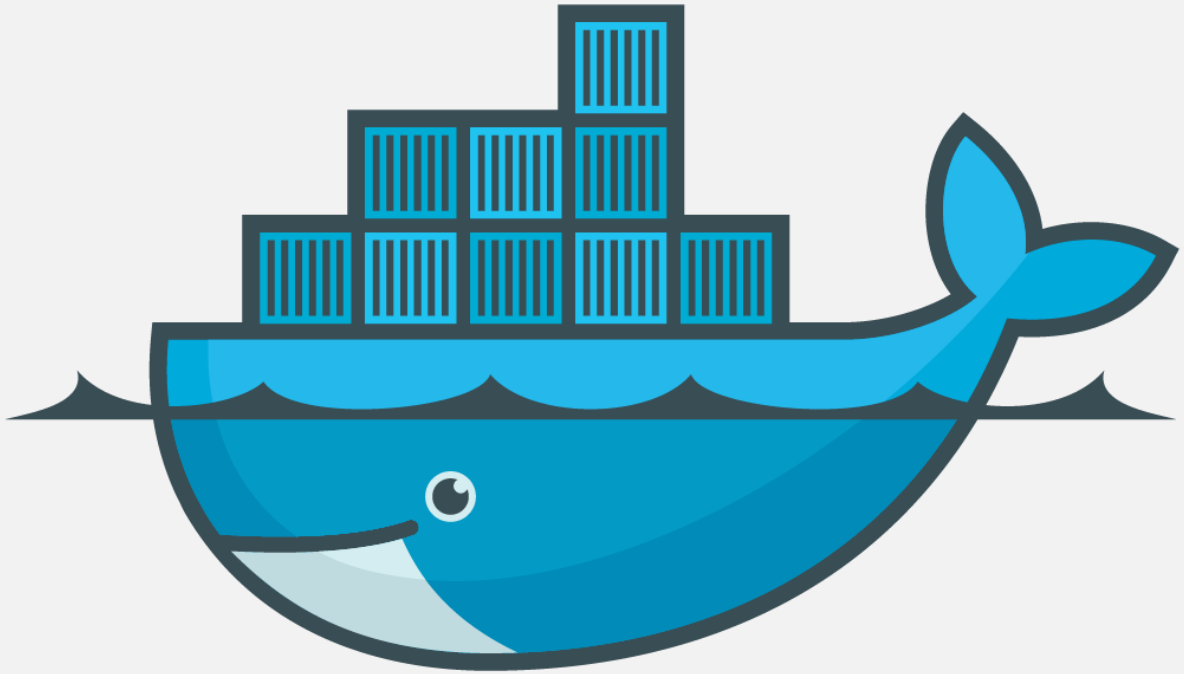
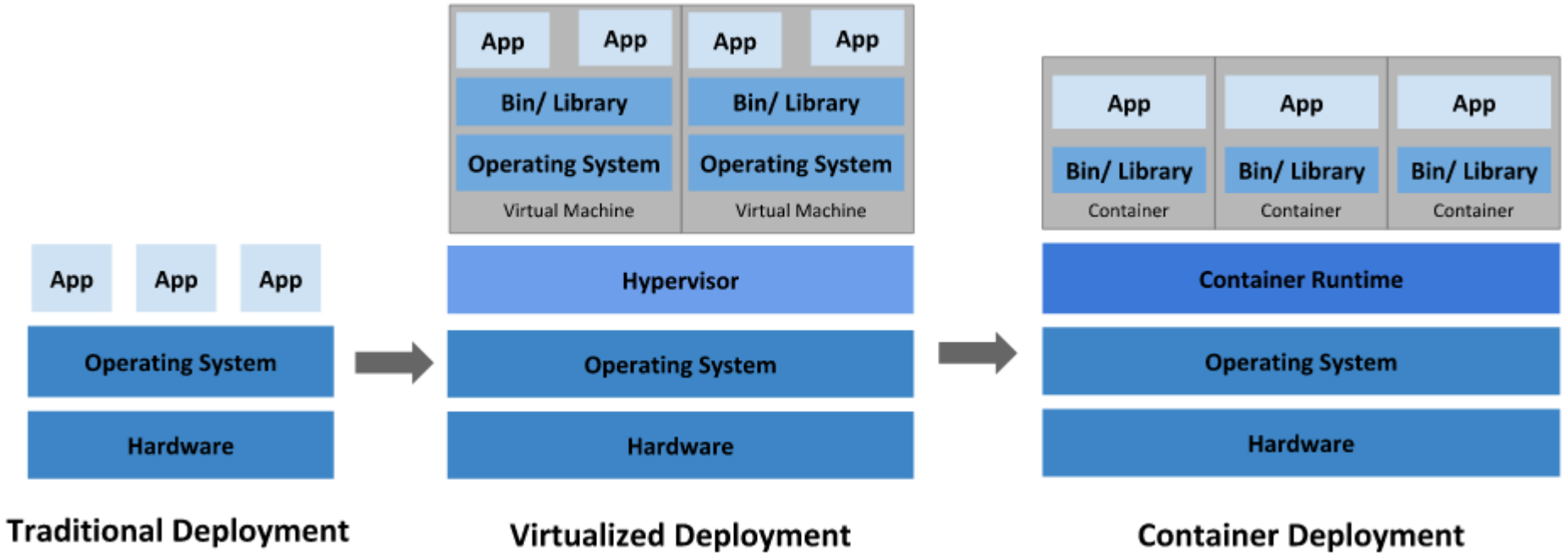


docker

With kubernetes

# 도커와 쿠버네티스

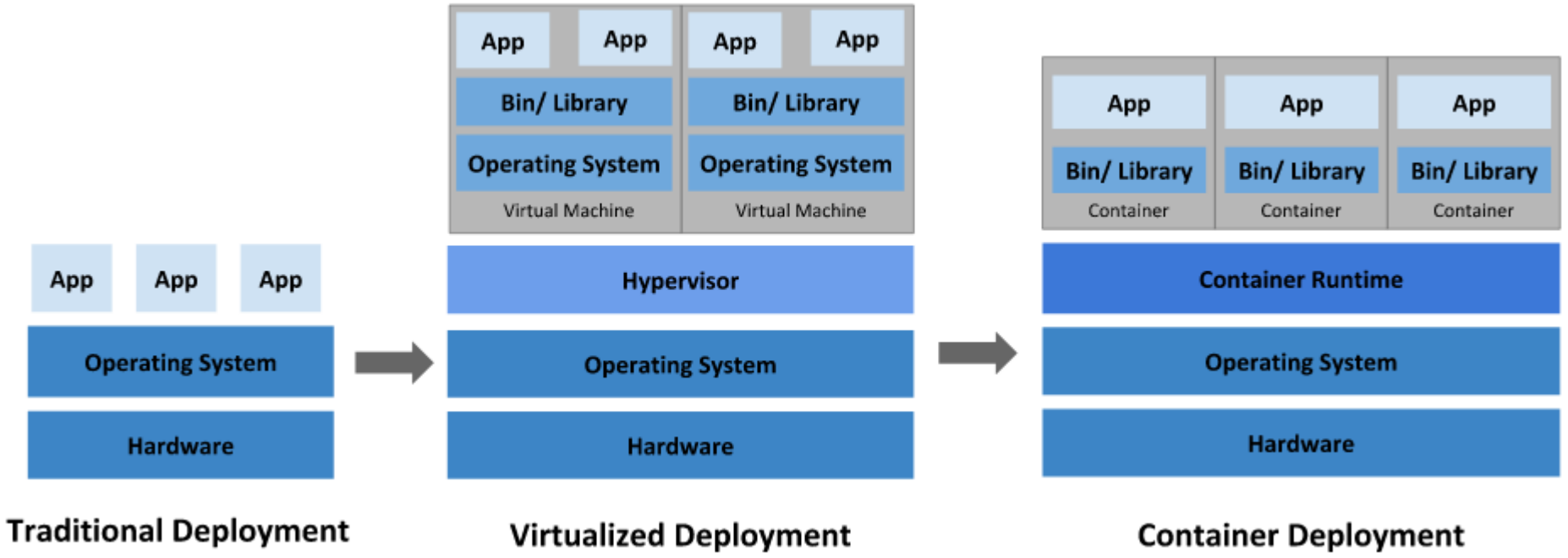
조세나라




## 개발과 배포의 연대기

**전통적인 배포 시대 :**

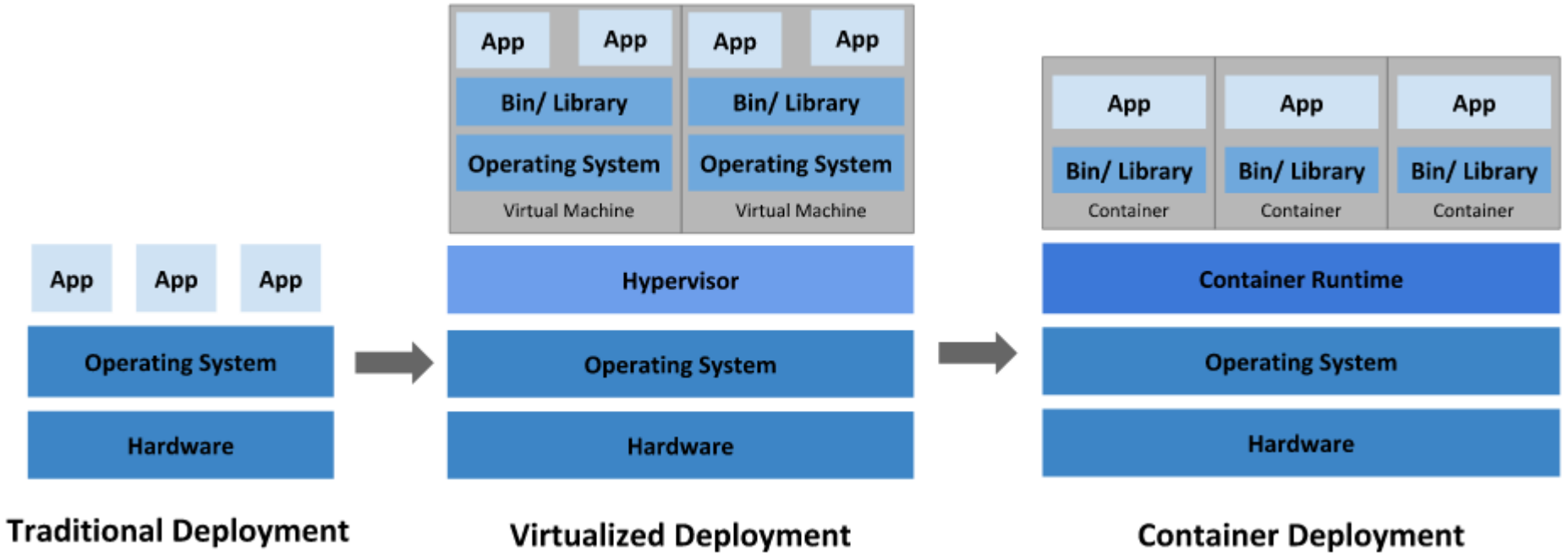
**초기엔 어플리케이션(이하 앱)을 물리 서버에서 실행시켰다.**
**이는 물리 서버에서 여러 앱의 리소스 한계를 정의 할 수 없었기에 리소스 할당의 문제가 발생하였다.**
**예를 들면 한 앱의 인스턴스가 대부분의 자원을 소모한다면 다른 앱은 성능이 저하되었고 이는**
**스케일 업을 통한 물리 서버의 비용 소모를 야기하였다.**




## 개발과 배포의 연대기

**가상 배포 시대 :**

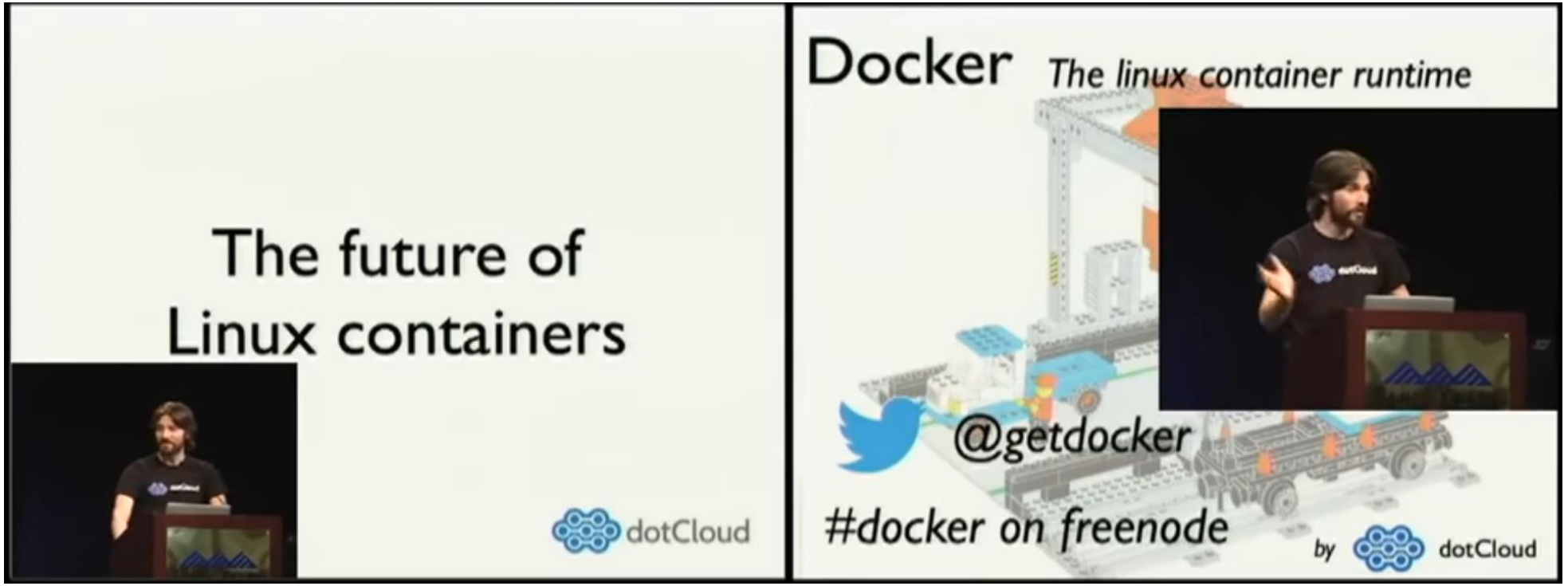
**전통적인 배포의 해결책으로 가상화가 도입되었다.**
**이는 단일 물리 서버의 CPU에 여러 VM을 실행하여 앱을 격리하고 보안성을 유지할 수 있게 하였다.**
**또한 리소스를 효율적으로 활용할 수 있게 되었고 쉽게 앱을 추가하고 업데이트 할 수 있고 물리 서버의 비용 절감과 확장성을 제공한다.**




## 개발과 배포의 연대기

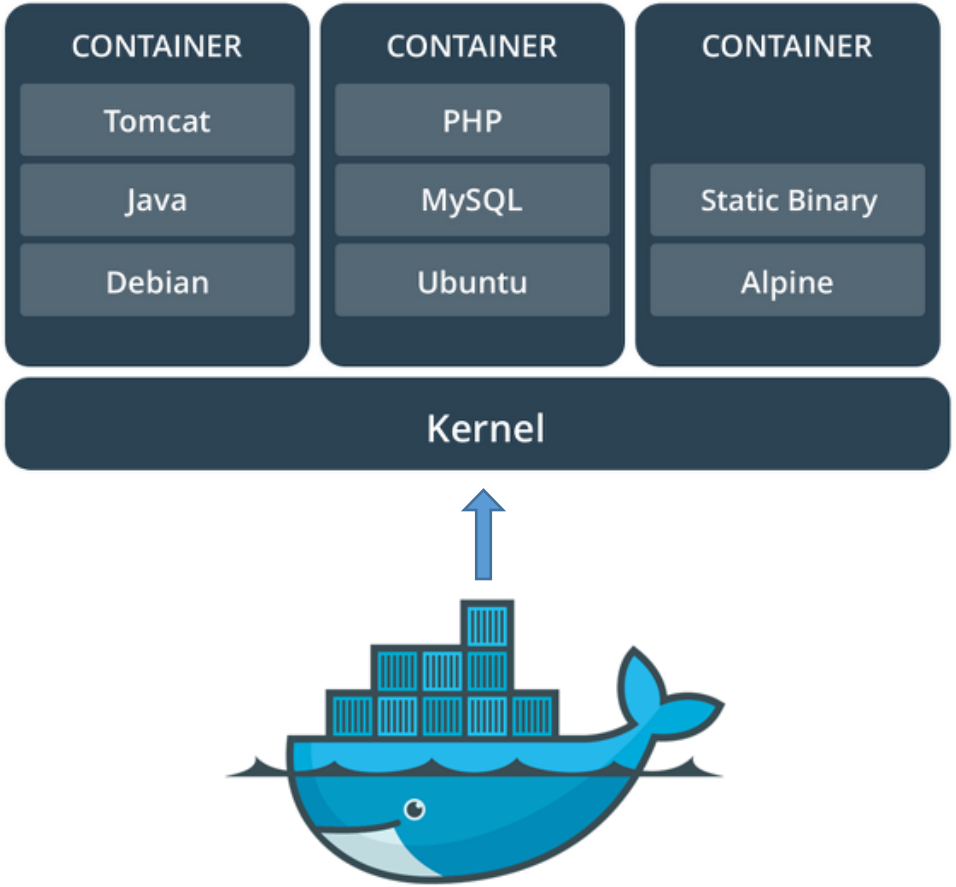
**컨테이너 개발 시대 :**

**컨테이너는 VM과 유사하지만 격리 속성을 완화하여 앱 간의 OS를 공유하여 필요한 리소스가 획기적으로 줄어들 게 된다.**
**그러므로 컨테이너는 가벼우며 기본 인프라와 종속성을 끊어 클라우드나 다른 OS에 쉽게 배포할 수 있게 된다.**
**또한 MSA의 발전으로 작은 서비스 단위가 많아지며 VM보다  경제적인 컨테이너 개발이 대세가 되었다.**




## Docker의 시작

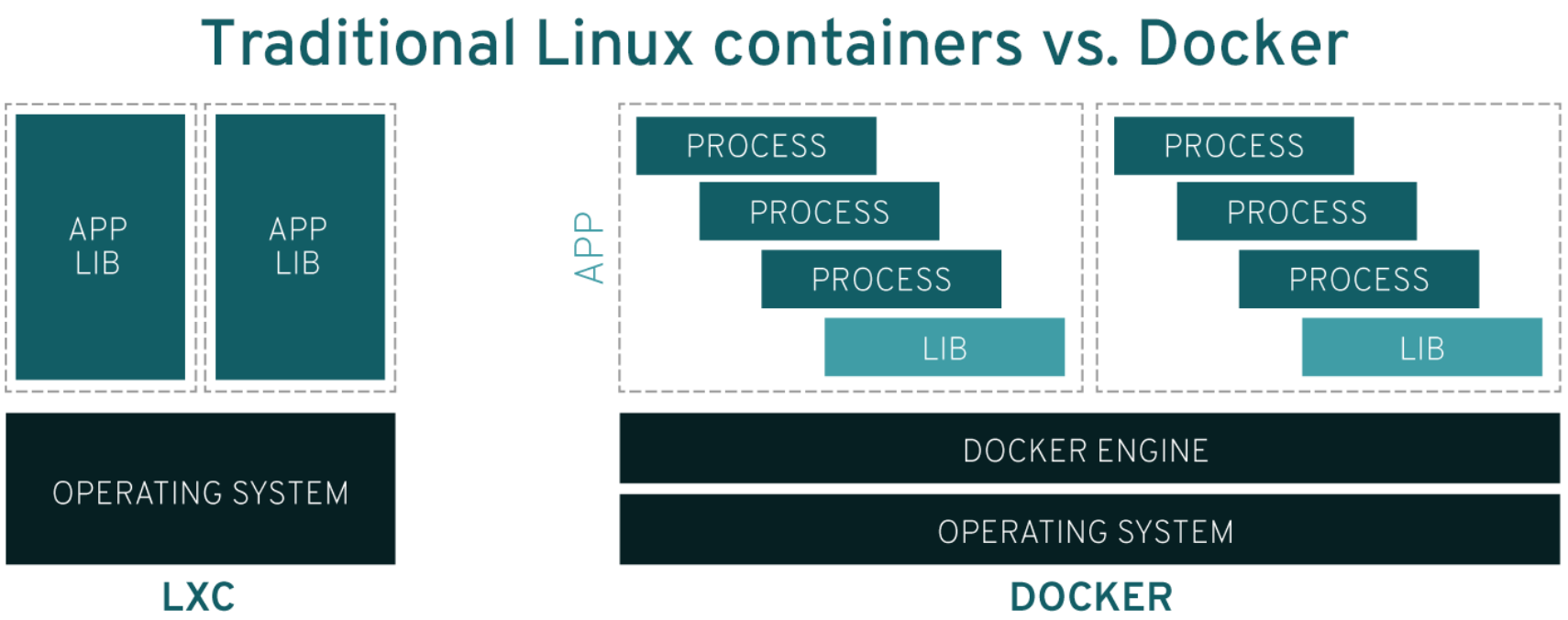
도커(Docker)는 dotCloud의 솔로몬 하익스에 의해 2013년에 공식적으로 등장하였다. 파이콘 US 2013에서“리눅스 컨테이너의 미래”라는 라이트닝 토크에서 도커가 처음으로 소개되었으며, 이 발표 이후 인기를 얻어 사명을 Docker Inc.로 바꾸었으며 이후 DockerCon 2014에서 1.0 버전이 발표되었다.




## Docker란?

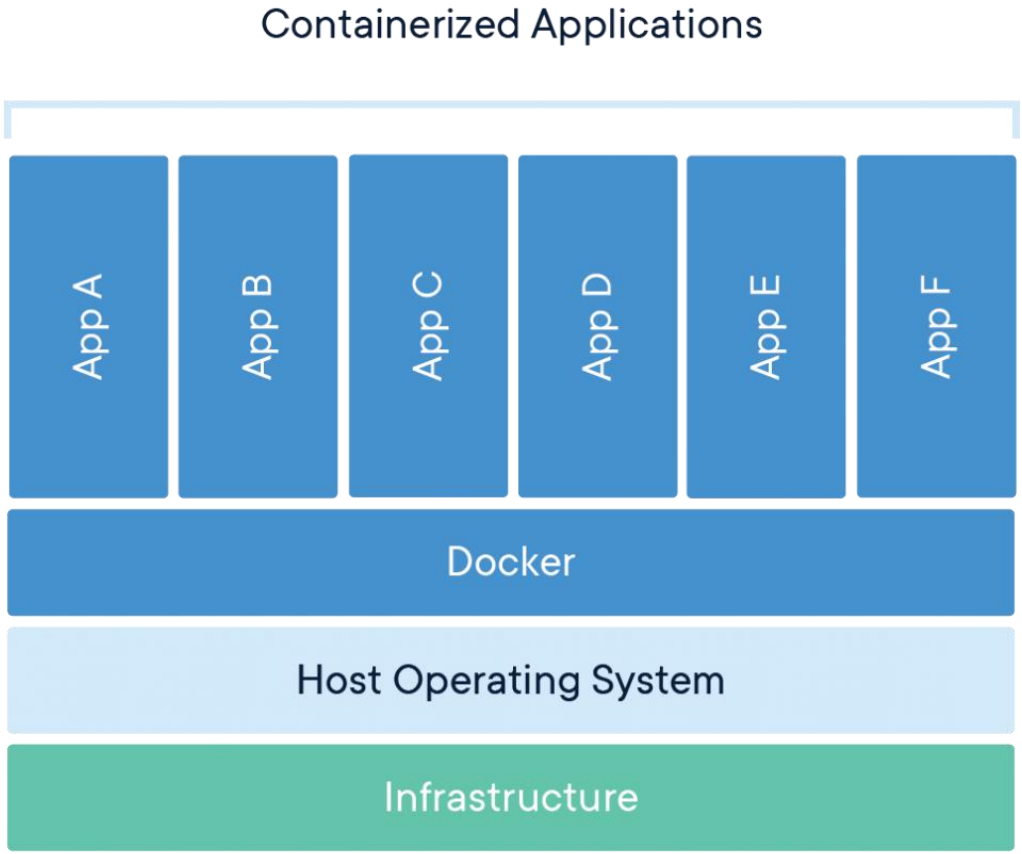
도커는 리눅스의 응용 프로그램들을“프로세스 격리”기술을 사용하여
컨테이너 형태로 실행하고 관리하는 오픈 소스 프로젝트이다.

특히 Linux 커널의 cgroup 및 네임스페이스로 알려진 기본 컴퓨팅 개념을
활용하여 개발되었기에 기술적 기반은 Linux에 있다고 볼 수 있다.




## Docker와 Linux Container

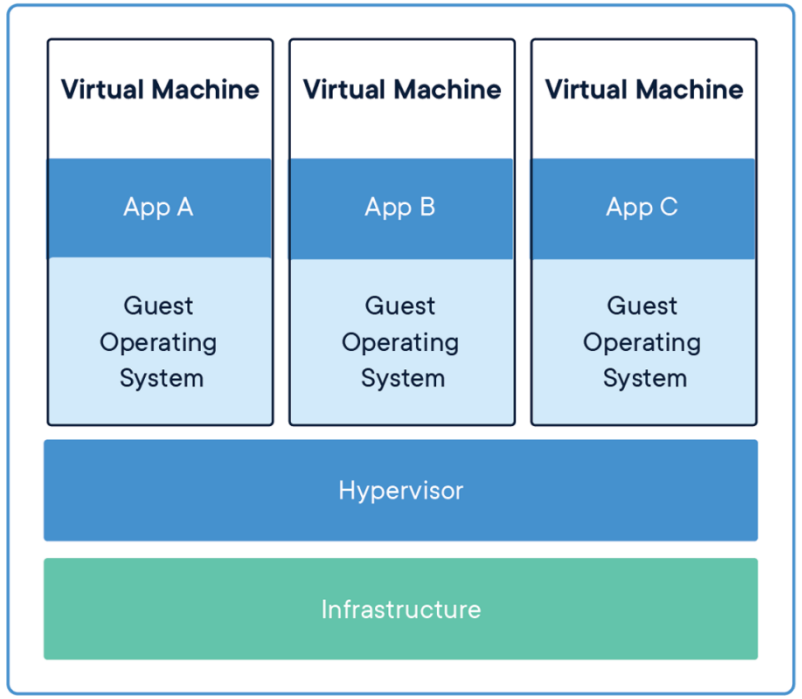
도커의 시작은 LXC 기술을 기반으로 구축되었으나, 현재는 종속 관계를 벗어났다. LXC는 경량화된 가상화 방법으로 유용하게 사용되었으나 사용자에겐 그 이상의 좋은 경험을 제공하진 못하였고 이는 도커가 계층화를 시키고 어플리케이션을 세분화하여 개별적인 프로세스를 통해 수행할 수 있도록 제공하여 LXC를 계승하며 도커만의 큰 차이점을 만들어냈다.



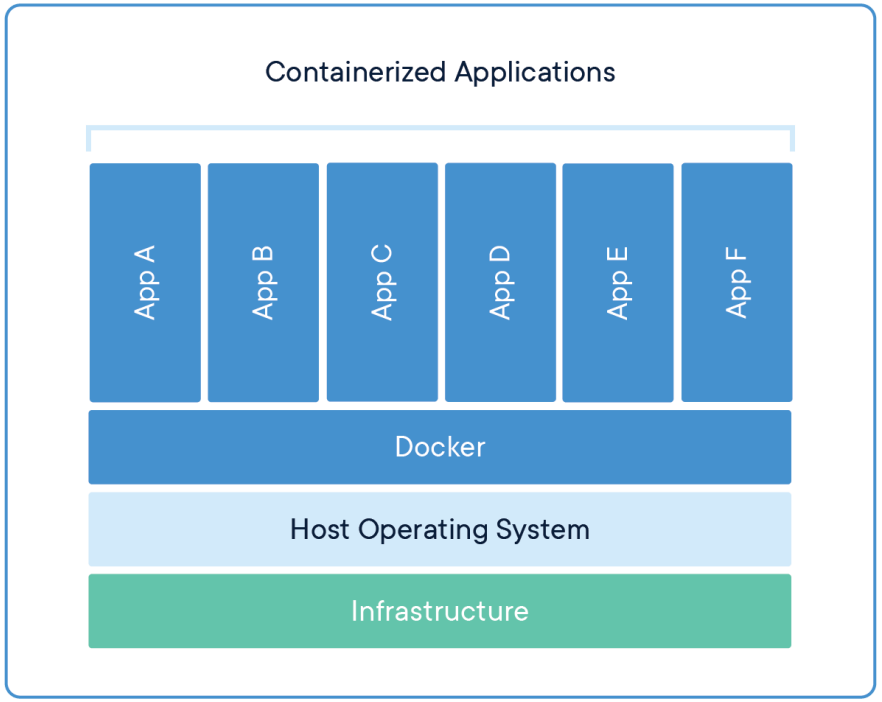

## Docker Container

도커에서 컨테이너는 표준화된 소프트웨어 단위를 의미한다.
소프트웨어의 실행에 필요한 모든 것을 포함하는 완전한 파일 시스템 안에 감싼다.
여기에는 코드, 런타임, 시스템 도구, 시스템 라이브러리 등 서버에 설치되는 무엇이든 아우른다.
이는 실행 중인 환경에 관계 없이 언제나 동일하게 실행될 것을 보증한다.

Vitual Machine




Docker Container

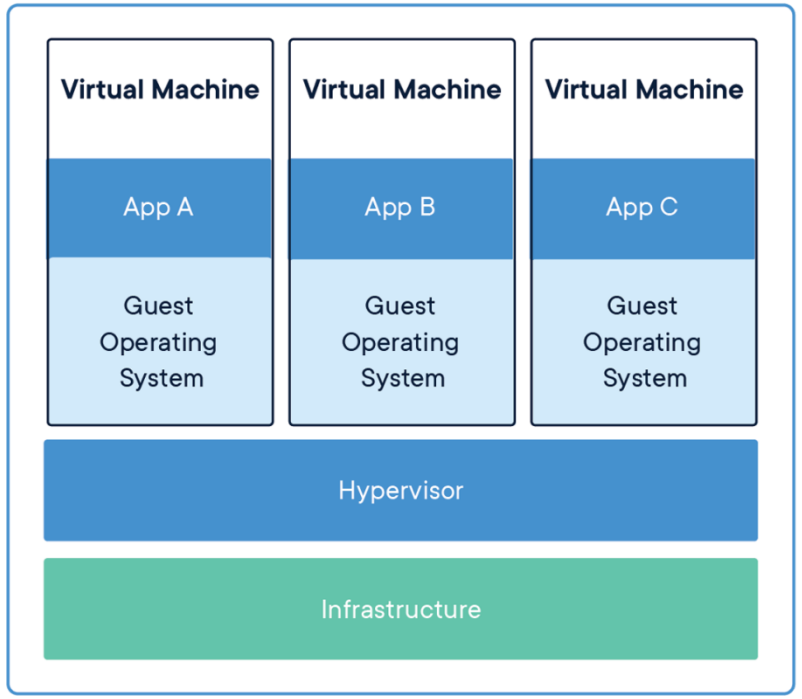



# Virtual Machine과 Docker Container의 차이점

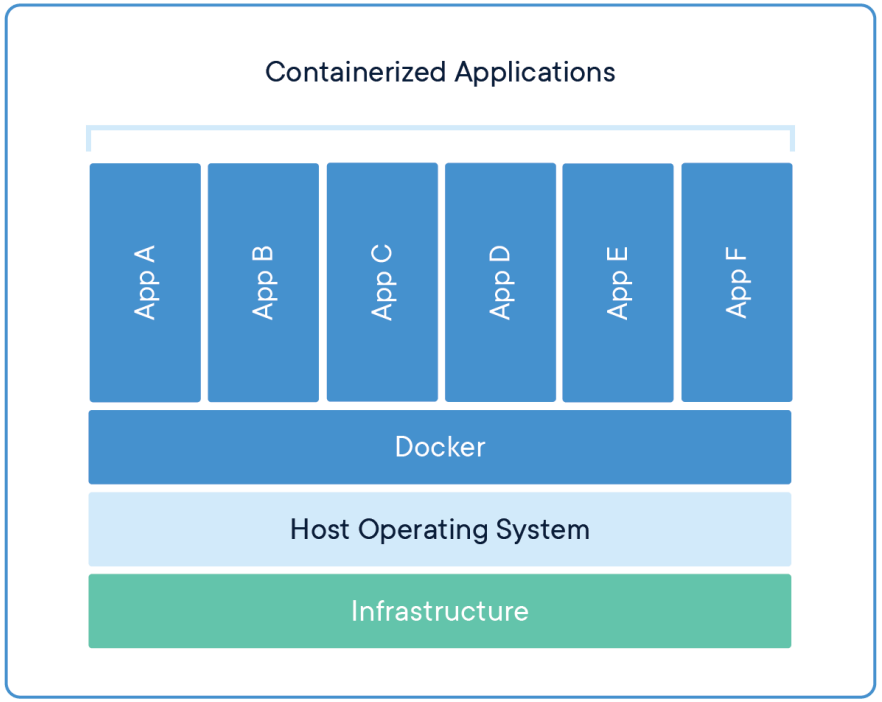
## VM(Virtual Machine)

VM은 하나의 서버를 여러 서버로 전환하는 물리적 하드웨어의 추상화.
Hypervisor를 사용하면 단일 시스템에서 여러 VM을 실행할 수 있다.
각 VM에는 운영 체제, 애플리케이션, 필요한 바이너리 및 라이브러리의 전체 사본이 포함되며 수십 GB를 차지하며 따라서 부팅 속도가 느릴 수도 있다.

Vitual Machine



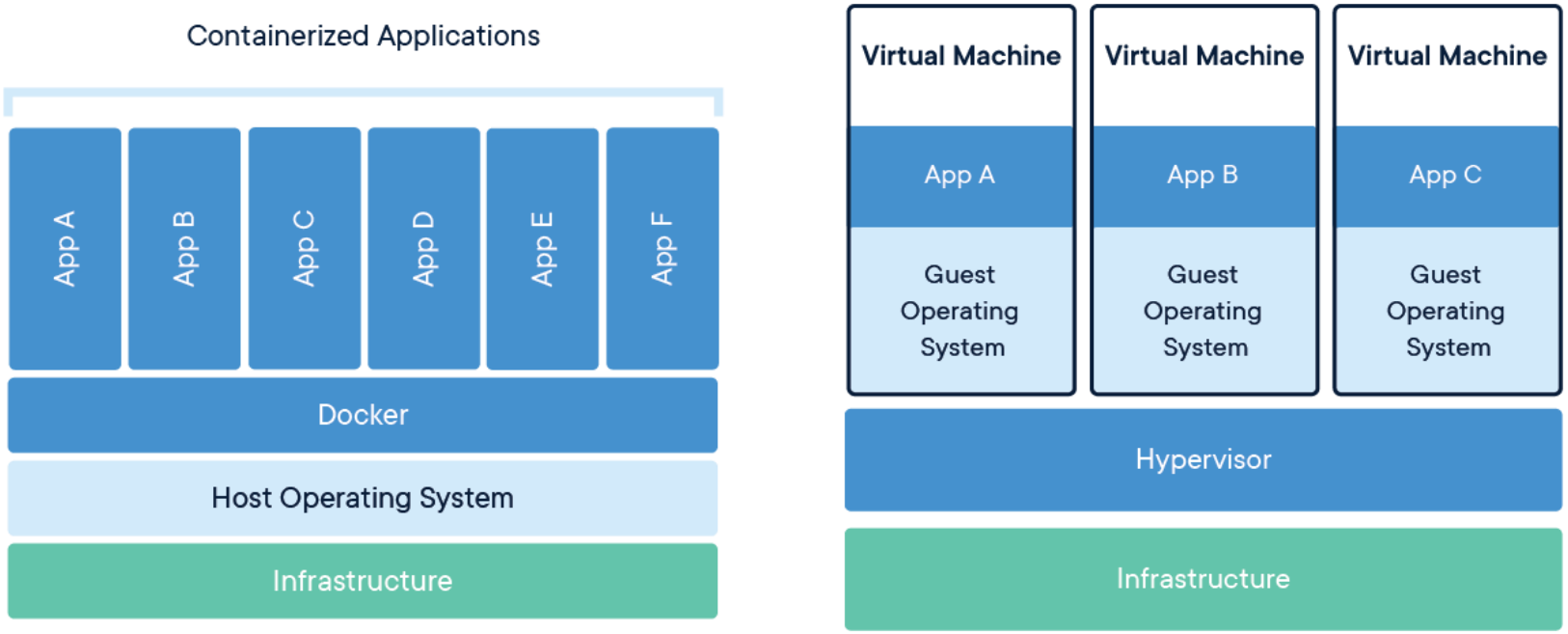

Docker Container




# Virtual Machine과 Docker Container의 차이점

## Docker Container

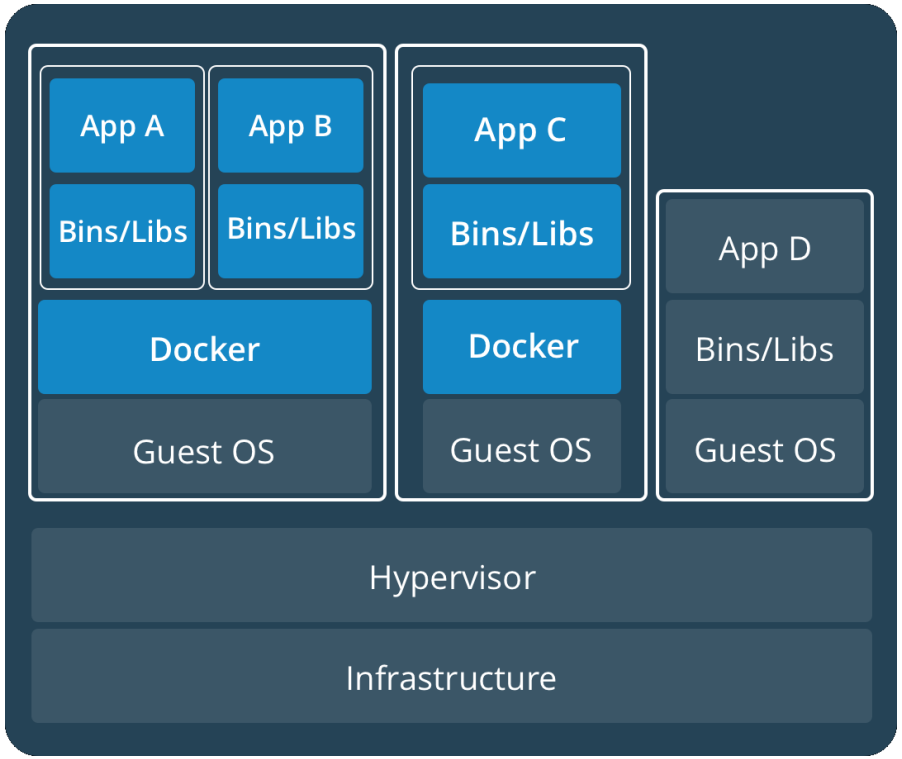
컨테이너는 코드와 종속성을 함께 패키징하는 앱 계층의 추상화이다.
여러 컨테이너가 동일한 컴퓨터에서 OS 커널을 다른 컨테이너와 공유 할 수 있으며,
각 컨테이너는 사용자 공간에서 격리 된 프로세스로 실행된다.
컨테이너는 VM보다 적은 공간을 차지하고 (이미지는 일반적으로 수십 MB.)
더 많은 애플리케이션을 처리 할 수 있으며 더 적은 수의 VM과 운영 체제를 필요로 한다.




# Virtual Machine과 Docker Container의 혼합

분명 컨테이너는 VM에 비해 아주 많은 이점을 지니고 있다.
이는 Scale Out을 통한 서버 관리에서도 특출난 이점을 지니고 있으며 이는 곧 서버를 운영하는 비용과 직결된다. 한 마디로 VM에 비해 보다 쉽고, 보다 경제적이다는 것이다.

그러나 모든 환경에서 컨테이너가 VM에 비해 장점이 있는 것은 아니다.
보안적인 측면에선 VM이 특출나며 컨테이너의 멀티 OS에 대한 문제도 해결되며
실제로 Docker측에선 VM과의 혼용으로 앱 배포 및 관리의 유연성이 제공된다고 한다.

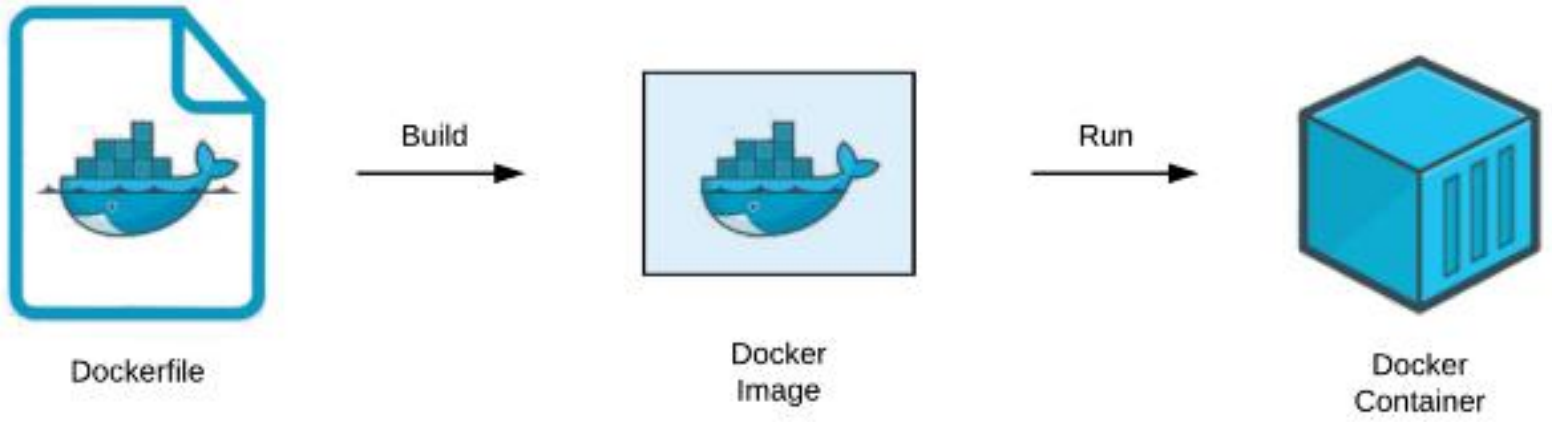



## Virtual Machine과 Docker Container의 혼합

일반적인 혼합 흐름은 다음과 같다.

베어메탈에 Host OS를 설치하고 그 위엔 경량화된 Guest OS를 설치한 여러대의 VM을 올린다. 그리고 개별 VM에 Docker-Engine을 올려 3중첩 시킨 환경을 구축할 수 있다.

이렇게 되면 멀티 OS 이슈와 보안적인 측면에서의 VM의 강점을 가져갈 수 있고 VM의 느리고 비싼 단점을 보완하며 새로운 장점을 지닌 환경이 구축된다.

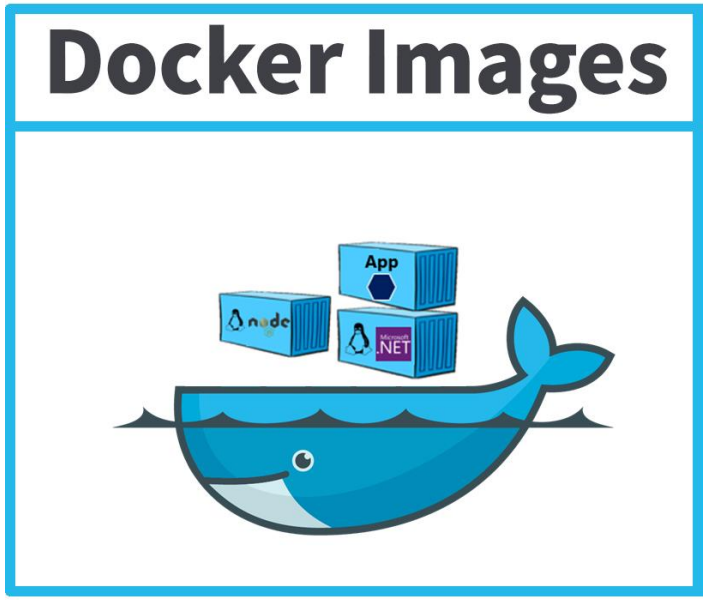


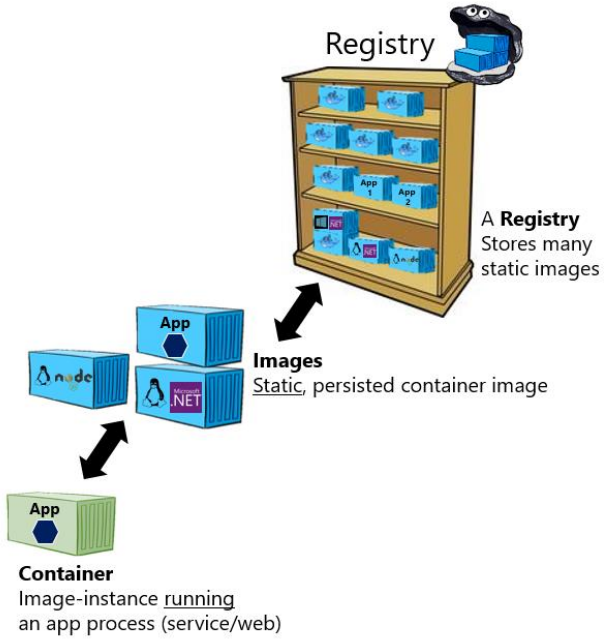

## Docker image

도커 이미지는 VM의 이미지와 비슷한 역할을 한다.
이미지는 컨테이너 실행에 필요한 파일과 설정 값들을 포함하고 있는 것으로서
상태 값을 가지거나 변하지 않는다(Immutable).

컨테이너는 이러한 이미지를 실행한 상태이며 추가되거나 변경이 되는 것은 이미지가
아닌 컨테이너에 반영이 되는 것이다.

이러한 이미지는 DockerFile을 통해 생성이 가능하며 이때 버전 관리가 이뤄진다.



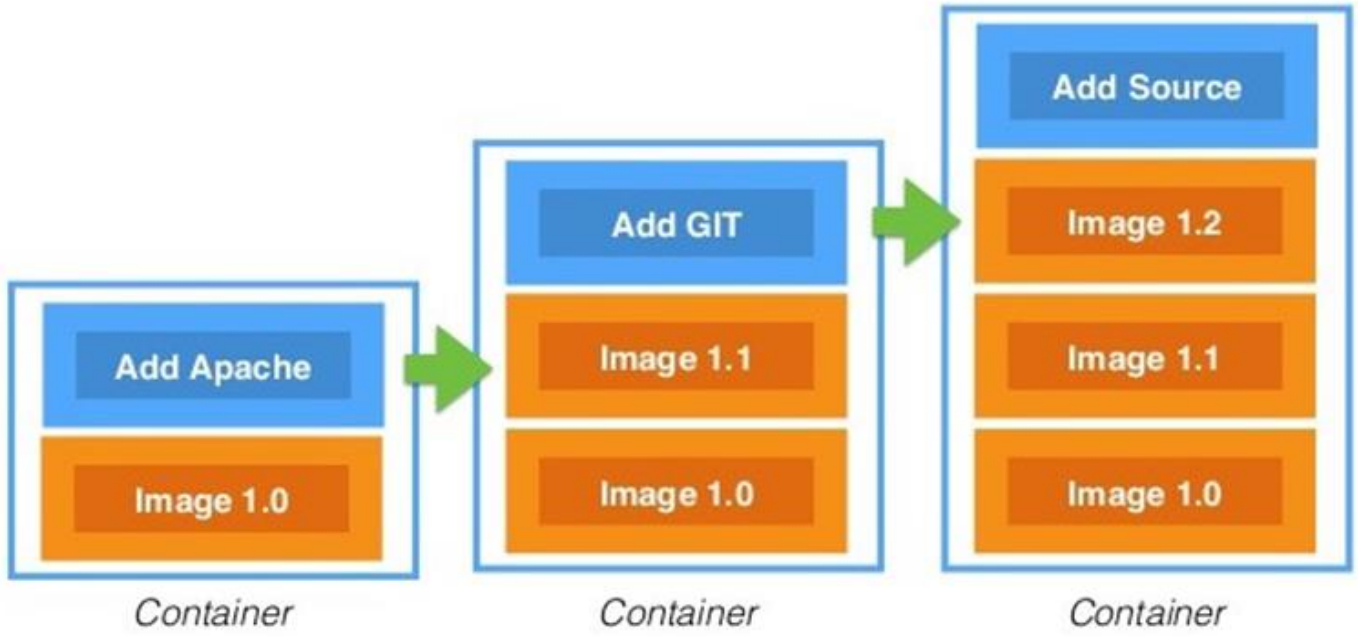




## Docker image

도커 이미지는 일반적으로 Docker Hub와 같은 공용 저장소를 통해 배포되지만
개인적으로 구성한 Docker Registry를 통해서도 저장 및 배포가 가능하다.
이미지는 OS, DB, Web Server, WAS등 다양한 종류의 이미지들이 있다.

예를 들면 CentOS 이미지는 CentOS를 실행하기 위한 모든 파일을 가지고 있고
MySQL 이미지는 debian OS를 기반으로 MySQL을 실행하는 데 필요한 모든 파일과 명령어,
포트 정보 등을 지니고 있으며 좀 더 복합적인 예로 GitLab은 CentOS 기반으로
Ruby, go, DB, redis, GitLab 소스, Ngnix 등을 가지고 있다.

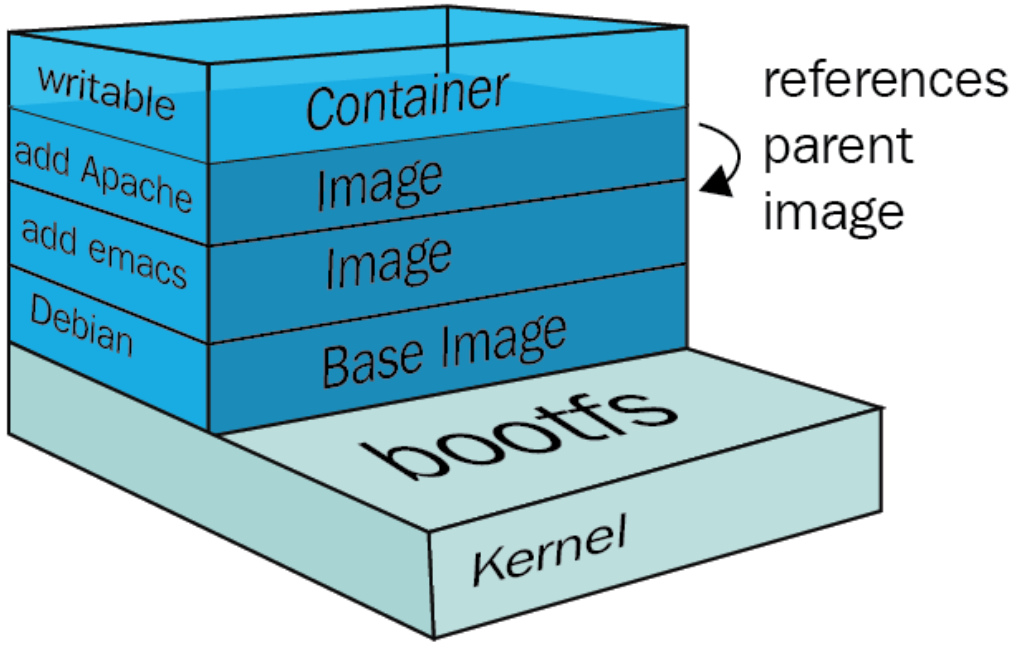
컨테이너는 이러한 이미지를 복합적으로 사용하여 구동이 된다.




## Docker Layer

도커 이미지는 모든 정보를 담고 있으며 불변하기 때문에 수정할 수가 없다.
만약 기존의 이미지에 파일 하나가 추가해야 한다면 이에 맞는 새로운 이미지를 받아야 할 것이다. 그러나 겨우 1MB도 되지 않는 파일 하나 때문에 수백MB의 이미지를 새로 받아야 한다는 것은 매우 비효율적이다.

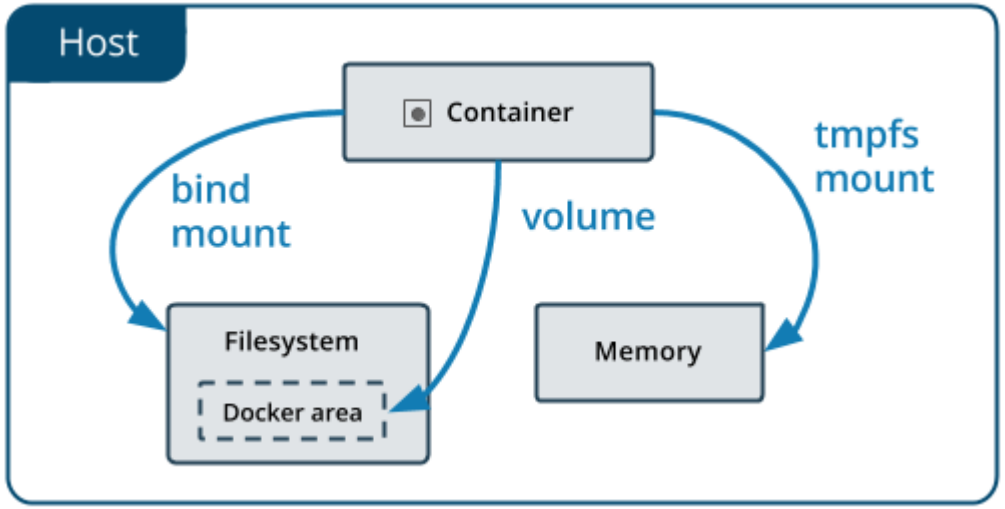
이러한 문제를 해결하기 위해 나온 것이 Docker Layer이다.




## Docker Layer

도커 레이어는 유니온 파일 시스템을 이용하여 여러 개의 레이어를 하나의 파일 시스템으로 사용할 수 있게 한다.
따라서 하나의 파일 시스템에 여러 이미지들이 Read Only 레이어로 구성이 되며 파일이 추가되거나 수정되면 새로운 레이어가 생성된다.

컨테이너 생성 시에도 레이어 방식을 사용하는 데 이때 컨테이너는 Read-Write 레이어를 추가하여 컨테이너 실행 중 생성되거나 변경되는 내용은 컨테이너 레이어에 저장된다.

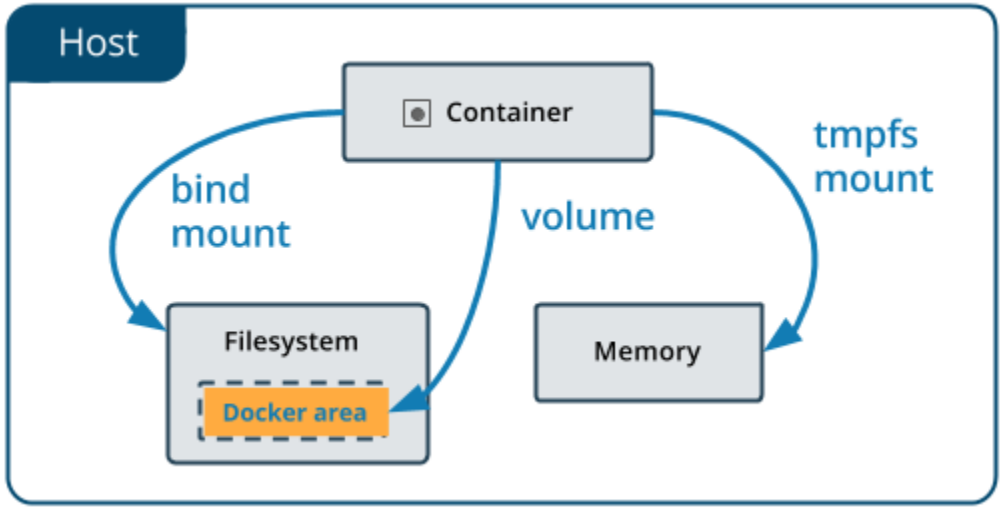



## Docker Volume과 Bind Mount

컨테이너가 실행되며 생성되거나 변경된 내용은 디폴트로 컨테이너 레이어에 저장이 된다.
하지만 컨테이너 레이어에 쓰여진 데이터는 컨테이너의 생명주기 종료와 함께 삭제가 된다.

도커에서 실행될 많은 어플리케이션들은 컨테이너의 생명주기와 별개로 데이터를 영속적으로 저장해야 하기 때문에 도커에선 이러한 기능을 별도로 제공하고 있다.

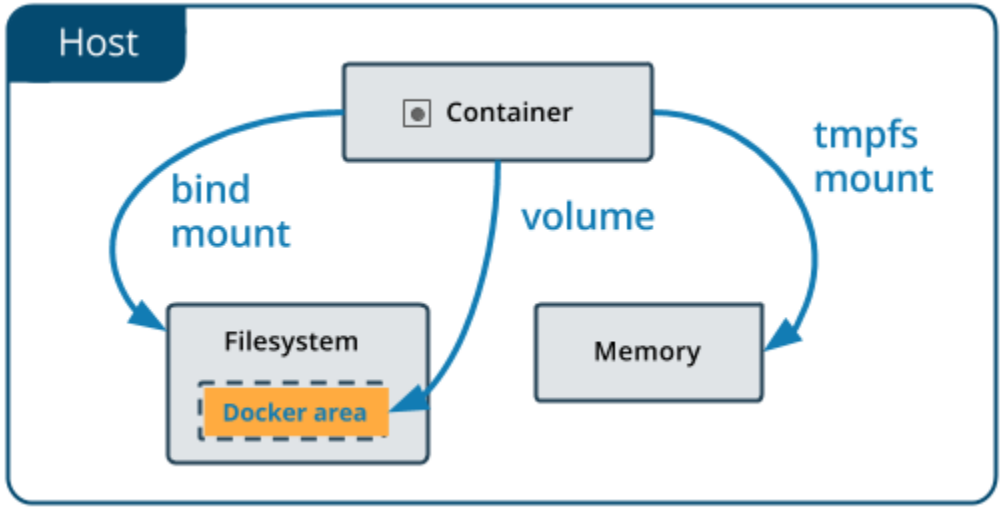
바로 도커 볼륨과 바인드 마운트이다.




## Docker Volume

볼륨은 Docker에서 관리하는(/var/lib/docker/volumes/ on Linux) 파일 시스템에 저장된다.
즉 Docker가 아닌 프로세스가 이 부분을 수정하여선 안된다.
그리고 데이터를 유지하는 데 있어 도커에서 가장 권장하는 방법이다.

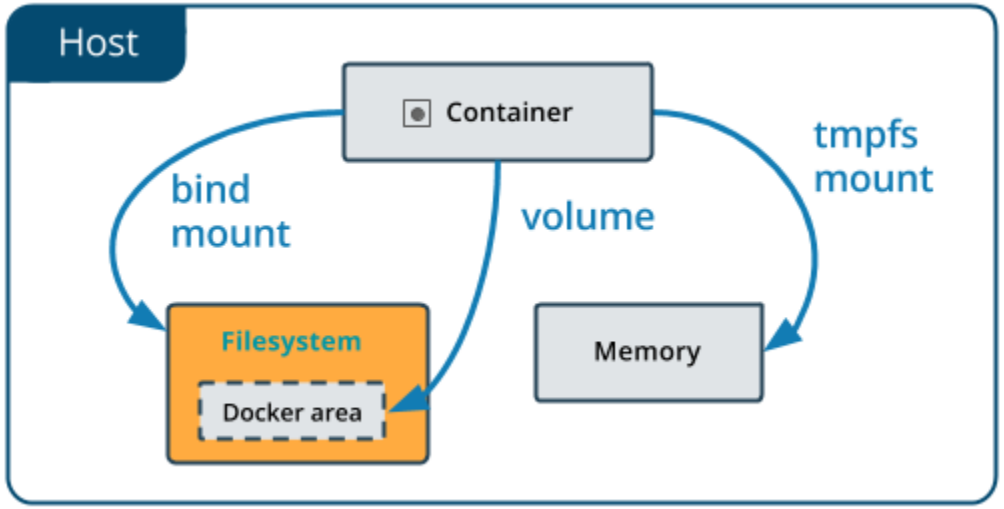
볼륨은 명령어를 통해 명시적으로 생성하거나 컨테이너 서비스 혹은 생성 중에 볼륨을 생성할 수 있다.
또한 볼륨은 여러 컨테이너에서 마운트하여 사용할 수 있으며 특정 컨테이너가 더 이상 볼륨을 사용하지 않더라도 볼륨은 자동으로 삭제되지 않으며 명령어를 통해 삭제할 수 있다.




## Docker Volume

볼륨은 바인드 마운트에 비해 여러 장점을 지니고 있다.

1. 백업 또는 마이그레이션이 쉬움.
2. Linux와 Windows 컨테이너 모두 작동.
3. 여러 컨테이너 간에 안전하게 공유.
4. Host OS가 아닌 원격 Host나 클라우드 공급자를 통해 저장하고 암호화 할 수 있음.
5. Docker Desktop 볼륨은 Mac 및 Windows 호스트의 바인드 마운트보다 높은 성능을 제공 한다.

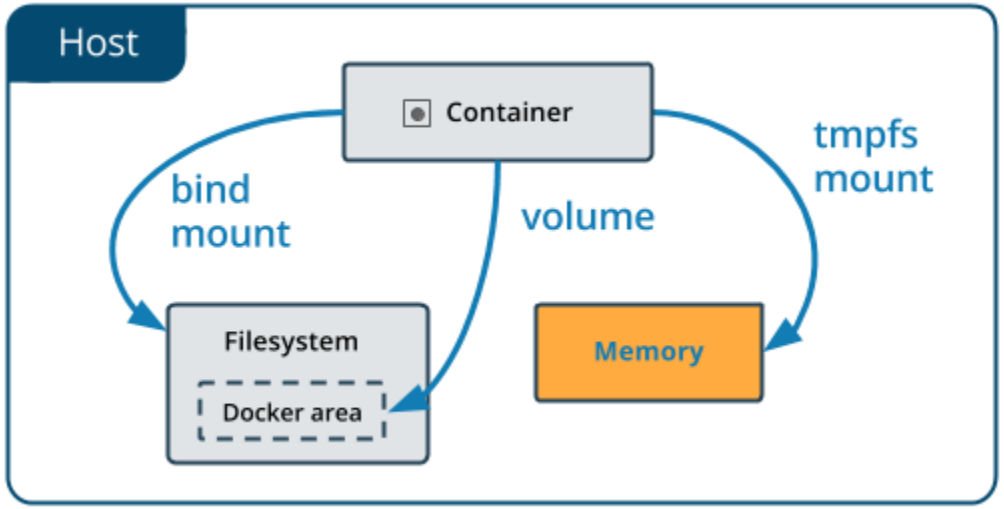



## Bind Mount

바인드 마운트는 볼륨에 비해 기능은 제한적이지만 성능이 매우 뛰어난 편이다.
바인드 마운트를 사용하면 호스트 시스템의 파일 또는 디렉토리가 컨테이너에 마운트된다.
따라서 Docker 호스트에 미리 존재할 필요가 없으며 필요 시 생성된다.

다만 이러한 특징으로 인해 컨테이너에서 호스트 시스템의 파일에 영향을 줄 수 있다는
보안적인 이슈가 존재한다.

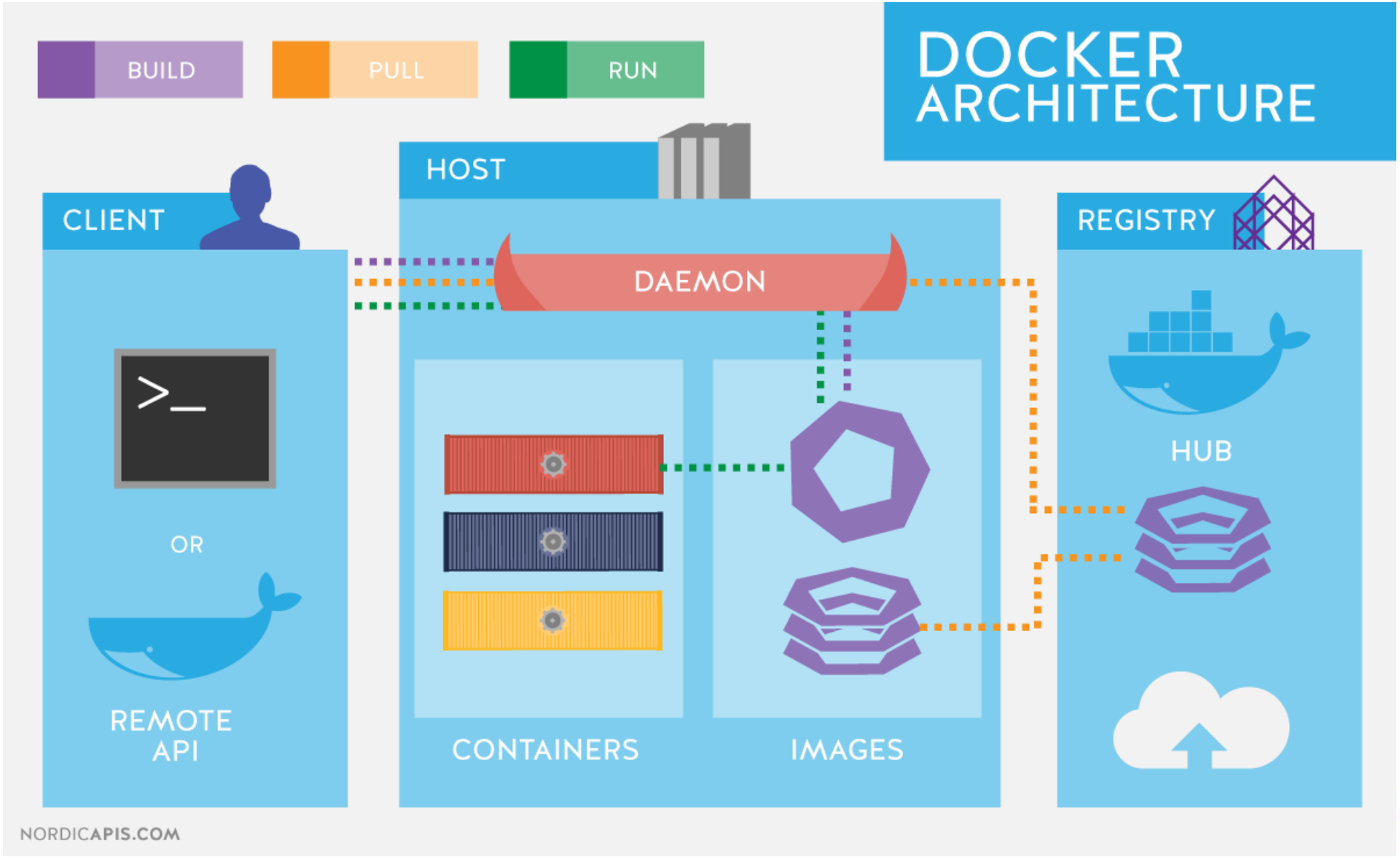
따라서 제한적인 경우에 한해서만 바인드 마운트가 권장된다.




## Tmpfs Mount

예외적인 사항으로 메모리에 저장하는 Tmpfs Mount를 사용하는 경우도 있다.
이는 Linux에서만 사용이 가능하며 컨테이너간 마운트 공유는 불가능하다.

주로 비 영구적이거나 민감한 정보를 저장하기 위한 용도의 컨테이너를 사용 시 사용을 권장하고 있다.


BUILD
PULL
RUN
DOCKER ARCHITECTURE
HOST
CLIENT
REGISTRY
DAEMON
>_
OR
REMOTE API
CONTAINERS
IMAGES
HUB
NORDICAPIS.COM


## 컨테이너 오케스트레이션

컨테이너 오케스트레이션은 컨테이너의 배포, 관리, 확장, 네트워킹을 자동화하는 것을 의미하며 Docker와 MSA 기반의 관리가 활발해지며 자동화에 대한 필요성이 대두되며 컨테이너 오케스트레이션이 등장하게 되었다.

만약 컨테이너가 다운 된다면 다른 컨테이너를 다시 시작시켜야 한다.
이러한 문제를 시스템이 자동으로 처리 할 수 있다면 많은 비용을 절감할 수 있을 것이다. 오케스트레이션을 쓰고자 하는 이유가 바로 이것이다.

# kubernetes

## Kubernetes란?

쿠버네티스는 컨테이너화 된 어플리케이션을 자동으로 배포, 확장 및 관리를 해주는 컨테이너 오케스트레이션 시스템이다.

주로 줄여서 케이(에이)츠(K8s) 혹은 큐브(kube)라고 부르기도 한다.

Google에 의해 개발되었고 2014년에 오픈 소스로서 공개되었으며 그 노하우와 수 많은 기업들의 참여로 현재 컨테이너 오케스트레이션 플랫폼으론 사실상 표준이 되었다.

# kubernetes

## Kubernetes란?

쿠버네티스는 단순한 컨테이너 플랫폼이 아닌 MSA 기반의 비즈니스 시스템과 클라우드 플랫폼을 지향하는 조직에 더 적합하다.

또한 오토 스케일링과 유효성, 자가 치유와 같이 매우 강력하고 고급적인 기능들이 많이 있어 이를 활용할 수 있는 기술과 지식이 있는 조직에겐 큰 효과를 발휘하지만 특유의 복잡성으로 인해 대부분의 사람들에겐 어려움이 있을 수 있다.

따라서 여러 기능에 대해 검토하고 조직에 적합한 지 신중히 고려할 필요가 있다.

# Kubernetes의 주요 기능


**자동화된 롤아웃과 롤백** : 어플리케이션의 변경 시 점진적 롤아웃과 문제 발생 시 롤백 제공.

**서비스 디스커버리와 로드 밸런싱** : 파드에게 고유 IP와 집합간에 DNS를 주고 로드 밸런싱 제공.

**스토리지 오케스트레이션** : 로컬 스토리지, 퍼블릭 클라우드 등에 시스템을 자동으로 마운트.

**시크릿 구성 관리** : 사용자의 이미지를 다시 빌드하거나 시크릿을 노출하지 않고 구성하고 배포.

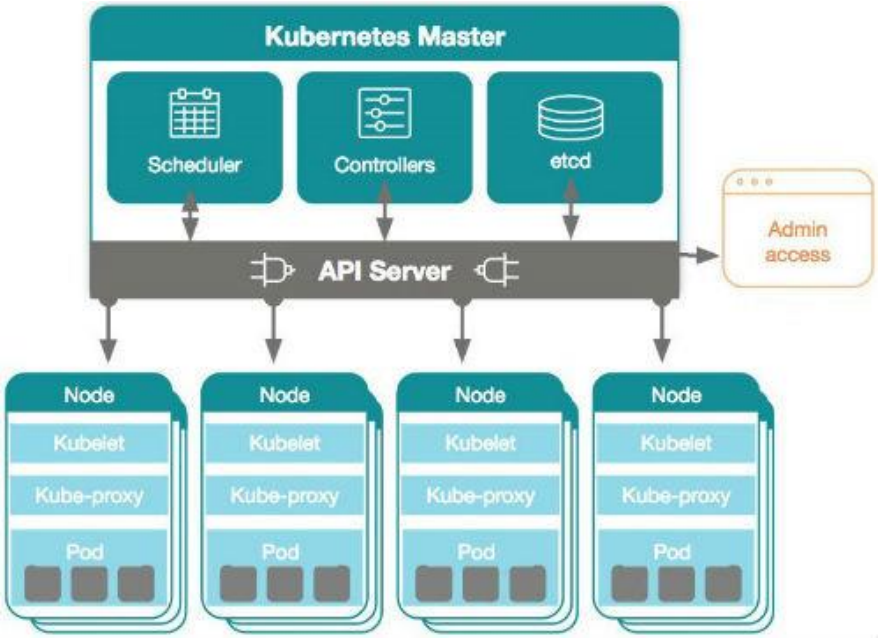
**자동 빈 패킹(bin packing)** : 리소스의 요구 사항과 제약 조건에 따라 컨테이너를 자동으로 배치.

**배치 실행** : 배치와 CI 워크로드 관리하고 실패한 컨테이너의 교체 기능 제공.

**IPv4/IPv6 이중 스택** : 파드와 서비스에 IPv4와 IPv6 주소를 모두 할당 가능.

**수평적인 스케일링** : CPU 사용량에 따라 파드 개수를 자동으로 스케일하는 기능 제공.

**자가 치유** : 오류가 발생한 컨테이너를 재시작하고, 노드가 죽었을 때 스케줄링하여 컨테이너를 교체하고, 상태 체크에 응답하지 않는 컨테이너를 제거하며, 서비스가 제공될 준비가 될 때까지 클라이언트에 해당 컨테이너를 알리지 않는다.

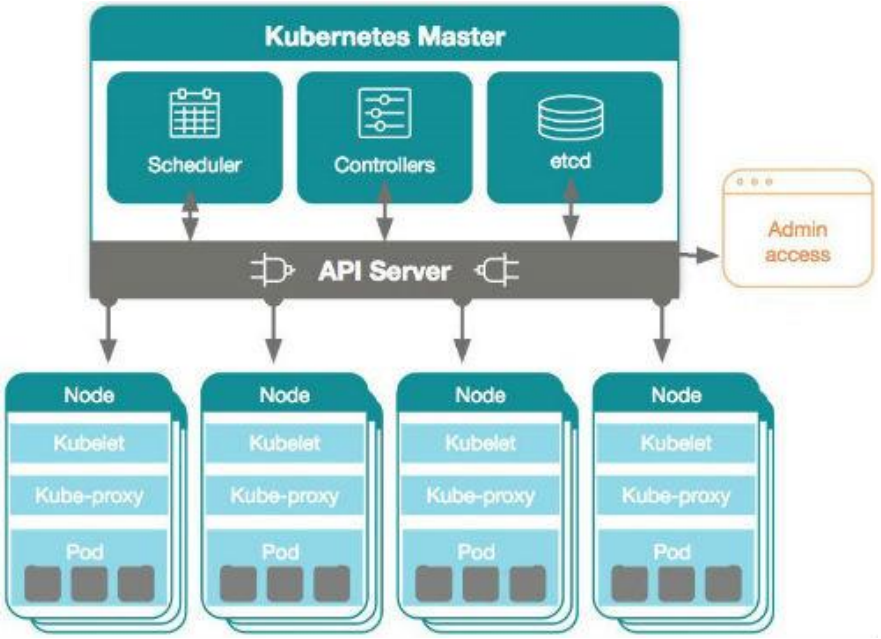



## Kubernetes의 구조

쿠버네티스는 기본적으로 클러스터라는 단위를 사용한다.
이 클러스터는 쿠버네티스의 여러 리소스를 관리하기 위한 집합체를 의미한다.

전체 클러스터를 관리하는 컨트롤러로써 마스터(Master)가 존재하고 컨테이너가
배포되고 어플리케이션이 실행되는 가상 혹은 물리적 머신인 노드(Node)가 존재한다.

노드는 한 개 이상의 파드(Pod)를 지니고 있으며 파드는 1개 이상의 컨테이너가 모인
집합체를 의미하며 이 파드는 결합성이 강한 컨테이너들 끼리 묶이게 된다.




## Kubernetes의 구조

쿠버네티스는 크게 마스터와 노드 두 개로 분리가 되며
마스터는 쿠버네티스의 환경 설정을 저장하고 전체 클러스터를 관리하는 역할을 맡고 있고 노드는 마스터에 의해 명령을 받고 실제 워크로드를 생성하여 서비스한다.

노드는 Kubelet이 API Server를 통해 마스터와 통신을 하게 되며 이 API Server는 REST API로 제공하고 처리한다.

즉 쿠버네티스는 단순하게 생각하면 중앙(Master)에서 API Server를 두고 각 서버(Node)를 에이전트(Kubelet)와 통신하고 이를 수행하는 구조라 볼 수 있다.

# 감사합니다.

- 출처

## 도커

https://www.docker.com/
https://docs.docker.com/
https://devopedia.org/docker
https://medium.com/@darkrasid/docker%EC%99%80-vm-d95d60e56fdd
https://www.redhat.com/ko/topics/containers/what-is-docker
https://www.44bits.io/ko/post/easy-deploy-with-docker
https://docs.microsoft.com/ko-kr/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries

## 쿠버네티스

https://kubernetes.io/ko/docs/concepts/overview/what-is-kubernetes/
https://bcho.tistory.com/1255?category=731548
https://subicura.com/2019/05/19/kubernetes-basic-1.html#%EC%BF%A0%EB%B2%84%EB%84%A4%ED%8B%B0%EC%8A%A4%EB%9E%80