



# Docker Container



2015.8.26

(주)파이오링크

SDN 개발실 백승훈([sh.baek@piolink.com](mailto:sh.baek@piolink.com))

- **Background**

- Virtualization
  - Immutable Infrastructure
  - Container

- **Docker**

- Overview
  - Technical Analysis
  - Services

- **Docker + OpenStack**

- **Docker + NFV**

- **Open Container Initiative**

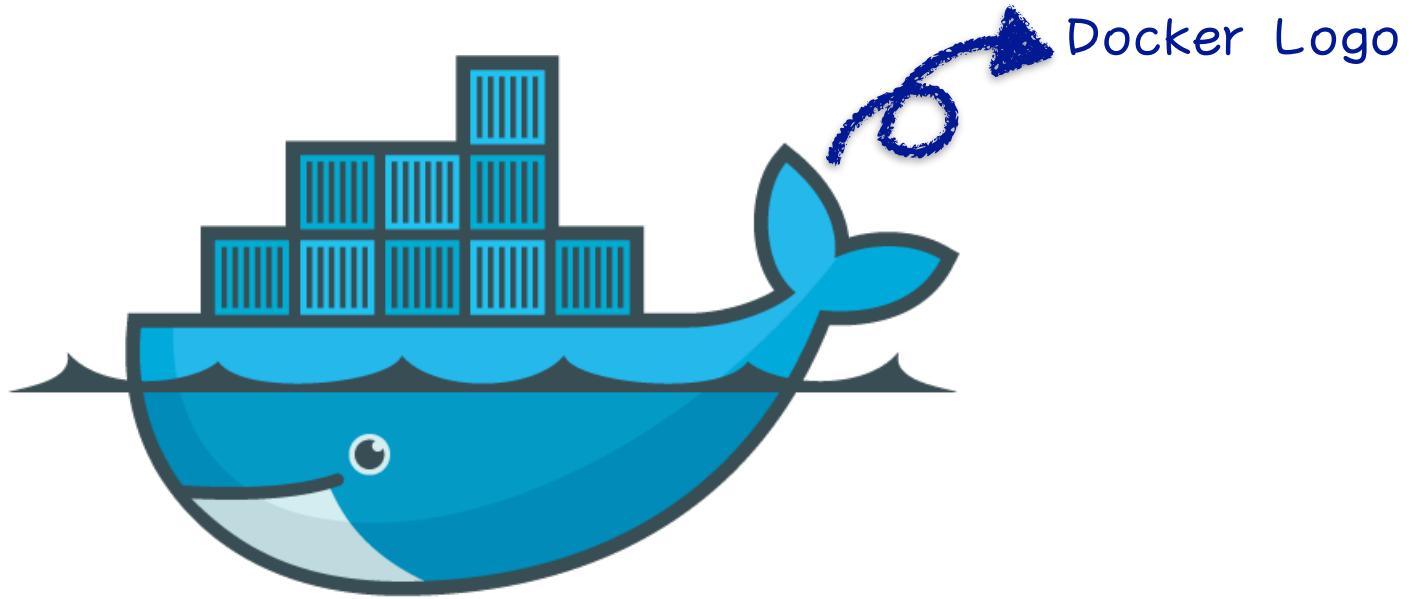
- What's Open Container Initiative?
  - The 5 Principles of Standard Container
  - Filesystem Bundle
  - Configuration File
  - Lifecycle & Runtime
  - Implementation

- **Summary**

- **References**

## : Docker?

---



**Docker**란 OS 가상화 기술인 컨테이너를  
관리, 생성, 실행, 배포하기 위한 오픈소스 엔진

# ■ Virtualization

- 가상화란 물리적인 자원을 동시에 다수의 논리적인 자원으로 사용하는 것

- 가상화 기술이 주는 이점

- 효율적인 자원 사용
- 개발 환경의 휴대화
- 설치가 빠르고 간편함

- 가상화 방법 (과거에는 Virtual Machine을 많이 사용했으나 Docker의 등장으로 Container의 관심 증가)

- Virtual Machine (eg. KVM, Xen)

- 하이퍼바이저(Hypervisor)를 통한 컴퓨팅 가상화로 여러 Guest OS를 동작시킴  
(하이퍼바이저: Virtual Machine을 생성, 실행, 관리하기 위한 소프트웨어 플랫폼)

- Container (eg. LXC, Docker)

- OS를 가상화해 여러 개의 고립된 리눅스 시스템을 동작시킴



# ■ Immutable Infrastructure

---

## ▪ VM(Virtual Machine) 사용의 문제점

- 가상 머신을 사용하면 가상 서버마다 필요한 소프트웨어 설치 필요
- 쉘 스크립트를 통한 설치 및 설정 자동화도 어렵고 안정성에 영향을 줌

\* 위의 문제를 해결하기 위해 “**Immutable Infrastructure**”라는 패러다임 등장

## ▪ Immutable Infrastructure?

- OS 커널과 서비스 환경을 분리해 커널의 수정 없이 서비스 환경의 교체가 가능함
- 서비스가 수정되면 이전 서비스 환경을 새로운 서비스 환경으로 교체
- “Immutable Infrastructure”的 장점
  - 편리한 관리: 서비스 운영환경을 이미지화해 중앙에서 관리 가능
  - 확장: 이미지 하나로 다수의 서버 생성 가능
  - 테스트: 이미지를 이용해 같은 서비스 환경 구성 가능
  - 경량화: OS와 서비스 환경을 분리해 가볍고 어디서든 실행 가능

## ▪ Docker는 “Immutable Infrastructure” 관련 프로젝트 중 하나

## ■ Google and Containers

“Everything at Google runs in a container”

- Google Cloud Platform 담당 수석 Joe Beda의 슬라이드 내용 중

## Google and Containers

Everything at Google runs in a container.

Internal usage:

- Resource isolation and predictability
- Quality of Services
  - batch vs. latency sensitive serving
- Overcommitment (not for GCE)
- Resource Accounting

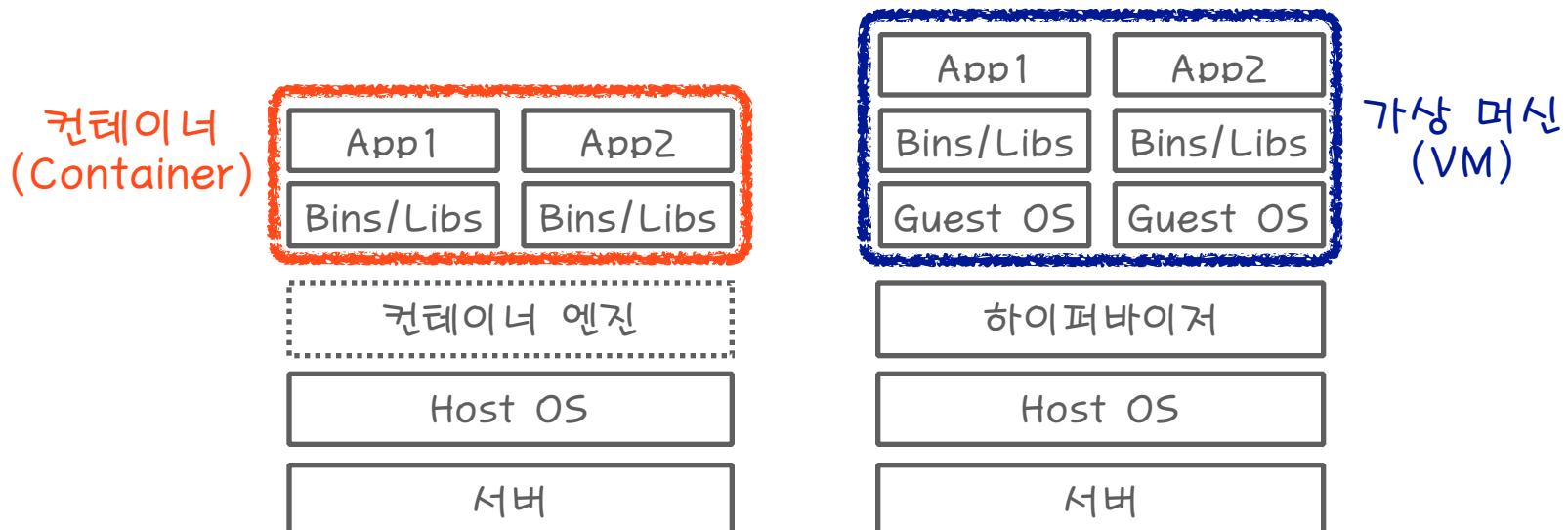
We start over 2 billion containers per week.



# ■ Container

## ▪ Container란 OS를 가상화해 여러 개의 고립된 리눅스 시스템을 실행하는 방법

- “chroot”가 기원 (chroot? 시스템을 보호하기 위해 특정 경로를 최상위 경로로 보이게 함)
- Guest OS가 필요 없이 Host OS의 자원을 공유하기 때문에 가상화의 오버헤드가 적음
  - VM은 각각의 VM마다 Guest OS를 설치해 오버헤드가 큼
  - Container는 Host OS의 커널 자원을 공유하고 유저 영역을 분리함
  - OS가 하나만 동작해 낮은 사양의 환경에서 적합
- 부팅 개념이 존재하지 않아 시작과 종료가 빠름
- 플랫폼 간 이동이 자유로움
- 불필요한 것을 제거하고 목적에 맞는 환경 구성 가능
  - eg) 웹 서버용 컨테이너는 Apache, Httpd와 같은 필요한 것만 존재



# ■ Container의 장단점

---

## ■ 장점

- 개발에서 운영까지의 과정이 단축됨
  - 휴대성이 좋아 개발, 테스트 환경을 그대로 운영환경에 적용 가능
- 빠른 시작과 종료
  - OS 입장에서는 단순한 프로세스 실행 및 종료
- 오버헤드가 낮음
  - 가상화를 위해 하이퍼바이저가 필요하지 않음
- 자원의 효율적 사용
  - 애플리케이션 동작에 필요한 자원으로만 구성

## ■ 단점

- Host OS에 종속적
  - 단일 하드웨어에 다양한 OS의 사용이 필요한 경우 VM을 사용해야 함
- 각각의 컨테이너에 독립된 커널 구성이 불가능함
  - 커널 자원을 공유하기 때문에 컨테이너가 사용하는 커널 환경은 동일

# ■ Container vs VM

## ▪ Container vs VM

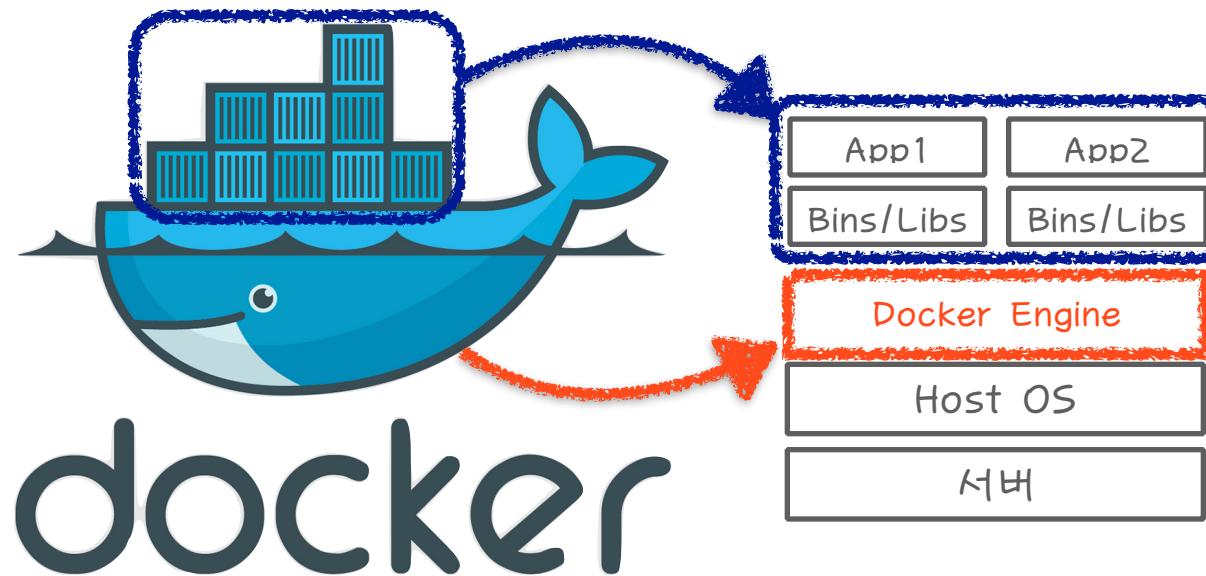


항목	컨테이너	가상 머신
하이퍼바이저	X	O
Guest OS	X	O
커널 자원 분리	X	O
시작 및 종료 시간	빠름	느림
자원 효율성	높음	낮음

## ■ Docker 란?

### ■ Docker란 컨테이너 관리, 생성, 실행, 배포를 위한 오픈소스 엔진

- 2013년 Docker, Inc(구 dotCloud)가 프로젝트를 처음 발표
- Docker의 사전적 의미는 부두 노동자로 컨테이너를 다루는 모습이 유사함
- 현재 1.7.1 버전까지 출시 (2015/7/14)
- Apache License 2.0
- Go 언어로 작성(Go 언어는 Google이 개발한 컴파일 언어)
- 개발을 위한 효율성 증가(개발 환경 구축시간 단축, 분리된 공간으로 실험환경 구성 가능)
- Linux Container, Control Group, Namespace, AUFS 등의 기술을 사용



# Docker의 장점

---

## ▪ Docker를 사용하는 이유

### - 애플리케이션의 빠른 배포

- 컨테이너 형식이 표준화됨
- 개발자와 관리자의 업무를 분리  
(개발자는 컨테이너 내부만, 관리자는 컨테이너 관리만)
- 애플리케이션에 대한 이해도를 높여줌
- 개발 테스트 및 수정이 빠름

### - 설치 및 확장이 쉽고, 휴대성이 좋음

- Docker는 어디서든 동작 가능
- 실험환경이 실제 동작하는 환경으로 이동 가능
- 컨테이너 규모 확장 및 축소가 쉽고 빠름
- 컨테이너 실행/종료가 빠름

### - 오버헤드가 낮음

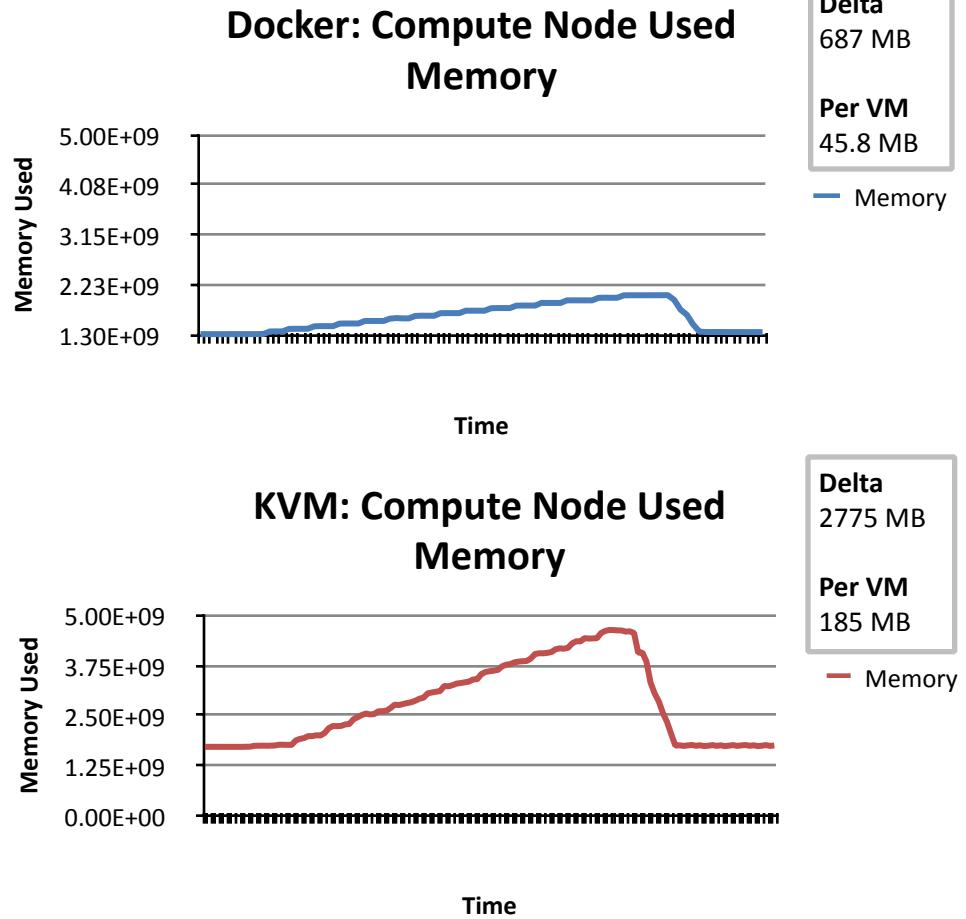
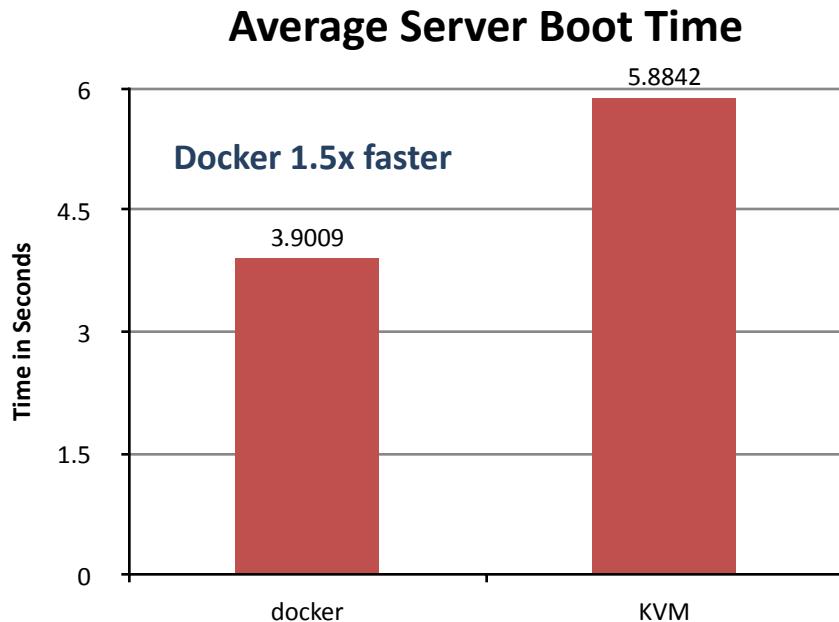
- 하이퍼바이저가 필요 없음
- 서버 자원을 더 효과적으로 사용 가능
- 장비 및 라이센스에 소비되는 비용 감소

### - 관리가 쉬움

- 수정된 부분만 소규모로 적용

## Docker vs VM(성능 비교)

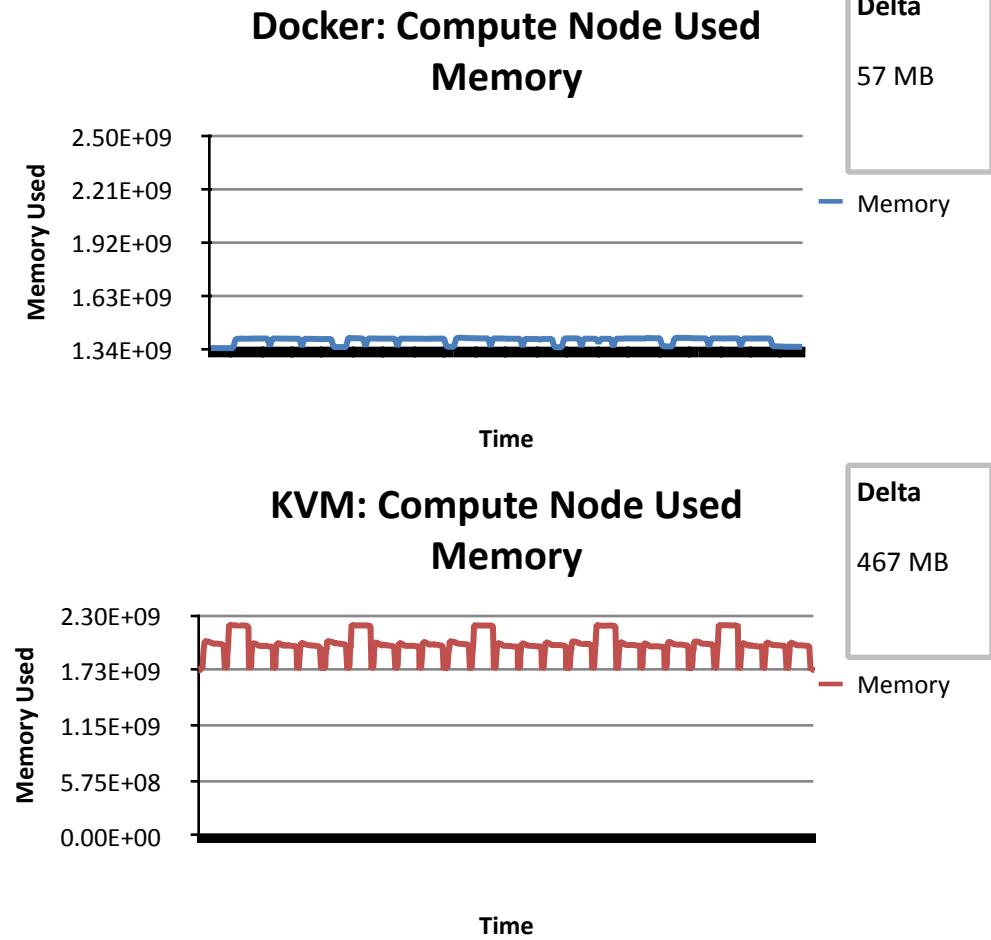
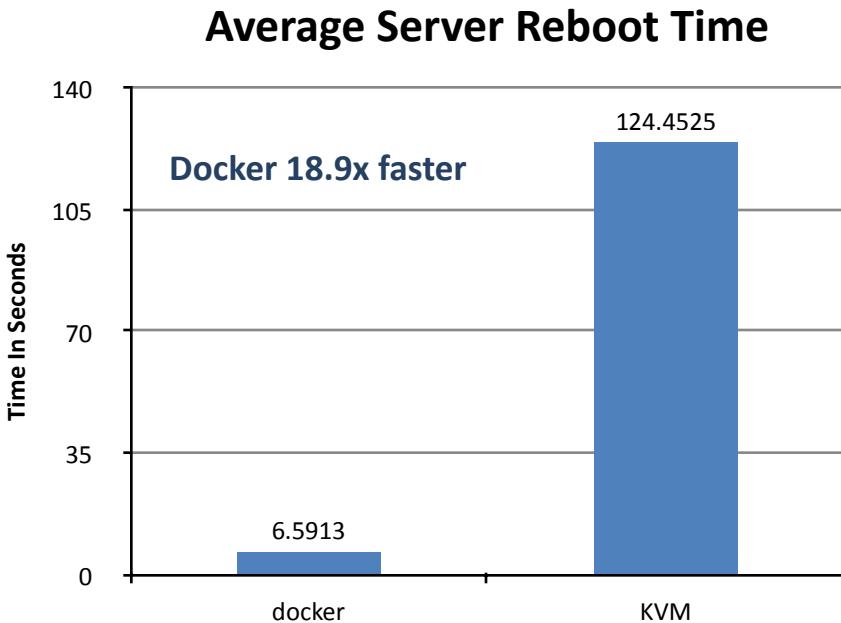
- Docker vs KVM: 부팅 시간 및 메모리 사용량 (15개의 VM 연속)



자료 출처: <http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack>

## Docker vs VM(성능 비교)

- Docker vs KVM: 재부팅 시간 및 메모리 사용량 (15개의 VM 연속)



자료 출처: <http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack>

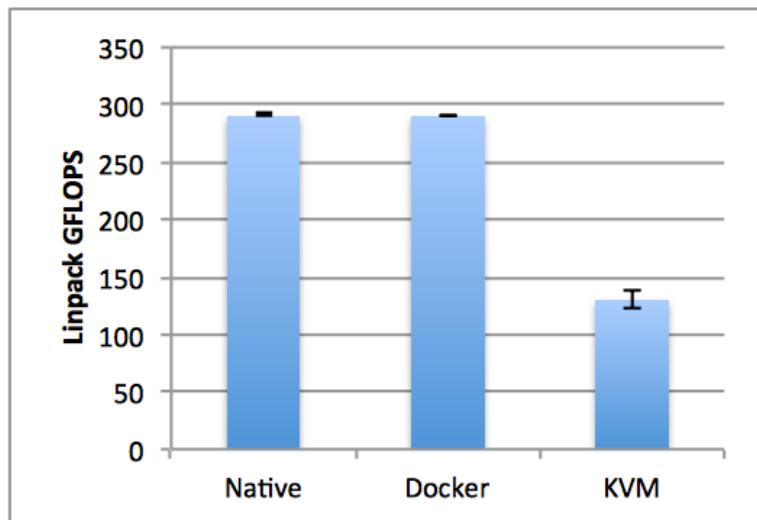
## Docker vs VM(성능 비교)

### Docker vs KVM: CPU 및 IO 성능 비교

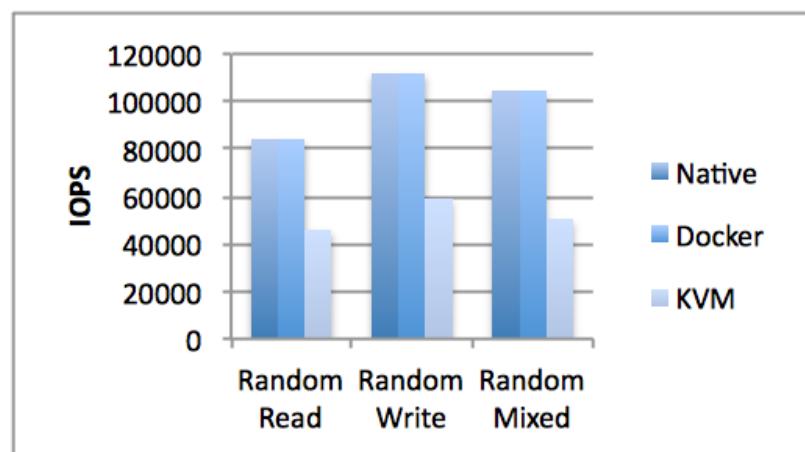
- Linpack 툴을 이용해 **CPU**의 연산 부하를 준 경우
  - Docker: Native와 비슷
  - KVM: 큰 성능 저하 발생

#### - **IOPS**(Input/Output Per Second) 측정

- Docker: Native와 비슷
- KVM: 큰 성능 저하 발생



<CPU 테스트>



<Block I/O 테스트>

자료 출처: [http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)

# Docker의 언론 평가

## ■ 2014~2015년도 계속 Docker는 주목해야 할 기술로 선정!!

- 개발 환경 구성의 단순화, 휴대성, 서비스 배치 속도 향상 등의 이유

### 올 한 해 계속 기억해야 할 오픈소스 수퍼 루키 10선

#### 2015년을 장식할 8가지 IT 이야기

제 6회 블랙 럭 오픈 소스 올해의 루키(Black Duck Open Source Project of the Year)

최고의 오픈소스 프로젝트들을 선정해 시상한다. 올해는

라이버시, 소셜 미디어, 사물 인터넷 등이다. 수상자 선정은 모든 예측이 100% 맞아떨어지는 것은 아니

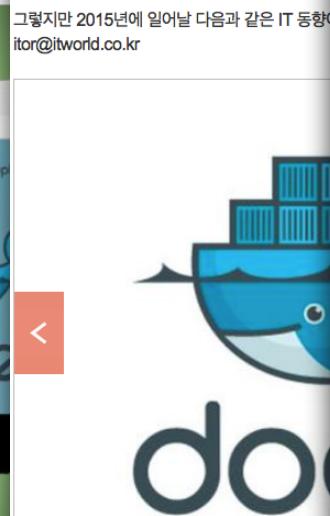
선도 오픈 소스 소프트웨어 디렉토리에 기반해 전체 커뮤니티를 찾아볼 수 있기 때문에 실제로 그 일이 일어났다.

커밋 배분, 소셜 미디어 트래픽 등을 기준으로 했다. ed 향상 조심스러울 수밖에 없다.



다커

다커(Docker, <https://www.docker.io>)는 오픈소스 프로젝트로, 컨테이너 기술을 활용해 애플리케이션을 빠르게 배포하는 데 사용되는 플랫폼이다. 컨테이너는 모든 애플리케이션의 경향, 이동형, 자동화된 컨테이너화된 형태로 제공된다. 컨테이너는 노트북에서 구축하고 테스트한 동일 컨테이너를 확장하여 클라우드, 공공 클라우드상에서 실행된다. 컨테이너는 컨테이너 기반이 튼튼해졌고 레드햇과 구글 등 대기업의 주도로 발전한 자체 앱에서 실행할 수 있고 동시에 개발자에게 인



컨테이너(그리고 도커(Docker), 모멘텀을 맞이한 오픈소스 컨테이너라는 개념이 구현될 시기까지 컨테이너는 많은 플랫폼에서 가능할 수 있는 앱을 제공하는 데 있어 중요한 것이 아니다.

#### 꼭 확인해 봐야 할 7가지 멋진 오픈소스 프로젝트

Fredric Paul | Network World

오픈소스 세계는 엄청난 속도로 계속 확대되고 있다. 이를 오픈소스 소프트웨어는 제품 및 기술 개발을 위한 중요한 "출발"점이 되고 있. 또한 그것들은 소중한 지원과 아이디어의 원천이 될 수 있는 열정적인 커뮤니티의 지원을 받고 있으며, 얻지 못하거나 수정할 수 없거나 대체할 수 없는 숨겨진 "비밀 양념" 따위는 존재하지 않는다. 본 슬라이드에서는 가장 멋진 7가지 무료 오픈소스 소프트웨어 프로젝트를 살펴보고자 한다. editor@itworld.co.kr



#### Docker, Containers, and the Future of Application Delivery

##### Docker

도커(Docker)는 소프트웨어 패키지화의 미래라는 생각이 든다. 도커는 리눅스 애플리케이션을 의존성에 따라 '컨테이너(Container)'로 묶는 과정을 자동화한다. 이런 컨테이너(특히 가벼운 형태의 가상화)는 앱을 호스트와 그 자원으로부터 고립시켜 배치 속도를 향상시키고 휴대성을 확보하며 앱과 그 호스트 환경을 더욱 견고하게 한다. 도커가 미래적인 방식이라는 증거는 아마존 웹 서비스에서 확인할 수 있으며, 해당 서비스는 얼마 전 도커를 엘라스틱 빙스토크(Elastic Beanstalk)에 통합했다고 발표한 바 있다. 라이선스 : 아파치(Apache) v2.0

자료 출처: CIO Korea

# Docker의 파트너

## ▪ Partner

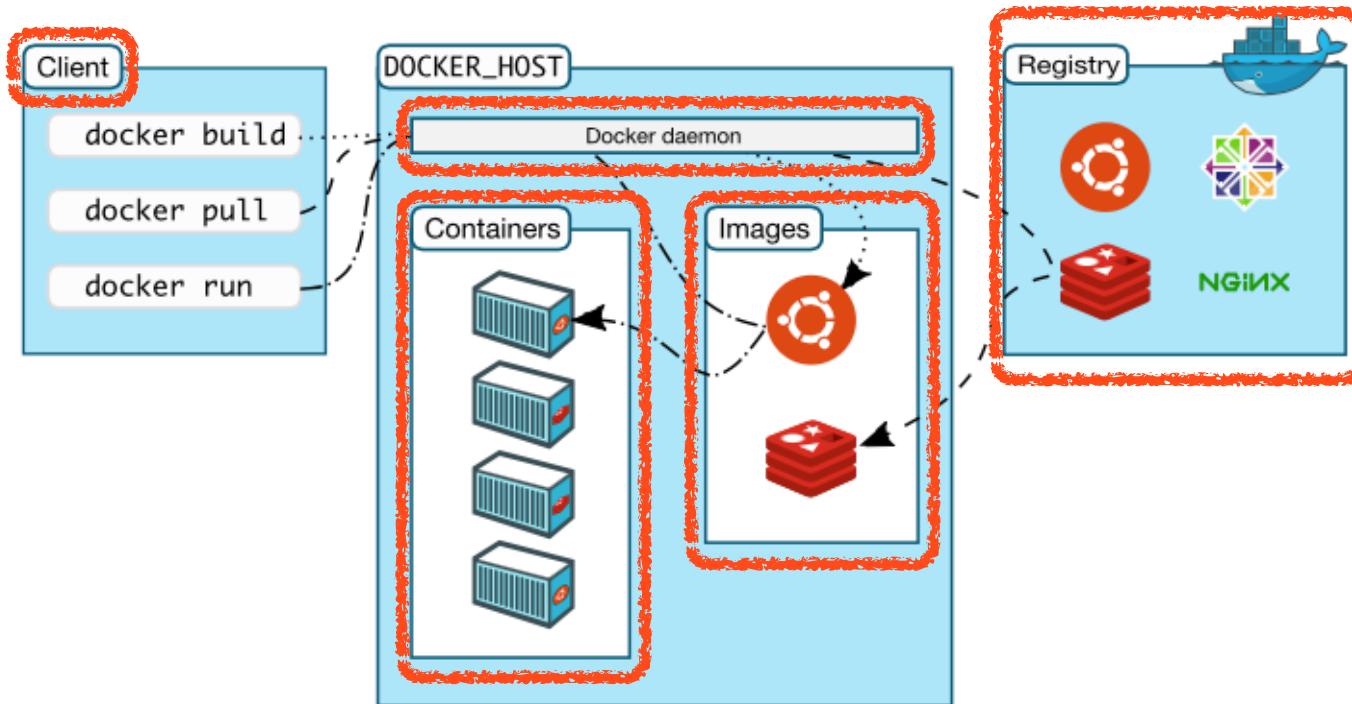
- 기술, 컨설팅, 교육, 서비스 공급업체 등 많은 기업이 프로젝트에 참여

Technology	Consulting	Service	Training
      	        	      	      

# Docker의 구성요소

## Docker Component

- **Server(Docker Daemon)** : 컨테이너를 관리하고 실행
- **Client** : 사용자와의 인터페이스 제공
- **Registry**: Image 저장소
- **Image**: 라이브러리, 실행 파일, 소스코드 등이 패키지화된 것
- **Container**: 하나 이상의 Image가 동작하는 것



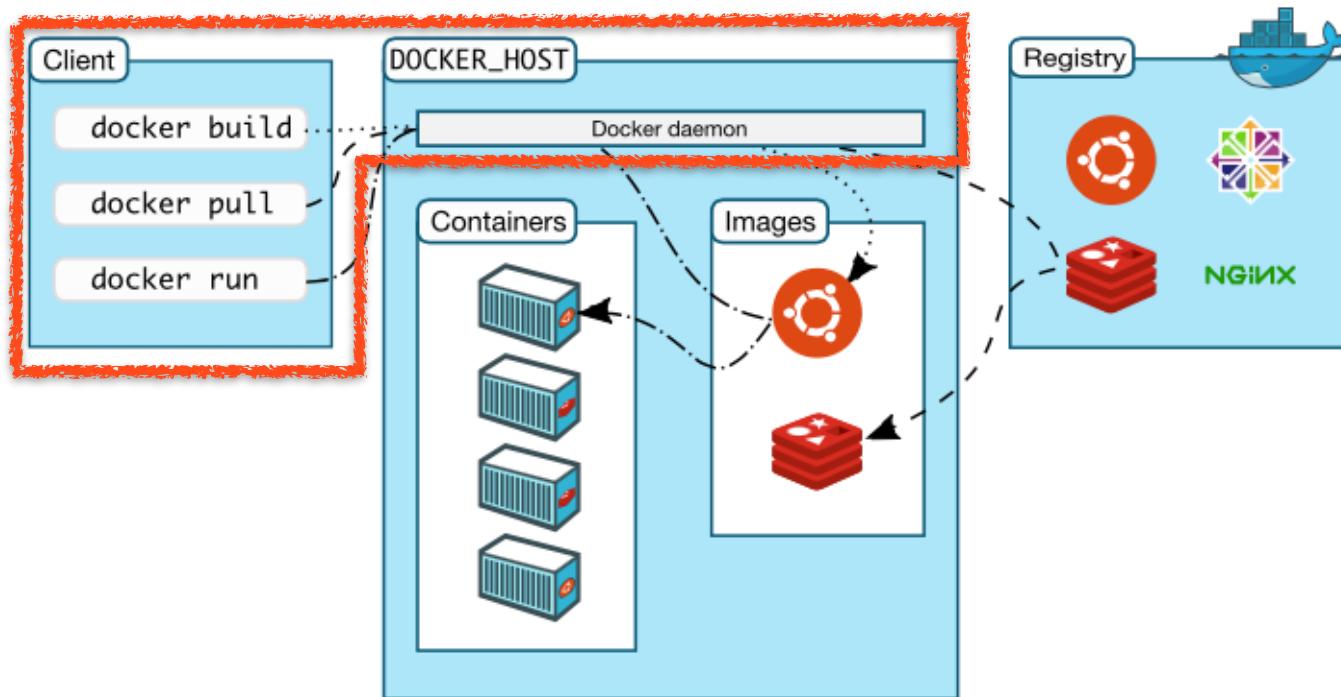
# Docker Server & Client

## ▪ Docker Server(Daemon)

- Host machine에서 컨테이너를 관리하고 실행하는 데몬
- 사용자와 Docker Client를 통해 연결됨

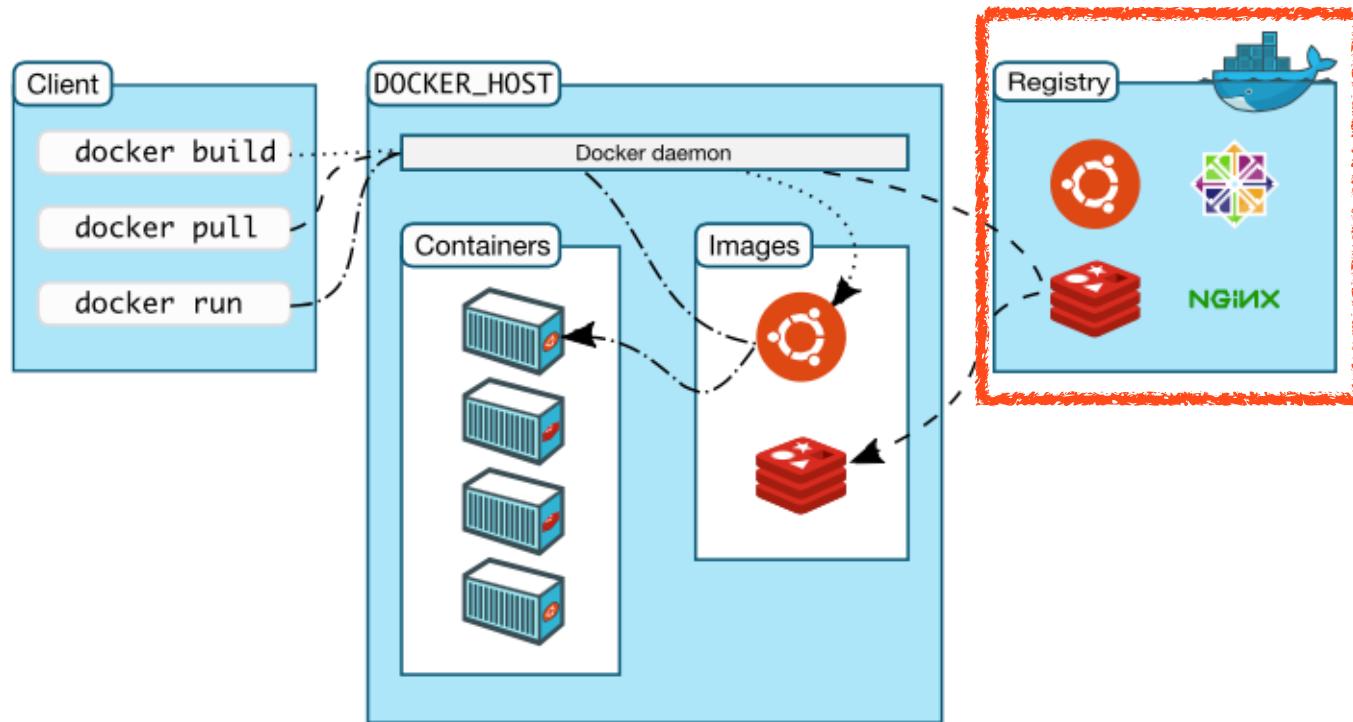
## ▪ Docker Client

- Docker와 사용자 간 인터페이스 제공
- 사용자 명령을 받아 Docker Daemon으로 전달 (소켓 통신사용)



# Docker Registry

- Docker Registry는 Image가 저장된 장소
- Private 또는 public registry가 존재
- Docker Hub는 Docker, Inc에서 제공하는 public registry
  - Docker Hub는 GitHub 같은 기능을 제공
  - Docker Hub에 있는 다른 사용자의 Image도 사용 가능



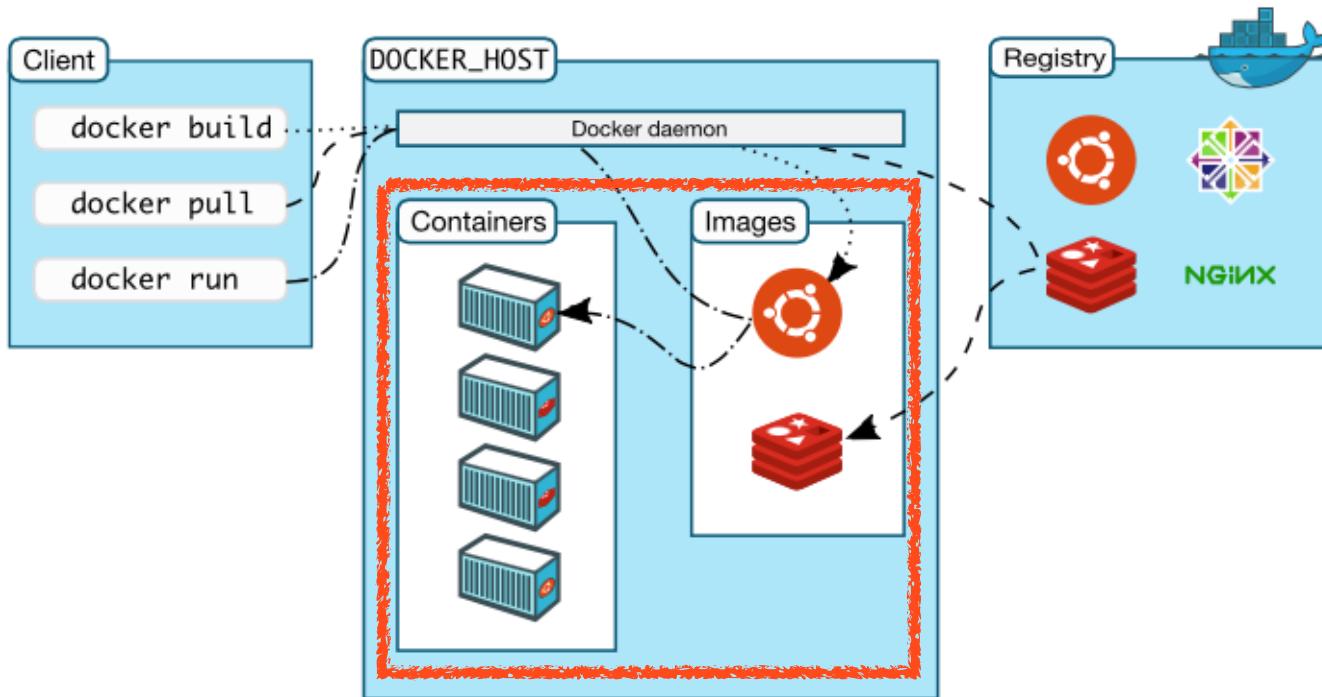
# Docker Image & Container

## ▪ Image

- Container를 만들기 위한 자원  
(실행 파일, 라이브러리, 애플리케이션 등)
- Union 파일 시스템 사용  
(여러 파일 시스템을 통합해 하나로 만듦)
- Public 또는 private registry에 저장
- 다른 사용자가 만든 것도 사용 가능

## ▪ Container

- Image를 실행한 상태
- 독립된 애플리케이션 플랫폼
- 하나 또는 그 이상의 Image로 구성
- 애플리케이션 실행을 위한 모든 것을 포함



# ■ Union Filesystem

- 여러 filesystem을 통합해 하나의 filesystem으로 보이게 하는 기술

- 수정된 layer만 새로 빌드 후 교체하는 경량화된 방식

- Layer(filesystem을 나눈 단위)

- Read-Only Layer (Container를 제외한 나머지)

- Boot filesystem(bootfs): Kernel
    - Base Image: debian, ubuntu, busybox, etc ...
    - Image: Application, code, etc ...

- Read-Write Layer (항상 최상위 Layer에 위치)

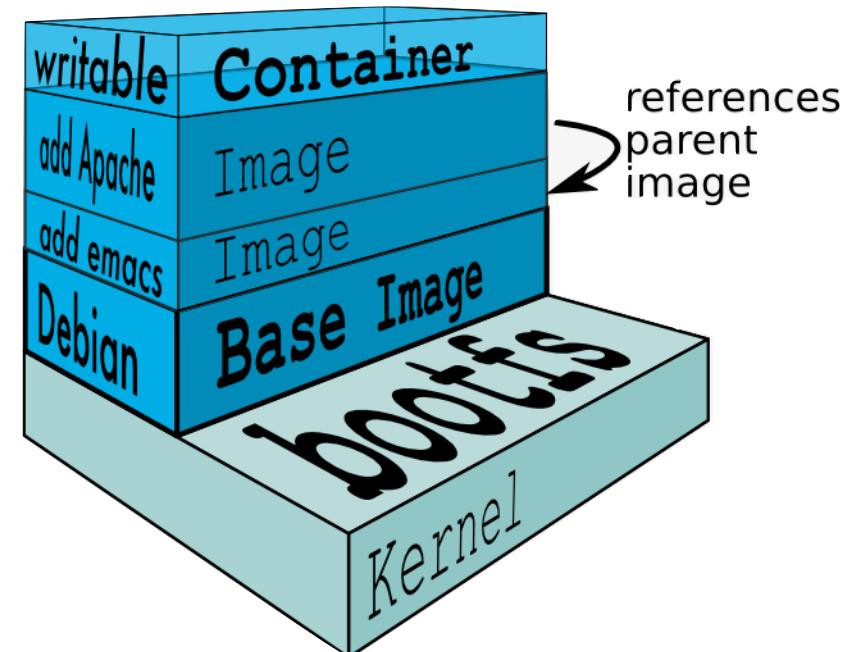
- Container: 수정 가능한 layer

- Copy-on-Write

- Image 내용을 수정할 땐 Container로  
수정할 파일을 복사한 후에 내용 수정  
(사용자는 Container의 내용이 보임)

- 현재 상태를 저장할 땐 Container를 Image로 만듦

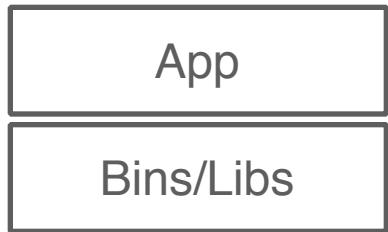
- 만들어진 Image는 Registry에 저장
  - Container가 Image로 만들어 지면  
새로운 Container를 위에 생성함



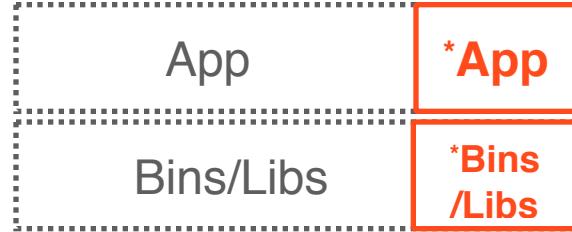
## : Image 수정 및 업데이트

- 기본 이미지에서 수정된 부분만 업데이트

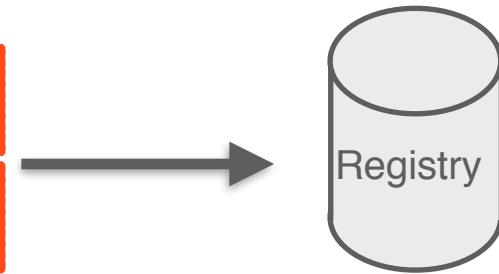
### 1. 수정 전 이미지



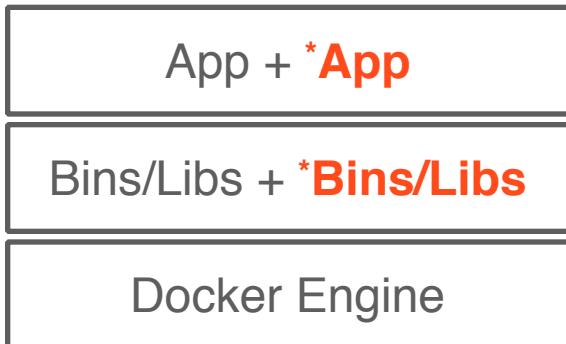
### 2. 이미지 수정



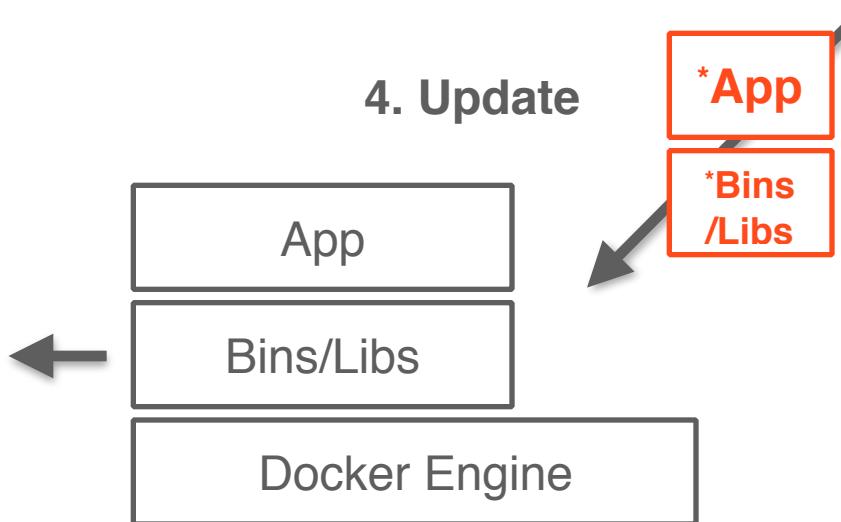
### 3. Push



### 5. 수정된 부분이 추가된 이미지로 동작



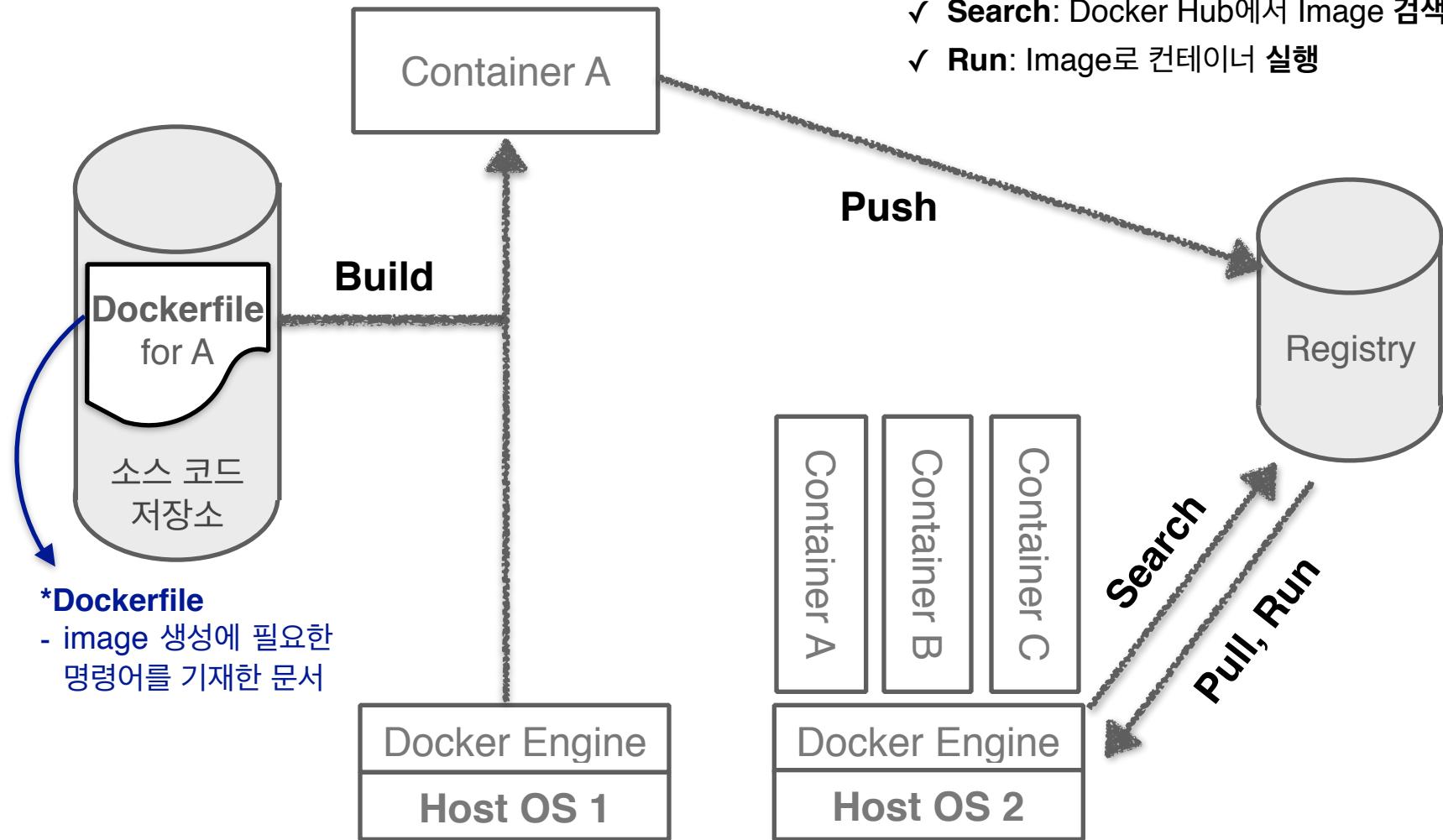
### 4. Update



# Docker 동작 방식

## ▪ Image 생성, 저장, 검색, 실행

- ✓ **Build:** Dockerfile로 Image 생성
- ✓ **Push:** Registry에 Image 저장
- ✓ **Pull:** Registry에서 Image 불러옴
- ✓ **Search:** Docker Hub에서 Image 검색
- ✓ **Run:** Image로 컨테이너 실행



# Dockerfile

- Dockerfile은 Docker image를 자동으로 만들기 위해 명령어를 모아둔 문서

- 사용자가 image 생성을 위한 명령어를 순차적으로 작성
- Docker client의 터미널에 “**docker build**” 명령을 입력해 image 생성

- Dockerfile 문법

- **FROM <이미지>:<태그>** - 생성될 이미지의 기반을 설정 (eg. ubuntu, red hat, etc...)
- **MAINTAINER** - 관리자 정보
- **RUN** - 쉘 스크립트 또는 명령어
- **VOLUME** - 호스트와 공유할 경로의 목록 (docker run 명령의 “-v” 옵션으로도 설정 가능)
- **CMD** - 컨테이너가 시작할 때 실행할 실행파일 또는 쉘 스크립트
- **WORKDIR** - CMD에 설정한 실행 파일이 실행될 경로
- **EXPOSE** - 호스트와 연결할 포트 번호

```
FROM ubuntu:14.04
MAINTAINER Foo Bar <foo@bar.com>
RUN apt-get update
RUN apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf
VOLUME ["/data", "etc/nginx/site-enabled", "/var/log/nginx"]
WORKDIR /etc/nginx
CMD ["nginx"]
EXPOSE 80
```

<dockerfile 예제>

Dockerfile 작성 후 이미지 생성 명령어  
: **docker build <옵션> <Dockerfile 경로>**

# ■ Docker 명령어

---

## ▪ Docker client의 CLI 명령어

### - **docker build**

: Dockerfile로 이미지 생성

### - **docker run**

: 이미지로 컨테이너를 생성

### - **docker create**

: 이미지를 이용해 컨테이너를 생성.  
(run과 다르게 컨테이너를 생성만 함)

### - **docker commit**

: 컨테이너의 변경 사항을 이미지로 생성

### - **docker search**

: Docker Hub에서 이미지를 검색

### - **docker pull**

: Registry에서 이미지를 받아 옴

### - **docker push**

: Registry에 이미지를 올림

### - **docker inspect**

: 컨테이너와 이미지의 세부 정보를  
JSON 형태로 출력

### - **docker images**

: 현재 사용 가능한 이미지 목록을 출력

### - **docker ps**

: 컨테이너 목록을 출력

### - **docker kill**

: 컨테이너에 KILL 시그널을 보내 종료

### - **docker rm**

: 컨테이너를 삭제

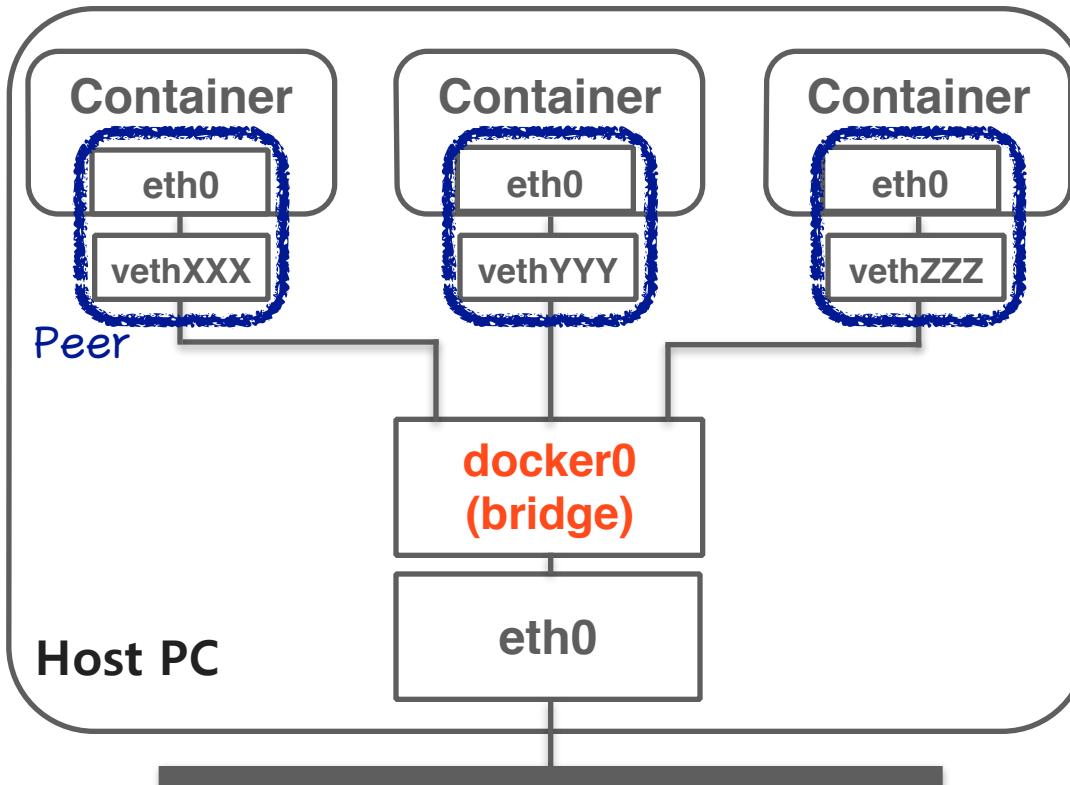
### - **docker rmi**

: 이미지를 삭제

### - **etc...**

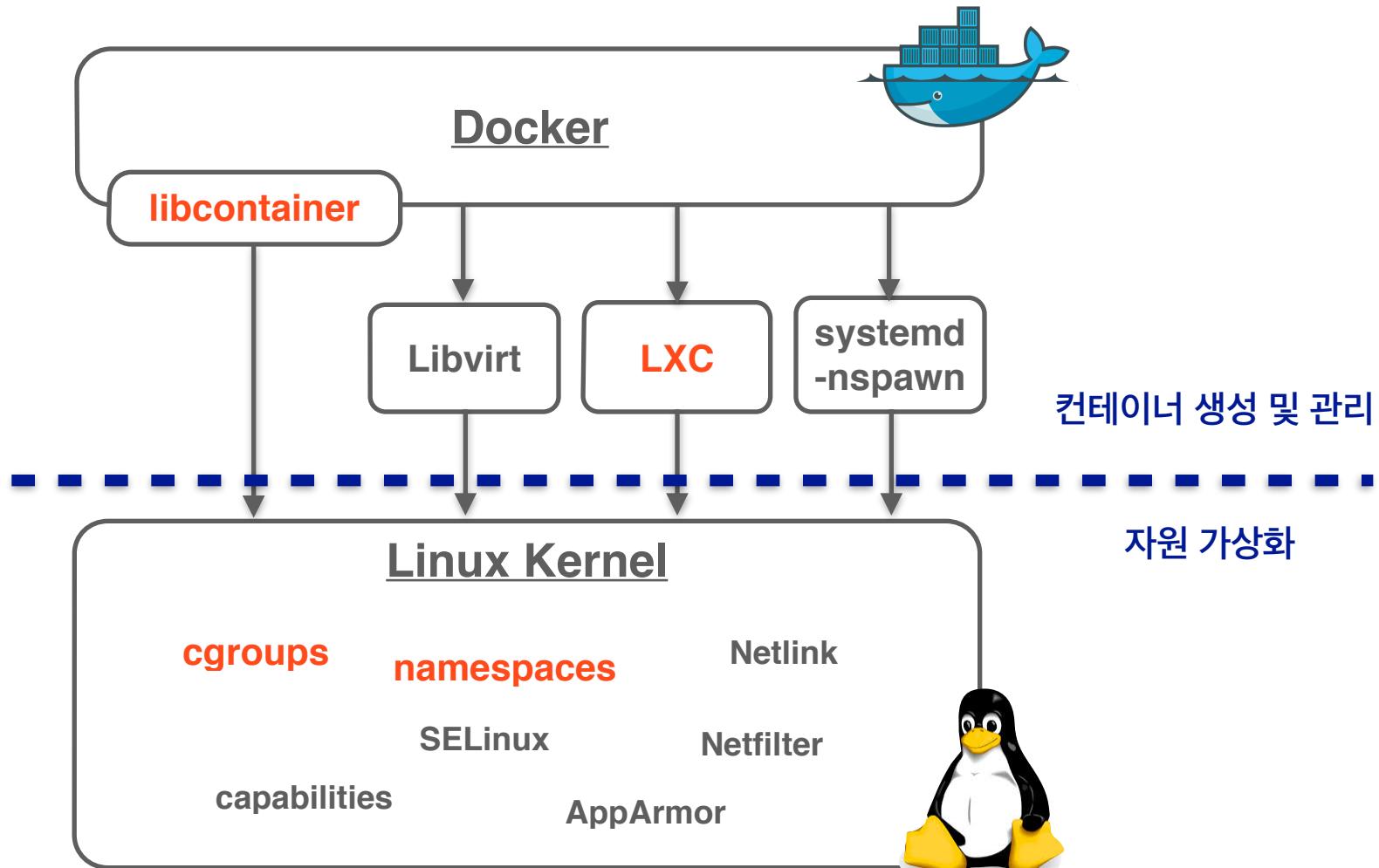
# Docker Networking

- Docker 컨테이너가 외부로 통신하기 위한 구조
  - Default gateway로 Linux bridge인 **docker0**를 만듦
    - 외부 네트워크와 교환하는 패킷에 NAT 작업 수행
    - 컨테이너 간 네트워크 연결
  - 각 컨테이너에 격리된 네트워크 공간을 만들고 컨테이너의 eth0에 static IP 할당
  - 컨테이너 내부의 eth0와 docker0로 연결된 veth 인터페이스를 연결해 컨테이너 외부와 통신



# Docker가 사용하는 기술

- Docker가 컨테이너를 관리하기 위해 사용하는 기술
  - libcontainer, LXC(LinuX Container), cgroups, namespaces, etc...



# ■ LXC & libcontainer

---

## ▪ LXC(LinuX Container) & libcontainer

### - 컨테이너 라이프사이클 관리 기술 (Docker 실행 옵션으로 선택 사용)

- 과거에는 “chroot” 명령어로 컨테이너를 생성함
  - 직접 하위 경로에 실행 파일과 공유 라이브러리를 설치해야 함
  - 완벽한 가상화 환경이 아니기에 제약 사항이 많음
  - 설정이 복잡함

### - LXC

- 초기 Docker에서 사용된 Linux용 컨테이너 관리 기술
- namespaces, cgroups를 사용해 컨테이너를 생성 및 관리함
- Linux 2.6.24에서 처음 발표

### - libcontainer

- Docker version 0.9 이후부터 default로 사용
- Linux 플랫폼에 의존적인 LXC를 대체하기 위해 Docker에서 만든 컨테이너 관련 기술
  - Host OS 의존성을 제거해 다양한 플랫폼 지원(Red Hat, Microsoft Windows, etc ...)
- Go 언어로 구현

## ■ Control Groups

---

### ▪ cgroups(control groups)

- 컨테이너의 컴퓨팅 자원을 제어하는 기술

- 프로세스 집단의 자원(CPU, memory, disk I/O, network, etc ...) 관리 기능

- 2006년 Google에서 개발 시작

• 초기 이름은 “process container”였으나 2007년 “control groups”으로 변경

- Linux 2.6.24 이상 버전부터 지원

- 기능

- Resource limitation: 설정한 메모리 이상 사용하지 못하도록 보호
- Prioritization: 그룹에 따라 CPU 사용량 또는 디스크 I/O 분배
- Accounting: 자원 사용량 측정
- Control: 사용 중단, 체크 포인트 설정, 재시작 등

# ■ Namespaces

---

## ▪ namespaces

- 컨테이너를 만들기 위한 프로세스 가상화 기술
- 하이퍼바이저를 사용하지 않고 격리된 공간을 제공

- 구현 기술

- clone() - 새로운 프로세스와 새로운 namespace 생성
- unshare() - 존재하는 프로세스에 새로운 namespace 생성
- sets() - 존재하는 namespace에 합류

- namespaces 종류

- Mount namespace (mount point, filesystem)
- Network namespace (network interface, routing, etc ...)
- UTS namespace (hostname, domain name)
- PID namespace (process)
- IPC namespace (IPC)
- User namespace (UID)

# ■ Namespaces

---

## ▪ namespaces 종류

### - Mount namespace

- 파일 시스템 추상화를 위해 사용
- Container 사용자에게는 VM과 같이 새로운 파일 시스템을 보여줌

### - UTS namespace

- user name과 domain name을 기반으로 초기화 및 설정 스크립트 작성 가능

### - PID namespace

- PID 재사용 가능
- Container는 자신의 PID 1을 가짐

### - Network namespace

- Container를 host 같이 다룰 수 있음
- network interface, routing table, IP4/6 network stack, firewall, port number

### - IPC namespace

- 공유메모리, 큐, 세마포어의 IPC 자원 분리

### - User namespace

- user와 group ID 공간을 격리
- namespace 외부에 있는 user나 group이 내부의 공간을 보거나 건드릴 수 없음

# Docker & CoreOS

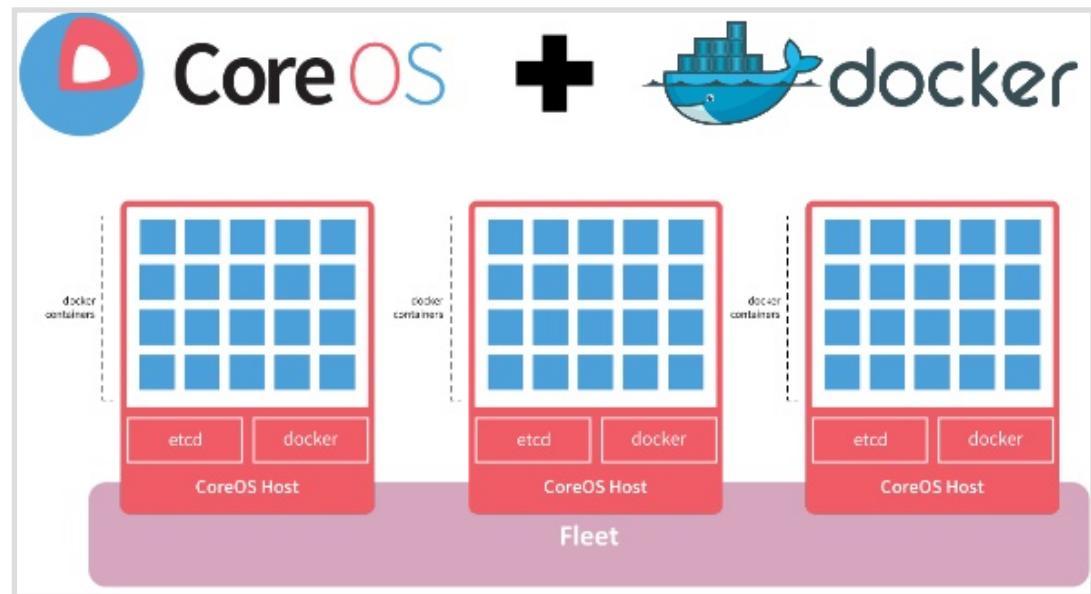
- Docker 서비스(CoreOS)
  - Docker 구동에 최적화된 OS

- 특징

- 간단한 컨테이너 배포
- Clustering
- 동적 확장
- HA 지원

- CoreOS 구성 요소

- etcd
  - 분산 key/value 저장소로 클러스터 설정값 및 노드 정보를 저장하고 공유함
- systemd
  - Linux 서비스 매니저
- fleet
  - etcd와 systemd를 이용한 분산 서비스 실행 시스템
  - systemd는 로컬 서비스를 관리하고 fleet은 원격에서 여러 서버의 서비스를 관리



# Docker & Google

## ▪ Docker 서비스(Google)

- Google Cloud Platform은 Docker 컨테이너 사용을 단순화함

- Google Compute Engine

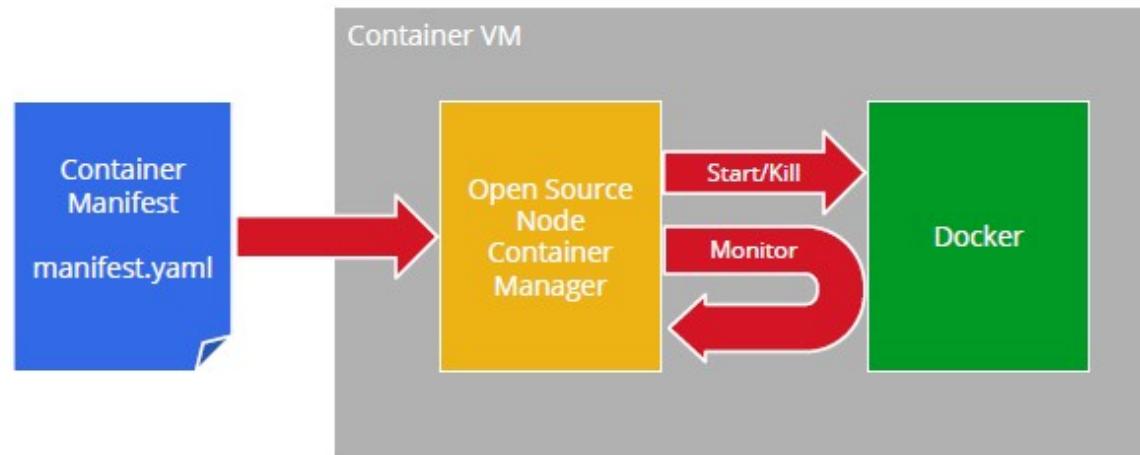
- Google 인프라에서 동작하는 가상 머신을 생성하고 실행

- Google Container Engine

- 컨테이너 오케스트레이션 시스템
  - Google Compute Engine에 Docker 컨테이너를 관리하고 실행함
  - 오픈소스 프로젝트로 Kubernetes 진행

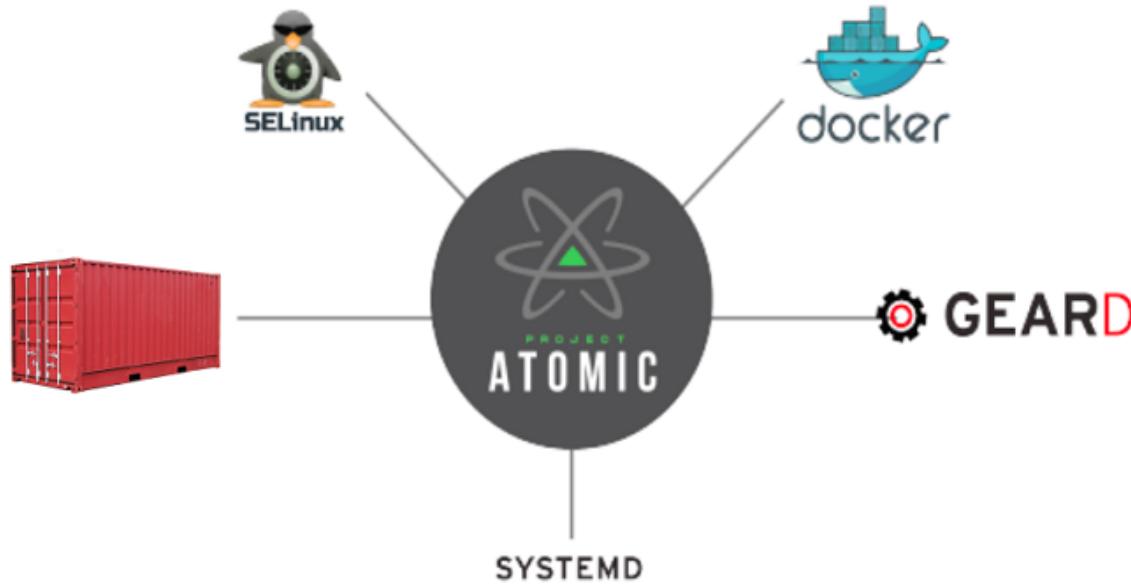
NEW!

### Container VMs in Google Compute Engine



# Docker & Red Hat

## ▪ Docker 서비스(Red Hat)



- RHEL 7(Red Hat Enterprise Linux 7) 부터 Docker 지원
  - 컨테이너 가상화에 최적화 된 OS 개발 프로젝트 **Atomic**을 시작
- OpenShift와 Docker를 통합해 컨테이너를 쉽게 활용할 수 있게 함
  - OpenShift는 클라우드 PaaS 환경에서 애플리케이션 개발, 배포 및 관리 환경을 제공하는 오픈 솔루션
  - systemd를 사용 가능한 Linux 환경에서 Docker 컨테이너를 설치하기 위한 CLI 클라이언트 **GEARD** 개발
- 2014년 3월 Docker 컨테이너 형식으로 패키징 된 애플리케이션 인증 프로그램 시작

# Docker & Amazon

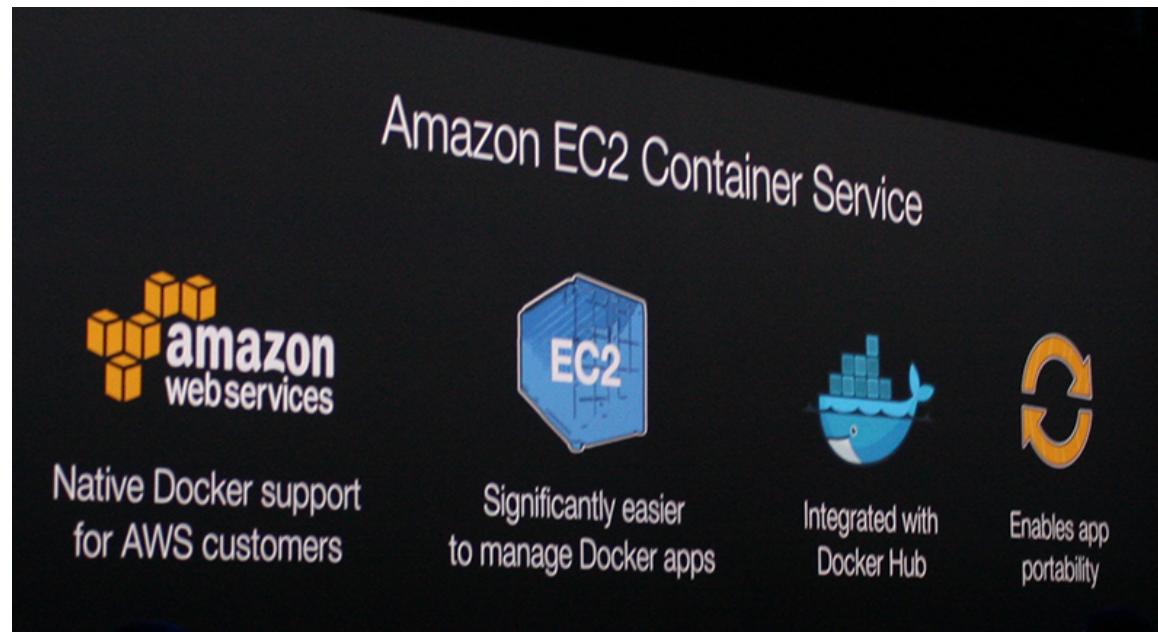
## ▪ Docker 서비스(Amazon)

### - E2C Container Service

- E2C 클러스터의 Docker 컨테이너 관리 서비스
- 클러스터 관리, 유연한 컨테이너 배치, 다른 AWS 서비스와 연동, 보안 강화

### - Elastic Beanstalk

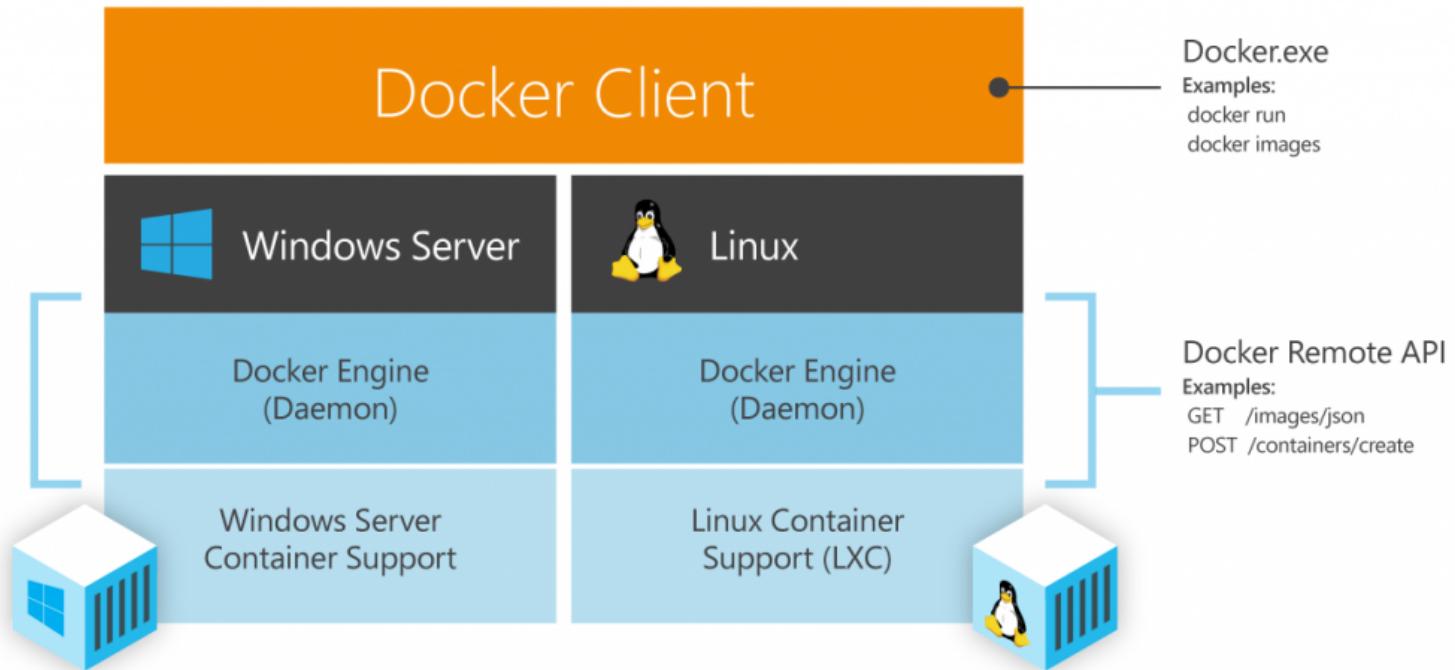
- AWS 리소스를 조합해 완성된 애플리케이션 플랫폼을 제공함
- Docker registry와 연동해 Docker 이미지로 만든 컨테이너를 제공



# Docker & Microsoft

## ■ Docker 서비스(Microsoft Azure)

- 2014년 10월 Window Server에 Docker 지원을 선언
- 리눅스 기반에서 작동한 모든 기능과 기술을 윈도우 서버에도 똑같이 지원 예정



# Docker + OpenStack

- OpenStack에 Docker를 적용하면 빠른 배포, 유연성, 자원의 효율적 사용 등의 장점이 있음

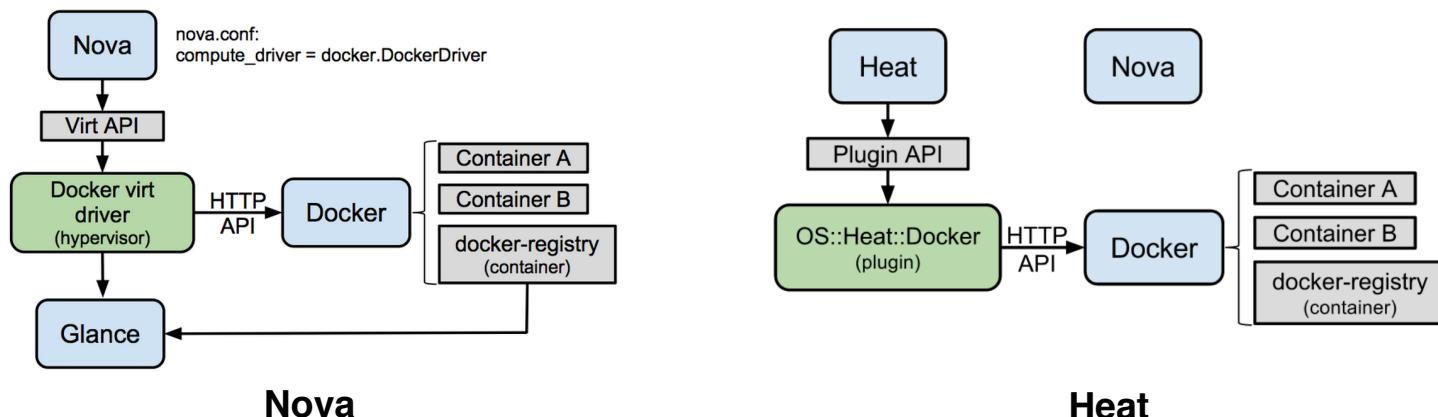
- OpenStack과 Docker 연동

- Nova(컴퓨팅 자원 관리 프로젝트)

- Nova의 가상 드라이버와 Docker를 연동
    - Docker는 새로운 하이퍼바이저로써 동작 (같은 API로 VM 대신 컨테이너 배치)
    - Glance에 Docker 이미지 저장

- Heat(오케스트레이션 프로젝트)

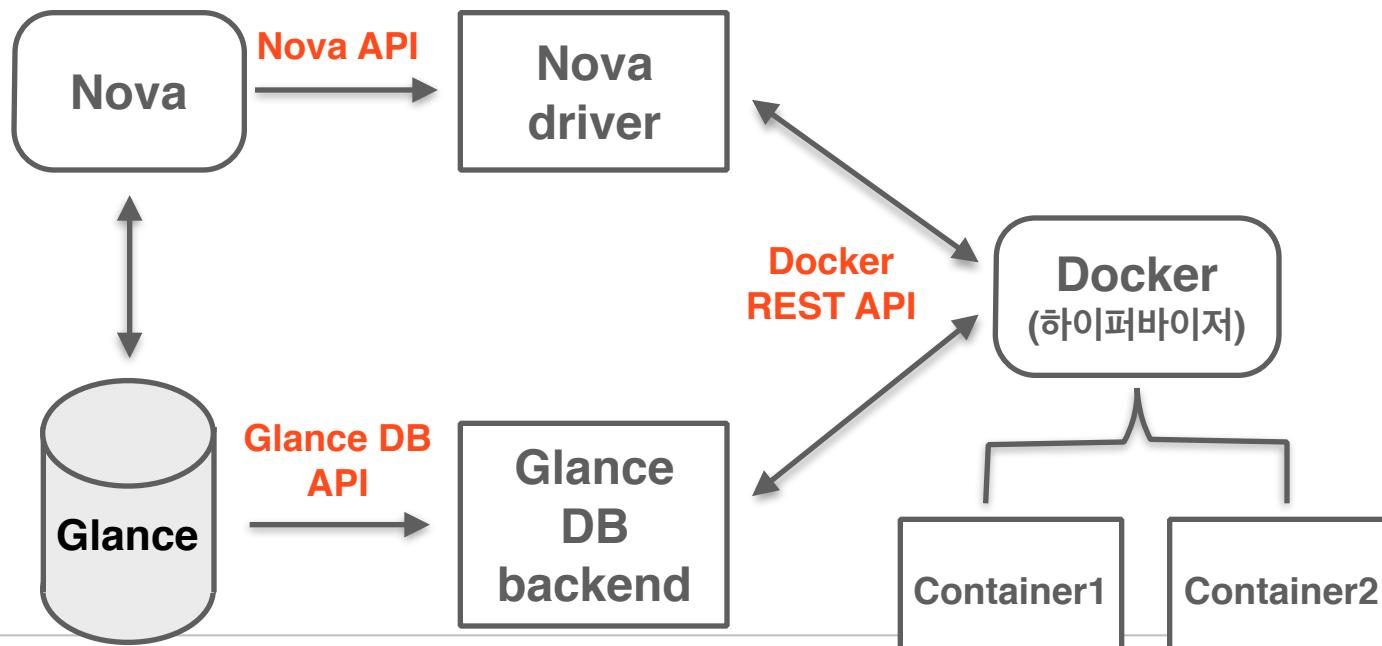
- Docker를 위한 Heat 플러그인을 추가
    - Docker API를 사용해 Heat과 Docker가 직접 통신



# Docker + OpenStack(Nova)

## ▪ Nova Driver를 이용한 Docker 연동

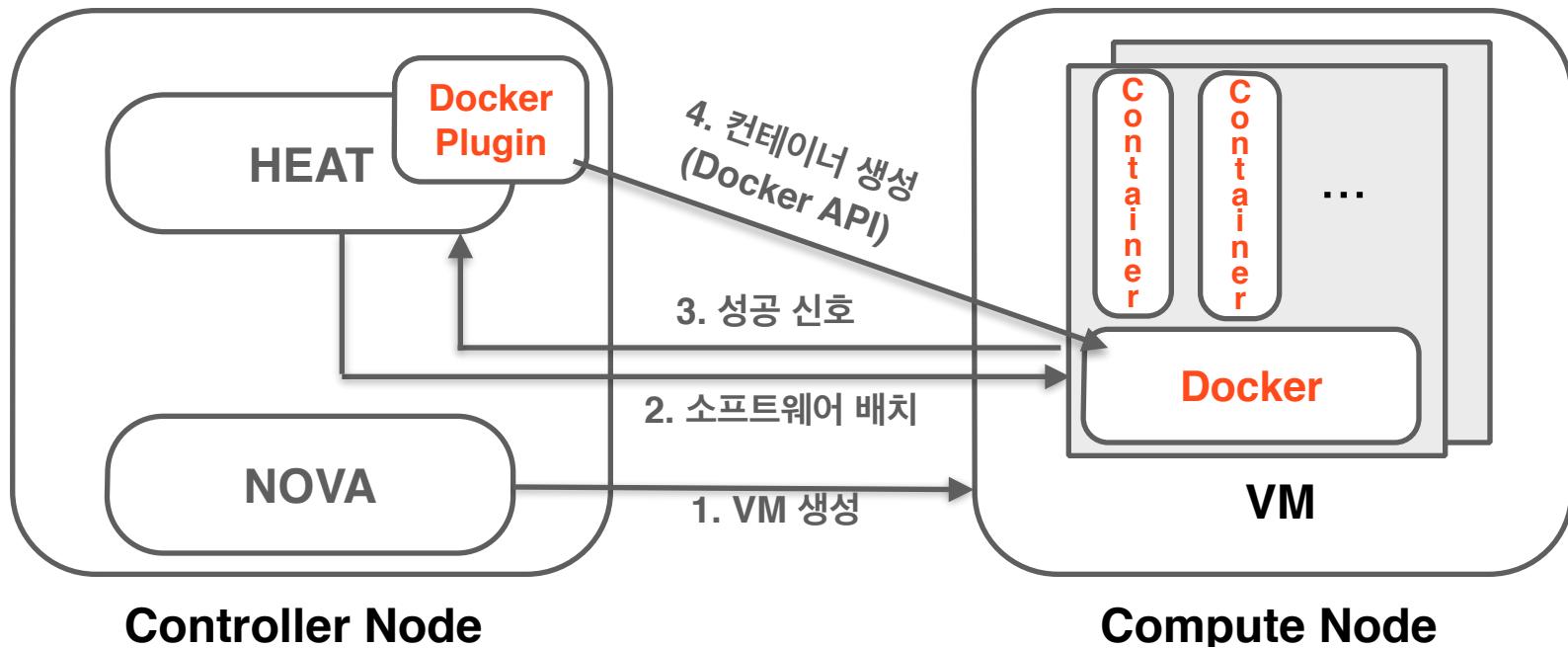
- OpenStack Havana 버전과 함께 소개
- Docker는 OpenStack Nova Compute의 하이퍼바이저로 동작
- Nova driver가 Docker와의 인터페이스를 위한 HTTP 클라이언트를 내장(컨테이너 관리)
- Nova driver는 Glance(OpenStack 이미지 서비스)로부터 이미지를 가져와 Docker에 올림
  - Docker에서 Glance로 이미지 저장 가능
- 컨테이너 시작, 종료, 재부팅, 스냅샷 등 제한된 기능 지원
- Icehouse, Juno, Kilo에서는 메인 소스 코드로 등록하지 않음 (빠른 업데이트 등의 이유)
- Horizon UI와 연동 가능



# Docker + OpenStack(Heat)

## ▪ Heat plugin을 이용한 Docker 연동

- OpenStack Icehouse 버전과 함께 소개
- “Docker + Nova”와 별개로 동시에 진행 중
- 오피스트레이션 프로젝트인 Heat을 이용해 Docker 컨테이너 관리 및 배치가 가능
- Heat 모듈에 Docker plugin을 장착
  - Docker API를 직접 호출 (전체 Docker API 사용 가능)
  - 기존 OpenStack 위에서 Docker 컨테이너 관리와 배치가 가능



# Docker + OpenStack(Magnum)

---

## ▪ OpenStack Magnum: Containers as a Service

### - OpenStack Container 서비스를 위한 프로젝트

- 2015년 1월 첫 소스 배포 후 3월 24일 프로젝트 공식 승인

### - Vision: Containers as a First class resource in OpenStack

- 오케스트레이션 엔진을 만들기 위한 OpenStack API 개발
- 백 엔드 컨테이너 기술로 Kubernetes 그리고 Docker를 사용
- VM 또는 베어메탈 클러스터에서 컨테이너 동작
- CoreOS 또는 Atomic 오케스트레이션 가능

### - Magnum vs (Nova + Docker)

- Magnum은 Nova API 이상의 기능 지원 가능
- Magnum도 인스턴스 할당에는 Nova를 사용

### - Magnum vs (Heat + Docker)

- Magnum은 컨테이너 기술 선택 가능
- Magnum은 Glance 사용 가능

MAGNUM

*Containers as a Service*

# Docker + OpenStack(Magnum Architecture)

## ■ Magnum Architecture

### - Magnum API(REST Server)

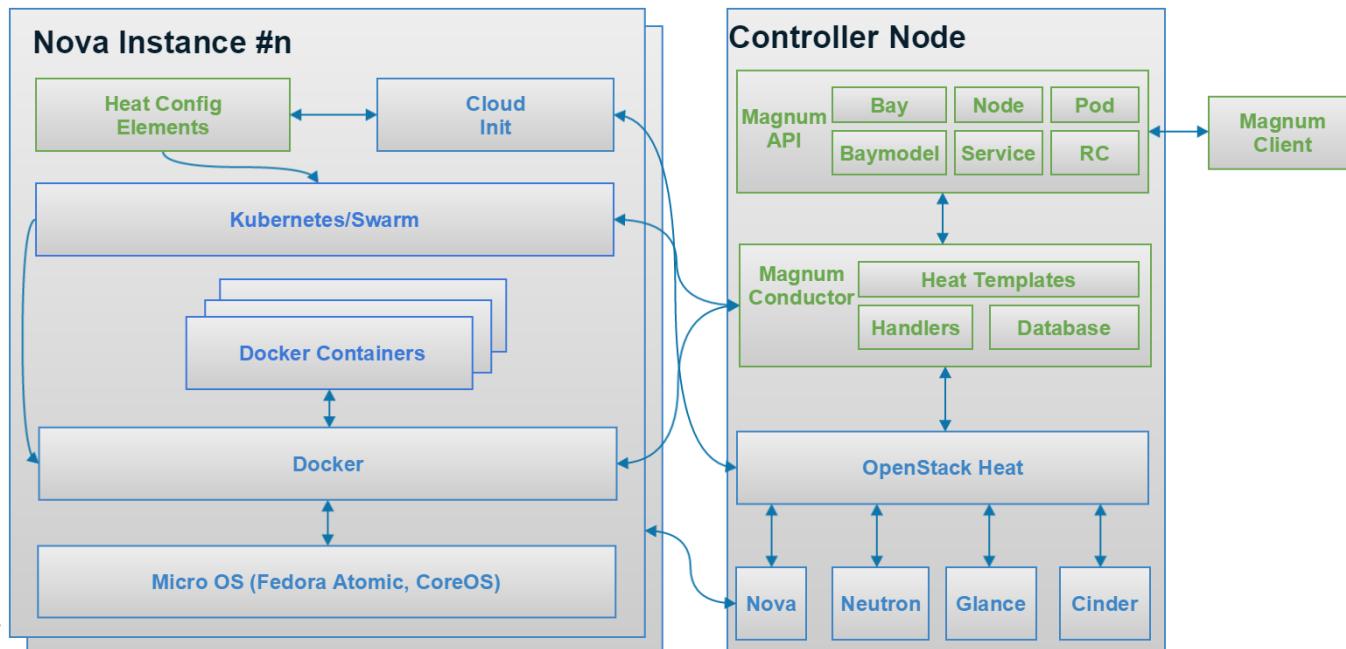
- Client에게 받은 요청을 AMQP를 이용해 conductor 프로세스로 전달  
(AMQP: 서로 다른 시스템 간 메시지를 교환하기 위한 MQ 프로토콜)

### - Magnum Client(REST Client)

- REST Server로 보낼 요청 메시지 생성

### - Magnum Conductor

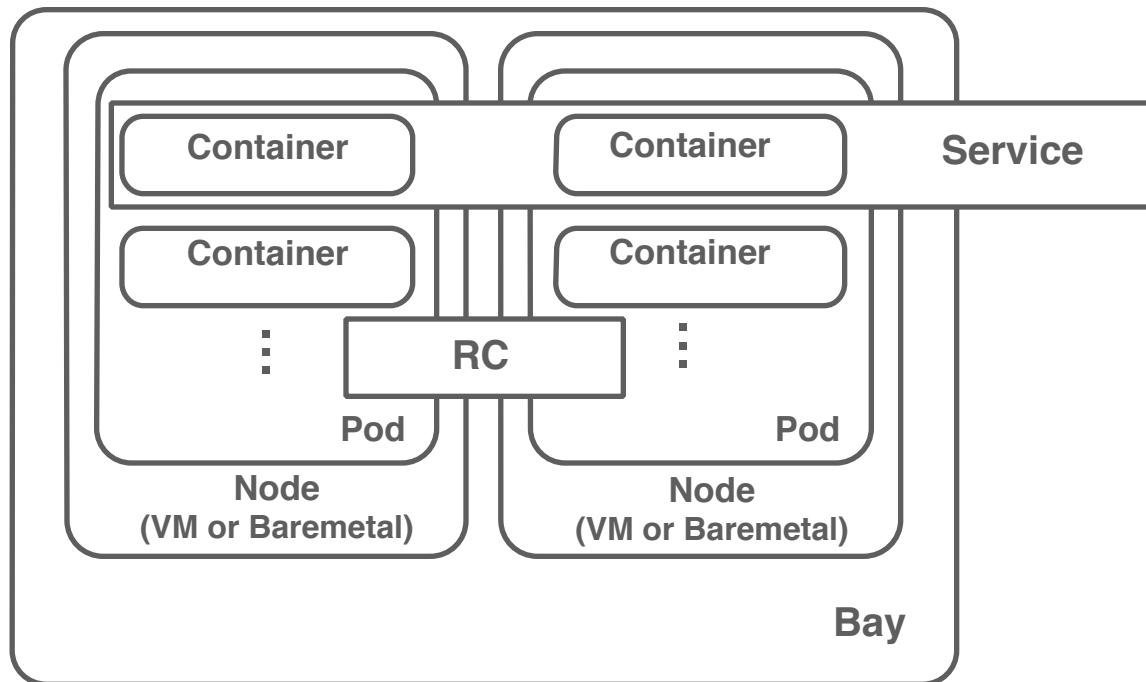
- Controller에서 동작
- Kubernetes 또는 Docker REST API endpoint와 연결



# Docker + OpenStack(Magnum Object)

## ▪ Magnum Object

- **Bay**: Node의 집합
- **BayModel**: 새로운 Bay를 생성하기 위한 템플릿 정보를 저장
- **Node**: 베어메탈 또는 가상 머신
- **Pod**: 물리 또는 가상 머신에 동작하는 컨테이너의 집합
- **Service**: Pod의 논리적 집합과 그들 사이에 접근하기 위한 정책을 정의하는 추상적 개념
- **RC(ReplicationController)**: Pod 그룹의 자원을 제한하기 위한 추상적 개념
- **Container**: Docker Container



## ■ Docker + NFV

---

- 컨테이너 위에 VNF를 올림 (ETSI/NFV-ISG 표준)

- 통신 사업자들이 VM을 컨테이너로 대체하는 것에 긍정적 반응

- 하드웨어에 VM을 여러 개 사용하면 무거움
    - 하이퍼바이저 사용에 따른 라이센스 비용 지급
    - VM 성능에 대한 걱정

- VNF를 컨테이너에서 동작할 때 장단점

- 장점

- 하이퍼바이저 없이 구현 가능
      - 라이센스 비용이 필요 없음
      - 속도, 성능, 휴대성이 우수함
      - 확장성이 좋음
      - VM 보다 자원 소비량이 적음
      - 데이터센터의 파워 소비량 절약

- 단점

- 보안이 약함
      - 컨테이너 관리 기술 필요
      - 컨테이너 내부의 완벽한 격리가 필요

# ■ Open Container Initiative

---

## ▪ OCI(Open Container Initiative)

- 2015년 6월 22일 OCP(Open Container Project)라는 이름으로 시작

- 2015년 7월 22일 첫 드래프트 공개와 OCI로 이름 변경

- 컨테이너 표준을 목표로 하는 오픈 프로젝트

- **업체의 종속성 없이 컨테이너가 모든 플랫폼에 동작**할 수 있게 하는 것이 목표

- 컨테이너 포맷과 런타임 관련 기술의 표준 제작

- 프로젝트 특징

- 특정 고객 또는 오케스트레이션 같은 높은 레벨의 개념은 관여하지 않음

- 특정 벤더 또는 프로젝트에 대한 의존도를 낮춤

- OS, 하드웨어, CPU 구조, 클라우드 등의 환경과 관계없이 표준 컨테이너로 동작 가능

- Docker의 표준화 기여도

- Docker가 자사의 컨테이너 이미지 및 runtime 코드를 제공

- Docker 이미지가 사실상의 OCI 표준 포맷으로 결정

- 과거 2년 동안 많은 업체가 Docker Image 포맷을 지원하는 제품을 만듦

- 첫 OCF(Open Container Format) 스펙의 초안 작성

## OCI의 파트너

### OCI 참여 기업

#### LINUX FOUNDATION COLLABORATIVE PROJECTS



APCERA™



DATERA



EMC<sup>2</sup>

FUJITSU



Joyent



Google



NUTANIX™

ORACLE

Pivotal

POLYVERSE



resin.io

SUSE

sysdig



Verizon Labs

vmware®



OPEN CONTAINER INITIATIVE

# ■ 표준 컨테이너

---

## ■ 표준 컨테이너의 5가지 원칙

### 1. Standard operation

- container tool : 생성, 시작, 종료
- filesystem tool : 복제, 스냅 샷
- network tool : 다운로드, 업로드

### 2. Content-agnostic

- content와 관계없이 동작

### 3. Infrastructure-agnostic

- 인프라와 관계없이 동작

### 4. Designed for automation

- Content 및 인프라와 관계없이 표준 동작하므로 자동화에 적합함

### 5. Industrial-grade delivery

- 어디든 빠르고 효과적으로 컨테이너를 전달 가능해야 함

# ■ Filesystem Bundle

---

## ▪ Filesystem Bundle

- 표준 동작을 위한 모든 정보를 수용하고 **container**를 인코딩하기 위한 포맷을 정의
  - 컨테이너가 로컬 파일시스템에 저장되는 방법만 정의함
    - 버전 관리, 컨테이너 전달 방법 등에 영향받지 않고 동작해야 함
- 표준 컨테이너 **bundle**은 컨테이너 동작에 필요한 모든 콘텐츠를 포함
  - 다른 환경에서도 같은 컨테이너로써 동작 가능해야 함
- 표준 컨테이너 **bundle** 구성 항목
  - 모든 것을 포함하는 하나의 Top-level 경로
  - 하나 또는 그 이상의 콘텐츠 경로
  - 한 개의 설정 파일(config.json)

# ■ Configuration File

---

- 컨테이너의 top-level 경로에는 config.json 파일이 존재해야 함
  - config.json: 표준 동작을 위해 필요한 메타데이터를 포함
- config.json의 설정 내용
  - Manifest version: OCF 버전 정보
  - Root Configuration: 컨테이너의 root filesystem 설정
  - Mount Configuration: 추가된 filesystem의 배열
  - Process Configuration: 프로세스 관리를 위한 설정
  - Hostname(option): hostname 설정
  - Platform-specific Configuration: OS 및 ARCH 설정
  - Linux-specific Configuration
    - namespaces, cgroups, capabilities, sysctl, etc...

# ■ Lifecycle & Runtime

---

## ■ Lifecycle

- Create: 컨테이너 생성 (filesystem, namespaces, cgroups, capabilities)
- Start: 컨테이너에 프로세스를 실행함
- Stop: 외부에서 프로세스를 종료시킴

## ■ Runtime

### - Hook

- 컨테이너의 다양한 lifecycle 전/후에 실행되는 코드
- Hook은 컨테이너 상태를 입력받아 작업에 필요한 정보를 얻음
- Hook은 절대 경로를 가지며, Host의 filesystem에서 실행

#### • Pre-start

- 컨테이너 프로세스 시작 후 그리고 사용자 입력 전에 불림
- 컨테이너를 커스터마이징 하기 위해 사용됨

#### • Post-stop

- 컨테이너 종료 후에 불림
- 디버깅 또는 클린업에 사용됨

## ■ runc

- OCI 런타임의 구현 레퍼런스
- OCF 표준을 따르는 컨테이너 동작을 위한 CLI 툴

## ■ runv

- OCI를 위한 하이퍼바이저 기반의 런타임
- 목표: “**Make VM run like Container**”
  - “App Centric”이라는 컨테이너 철학을 VM에 적용
- 지원 가능한 환경
  - 하이퍼바이저
    - KVM, Xen, VirtualBox
  - 리눅스 배포판
    - Ubuntu 64bit, CentOS 64bit, Fedora 64bit, Debian 64bit

## ■ OCT(Open Container Testing)

- OCI 설정 및 런타임을 위한 테스트 프레임워크

# ■ 요약(Docker)

---

## ▪ Docker

- Docker란 컨테이너 관리, 생성, 실행, 배포를 위한 오픈소스 엔진

### - Docker의 장점

- 애플리케이션의 빠른 배포
- 설치 및 확장이 쉽고, 휴대성이 좋음
- 오버헤드가 낮음
- 관리가 쉬움

### - Docker Component

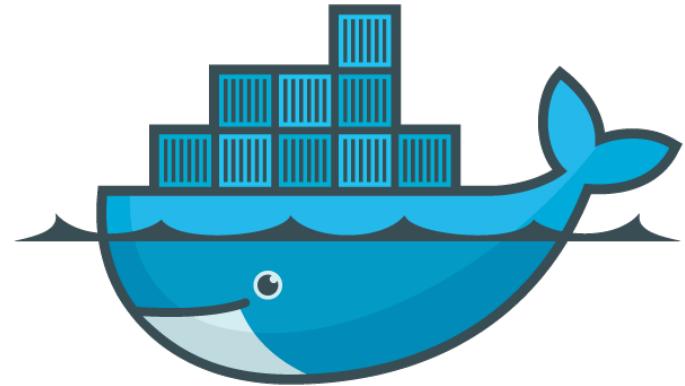
- Server (컨테이너 관리 데몬)/ Client (사용자 인터페이스 제공)
- Image (Container 자원)/ Container (한개 이상의 Image가 동작 상태)
- Registry (Public/Private Image 저장소이며, Docker, Inc가 제공하는 Docker Hub가 존재)

### - Docker 핵심 기술

- Union Filesystem: 여러 계층으로 나뉜 Filesystem을 하나로 통합
- LXC/libcontainer: 컨테이너를 생성하고 관리하기 위해 사용
- cgroups: 프로세스 그룹의 CPU, memory, I/O 자원을 관리
- namespaces: 컨테이너를 만들기 위해 시스템 공간을 격리

### - Docker 연동 서비스

- CoreOS, Google Cloud Platform, RHEL, AWS, MS Azure



## ■ 요약(Docker + OpenStack/NFV)

---

### ▪ Docker + OpenStack

- 빠른 배포, 유연성, 자원의 효율적 사용 등의 장점이 있음
- Nova (컴퓨팅 자원 관리 프로젝트)
  - Dokcer를 연동하기 위한 Nova Driver 필요
  - Docker를 하나의 하이퍼바이저로 취급
  - Glance를 이용해 Docker Image 저장 가능
  - Horizon UI 이용 가능
- Heat (오케스트레이션 프로젝트)
  - Docker와 Heat plugin으로 연동
  - Docker API 모두 직접 사용 가능
  - 기존 OpenStack 자원에 Docker container 배치 가능
- Magnum (Container as a Service)
  - OpenStack Container 서비스를 위한 프로젝트
  - 오케스트레이션 엔진을 만들기 위한 OpenStack API 개발
  - 백 엔드 컨테이너 기술로 Kubernetes 그리고 Docker를 사용

### ▪ Docker + NFV

- Container위에 VNF를 올림 (ETSI/NFV-ISG 표준)
- 통신 사업자들이 VM을 container로 대체하는 것에 긍정적 반응
  - 장점: 확장성, 휴대성, 자원 효율성, 운영 비용 절약
  - 단점: 관리의 어려움, 보안, 컨테이너 간 확실한 자원 격리 필요

## ■ 요약(Open Container Initiative)

---

### ▪ OCI(Open Container Initiative)

#### - 목표

- **업체의 종속성 없이 컨테이너가 모든 플랫폼에 동작**할 수 있게 하는 것
- 컨테이너 포맷과 런타임 관련 기술의 표준 제작

#### - Docker의 표준화 기여도

- Docker가 자사의 컨테이너 이미지 및 runtime 코드를 제공
- Docker 이미지가 사실상의 OCI 표준 포맷으로 결정
- 첫 OCF 스펙의 초안 작성

#### - OCF(Open Container Format)

##### • 컨테이너 포맷의 표준

- 표준 동작을 지원
- 어디서든 동작 가능
- 자동화에 적합한 설계
- 어디든 효과적으로 컨테이너 전달 가능

#### - Implementation

- **runc** : 표준 컨테이너 관리를 위한 CLI 툴
- **runv** : OCI를 위한 하이퍼바이저 기반의 런타임
- **OCT** : OCI 설정 및 런타임을 위한 테스트 프레임워크

## ■ 참조

---

- <https://www.docker.com>
- <http://docs.docker.com/reference/glossary/#boot2docker>
- <http://www.joinc.co.kr/modules/moniwiki/wiki.php/man/12/docker>
- <http://www.slideshare.net/dotCloud/why-docker>
- <http://www.slideshare.net/allingeek/docker-aspects-of-container-isolation>
- [https://en.wikipedia.org/wiki/Docker\\_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))
- <https://wiki.openstack.org/wiki/Docker>
- <http://docs.openstack.org/developer/magnum/>
- <https://wiki.openstack.org/wiki/Magnum>
- <https://wiki.openstack.org/wiki/Kolla>
- <https://github.com/MarouenMechtri/Docker-containers-deployment-with-OpenStack-Heat>
- <http://www.slideshare.net/AnimeshSingh/cloud-foundry-docker-openstack-leading-open-source-triumvirate>
- <https://github.com/MarouenMechtri/Docker-containers-deployment-with-OpenStack-Heat>
- <https://www.opencontainers.org>
- [https://hyper.sh/blog/post/2015/08/11/a\\_step\\_towards\\_the\\_open\\_container\\_initiative.html](https://hyper.sh/blog/post/2015/08/11/a_step_towards_the_open_container_initiative.html)
- [http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)
- <http://www.slideshare.net/Flux7Labs/performance-of-docker-vs-vms>
- <http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack>
- “The Docker Book” by James Turnbull
- “가장 빨리 만나는 Docker” by 이재홍
- “Docker: Up&Running” by Karl Matthias & Sean P.Kane



# 감사합니다.



(주)파이오링크  
서울시 금천구 가산디지털2로 98  
(가산동 550-1) IT캐슬 1동 401호  
TEL: 02-2025-6900  
FAX: 02-2025-6901  
[www.PIOLINK.com](http://www.PIOLINK.com)