may be either read-only or read/write file systems**, so that writes to the virtual, merged copy are directed to a specific real file system.
- This allows a file system to appear as writable, but without actually allowing writes to change the file system, also known as copy-on-write.

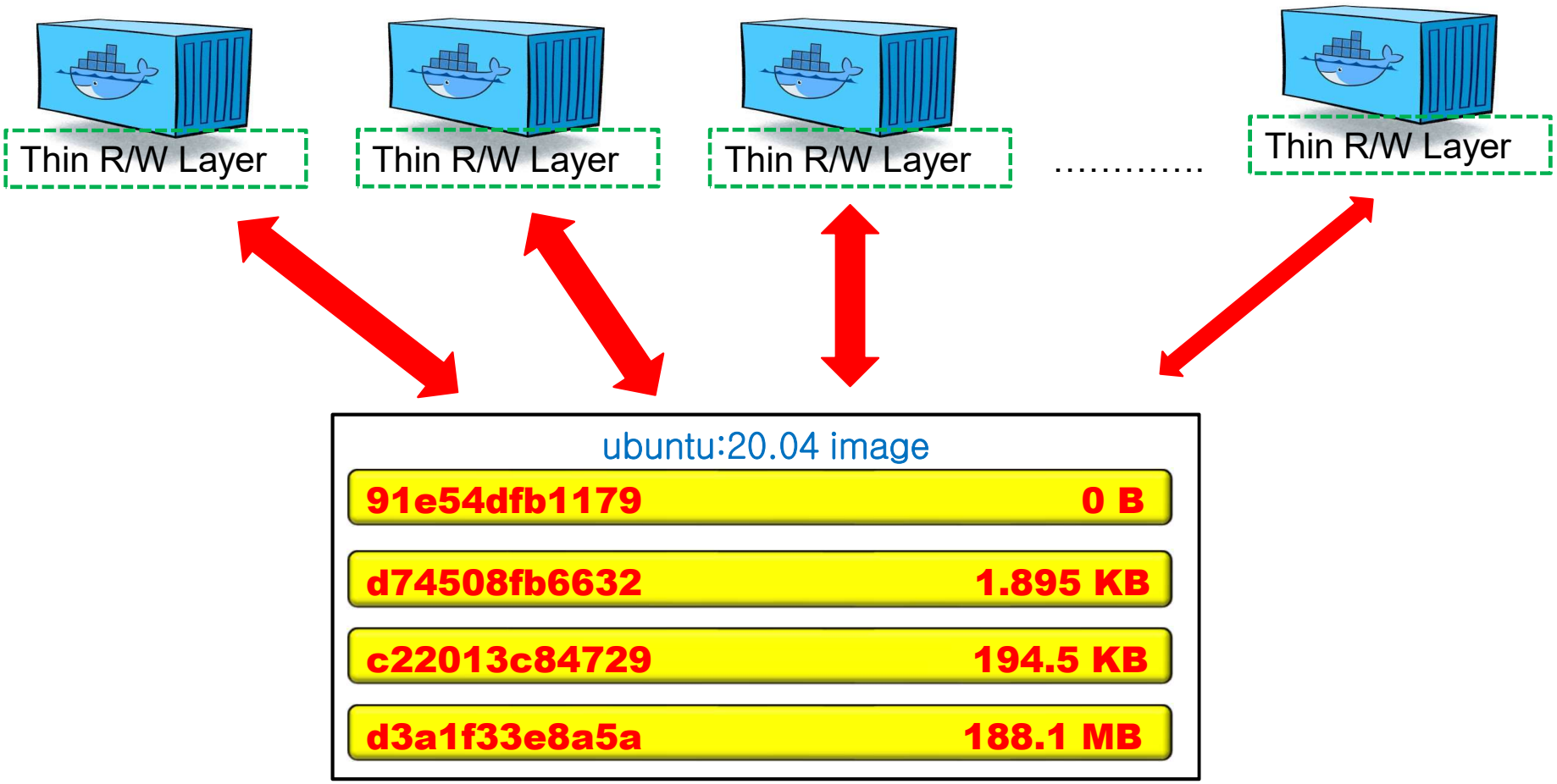
Docker Container Storage (Cont.)

Docker and Union FS

- Docker uses file systems inspired by **Unionfs**, such as **Aufs**, to layer Docker images.
- As actions are done to a base image, layers are created and documented, such that each layer fully describes how to recreate an action.
- This strategy enables Docker's lightweight images, as only layer updates need to be propagated (compared to full VMs, for example).

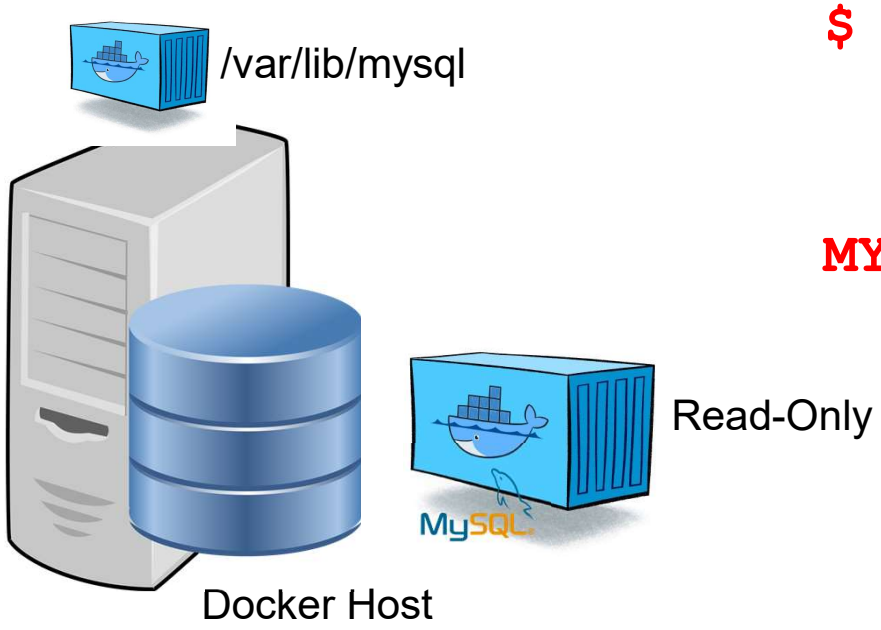
Docker Container Storage (Cont.)

Docker and Union FS



Docker Container Storage (Cont.)

Container Volume




```
$ docker run -d --name db \
    -v /dbdata:/var/lib/mysql
    -e
MYSQL_ALLOW_EMPTY_PASSWORD=true \
  mysql:latest
```

Docker Container Storage (Cont.)

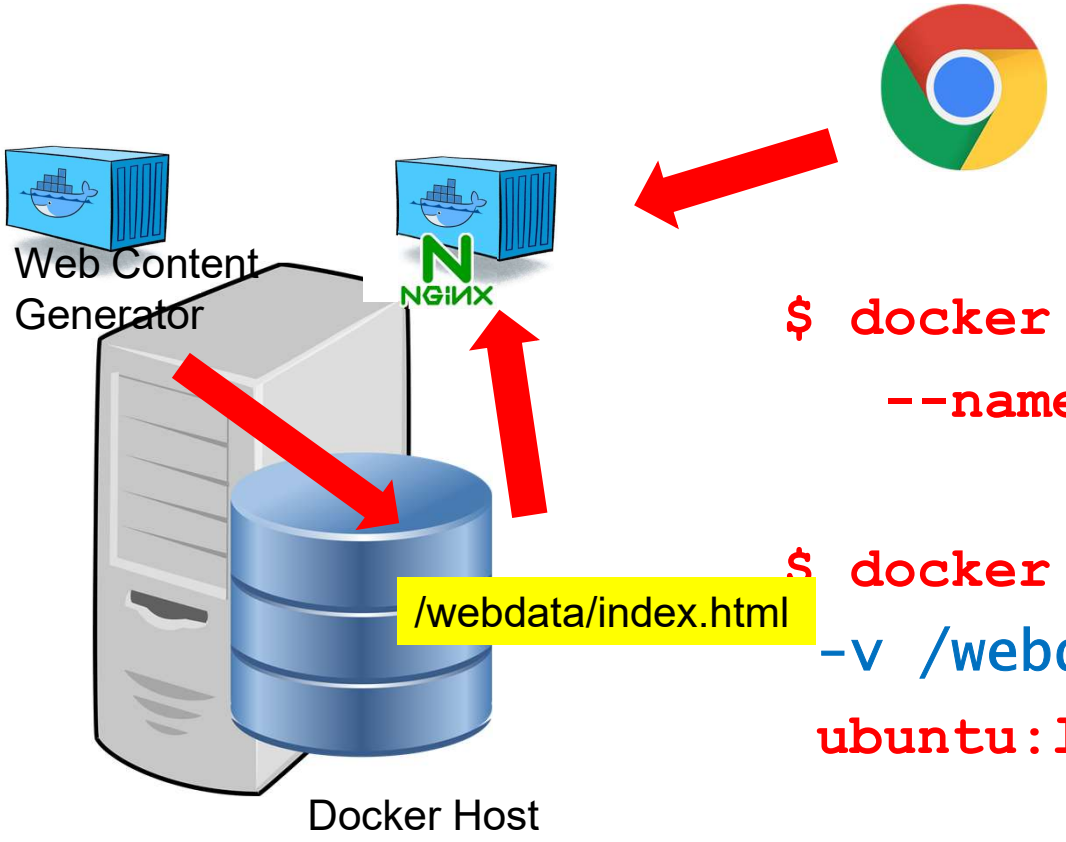
Container Volume

- volume 옵션 사용
  - **-v** {Host path} **:** {Container Mount path}
  - **-v** {Host path} **:** {Container Mount path} **:** {Read Write Mode} - **ro**, **rw**(기본)
  - **-v** {Container mount path}

```
$ docker run -d -v /dbdata:/var/lib/mysql -e MY…
$ docker run -d -v /webdata:/var/www/html:ro httpd:latest
$ docker run -d -v /var/lib/mysql -e MY…
```

Docker Container Storage (Cont.)

Container 間 데이터 공유하기




```
$ docker run -v /webdata:/webdata -d \
   --name df pythonexpert/df:latest
```

```
$ docker run -d \
-v /webdata:/usr/share/nginx/html:ro \
ubuntu:latest
```

Lab8. Docker Container Storage

Docker Network

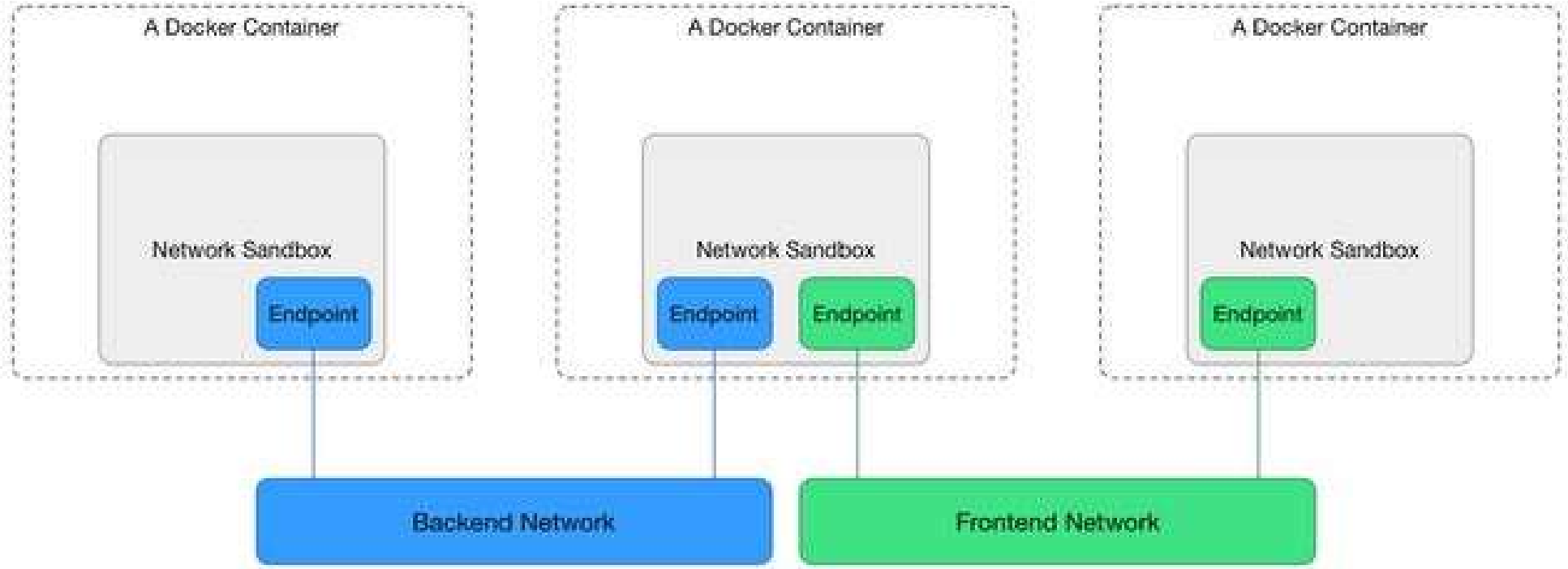
Docker Network

Container Network Model

- *libnetwork* project
- *Libnetwork* provides a native Go implementation for connecting containers.
- The goal of *libnetwork* is to deliver a robust Container Network Model that provides a consistent programming interface and the required network abstractions for applications.

Docker Network (Cont.)

Container Network Model



https://github.com/moby/libnetwork/blob/master/docs/design.md

Docker Network (Cont.)

Container Network Model

- ***Sandbox***
  - Contains the configuration of a container's network stack.
  - Includes management of the container's interfaces(eth), routing table and DNS settings.
  - May contain many endpoints from multiple networks.
- ***EndPoint***
  - Joins a Sandbox to a Network.
  - An implementation of an Endpoint could be a **`veth`** pair(eth-veth pair), an Open vSwitch internal port or similar.
  - Can belong to only one network and it can belong to only one Sandbox, if connected.

Docker Network (Cont.)

Container Network Model

- *Network*
  - Is a group of Endpoints
  - Is able to communicate with each-other directly.
  - An implementation of a Network could be a Linux bridge, a VLAN, etc.
  - Consist of many endpoints.
- Reference
  - https://github.com/moby/libnetwork/blob/master/docs/design.md
  - http://cloudrain21.com/container-networking-model
  - https://medium.com/@xiaopeng163/docker-bridge-networking-deep-dive-3e2e0549e8a0

Docker Network (Cont.)

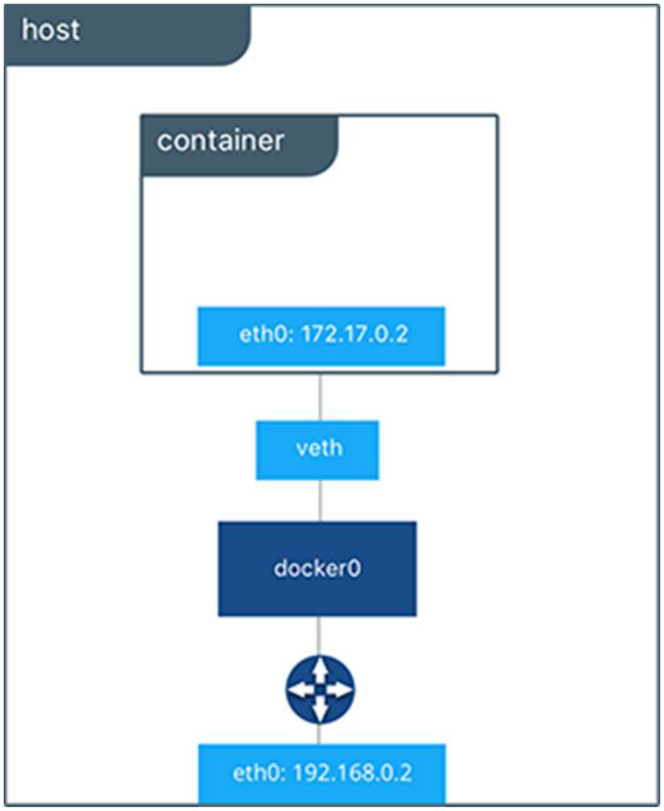
Launch a container on the default network

- Docker includes support for networking containers through the use of ***network drivers***.
- By default, Docker provides two network drivers., the ***bridge*** and the ***overlay*** drivers.
- Can also write a network driver plugin so that can create your own drivers but that is an advanced task.
- Every installation of the Docker Engine automatically includes three default networks.

```
$ docker network ls

NETWORK ID          NAME                DRIVER
18a2866682b8        none                null
c288470c46f6        host                host
7b369448dccb        bridge              bridge
```

Docker Network (Cont.)

Launch a container on the defa

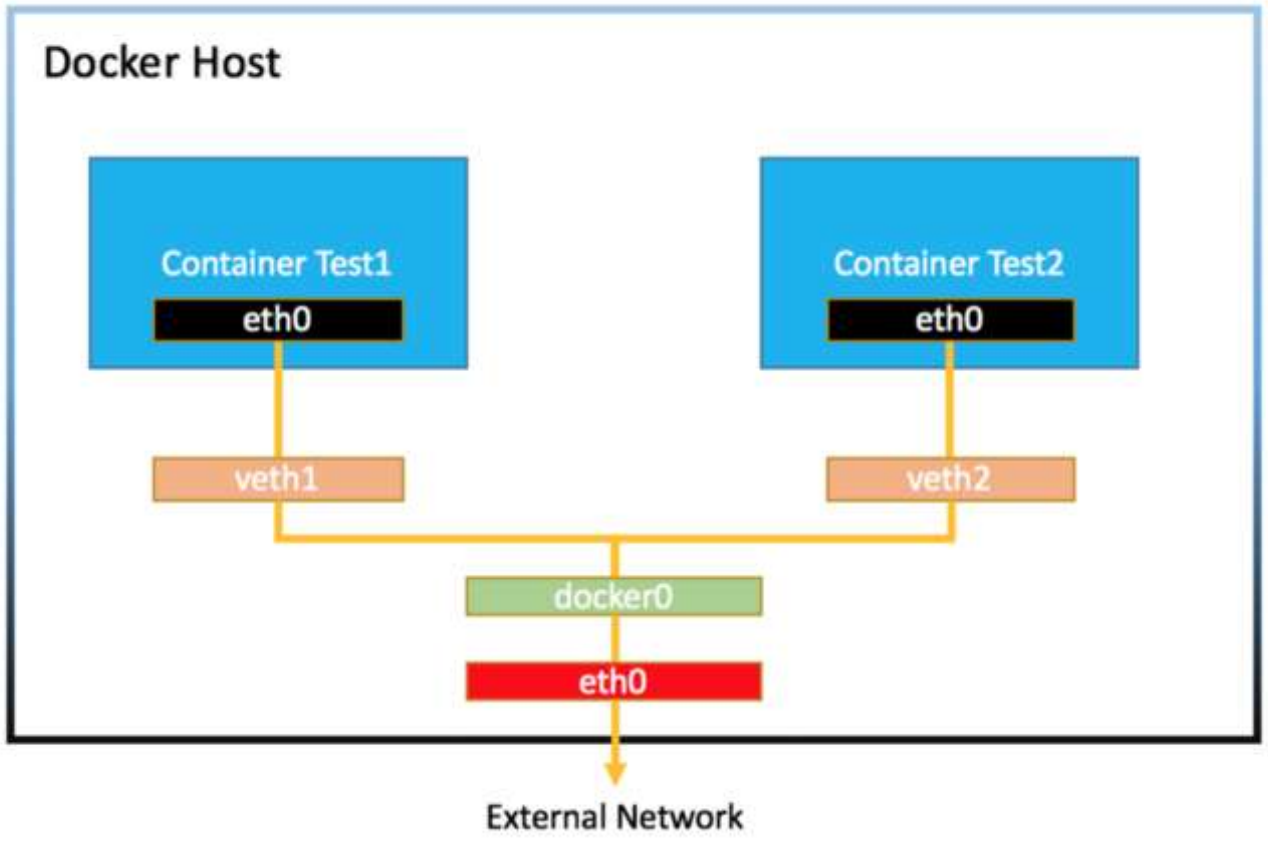



```
$ docker network inspect bridge

[
    {
        "Name": "bridge",
        "Id": "f7ab26d71dbd6f557852c7156ae0574bbf62c42f539b50c8ebde0f728a253b6f",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.1/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Containers": {
            "3386a527aa08b37ea9232cbcace2d2458d49f44bb05a6b775fba7ddd40d8f92c": {
                "Name": "networktest",
                "EndpointID": "647c12443e91faf0fd508b6edfe59c30b642abb60dfab890b4bdccee38750bc1",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "9001"
        },
        "Labels": {}
    }
]
```

https://docs.docker.com/engine/tutorials/networkingcontainers/

Docker Network (Cont.)

Launch a container on the default network



https://medium.com/@xiaopeng163/docker-bridge-networking-deep-dive-3e2e0549e8a0

Docker Network (Cont.)

Launch a container on the default network

```
docker-ubuntu@ubuntu-server:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d3:2e:d7 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.105/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed3:2ed7/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:75:8e:37 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s8
       valid_lft 345sec preferred_lft 345sec
    inet6 fe80::a00:27ff:fe75:8e37/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:ca:af:af:14 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:caff:feaf:af14/64 scope link
       valid_lft forever preferred_lft forever
```

Docker Network (Cont.)

Launch a container on the default network

- Through ***docker network*** command get more details about the **`docker0 bridge`**, and from the output, see there is no Container connected with the bridge now.

```
docker-ubuntu@ubuntu-server:~$ sudo docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
32ce6dec4771   bridge    bridge    local
ef8f1c31a15d   host      host      local
ee449dfed7eb   none      null      local
```

Docker Network (Cont

Launch a container on the

```
docker-ubuntu@ubuntu-server:~$ sudo docker network inspect 32ce6dec4771
[
    {
        "Name": "bridge",
        "Id": "32ce6dec4771b968ec9f13ef606890a860c64987df713dc09d4b71b8f225bafa",
        "Created": "2021-10-18T05:49:06.372801429Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Docker Network (Cont.)

Launch a container on the default network

- Can use **brctl** command to get bridge **docker0** information

```
docker-ubuntu@ubuntu-server:~$ brctl show

Command 'brctl' not found, but can be installed with:

sudo apt install bridge-utils

docker-ubuntu@ubuntu-server:~$ sudo apt install bridge-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  ifupdown
The following NEW packages will be installed:
  bridge-utils
```

for CentOS
 # yum install bridge-utils

```
docker-ubuntu@ubuntu-server:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242caafaf14       no
```

Docker Network (Cont.)

docker0

- virtual ethernet bridge : 172.17.0.0/16
- L2 통신 기반
- Container 생성시 **veth** 인터페이스 생성(sandbox)
- 모든 Container는 외부 통신을 **docker0**을 통해 진행
- Container running 할 때 **172.17.x.y**로 IP 주소 할당

Docker Network (Cont.)

docker0

```
docker-ubuntu@ubuntu-server:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242caafaf14       no
```

```
docker-ubuntu@ubuntu-server:~$ sudo docker pull mysql:5.7.34
5.7.34: Pulling from library/mysql
b4d181a07f80: Pull complete
a462b60610f5: Pull complete
578fafb77ab8: Pull complete
524046006037: Pull complete
d0cbe54c8855: Pull complete
aa18e05cc46d: Pull complete
32ca814c833f: Pull complete
52645b4af634: Pull complete
bca6a5b14385: Pull complete
309f36297c75: Pull complete
7d75cacde0f8: Pull complete
Digest: sha256:1a2f9cd257e75cc80e9118b303d1648366bc2049101449bf2c8d82b022ea86b7
Status: Downloaded newer image for mysql:5.7.34
```

Docker Network (Cont.)

docker0

```
docker-ubuntu@ubuntu-server:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242caafaf14       no
```

```
docker-ubuntu@ubuntu-server:~$ sudo docker run --name mysql \
> -e MYSQL_ROOT_PASSWORD=password -d -p 3306:3306 mysql:5.7.34
3d15bd9a3767348531233fe951f82fbe9b0fe905b88963469958c5cfeda9708f
docker-ubuntu@ubuntu-server:~$ sudo docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS
                    NAMES
3d15bd9a3767   mysql:5.7.34   "docker-entrypoint.s…"   6 seconds ago    Up 6 seconds    0.0.0.0:3306->3306/tcp
>3306/tcp, 33060/tcp   mysql
docker-ubuntu@ubuntu-server:~$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242caafaf14       no              veth505d121
```

Docker Network (Cont.)

docker0

```
docker-ubuntu@ubuntu-server:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d3:2e:d7 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.105/24 brd 10.0.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed3:2ed7/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:75:8e:37 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s8
       valid_lft 356sec preferred_lft 356sec
    inet6 fe80::a00:27ff:fe75:8e37/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ca:af:af:14 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:caff:feaf:af14/64 scope link
       valid_lft forever preferred_lft forever
163: veth505d121@if162: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP gro

    link/ether 2a:f9:50:4a:d1:99 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::28f9:50ff:fe4a:d199/64 scope link
       valid_lft forever preferred_lft forever
```

Docker Network (Cont.)

docker0

```
docker-ubuntu@ubuntu-server:~$ sudo docker inspect mysql
[
    {
        "Id": "3d15bd9a3767348531233fe951f82fbe9b0fe905b88963469958c5cfeda9708f",
        "EndpointID": "257b4b0707514f495d377fb615dd24e59fe18e992928230d474bda35c8468503",
        "Gateway": "172.17.0.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "MacAddress": "02:42:ac:11:00:02",
        "Networks": {
            "bridge": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": null,
                "NetworkID": "32ce6dec4771b968ec9f13ef606890a860c64987df713dc09d4b71b8f225bafa",
                "EndpointID": "257b4b0707514f495d377fb615dd24e59fe18e992928230d474bda35c8468503",
                "Gateway": "172.17.0.1",
                "IPAddress": "172.17.0.2"
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:11:00:02",
                "DriverOpts": null
            }
        }
}
```

Docker Network (Cont.)

port-forwarding

- Container Port를 외부로 노출시켜 외부 연결 허용
- iptables rule을 통한 Port 노출

```
-p hostPort : containerPort
-p containerPort
-P(대문자)
```

```
$ docker run -name web -d -p 80:80 nginx:1.14
$ iptables -t nat -L -n -v
```

```
Chain POSTROUTING (policy ACCEPT 3 packets, 446 bytes)
 pkts bytes target     prot opt in     out     source               destination
 1512 93869 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0
    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2           tcp dpt:80

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0
    2   104 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0            tcp dpt:80 to:172.17.0.2:80
```

Docker Network (Cont.)

Create your own bridge network

- Docker Engine natively supports both ***bridge networks*** and ***overlay networks***.
- A bridge network is limited to a single host running Docker Engine.
- An overlay network can include multiple hosts and is a more advanced topic.
- For this example, create a bridge network:

```
$ docker network create -d bridge my_bridge
```

Docker Network (Cont.)

Create your own bridge network

- The `-d` flag tells Docker to use the bridge driver for the new network.
- You could have left this flag off as bridge is the default value for this flag.
- Go ahead and list the networks on your machine:

```
$ docker network ls

NETWORK ID          NAME                DRIVER
7b369448dccb        bridge              bridge
615d565d498c        my_bridge           bridge
18a2866682b8        none                null
c288470c46f6        host                host
```

Docker Network (Cont.)

Create your own bridge network

- If you inspect the network, it has nothing in it.

```
$ docker network inspect my_bridge

[
    {
        "Name": "my_bridge",
        "Id": "5a8afc6364bccb199540e133e63adb76a557906dd9ff82b94183fc48c40857ac",
        "Scope": "local",
        "Driver": "bridge",
        "IPAM": {
            "Driver": "default",
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Docker Network (Cont.)

Add containers to a network

- To build web applications that act in concert but do so securely, create a network.
- Networks, by definition, provide complete isolation for containers.
- Can add containers to a network when you first run a container.
- Launch a container running a PostgreSQL database and pass it the `--net=my_bridge` flag to connect it to your new network:

```
$ docker run -d --net=my_bridge --name db training/postgres
```

- If you inspect your `my_bridge` you can see it has a container attached. You can also inspect your container to see where it is connected:

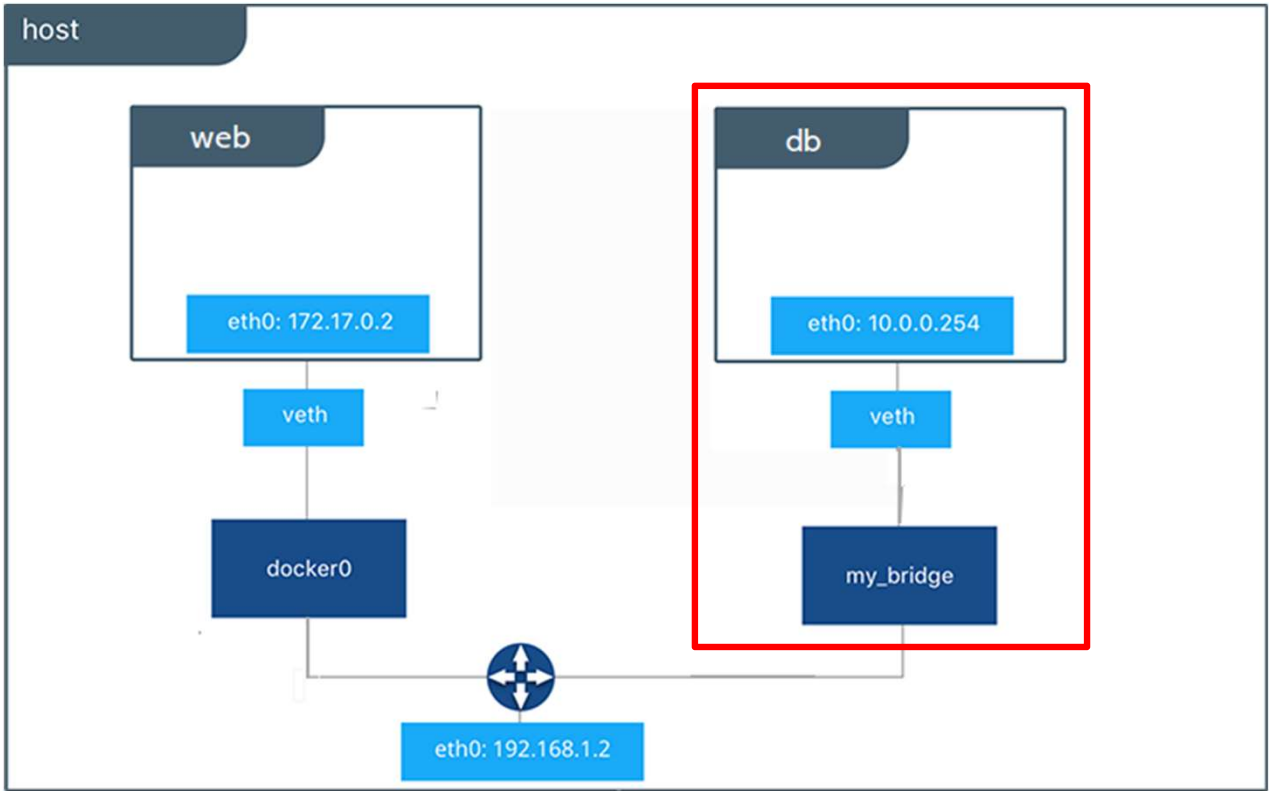
```
$ docker inspect --format='{{json .NetworkSettings.Networks}}'  db


{"my_bridge":{"NetworkID":"7d86d31b1478e7cca9ebed7e73aa0fdeec46c5ca29497431d3007d2d9e15ed99",
"EndpointID":"508b170d56b2ac9e4ef86694b0a76a22dd3df1983404f7321da5649645bf7043","Gateway":"10.0.0.1","IPAddress":"10.0.0.254","IPPr
```

Docker Network (Cont.)

Add containers to a network

- Now, go ahead and start your by now familiar **web** application.

```
$ docker run -d -name web training/webapp python app.py
```



Docker Network (Cont.)

Add containers to a network

- Which network is your **web** application running under?
- Inspect the application to verify that it is running in the default bridge network.

```
$ docker inspect --format='{{json .NetworkSettings.Networks}}'  web


{"bridge":{"NetworkID":"7ea29fc1412292a2d7bba362f9253545fecdfa8ce9a6e37dd10ba8bee7129812",
"EndpointID":"508b170d56b2ac9e4ef86694b0a76a22dd3df1983404f7321da5649645bf7043","Gateway":"172.17.0.1","IPAddress":"10.0.0.2","IPPr
```

- Then, get the IP address of your **web**.

```
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' web


172.17.0.2
```

Docker Network (Cont.)

Add containers to a network

- Now, open a shell to your running `db` container:

```
$ docker container exec -it db bash

root@a205f0dd33b2:/# ping 172.17.0.2
ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
^C
--- 172.17.0.2 ping statistics ---
44 packets transmitted, 0 received, 100% packet loss, time 43185ms
```

Docker Network (Cont.)

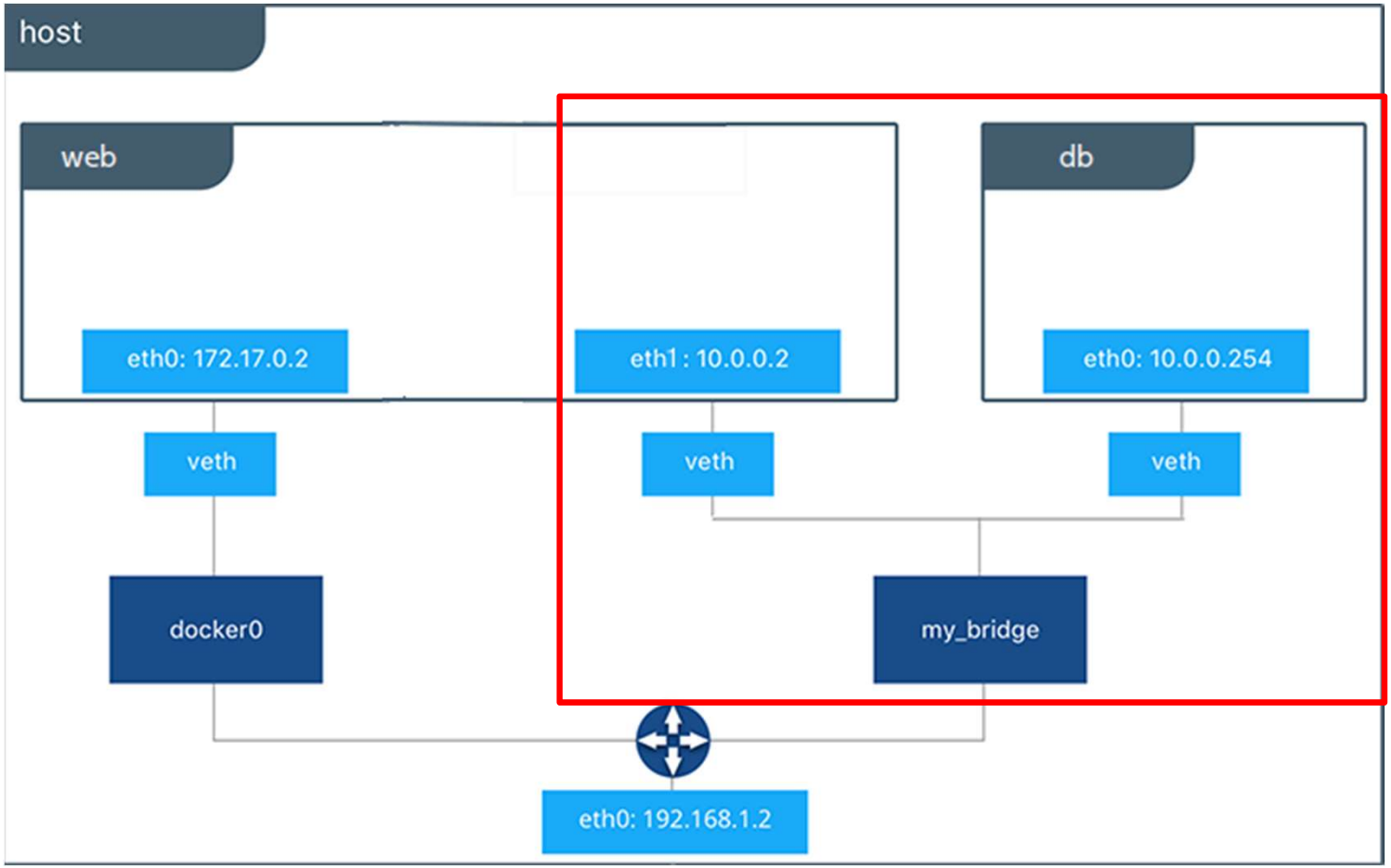
Add containers to a network

- Docker networking allows you to attach a container to as many networks as you like.
- You can also attach an already running container.
- Go ahead and attach your running web app to the `my_bridge`.

```
$ docker network connect my_bride web
```

Docker Network (Cont.)

Add containers to a network



Docker Network (Cont.)

Add containers to a network

- Open a shell into the **db** application and try the **ping** command.
- This time just use the container name web rather than the IP address.

```
$ docker container exec -it db bash

root@a205f0dd33b2:/# ping web
PING web (10.0.0.2) 56(84) bytes of data.
64 bytes from web (10.0.0.2): icmp_seq=1 ttl=64 time=0.095 ms
64 bytes from web (10.0.0.2): icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from web (10.0.0.2): icmp_seq=3 ttl=64 time=0.066 ms
^C
--- web ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.060/0.073/0.095/0.018 ms
```

- The **ping** shows it is contacting a different IP address, the address on the **my_bridge** which is different from its address on the bridge network.

Docker Network (Cont.)

사용자 정의형 Network Driver 생성 例

```
$ docker network create --driver bridge \
   --subnet 192.168.100.0/24  --gateway 192.168.100.254  mynet
$ docker network ls

$ docker run -d --name web -p 80:80 nginx:1.14
$ curl localhost

$ docker run -d --name hellojs --net mynet  --ip 192.168.100.100 \
   -p 8080:8080 hellojs
$ curl localhost:8080
```

Docker Network (Cont.)

Container 間 통신

```
$ docker network create app-network
$ docker network connect app-network mysql
$ docker run -dp 8080:80 \
  > --network=app-network \
  > -e WORDPRESS_DB_HOST=mysql \
  > -e WORDPRESS_DB_NAME=wp \
  > -e WORDPRESS_DB_USER=wp \
  > -e WORDPRESS_DB_PASSWORD=wp \
  > wordpress
```

Docker Network (Cont.)

Container 間 통신

```
$ docker run -d --name mysql -v /dbdata:/var/lib/mysql \
  > -e MYSQL_ROOT_PASSWORD=wordpress \
  > -e MYSQL_PASSWORD=wordpress  mysql:5.7

$ docker run -d --name wordpress --link mysql:mysql \
  > -e WORDPRESS_DB_PASSWORD=wordpress -p 80:80 wordpress:4
```

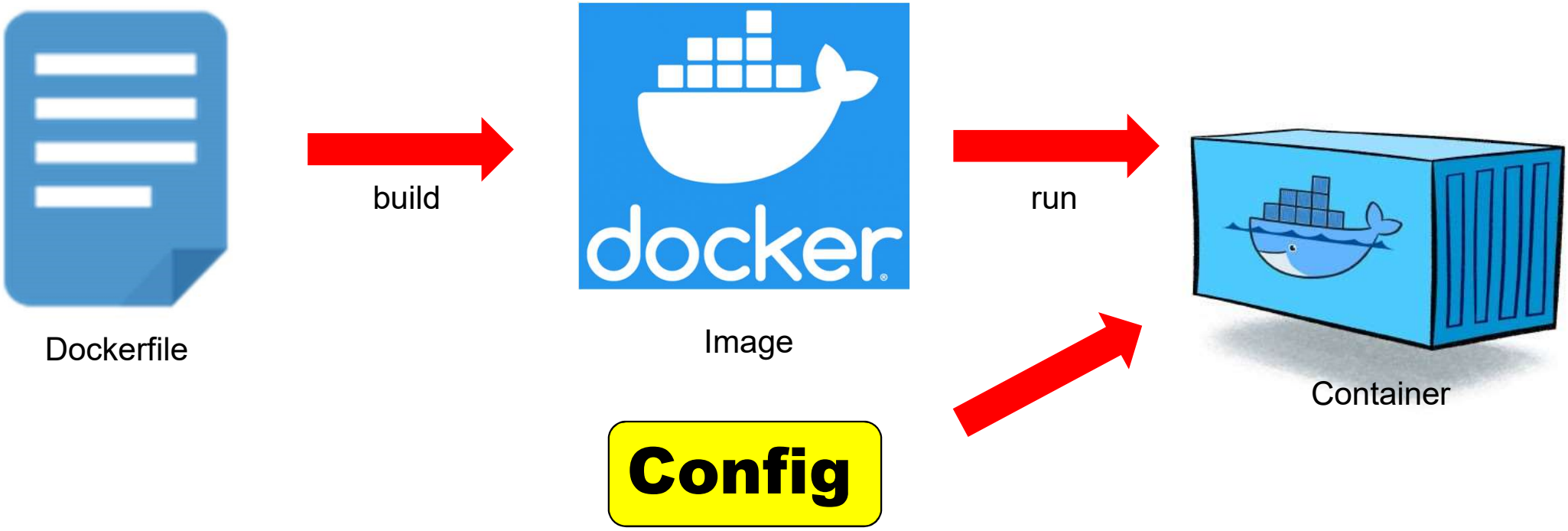
Lab9. Container Network

Docker Compose

Docker Compose

What's Docker Compose ?

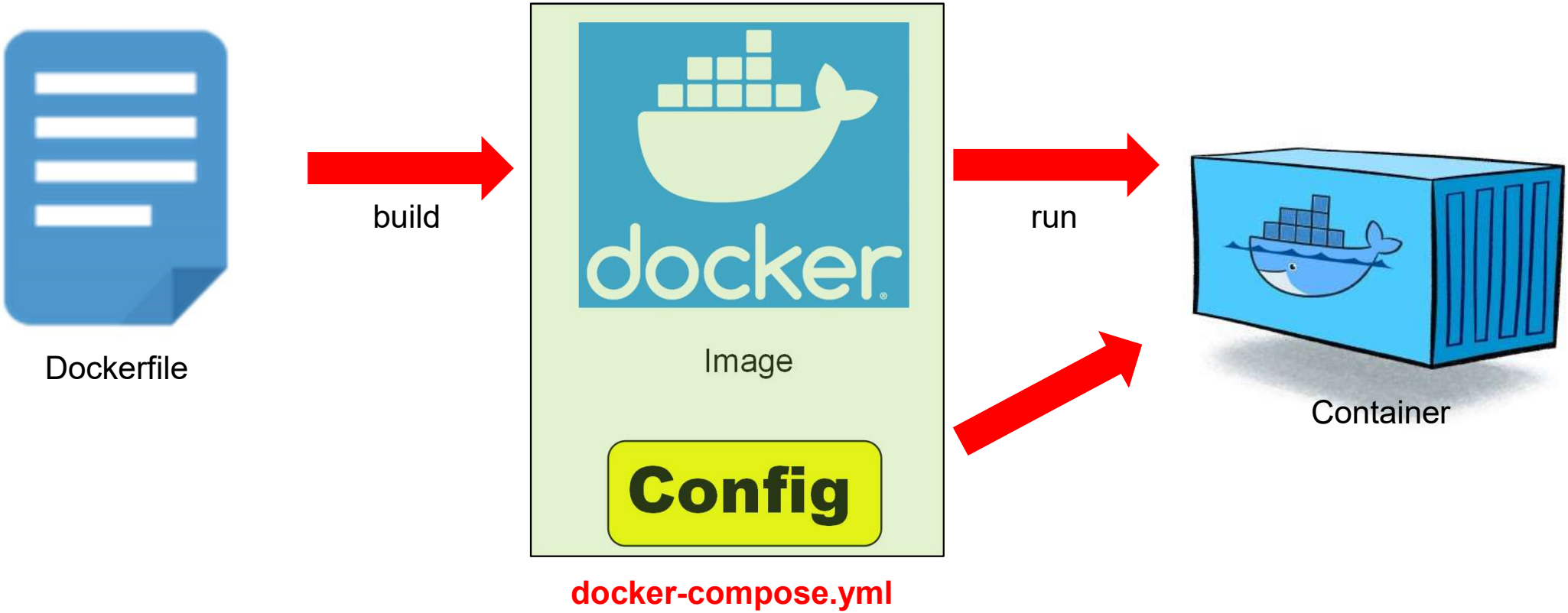
- From Docker Image to Container Process



Docker Compose (Cont.)

What's Docker Compose ?

- From Docker Image to Container Process



Docker Compose (Cont.)

What's Docker Compose ?

- Is a tool for defining and running multi-container Docker applications.
- With Compose, use a *YAML* file to configure your application's services.
- Then, with a single command, create and start all the services from configuration.
- Reference
  - https://docs.docker.com/compose/compose-file/
  - https://docs.docker.com/samples/wordpress/
  - https://docs.docker.com/compose/

Docker Compose (Cont.)

What's Docker Compose ?

- Using Compose is basically a three-step process:
  1. Define your app's environment with a ***Dockerfile*** so it can be reproduced anywhere.
  2. Define the services that make up your app in ***docker-compose.yml*** so they can be run together in an isolated environment.
  3. Run **`docker compose up`** and the Docker compose command starts and runs your entire app. You can alternatively run **`docker-compose up`** using the docker-compose binary.

Docker Compose (Cont.)

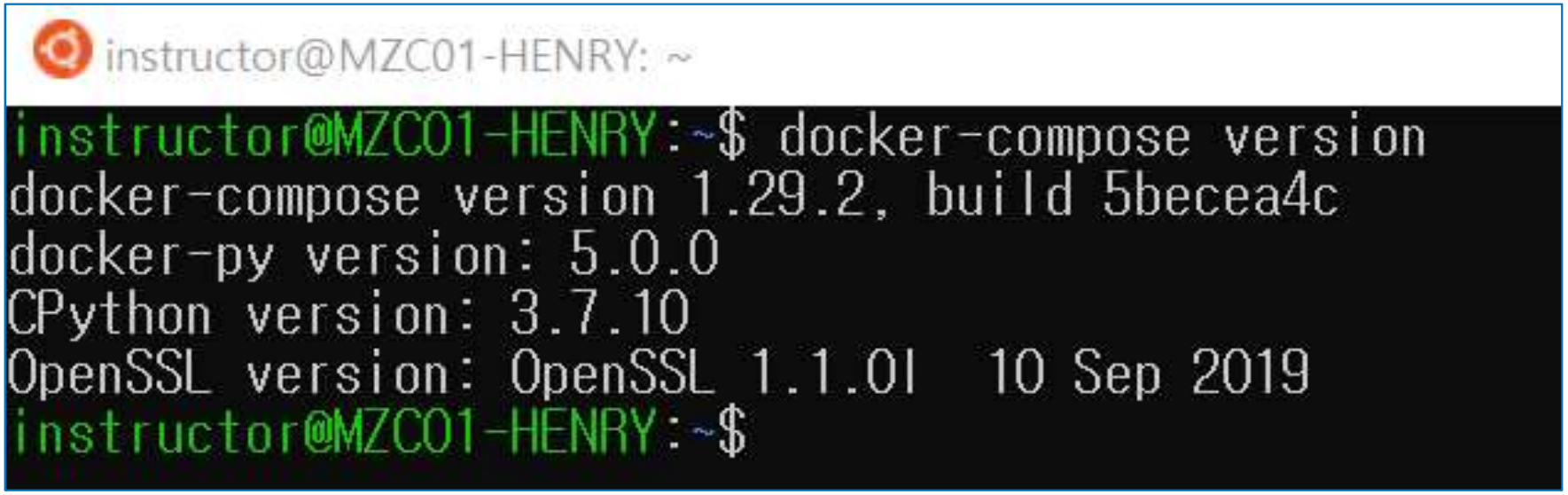
docker-compose.yml

```yaml
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Docker Compose

Docker Compose 설치하기

- Docker for Windows or Docker for Mac 설치시 기본 설치됨.



Docker Compose

Docker Compose 설치하기

- Docker Compose Installation on Linux
  1. Run this command to download the current stable release of Docker Compose:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

  2. Apply executable permissions to the binary:

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

```
instructor@docker-ubuntu:~$ docker-compose version
docker-compose version 1.29.2, build 5becea4c
docker-py version: 5.0.0
CPython version: 3.7.10
OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019
```

Lab10. Docker Compose Installation on Ubuntu

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

1. Docker 실행 명령어를 일일이 입력하기가 복잡해서
   - Nginx 실행하기

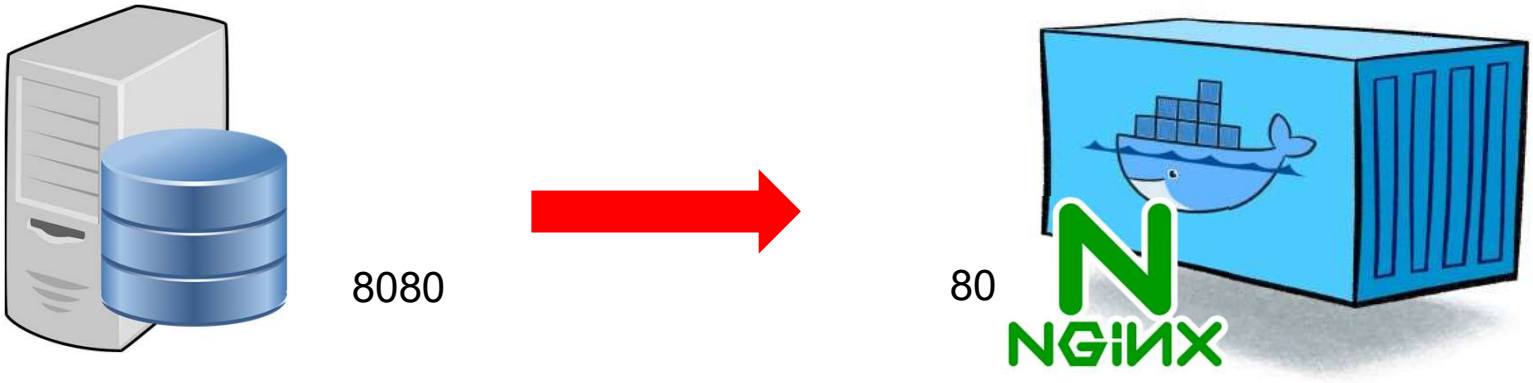
```
$ docker run -it nginx
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

1. Docker 실행 명령어를 일일이 입력하기가 복잡해서
   - Nginx Container 실행 + Host 8080 Port 연결하기

```
$ docker run -it -p 8080:80 nginx
```

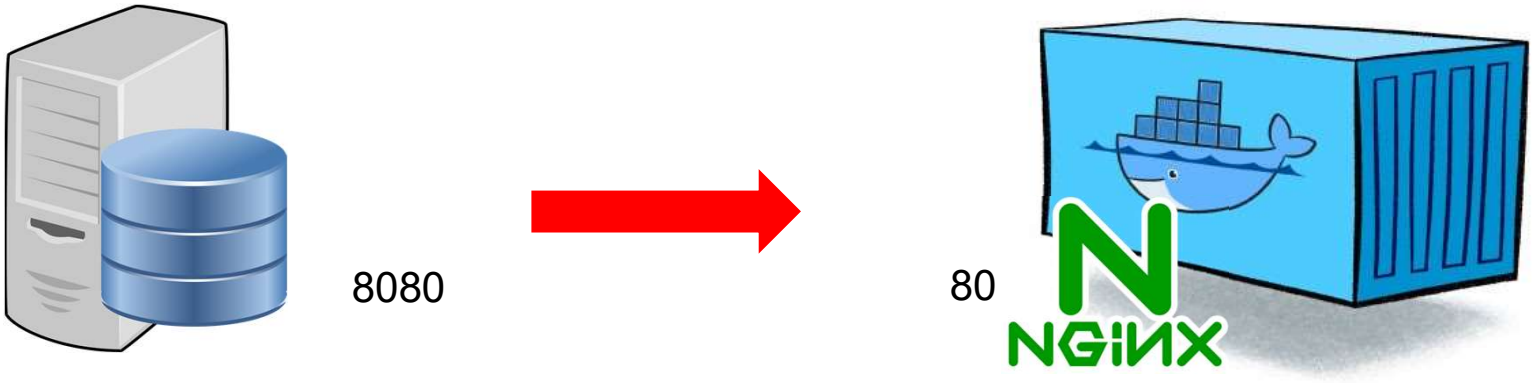


Docker Compose (Cont.)

Docker Compose를 사용하는 이유

1. Docker 실행 명령어를 일일이 입력하기가 복잡해서
   - Nginx Container 실행 + Host 8080 Port 연결 + Container 종료시 자동 삭제

```
$ docker run -it -p 8080:80 --rm nginx
```

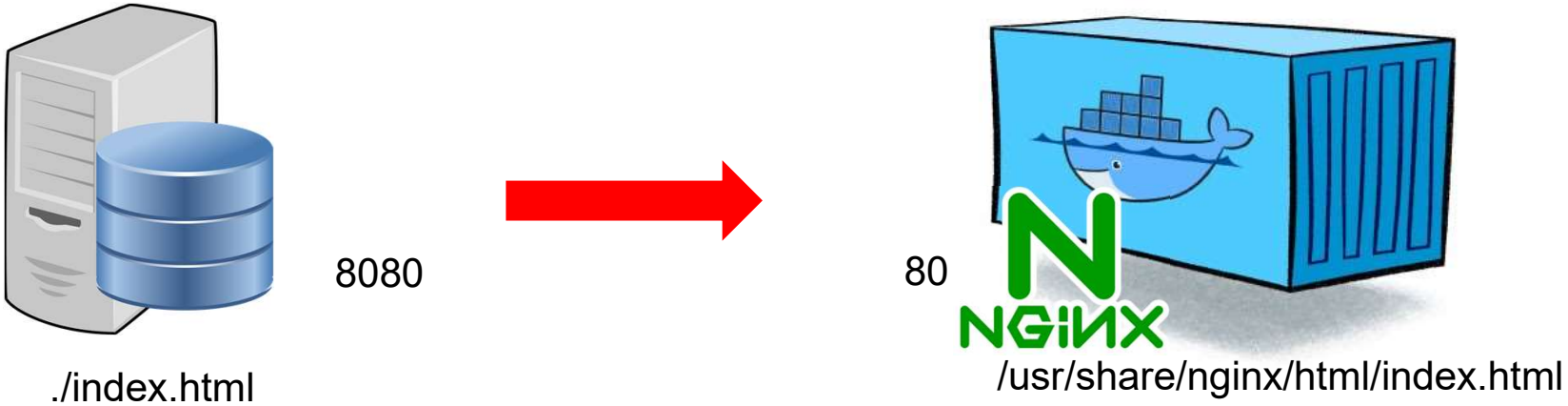


Docker Compose (Cont.)

Docker Compose를 사용하는 이유

1. Docker 실행 명령어를 일일이 입력하기가 복잡해서
   - Nginx 컨테이너 실행 + Host 8080 Port 연결 + Container 종료시 자동 삭제 + Host의 Directory를 Container 안에서 링크하기

```
$ vi index.html
$ docker run -it -p 8080:80 --rm -v ${PWD}:/usr/share/nginx/html/
nginx
```



Docker Compose (Cont.)

Docker Compose를 사용하는 이유

2. Container間 연결하기가 불편해서
   - django-sample Image build

```
$ git clone https://github.com/raccoonyy/django-sample-for-docker-compose.git django-sample
$ cd django-sample
$ docker build -t django-sample .
```

   - django Container 실행 + Postgres Container 실행 + 서로 연결하기

```
$ docker run --rm -d --name django -p 8000:8000 django-sample
$ docker run --rm -d --name postgres -e POSTGRES_DB=djangosample \
  -e POSTGRES_USER=sampleuser -e POSTGRES_PASSWORD=samplesecret \
  postgres
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

2. Container間 연결하기가 불편해서
   - postgres Container 실행 + django Container 실행 + 서로 연결하기

```
$ docker run --rm -d --name postgres -e POSTGRES_DB=djangosample \
 -e POSTGRES_USER=sampleuser \
 -e POSTGRES_PASSWORD=samplesecret \
 postgres

$ docker run -d --rm -p 8000:8000 -e DJANGO_DB_HOST=db \
 --link postgres:db \
 django-sample
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

2. Container間 연결하기가 불편해서
   - postgres Container 실행 + django Container 실행 + 서로 연결하기

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - postgres Container 실행 + django1 Container 연결

```
$ docker run --rm -d --name postgres \
 -e POSTGRES_DB=djangosample -e POSTGRES_USER=sampleuser \
 -e POSTGRES_PASSWORD=samplesecret postgres

$ docker run -d --rm --name django1 -p 8000:8000 \
-e DJANGO_DB_HOST=db \
--link postgres:db django-sample
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

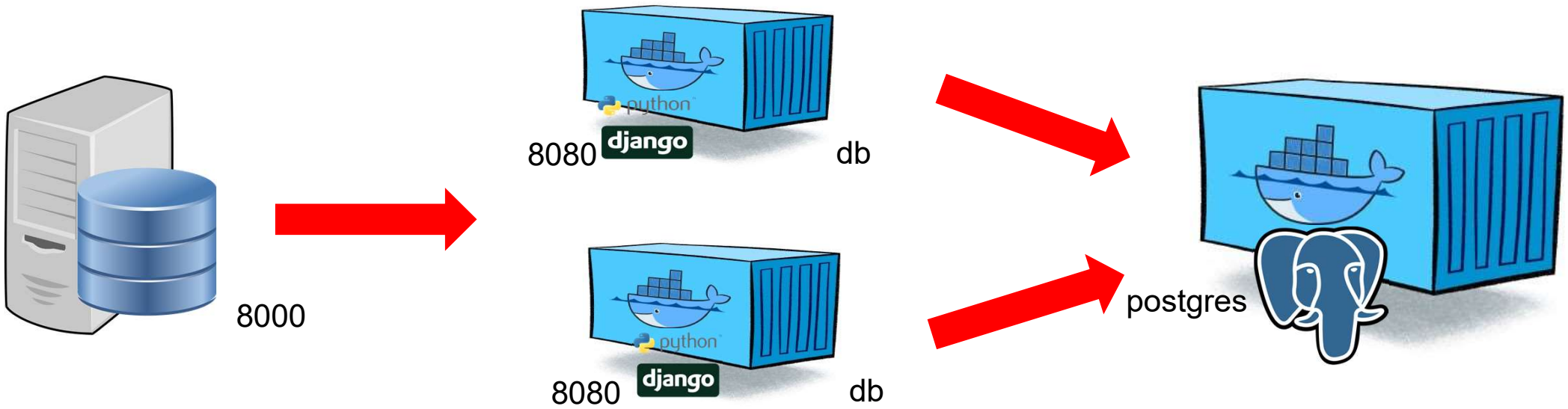
3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - postgres Container는 Host의 다른 Container들이 모두 접근할 수 있음

```
$ docker run -d --rm --name django2 \
-p 8001:8000 \
-e DJANGO_DB_HOST=db \
--link postgres:db \
django-sample
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - postgres Container 실행 + django Container 실행 + 서로 연결하기

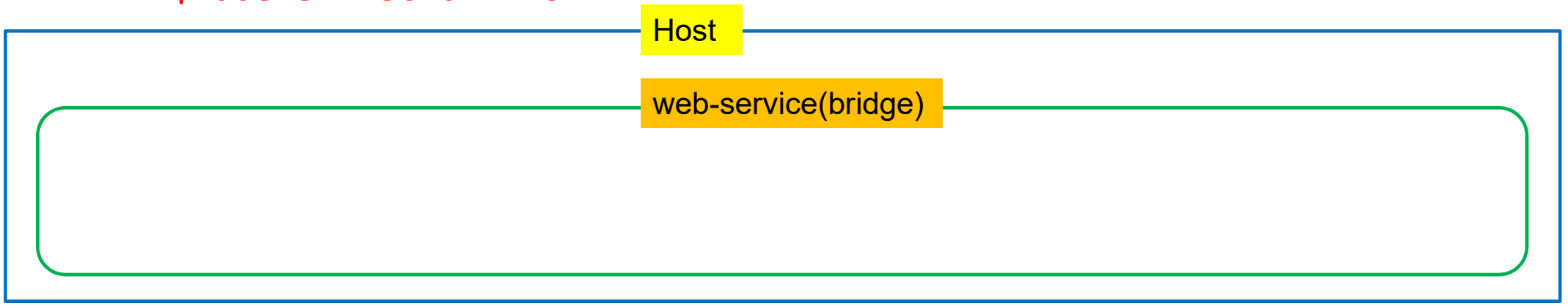


Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - postgres Container + django1 Container만 통신할 수 있는 가상 네트워크 생성

```
$ docker network ls

$ docker network create --driver bridge web-service
$ docker network ls
```



Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - Docker Network Modes
     - bridge : 해당 Network 안에서만 통신 가능
     - host : Host와 똑같은 Network 환경
     - none : 어떤 Network도 사용하지 않음.

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

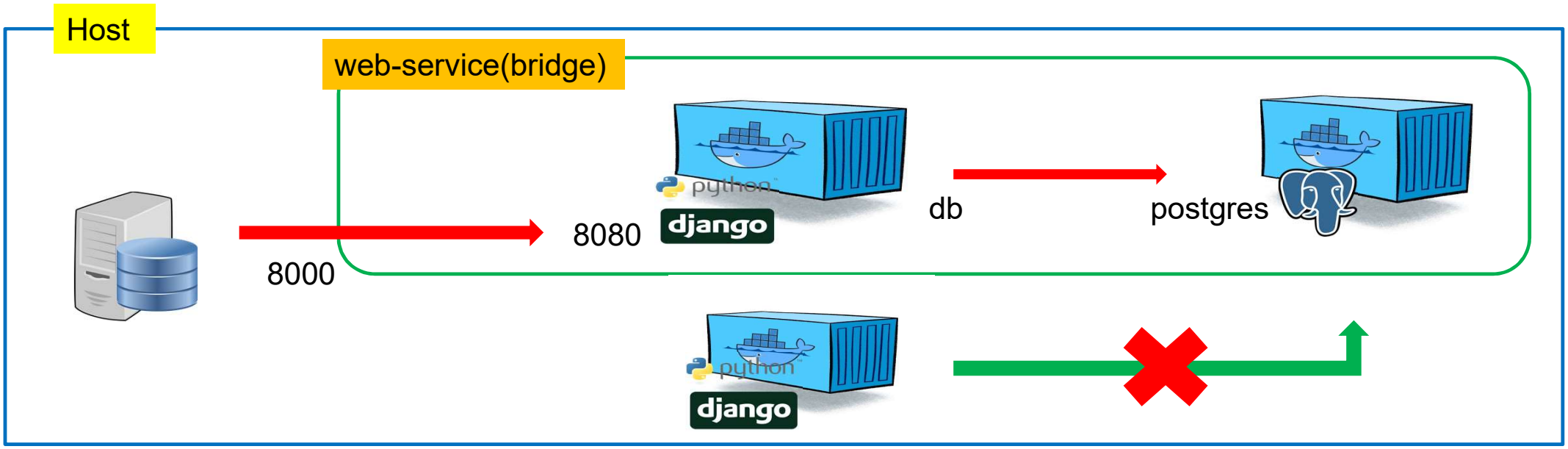
3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - 생성한 가상 Network로 Container 실행하기

```
$ docker run --rm -d --name postgres --network web-service \
  -e POSTGRES_DB=djangosample -e POSTGRES_USER=sampleuser \
  -e POSTGRES_PASSWORD=samplesecret postgres

$ docker run -d --rm --name django1 --network web-service \
  -p 8000:8000 -e DJANGO_DB_HOST=db --link postgres:db django-sample

$ docker run -d --rm --name django2 \
  -p 8001:8000 -e DJANGO_DB_HOST=db --link postgres:db django-sample
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - 생성한 가상 Network로 Container 실행하기



Docker Compose (Cont.)

Docker Compose를 사용하는 이유

3. 특정 Container 끼리만 통신할 수 있는 가상 네트워크 환경을 편리하게 관리하고 싶어서
   - 생성한 가상 Network로 Container 실행하기

```
docker: Error response from daemon: Cannot link to /postgres, as it does not bel
ong to the default network.
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

4. 이 모든 것을 간단한 명령어로 관리하고 싶어서
   - 실행 명령어와 종료 명령어

```
$ docker network create --driver bridge web-service

$ docker run --rm -d --name postgres --network web-service \
  -p 5432:5432 -e POSTGRES_DB=djangosample \
  -e POSTGRES_USER=sampleuser \
  -e POSTGRES_PASSWORD=samplesecret postgres

$ docker run -d --rm --name django1 --network web-service \
  -p 8000:8000 -e DJANGO_DB_HOST=db --link postgres:db django-sample

$ docker kill django1 postgres
$ docker network rm web-service
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

4. 이 모든 것을 간단한 명령어

- docker-compose.yml

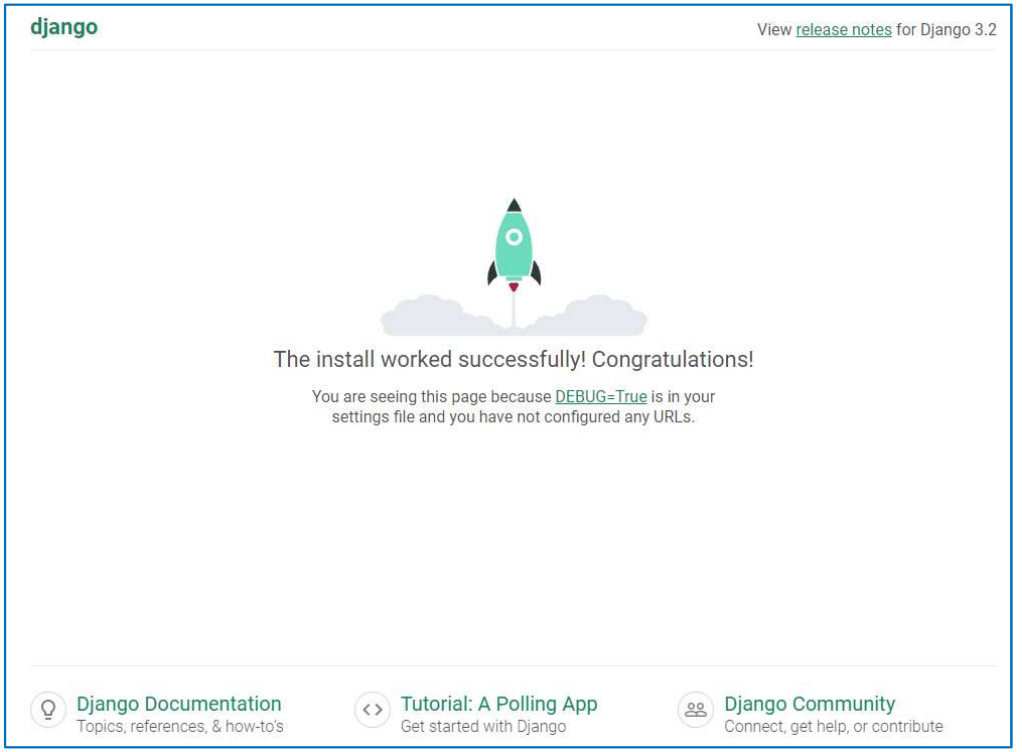
```yaml
version: '3'

volumes:
  postgres_data: {}

services:
  db:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgres/data
    environment:
      - POSTGRES_DB=djangosample
      - POSTGRES_USER=sampleuser
      - POSTGRES_PASSWORD=samplesecret
  django:
    build:
      context: .
      dockerfile: ./compose/django/Dockerfile-dev
    volumes:
      - ./:/app/
    command: ["./manage.py", "runserver", "0:8000"]
    environment:
      - DJANGO_DB_HOST=db
    depends_on:
      - db
    restart: always
    ports:
      - 8000:8000
```

Docker Compose (Cont.)

Docker Compose를 사용하는 이유

4. 이 모든 것을 간단한 명령어로 관리하고 싶어서
   - Docker Compose로 실행하고 종료하기

```
$ docker-compose up -d
$ docker-compose down
```



Lab11. Docker Compose를 사용하는 이유

Docker Compose

Docker Compose 실습하기

- Nginx Server를 Docker Compose로 실행하기
  1. docker run으로 nginx 실행하기
     - Image : nginx:latest
     - Listening Port : 80
     - Host Connection Port : 8080
     - index.html : Hello Docker Compose
     - HTML Directory : /usr/share/nginx/html

Docker Compose (Cont.)

Docker Compose 실습하기

- Nginx Server를 Docker Compose로 실행하기
  2. docker-compose.yml 작성하기

Docker Compose (Cont.)

Docker Compose 실습하기

- Nginx Server를 Docker Compose로 실행하기
  2. docker-compose.yml 작성하기

```
version: '3'

services:
  nginx:
    image: nginx
    ports:
      - 8080:80
    volumes:
      - ./:/usr/share/nginx/html/
```

Docker Compose (Cont.)

Docker Compose 실습하기

- Nginx Server를 Docker Compose로 실행하기
  3. 실행하기

```
$ docker-compose up -d
```

Lab12. Docker Compose 실습하기1

Docker Compose (Cont.)

Docker Compose Syntax

- File 확장자 : **.yml**
- 키와 값은 **:** 기호로 구분
- Block 안에서는 **두 칸** 들여쓰기
- 목록은 **–** 기호 사용
- 주석은 **#** 으로 표시

```
version: '3'

services:
  nginx:
    image: nginx
    ports:
      - 8080:80
    volumes:
      - ./:/usr/share/nginx/html/
```

Docker Compose (Cont.)

Docker Compose Syntax

- YAML File Sample
  - List

```
# A list of fruits
- Apple
- Orange
- Strawberry
- Mango
```

- YAML File Sample
  - Dictionary

```
# An employee record
martin :
    name :  Martin Henry
    job :  Developer
    level :  Senior
```

- YAML File Sample
  - Complicated Data

```
# Employees record
- Martin :
      name :  Martin Henry
      job :  Developer
      skills :
        - python
        - Java
        - C#
- Kaly :
      name :  Kaly Smith
      job :  Designer
```

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- version
  - `version : '3'`
  - docker-compose.yml 파일의 명세서 버전
  - docker-compose.yml 버전에 따라 지원하는 Docker Engine Version이 다름.

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- services
  - `services:`
    `postgres:`
    …
    `django:`
    …
  - 실행할 Container 정의
  - `docker run --name django`와 의미 같음.

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- image
  - `services:`
    `django:`
    `image: django-sample`
  - Container에 사용할 Image 이름과 Tag
  - Tag를 생략하면 latest
  - Image가 없으면 자동으로 pull

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- build
  - **services:**
    nginx:
    build: .
  - Container 빌드
  - dockerfile을 이용해서 build할 때 사용했던 경로와 같은 의미

```
$ docker build -t sample:1 .
```

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- context
  - services:
    django:
    build:
    context: .
    dockerfile: ./compose/django/Dockerfile-dev
  - Image를 자체 Build 후 사용
  - image 속성 대신 사용
  - 여기에 사용할 별도의 Docker 파일이 필요

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- command / entrypoint
  - services:
    app:
      image: node:12-alpine
      command: sh -c "yarn install && yarn run dev"
  - Container에서 실행될 명령어 지정
  - Dockerfile의 command/entrypoint와 거의 같음.

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- ports
  - services:
    django:
      …
      ports:
        - "8080:80"
  - Container와 연결할 Port(s)
  - {Host Port} : {Container Port}

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- environment
  - services:
    mysql:
      …
      environment:
        - MYSQL_ROOT_PASSWORD=password
  - Container에서 사용할 환경변수(들)
  - {환경변수 이름} : {값}

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- volumes
  - services:
    django:
    …
    volumes:
    - ./app:/app
  - Mount 하려는 Director[y,(ies)]
  - {Host Directory} : {Container Directory}

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- link
  - services:
    mysql:
      …
    django:
      …
      link:
        - mysql:db
  - 다른 Container와 연결
  - {연결할 Container 이름} : {해당 Container에서 참조할 이름}

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- expose
  - **services:**
    nginx:
    build: .
    expose:
    - "8080"
  - Host에는 Port를 공개하지 않고, Container에만 Port 공개한다.
  - Host와 직접 연결되지 않고 link등으로 연결된 Container-Container간의 통신만이 필요한 경우 등에 사용

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- depends_on
  - services:
    mysql:
      …
    django:
      …
      depend_on:
        - mysql
  - Container 의존성 추가

Docker Compose (Cont.)

docker-compose.yml 파일 작성법

- restart
  - **services:**
    db:
    image: mysql:5.7
    restart: always
  - Container가 종료될 때 적용할 restart 정책
    - no : 재시작 안됨
    - always : Container를 수동으로 끄지 전까지 항상 재시작
    - on-failure : 오류가 있을 시 재시작

Docker Compose (Cont.)

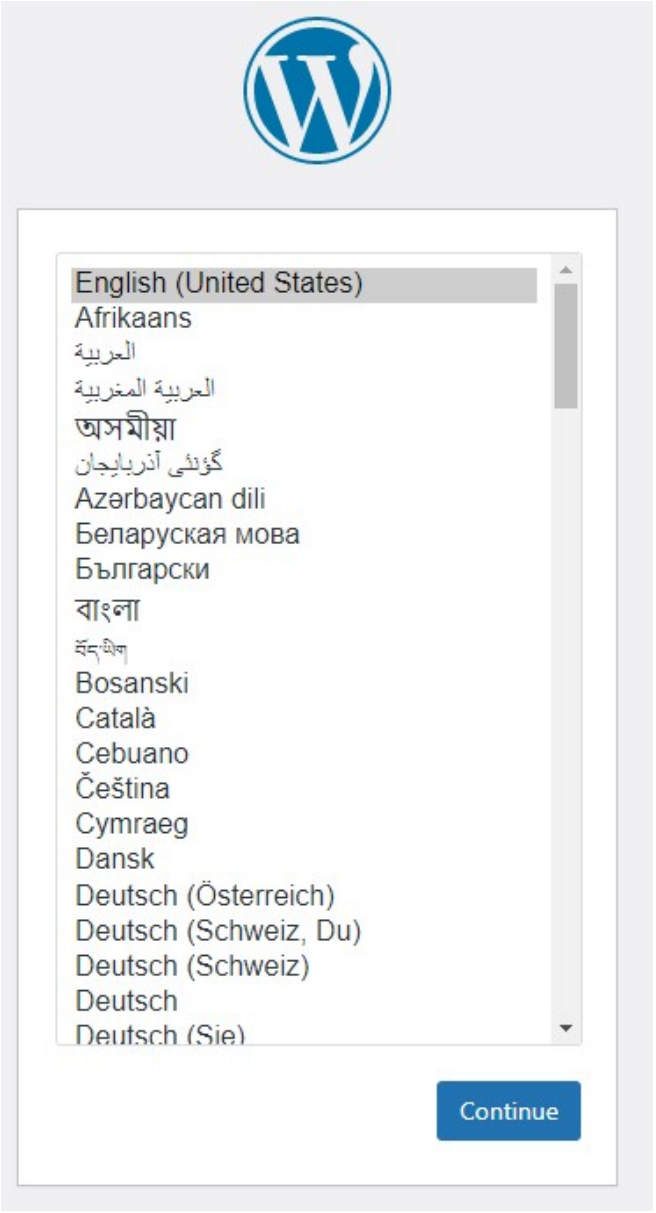
Docker Compose 실습하기

- WordPress를 Docker Compose로 실행하기
  1. WordPress Container
     - Image : wordpress
     - Listening Port : 80
     - WORDPRESS_DB_HOST : 3306
     - WORDPRESS_DB_USER : wp
     - WORDPRESS_DB_PASSWORD : wp
     - WORDPRESS_DB_NAME : wp

Docker Compose (Cont.)

Docker Compose 실습하기

- WordPress를 Docker Compose로 실행하기
  2. MySQL Container
     - Image : mysql:5.7
     - Listening Port : 3306
     - MYSQL_ROOT_PASSWORD : password
     - MYSQL_DATABASE : wp
     - MYSQL_USER : wp
     - MYSQL_PASSWORD : wp
     - Data Directory : /var/lib/mysql



Lab13. Docker Compose 실습하기2

Docker Compose (Cont.)

docker-compose 명령어(https://docs.docker.com/compose/reference/)

```
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [--profile <name>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name
                              (default: directory name)
  --profile NAME              Specify a profile to enable
  --verbose                   Show more output
  --log-level LEVEL           Set log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the
                              name specified in the client certificate
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
  --compatibility             If set, Compose will attempt to convert deploy
                              keys in v3 files to their non-Swarm equivalent
```

```
Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

Docker Compose (Cont.)

docker-compose 명령어

- **`docker-compose pull [service]`**
  - 필요한 Image 다운로드
- **`docker-compose build [service]`**
  - 필요한 Image 빌드

  $ docker-compose build

  $ docker-compose build wordpress
- **`docker-compose up [service]`**
  - 서비스 구동
  - **`--build`** : 강제로 Image 재 빌드
  - **`--force-recreate`** : Container를 새로 생성
  - **`--d`** : Daemon Mode로 실행, docker run의 –d와 동일

Docker Compose (Cont.)

docker-compose 명령어

- **`docker-compose ps`**
  - 현재 실행중인 Service 목록을 보여줌
- **`docker-compose logs [service]`**
  - 로그 보기
  - -f : 로그 계속 보기

  $ docker-compose logs

  $ docker-compose logs -f
- **`docker-compose top`**
  - 서비스 내에서 실행중인 Process 목록 보여줌

Docker Compose (Cont.)

docker-compose 명령어

- **`docker-compose run {service} {command}`**
  - 해당 Service에 Container를 하나 더 실행
  - -e : 환경변수 설정
  - -p : 연결할 port 설정
  - --rm : Container 종료시 자동으로 삭제
- **`docker-compose exec {container}{command}`**
  - 해당 Service의 Container에서 명령어를 실행
  - -e : 환경변수 설정

  $ docker-compose exec wordpress bash

Docker Compose (Cont.)

docker-compose 명령어

- **`docker-compose down [service]`**
  - Service를 멈추고 Container 삭제
  - Stop + kill
  - -v : Docker Volume도 함께 삭제

  $ docker-compose down
- **`docker-compose restart [service]`**
  - Container 재시작

  $ docker-compose restart

  $ docker-compose restart wordpress

Docker Compose (Cont.)

docker-compose 명령어

- **`docker-compose stop [service]`**
  - 서비스 중지

  $ docker-compose stop

  $ docker-compose stop wordpress
- **`docker-compose start [service]`**
  - Stop 중인 Service의 Container 실행

  $ docker-compose start

  $ docker-compose start wordpress

Lab14. docker-compose Commands

Docker Compose (Cont.)

Docker Compose 실습하기

- Flask App을 Docker Compose로 실행하기
  1. Flask Container
     - Connection Port : 5000
     - Redis Host Name : redis
  2. Redis Container
     - Image : redis

Docker Compose (Cont.)

Docker Compose 실습하기

- Flask App을 Docker Compose로 실행하기

3. app.py

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)

def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

Docker Compose (Cont.)

Docker Compose 실습하기

- Flask App을 Docker Compose로 실행하기
  4. requirements.txt

```
flask
redis
```

Docker Compose (Cont.)

Docker Compose 실습하기

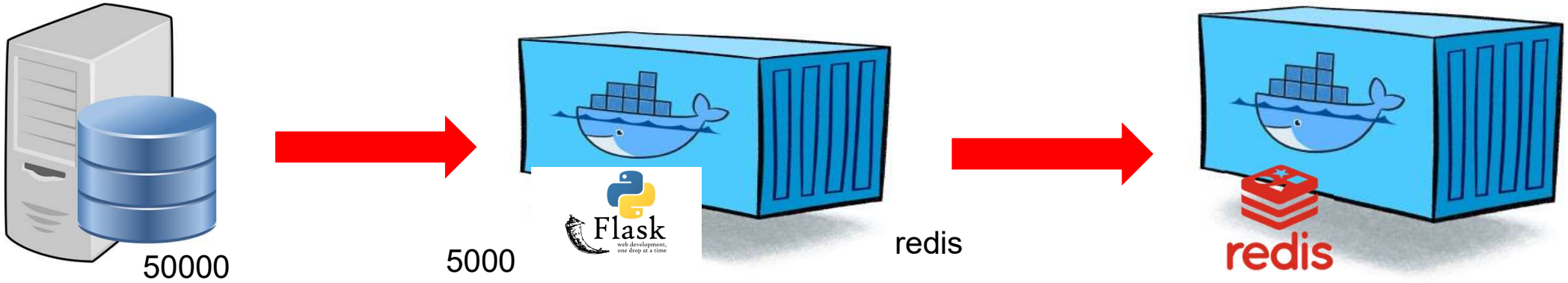
- Flask App을 Docker Compose로 실행하기
  5. Dockerfile

```
FROM        python:3.7-alpine
WORKDIR     /code
ENV         FLASK_APP    app.py
ENV         FLASK_RUN_HOST     0.0.0.0
RUN         apk  add --no-cache  gcc   musl-dev  linux-headers
COPY        requirements.txt  requirements.txt
RUN          pip install -r  requirements.txt
COPY        .    .
CMD         ["flask", "run"]
```

Docker Compose (Cont.)

Docker Compose 실습하기

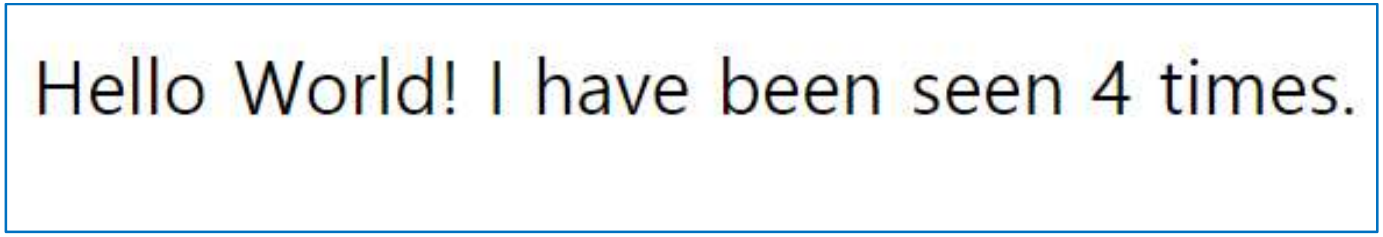
- Flask App을 Docker Compose로 실행하기
  6. Service 구성도



Docker Compose (Cont.)

Docker Compose 실습하기

- Flask App을 Docker Compose로 실행하기
  7. 확인 순서
     - flask Application을 Build하여 Image를 생성
     - 50000 Port로 접속할 수 있게 docker-compose.yml 작성
     - Docker Compose를 실행



Docker Compose (Cont.)

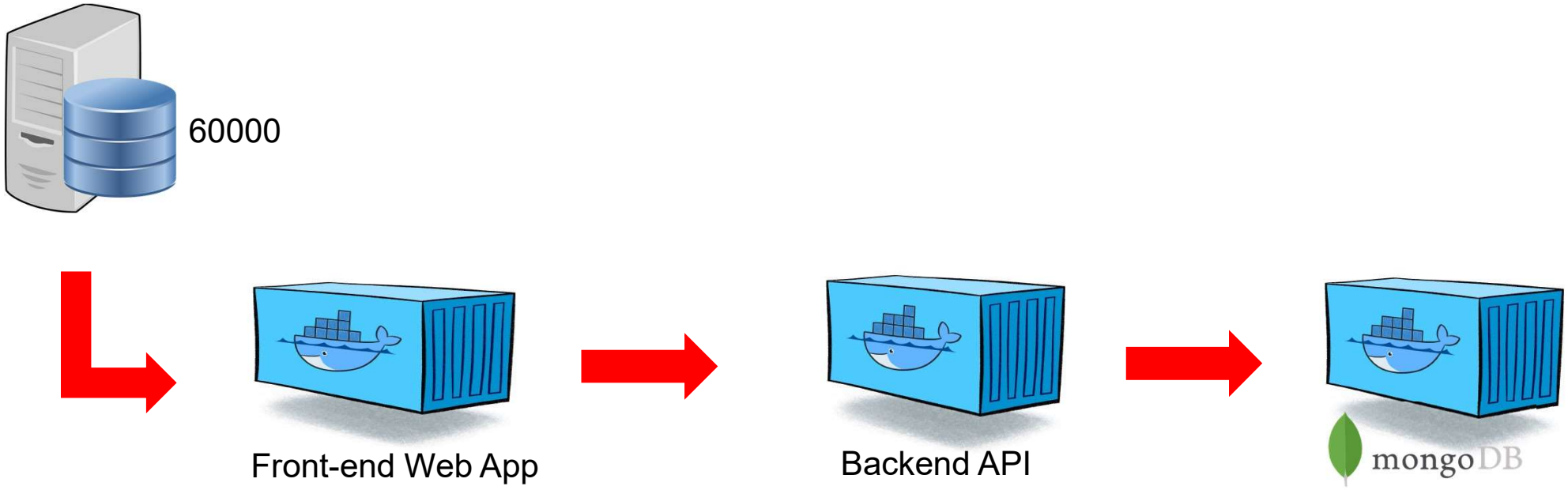
Docker Compose 실습하기

- Front-end, Back-end, Database로 구성된 방명록 서비스 실행하기
  1. Front-end
     - Image : subicura/guestbook-frontend:latest
     - Port : 60000
     - PORT 환경변수 : Service를 실행할 Port
     - GUESTBOOK_API_ADDR 환경변수 : Back-end Server 주소 ex)backend:8000
  2. Back-end
     - Image : subicura/guestbook-backend:latest
     - PORT 환경변수 : Service를 실행할 Port
     - GUESTBOOK_DB_ADDR 환경변수 : Database Server 주소 ex)mongodb:27017
  3. Database
     - Image : mongo:4
     - 연결되는 Port : 27017
     - Volume 설정 : /data/db

Docker Compose (Cont.)

Docker Compose 실습하기

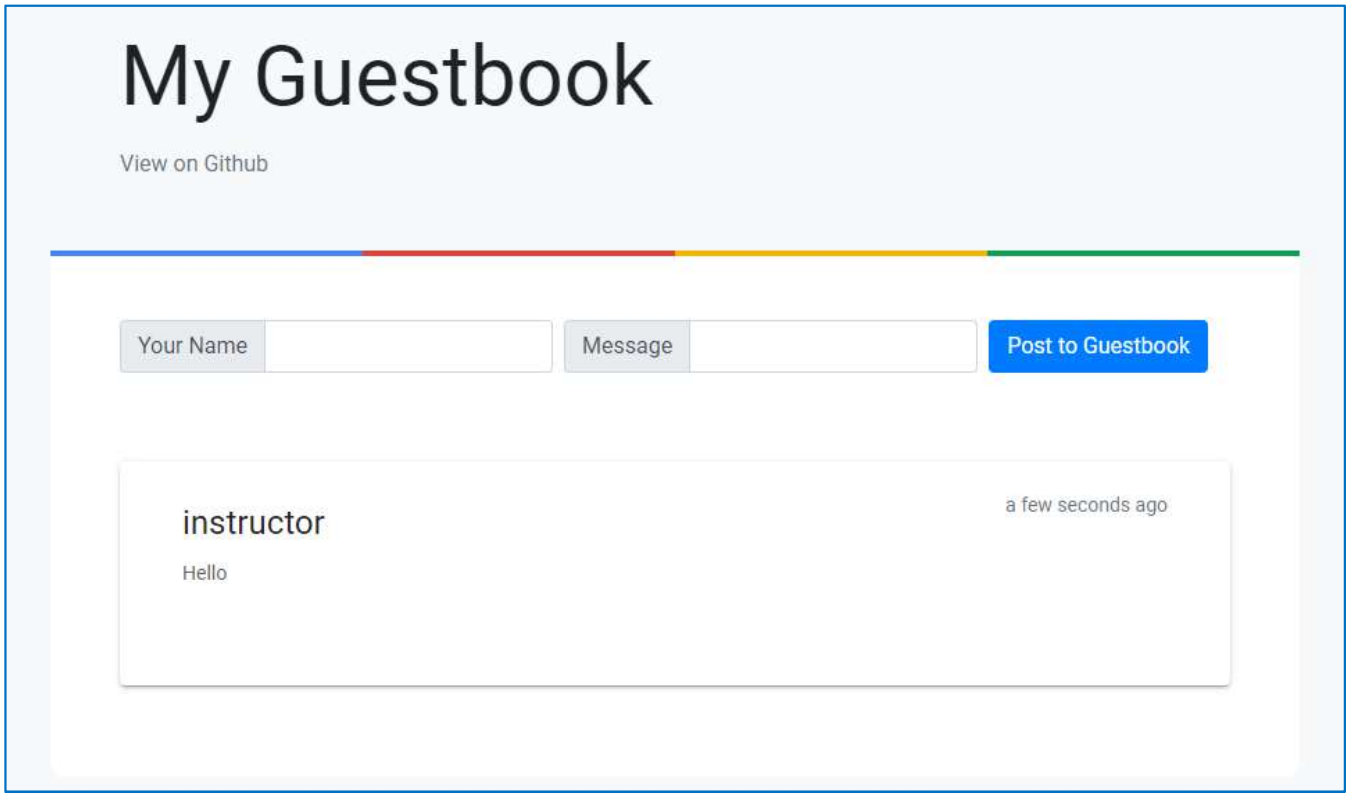
- Front-end, Back-end, Database로 구성된 방명록 서비스 실행하기
  4. Service 구성도



Docker Compose (Cont.)

Docker Compose 실습하기

- Front-end, Back-end, Database로 구성된 방명록 서비스 실행하기



Lab15. Docker Compose

Docker Extension

Docker Extension

Docker Know-How

- Image name convention
  - 소문자만 가능
- Private Registry
  - URI 포함 필요
- Runtime 시 foreground mode만 가능(Event stream 형태의 log 출력)
- 하나의 Docker container에는 하나의 Process만 운영할 것

Docker Extension (Cont.)

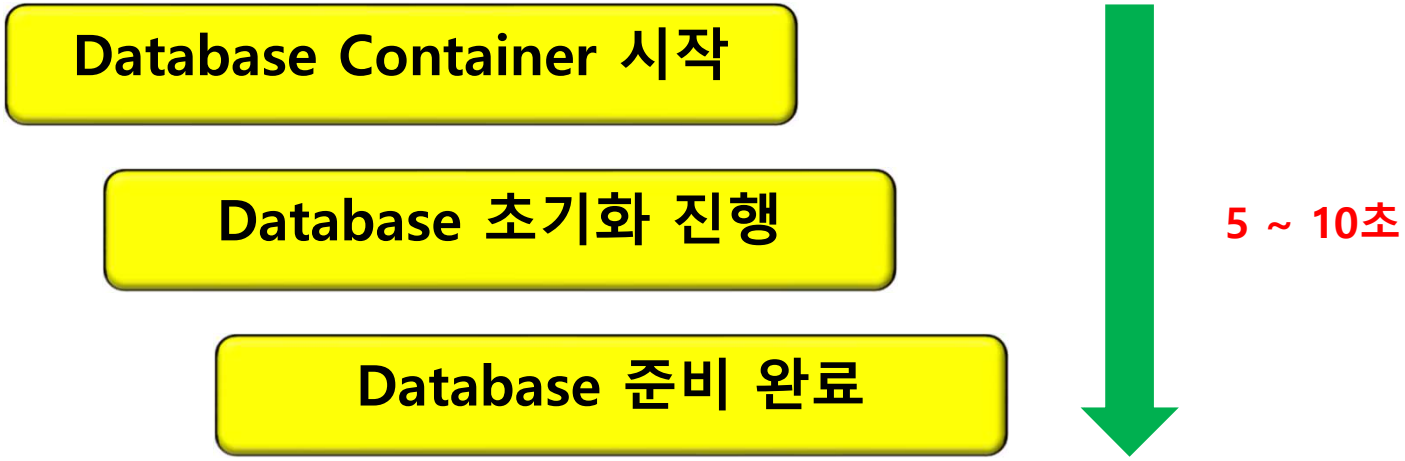
Docker Know-How

- Ubuntu host 환경 추천
  - aufs & Device-mapper 문제 고려
- Privileged mode를 통한 Host System 관리
  - Container가 상위 시스템을 관리할 필요시(보안 중요)
- 12 Factor 필독
  - https://12factor.net/ko/
  - Cloud Native 개발에 많은 도움

Docker Extension (Cont.)

Docker Compose `restart` option

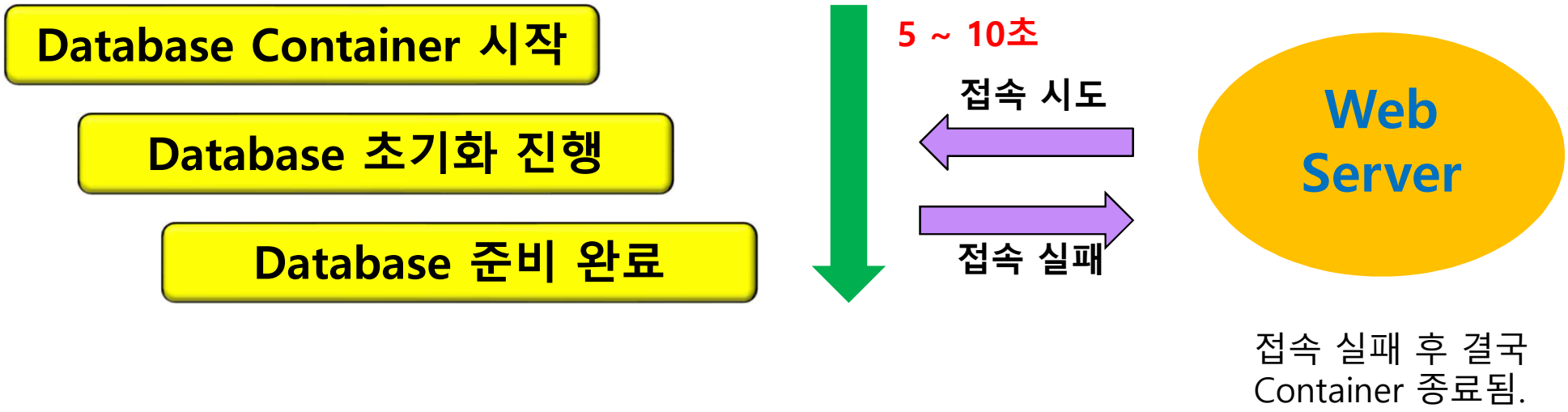
- Database 준비 과정



Docker Extension (Cont.)

Docker Compose `restart` option

- Web Server 실행 후 접속 시도



접속 실패 후 결국 Container 종료됨.

Docker Extension (Cont.)

Docker Compose `restart` option

- Container가 예기치 않게 종료 되었을 때 어떻게 할 것인지 결정

```
services:
  web:
    …
    restart: always
```

- "no" : 기본값, 다시 시작하지 않음.
- always : 항상 다시 시작
- on-failure : 오작동하면서 종료되었을 때만 다시 시작

Docker Extension (Cont.)

Docker Compose에서 환경변수 추가시 적용 순서

- 환경변수를 선언할 수 있는 위치
  - Dockerfile의 ENV
  - docker-compose.yml의 environment
  - docker-compose [run / exec] –e {key}:{value}
- 환경변수 우선순위
  1. docker / docker-compose 명령어 옵션
  2. docker-compose.yml의 environment 설정
  3. Dockerfile의 ENV

Lab16. Docker Compose 환경변수 우선순위

Lab17. DockerHub + GitHub + Jenkins