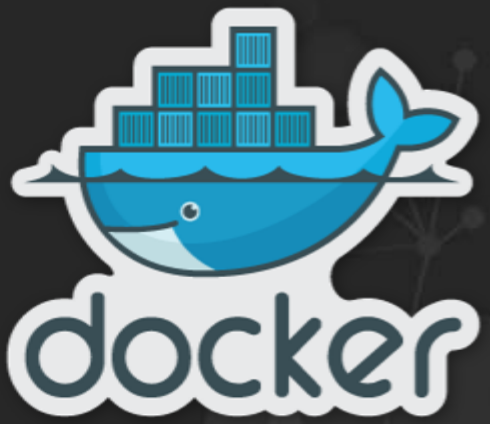

openmaru
APM
Container Technology
(Cloud Native System)

Container 시대를 향하여

openmaru APM

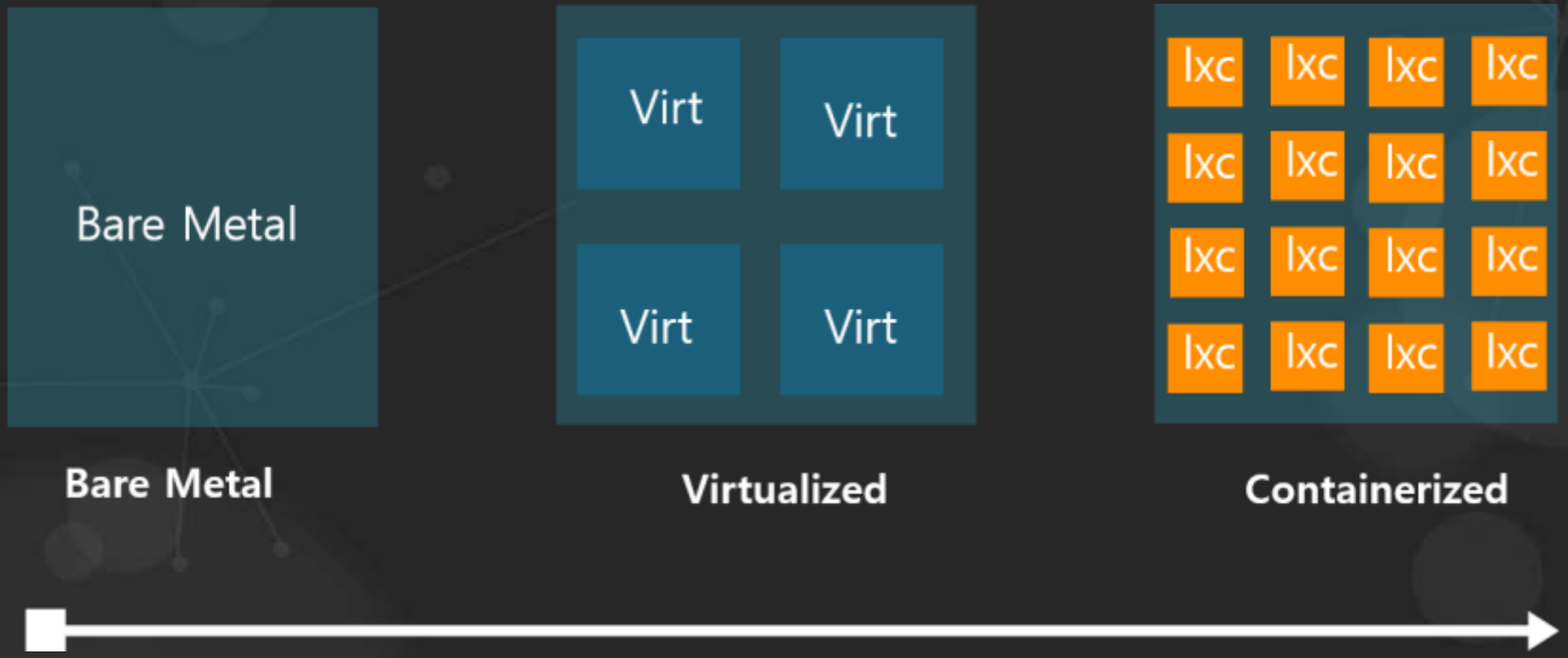


1. 컨테이너란 무엇인가? 가상화와의 차이는 무엇인가?
   - ☐ 기존의 가상화와 컨테이너는 무엇이 다른지?
   - ☐ 컨테이너에는 어떤 구조로 되어 있는 건가?
   - ☐ Linux OS만 있으면 컨테이너를 바로 사용하나?
2. 컨테이너 적용시의 기대효과?
   - ☐ 컨테이너를 사용하면 무엇이 좋은 것인가?
   - ☐ 컨테이너는 무조건 좋은 건가? 단점은 무엇인가?
   - ☐ 컨테이너를 적용하기에 좋은 시스템과 그렇지 않은 시스템은?
3. 컨테이너를 운영 환경에서 사용하려면? 최근 화제인 컨테이너 플랫폼이란?
   - ☐ 컨테이너기반 시스템을 구축 하는 방법은?
   - ☐ 컨테이너는 실제 운영환경에서 사용할 만큼 성숙된 기술인가?
   - ☐ 컨테이너 플랫폼은 무엇이 있으며 향후 발전 방향은?

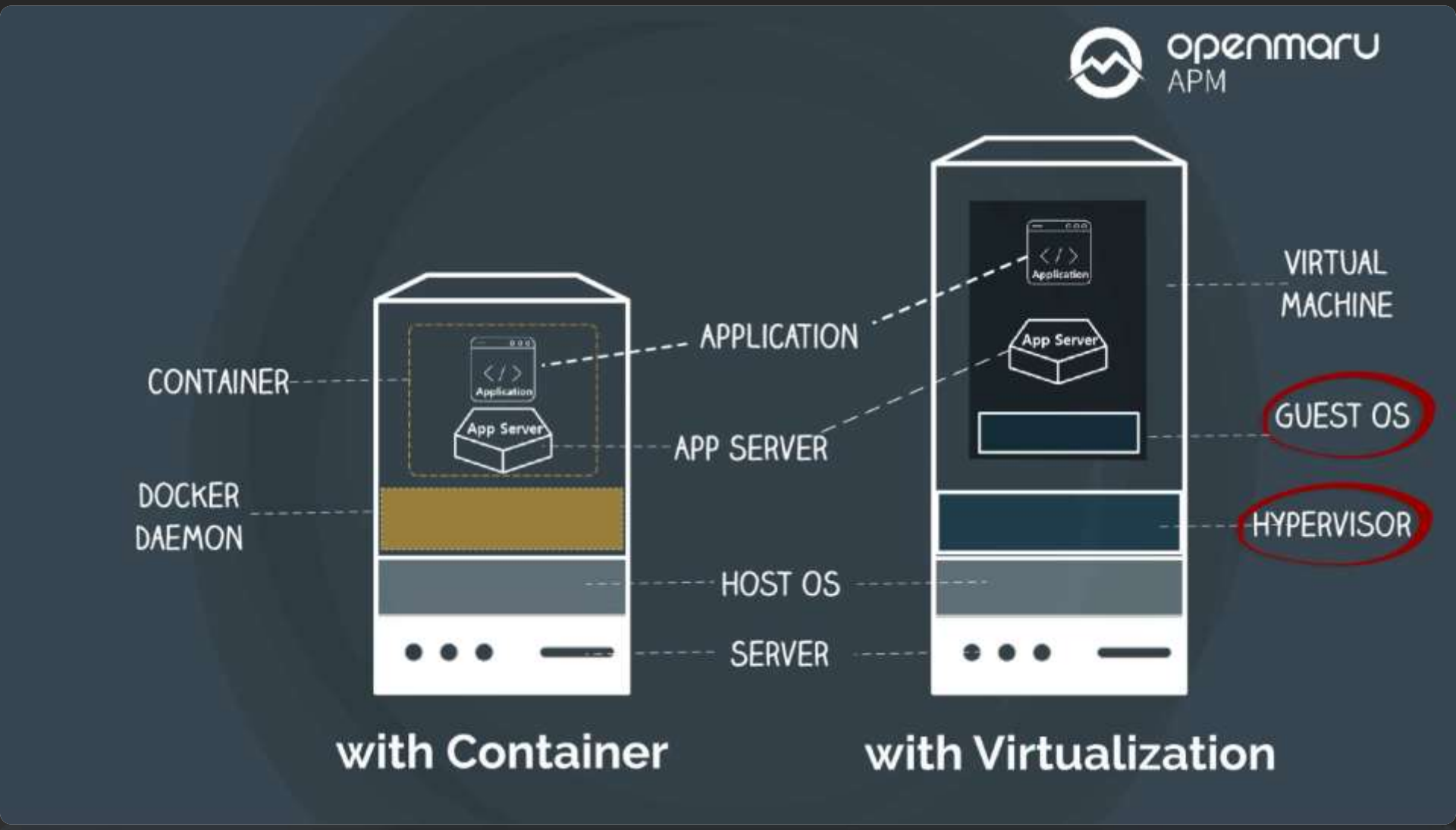
Before Container

After Container

Evolution of Infrastructure Architectures



After Container



하드웨어 가상화와 OS 가상화 비교

- 서버용 가상화 기술로 VMWare, RHV 나 KVM등의 가상머신모니터(VMM)/하이퍼바이저에 의한 "하드웨어 가상화"
- 하드웨어와 리소스를 가상머신에 할당하기 위해서는 하드웨어 전체가 가상화되어야 함

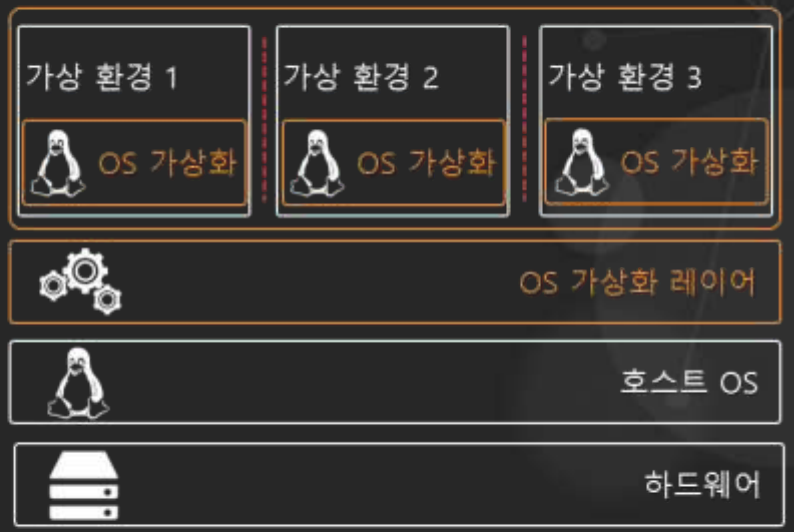
하이퍼바이저에 의한 하드웨어 가상화

호스트 OS와 게스트OS가 다를 경우도 가능

디스크/메모리 소비 큼

구성 자유도가 높은 가상화 기술

- 가상화라기보다는 격리된 가상적인 OS환경을 제공



OS 가상화

호스트OS와 게스트OS 동일

디스크/메모리 소비 작음

가볍고 휴대 성이 높은 가상화 기술

컨테이너 vs. 가상화

	컨테이너 형 가상화 (Docker)	하이퍼 바이저 형 가상화 (VMWare ESXi)	호스트 형 가상화 (Linux KVM)
가상 머신	OS를 호스트 OS와 공유하기 때문에 VM 마다 OS 설치를 할 필요는 없다	VM마다 OS 설치	VM마다 OS 설치
지원 OS	• Linux • Windows	• Windows, Linux • 일부 Unix도 지원	• Windows, Linux • 일부 Unix도 지원
부팅 시간	OS 설치 불필요하기 때문에 사용 시작까지의 시간이 매우 짧음	초기 구축 시에는 네트워크 OS 설치 등의 작업이 발생하기 때문에 이용 개시까지의 시간이 많이 소요	초기 구축 시에는 네트워크 OS 설치 등의 작업이 발생하기 때문에 이용 개시까지의 시간이 많이 소요
네트워크	호스트 측에 작성된 Docker 전용 NIC와 통신	• 네트워크의 생성이 가능 • VM에 임의의 숫자 vNIC를 부여 가능	• 네트워크의 생성이 가능 • VM에 임의의 숫자 vNIC를 부여 가능
자원	표준에서는 HDD 자원을 지정할 수 없다. CPU, 메모리에 대한 자원 할당 지정 가능	CPU, 메모리, HDD의 자원 할당을 지정	CPU, 메모리, HDD의 자원 할당을 지정
오버 헤드	컨테이너는 호스트 OS에서 보면 하나의 프로세스이며, 오버 헤드는 거의 없음	VM에서 기기까지의 액세스 경로를 하이퍼바이저 뿐이므로 호스트 형 가상화에 비해 오버 헤드가 적은	VM에서 기기까지의 액세스 경로가 다른 가상화 기술에 비해 길기 때문에 비교했을 경우에는 가장 오버 헤드가 높음

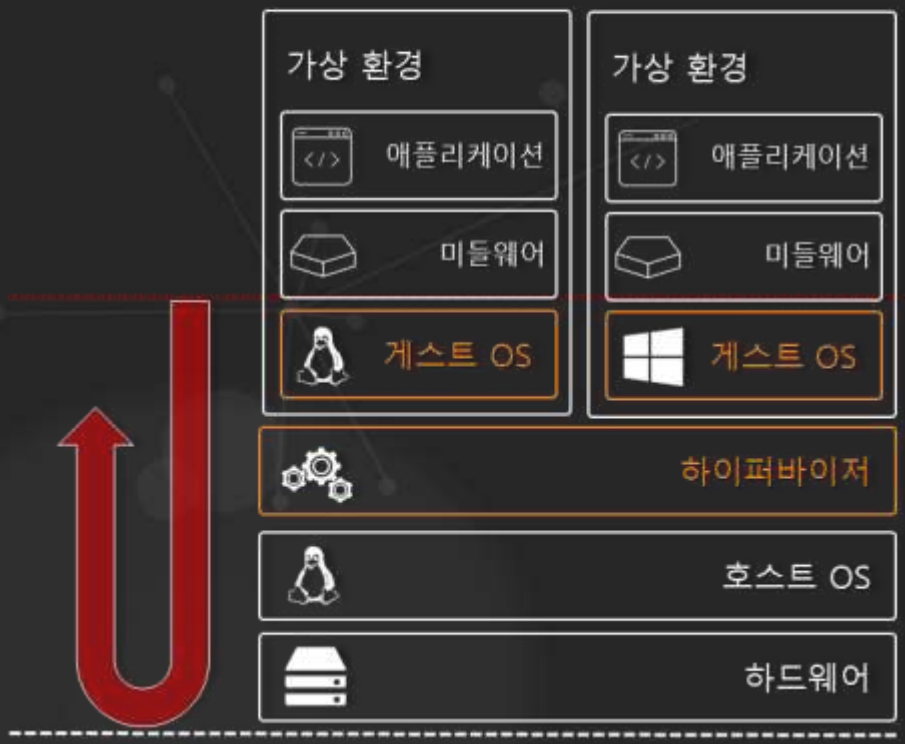
시작 시간 - Containers vs. VMs

- 하드웨어 가상화는 CPU, 메모리, 하드 디스크 등의 하드웨어를 가상화하고 있기 때문에 하드웨어 나 OS 부팅해야 부팅에 분 단위 시간 소요
- 컨테이너 형 가상화에서는 컨테이너 부팅 시 OS는 이미 시작하고 프로세스의 시작 만 할 초 단위 시간

오버헤드 - Containers vs. VMs

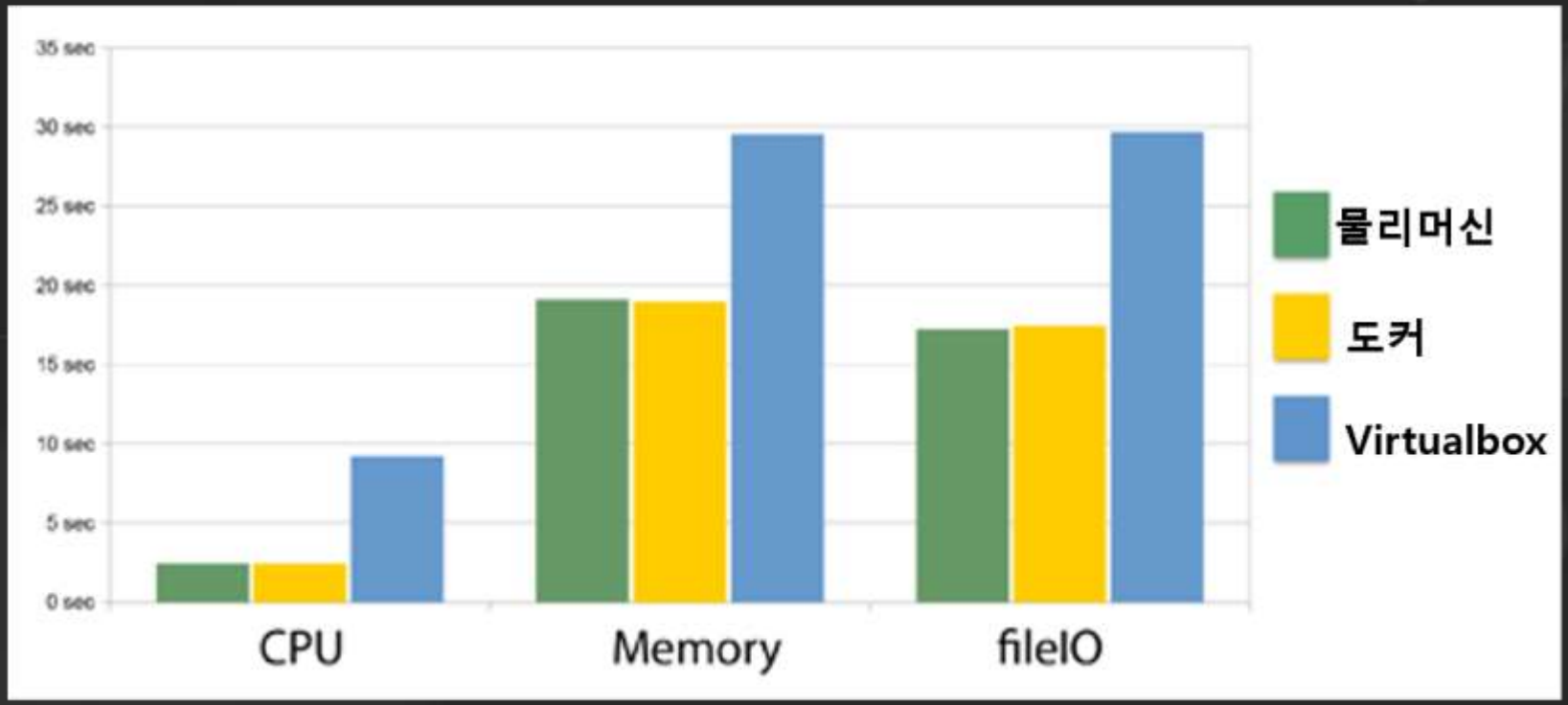
openmaru

- OS에서 응용 프로그램을 작동하는 경우, 하드웨어 가상화에서는 가상화 된 하드웨어 및 하이퍼바이저를 통해 처리하기 때문에 물리적 시스템보다 처리에 부가적인 시간 (오버 헤드)가 필요
- 컨테이너 형 가상화 커널을 공유하고 개별 프로세스가 작업을 하는 것과 같은 정도의 시간 밖에 걸리지 않기 때문에 대부분 오버 헤드가 없음



성능 - Containers vs. VMs

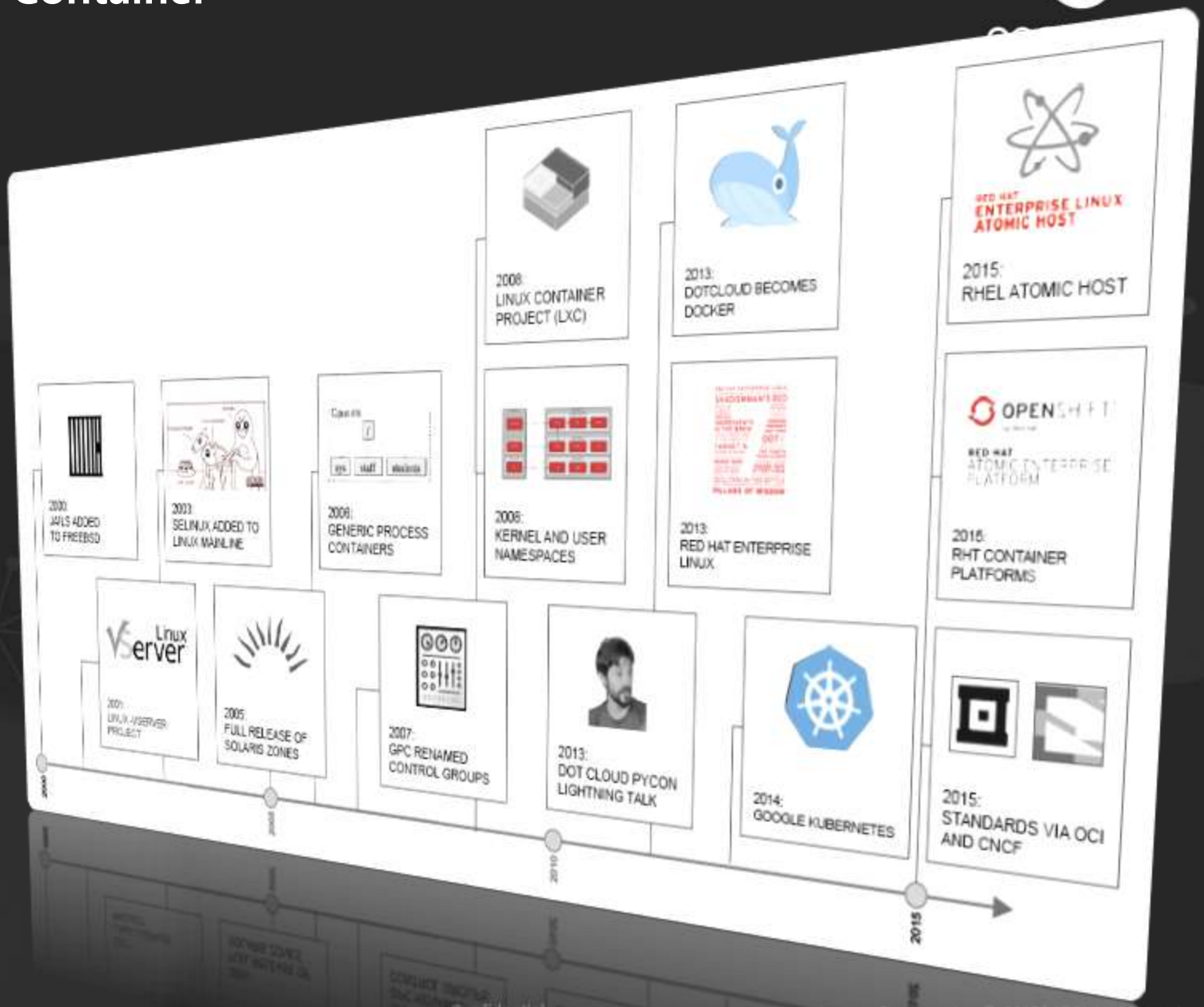
- "sysbench"라는 벤치 마크 도구를 사용하여 성능 측정
- 물리적 시스템과 컨테이너 형 가상화 성능은 모든 항목에서 거의 같은 결과
- 하드웨어 가상화는 메모리, 파일 IO는 약 2 배, CPU는 약 5 배의 시간
- 물리 머신과 비교해도 성능 저하가 거의 없음



컨테이너 (OS 가상화) 비교

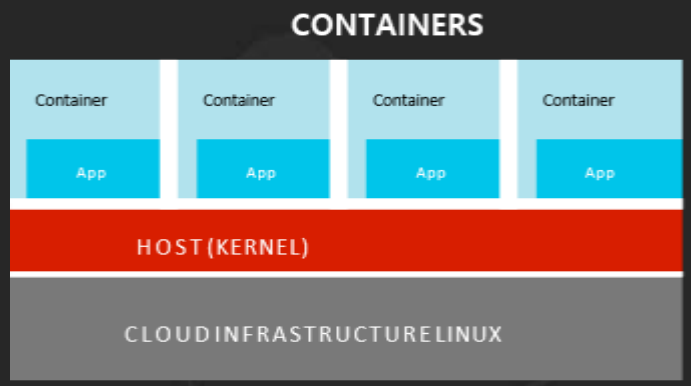
	LXD	Docker	LXC	OpenVZ	Virtuozzo
Release	2015	2013	2008	2005	2001
개발사	Canonical	Docker Inc.	IBM, Parallels, Canonical	OpenVZ Community	Parallels
지원OS	Linux	Linux, Windows, MacOS	Linux	Linux	Linux, Windows
가격	무상	무상	무상	무상	유상
형태	오픈소스	오픈소스	오픈소스	오픈소스	독점

History of Container



Linux 컨테이너 란?

- 컨테이너 3가지 기대 효과



- 애플리케이션에 요구되는 종속성을 패키지
- Linux OS 통합
- 애플리케이션 사이의 완전하고 안전한 분리
- VM 하이퍼바이저 불필요
- 모든 클라우드 환경에서 작동

DEVELOPERS

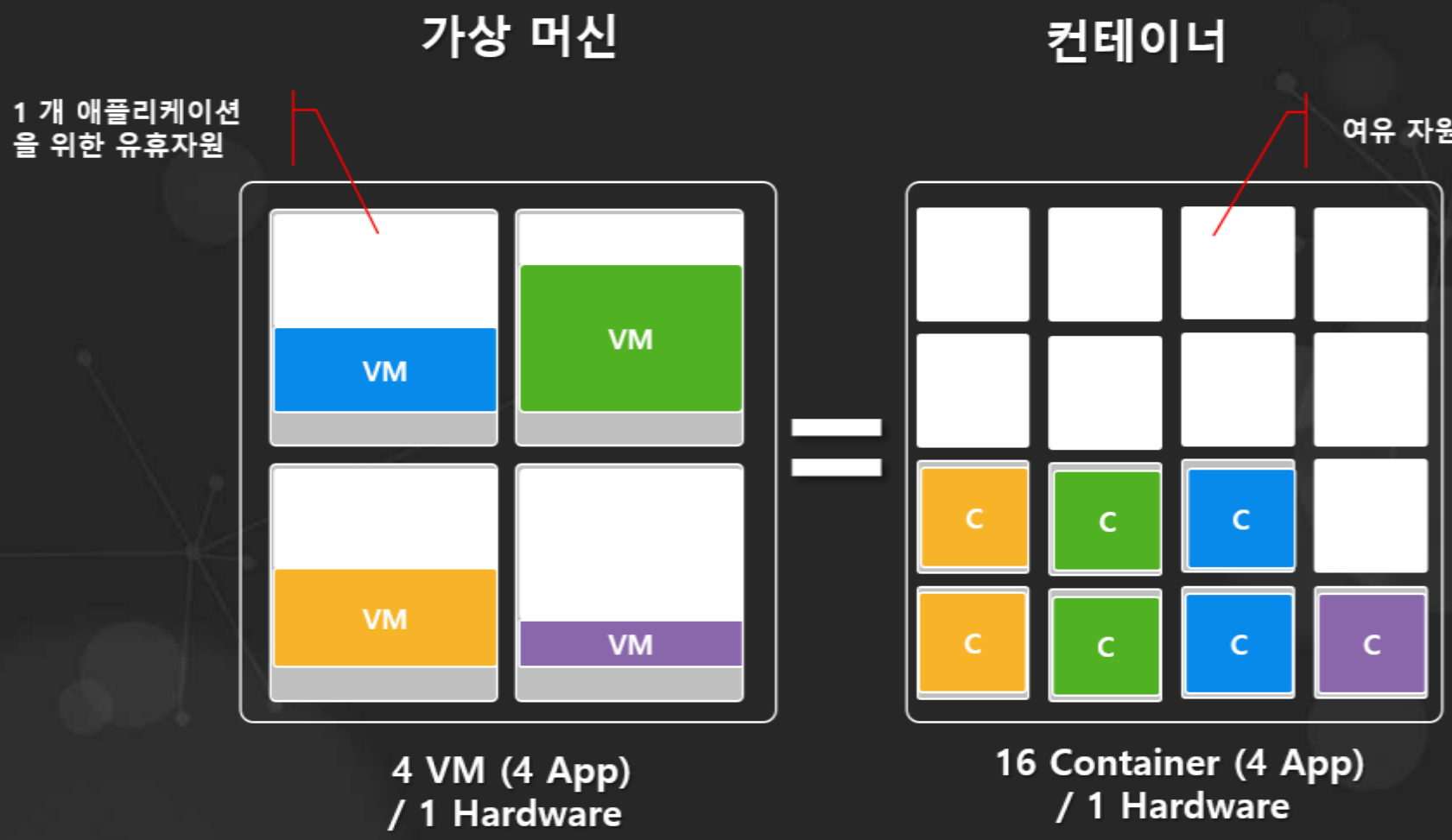
- 애플리케이션 배포 절차 통합
- 애플리케이션 배포 자동화
- 애플리케이션 성능 향상
- 멀티 클라우드 지원

IT OPERATIONS

BUSINESS LEADERS

- DevOps 문화를 촉진
- 하이브리드 클라우드를 촉진
- VM 라이센스 비용 절감
- 애플리케이션 개발주기 가속화

VM vs. Container - 집적도



Container 구조

openmaru APM

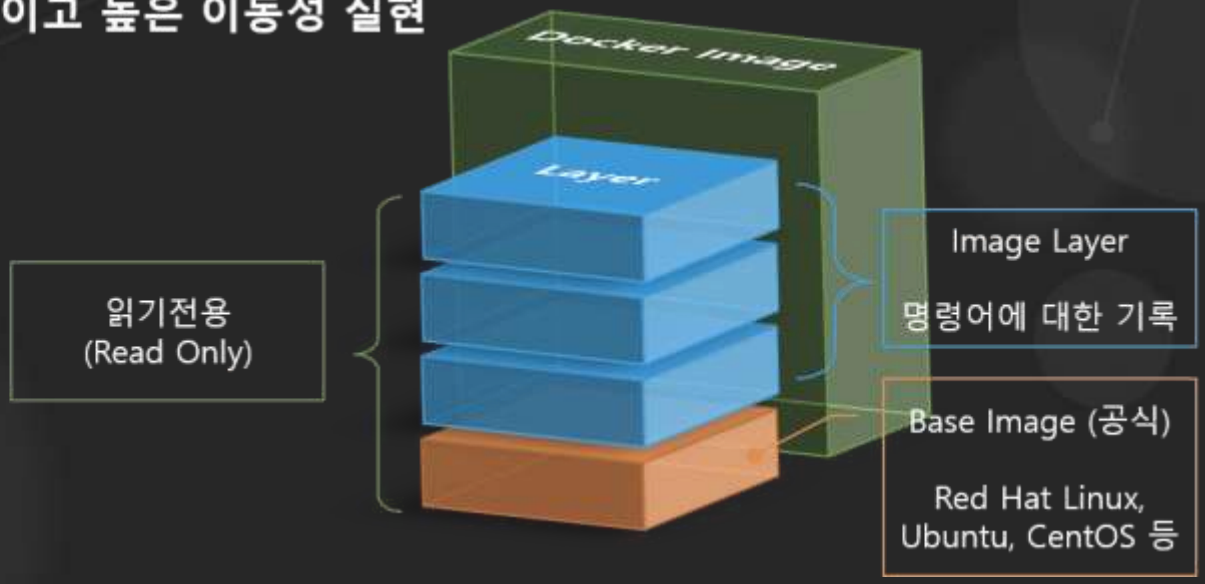
Docker 이미지 구조 예시

Container 구조

- 다른 프로세스와 자원 격리
  - namespaces :시스템
  - cgroups : 하드웨어
- 실행하는 프로세스
- 프로세스 종속 파일
  - 다른 프로세스에 독립된 파일

Docker Image

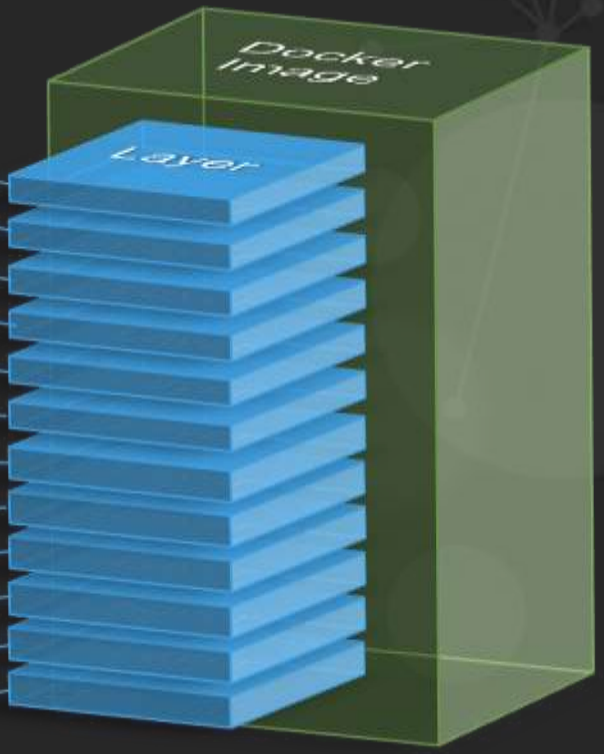
- 컨테이너를 실행할 때 필요한 파일시스템
  - 이미지 레이어의 집합체
    - 파일 내용과 메타 정보를 포함
  - 레이어는 부모와 자식 관계
  - 변경분만 기록
  - Read Only (읽기 전용) 으로 쓸 수 없음
- 공통 레이어를 이미지 간에 공유
  - 디스크 용량을 줄이고 높은 이동성 실현



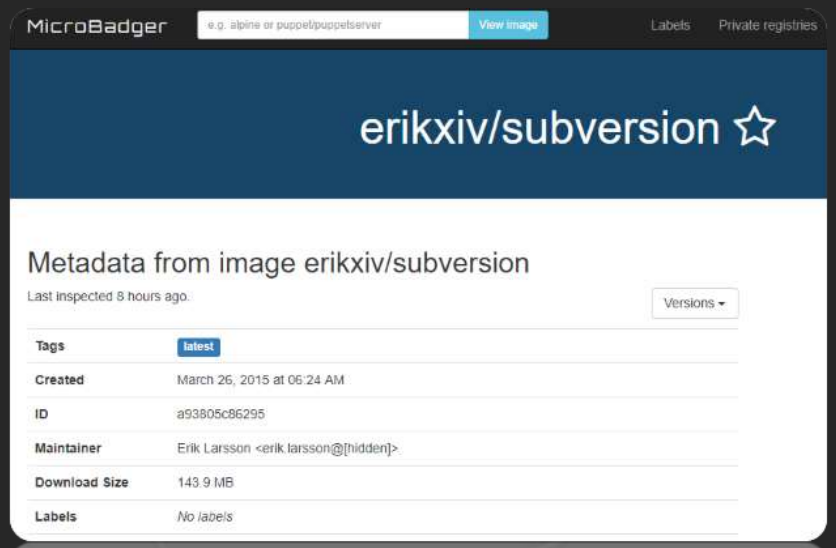
Docker Image 구조

- **docker history <image id / name>**

CREATED	CREATED BY	SIZE
a486da044a3f 10 weeks ago	/bin/sh -c # (nop) CMD ["nginx" "-g" "daemon o	0 B
3cb7f49c6bc4 10 weeks ago	/bin/sh -c # (nop) EXPOSE 443 /tcp 80 /tcp	0 B
42d2189f6cbe 10 weeks ago	/bin/sh -c # (nop) VOLUME [/var/cache/nginx]	0 B
6dda3f3a8c05 10 weeks ago	/bin/sh -c ln -sf /dev/stderr /var/log/nginx/	11 B
9108e25be489 10 weeks ago	/bin/sh -c ln -sf /dev/stdout /var/log/nginx/	11 B
72b67c8ad0ca 10 weeks ago	/bin/sh -c apt-get update &&	7.695 MB
e7e7a55e9264 10 weeks ago	/bin/sh -c # (nop) ENV NGINX_VERSION=1.9.4-1 ~ j	0 B
97df1ddba09e 3 months ago	/bin/sh -c echo "deb http://nginx.org/package	221 B
5dd2638d10a1 3 months ago	/bin/sh -c apt-key adv --keyserver hkp : // pgp.	1.997 kB
aface2a79f55 3 months ago	/bin/sh -c # (nop) MAINTAINER NGINX Docker Mai	0 B
9a61b6b1315e 3 months ago	/bin/sh -c # (nop) CMD ["/bin/bash"]	0 B
902b87aaaec9 3 months ago	/bin/sh -c # (nop) ADD file:e1dd18493a216ecd0c	125.2 MB



Docker Image에 대한 Layer 정보

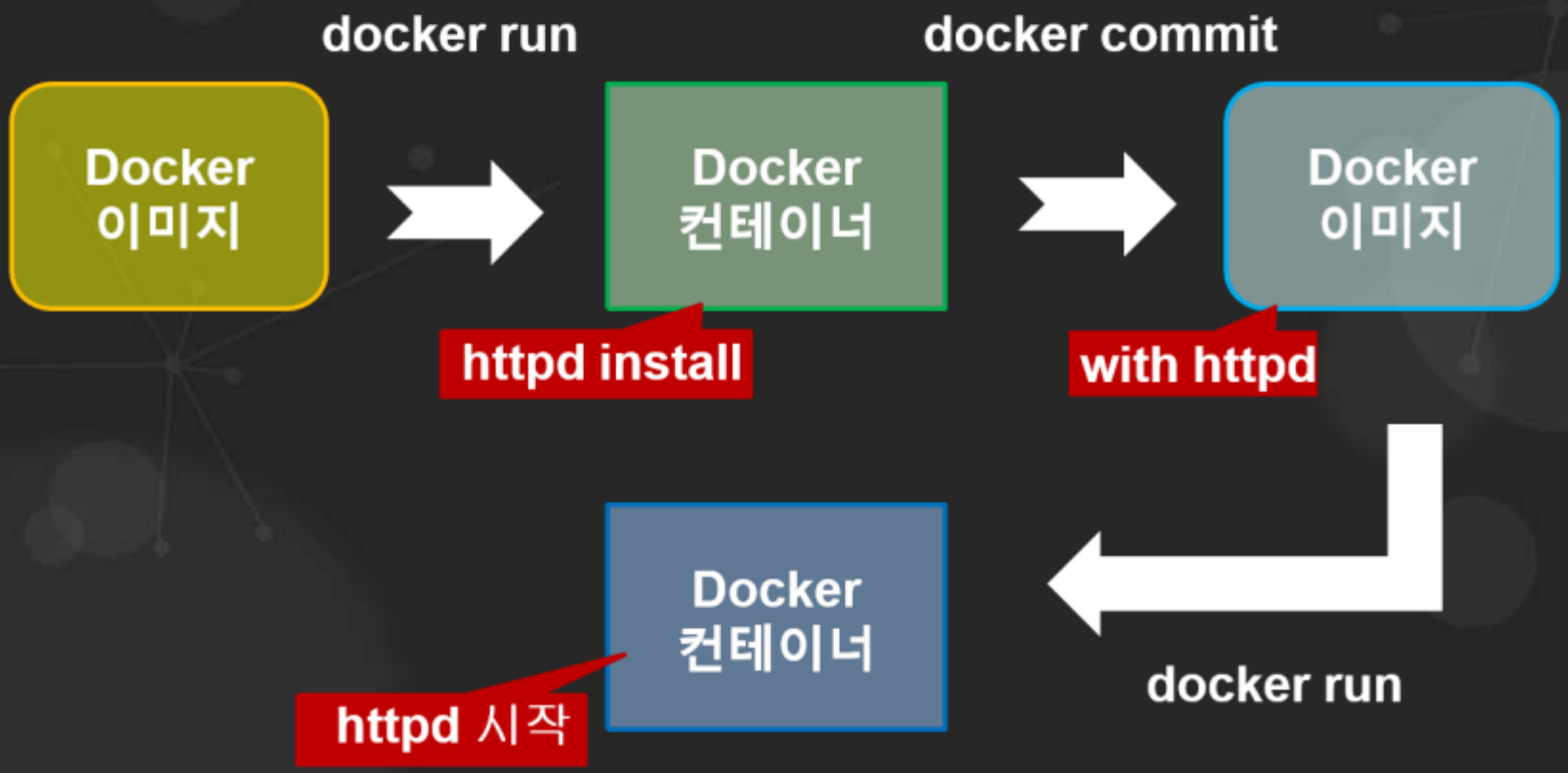


Source: https://microbadger.com/images/erikxiv/subversion

사용자 정의 컨테이너 이미지

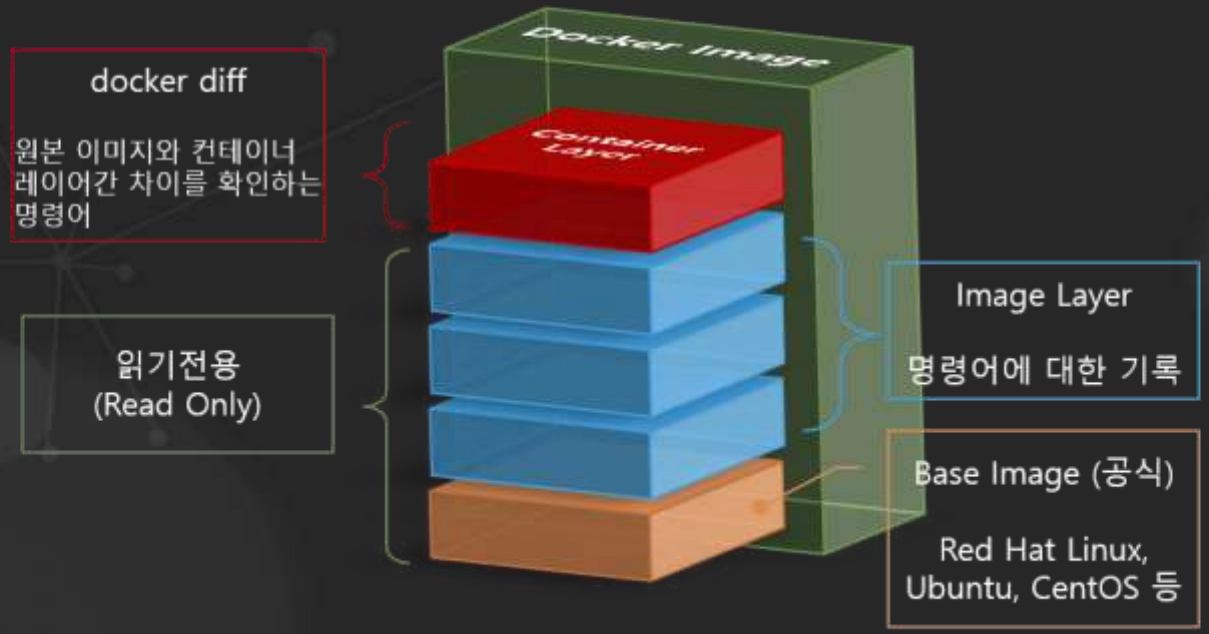
openmaru

- Docker 이미지는 Docker 컨테이너 생성의 기반이 되는 이미지
- 대표적으로 Red Hat Linux, ubuntu , centos 등이 이미지화 됨
- Docker 컨테이너는 한 Docker 이미지를 기반 작성된 각각의 애플리케이션 환경 제공

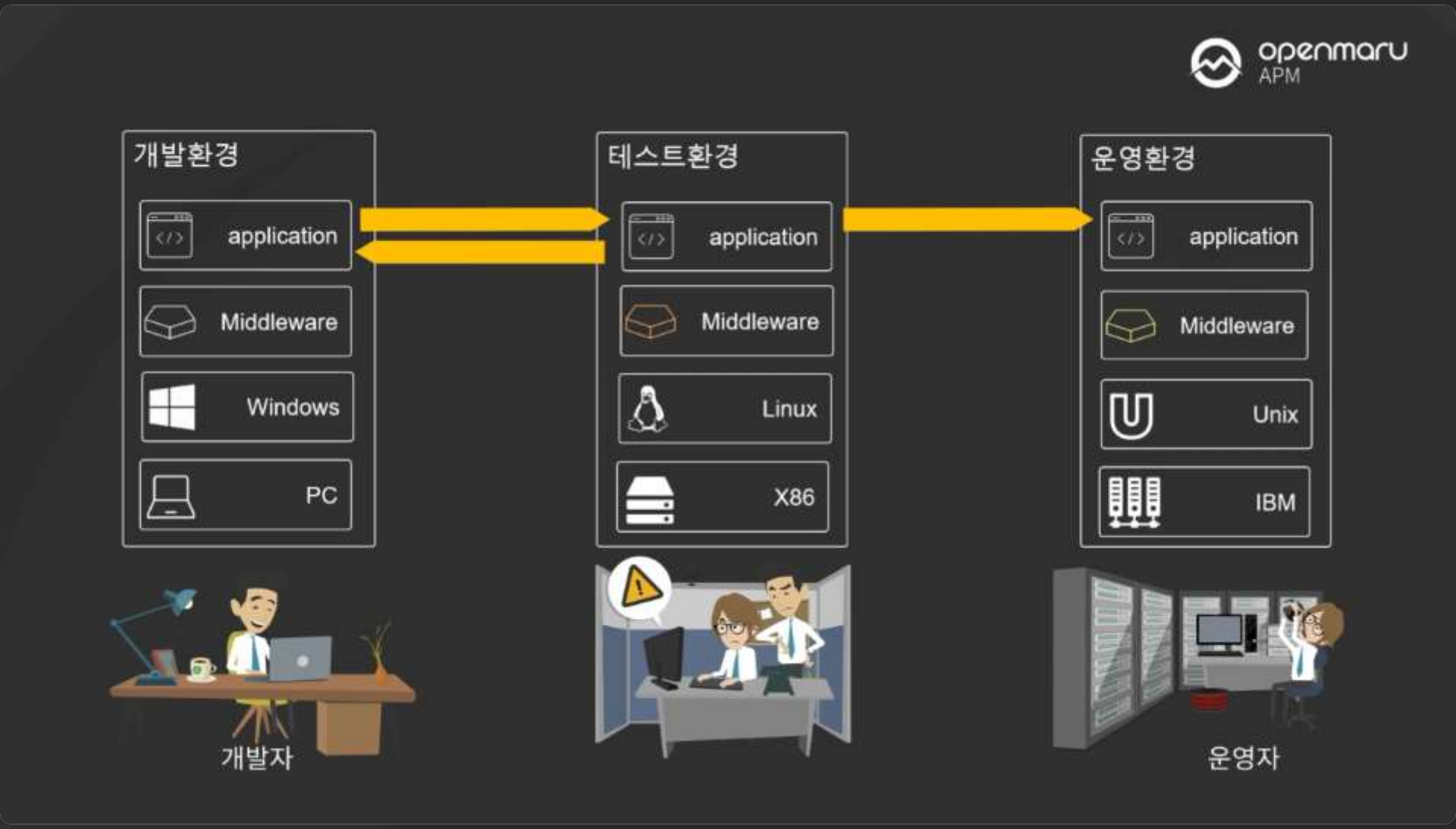


Docker Container 실행

- 새로운 이미지 레이어 자동 할당
- 이미지를 기반으로 독립 한 프로세스로 실행 (컨테이너 상태)
- docker run <opts> <image : tag>

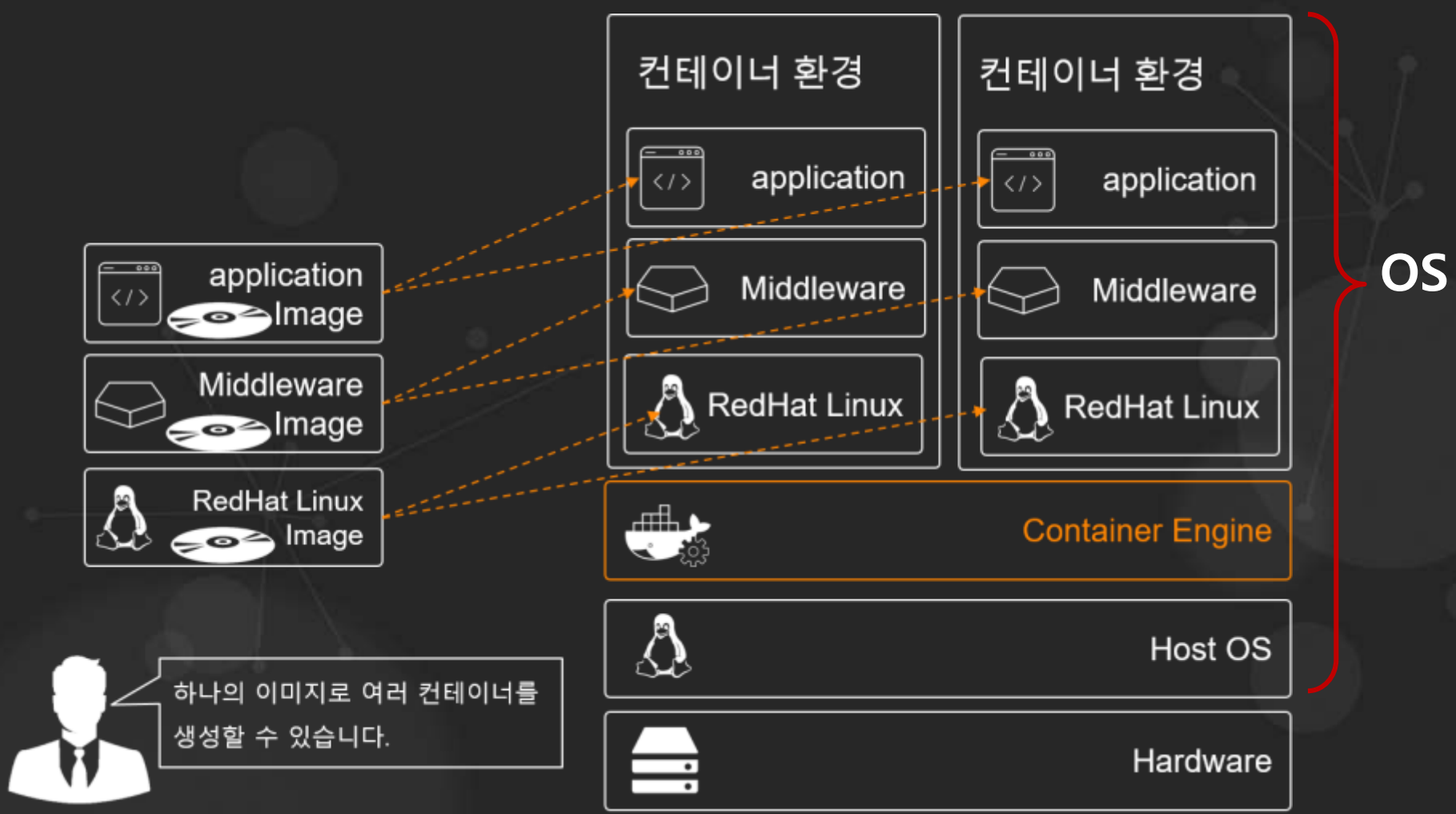


애플리케이션 배포 이슈



애플리케이션 배포 이슈

컨테이너의 동작



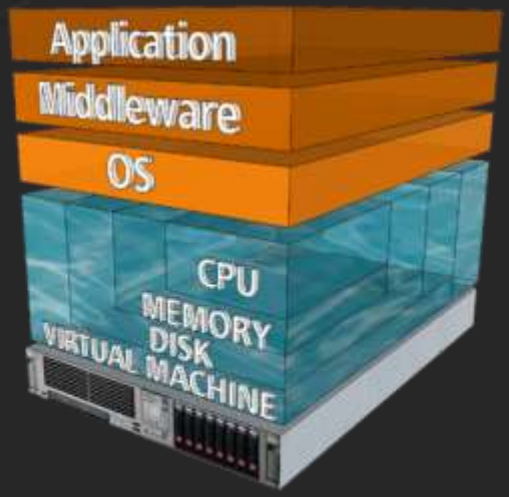
애플리케이션 배포 이슈

VM 의 고질적인 문제점

거대한 이미지 사이즈

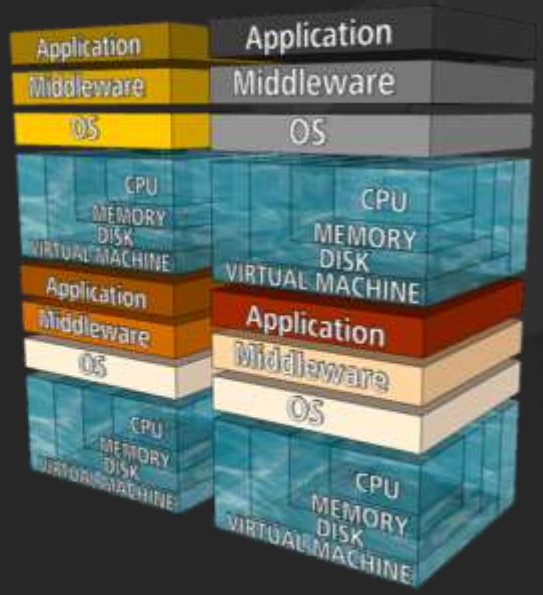
느린 시작 시간

- 부팅 시 Hypervisor – OS- 미들웨어 – 애플리케이션까지 실행 되어야함



VM 간의 환경 불일치

- VM 생성 후 개별로 변경 사항을 관리하기 때문에 VM 간 구성이나 환경이 불일치



컨테이너를 통한 VM 문제 해결

작은 이미지 사이즈

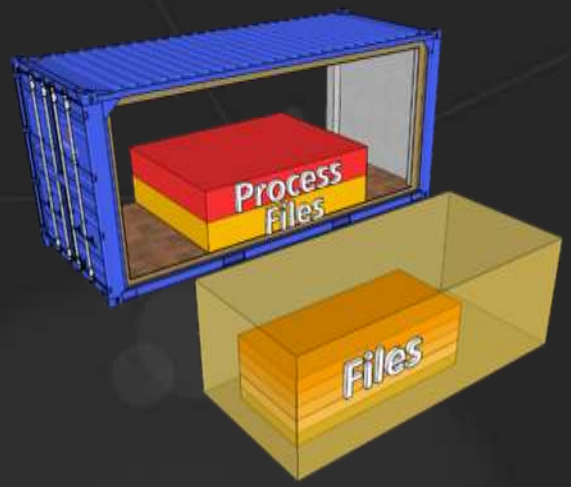
- 컨테이너는 레이어 개념으로 이미지에 파일을 추가/삭제하여 관리함
- 레이어 사이즈를 최적화하여 이미지 사이즈를 최소화

빠른 시작 시간

- 컨테이너는 분리된 프로세스 형식으로 OS 부팅이 필요 없기 때문에 부팅 시간을 최소화 할 수 있음

높은 이동성(Portability)

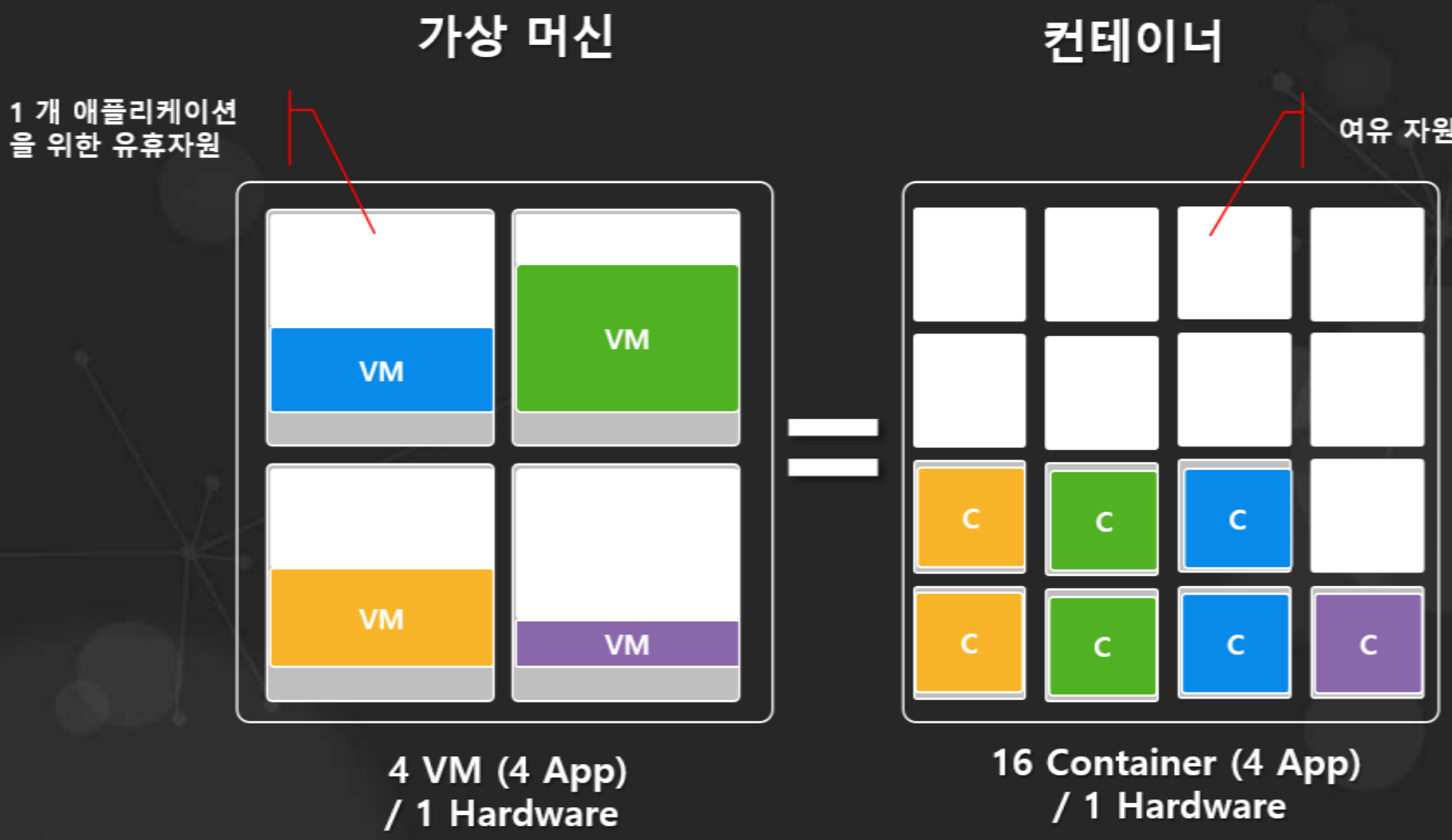
- 애플리케이션에 필요한 라이브러리나 의존 파일들을 이미지에 포함하기 때문에 환경에 의한 발행되는 문제가 거의 없음



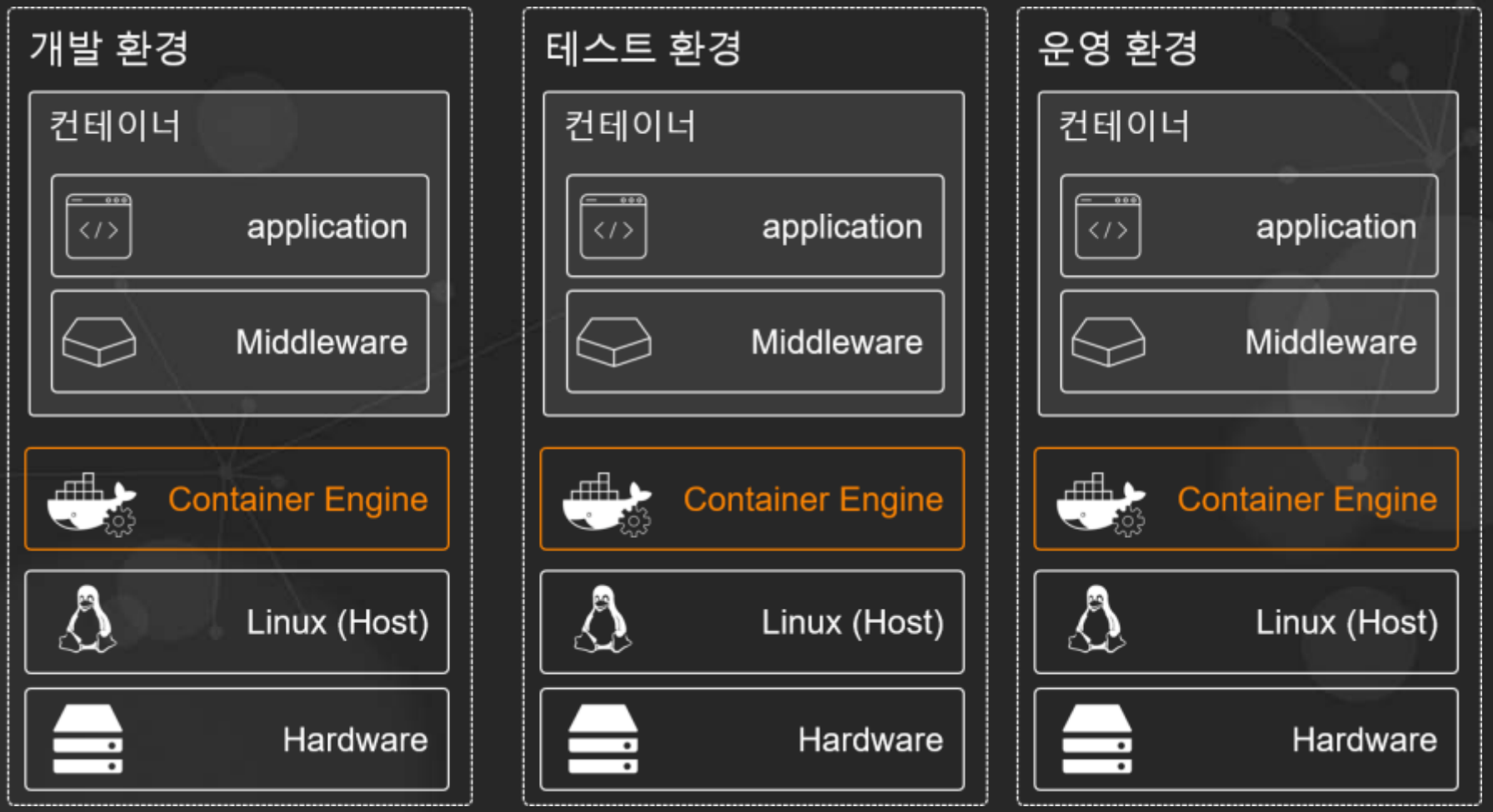
Container =
Process

VM vs. Container - 집적도

openmaru



컨테이너를 활용한 개발 프로세스



MIRANTIS 가 DOCKER ENTERPRISE 사업을 인수

Mirantis는 Docker Enterprise Technology Platform과 관련된 모든 지적 재산권 (Docker Enterprise Engine, Docker Trusted Registry, Docker Unified Control Plane, Docker CLI 등)을 취득했습니다.

앞으로 Mirantis 와 Docker는 함께 Docker 플랫폼의 오픈소스 제품 개발을 지속할 것이라고 밝혔습니다.

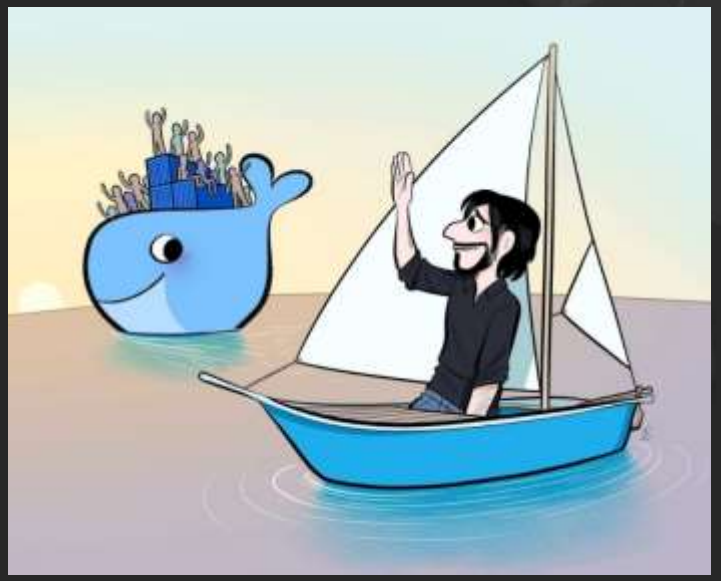
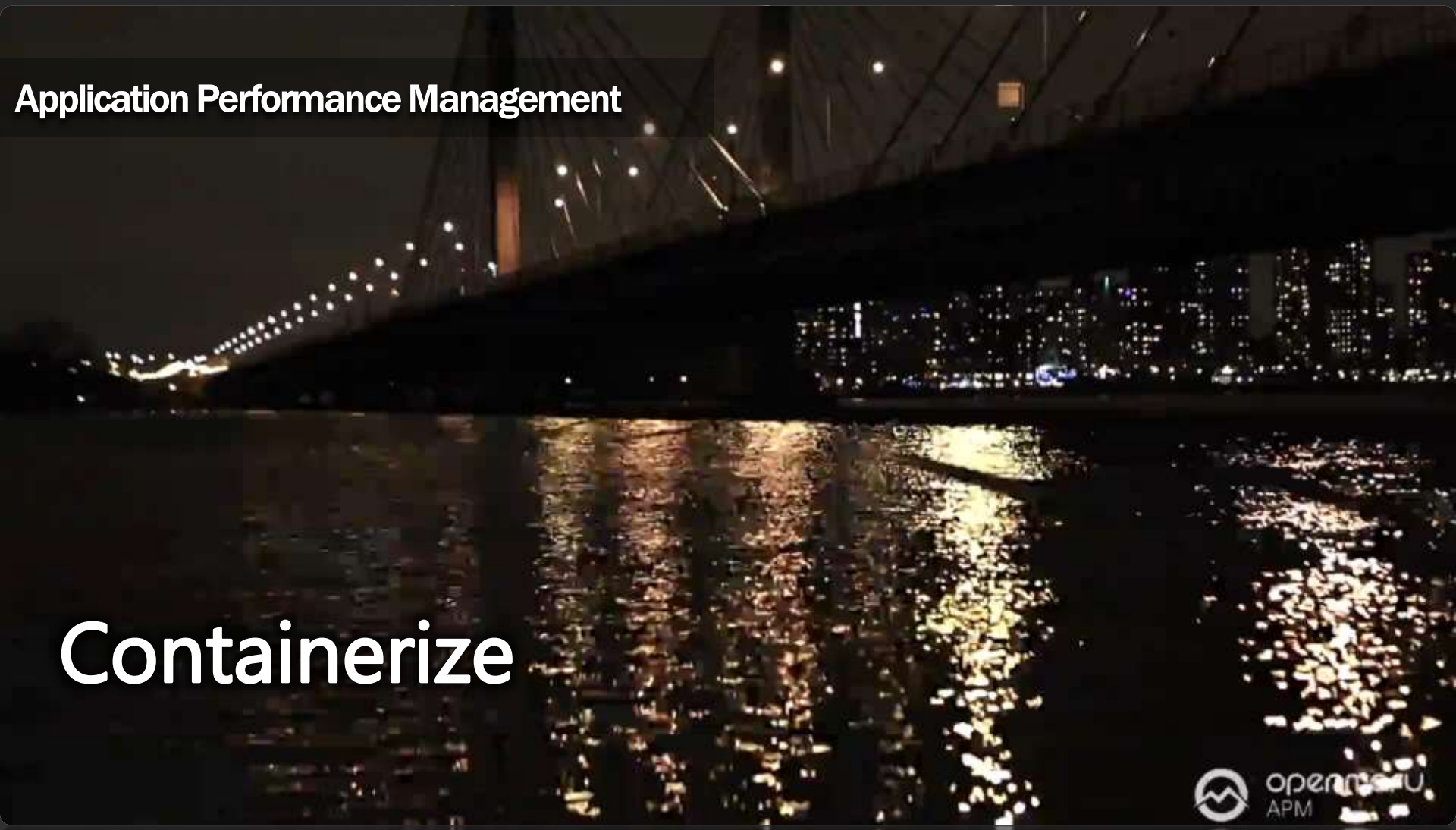


Image Source : https://www.docker.com/blog/au-revoir



컨테이너의 특징에 따른 고려사항

openmaru APM

특징	상세
POD 단위로 확장이 쉬움 (IP와 Hostname이 임의로 부여됨)	• 기존 이미지에서 동일한 POD를 만들기 때문에 쉽게 Scale Out 가능 • POD는 Scale Out이 가능하도록 지정된 IP Range에서 임의로 부여됨. 고정된 IP를 사용할 수 없음. • 컨테이너 Hostname도 확장을 고려하여 [애플리케이션명]-XXXX 형태로 부여됨 • 컨테이너를 임의로 확장할 수 있기 때문에 확장할 수 있는 애플리케이션 구조가 되어야 함
POD 재생성시 컨테이너 내부 데이터는 초기화 됨	• 반드시 저장해야 업로드 데이터와 같은 데이터는 NFS와 같은 공유 스토리지를 연결해야 함 • NFS는 로컬보다 느림
보안상 컨테이너 실행 시 임의의 사용자가 만들어 짐	• root 사용자 및 특정 User는 사용이 제한됨(옵션으로 허용 가능하지만 보안상 취약) • OpenShift POD내의 User는 임의의 숫자 형태로 사용됨 (예 : 10000020000) Group 권한으로 root가 지정됨) • 컨테이너 내에서 Host의 Network 및 리소스에 접근할 수 없음 (옵션으로 허용 가능하지만 보안상 취약)
표준출력으로 전달되는 로그는 Web Console에서 조회가능	• 컨테이너는 Foreground로 실행되기 때문에 표준 출력 파일을 별도로 지정할 수 없음 • 표준 출력에 나오는 메시지는 CLI나 웹 콘솔에서 조회 가능 • 일반 로그 File을 NFS를 사용하여 저장 가능하지만, 같은 디렉터리에 생기기 때문에 hostname 을 파일명 앞에 사용하여 동일 파일이 생성되지 않도록 해야 함

애플리케이션 Containerize시 고려사항 (1/2)

openmaru APM

구분	내용	비고
Containerize 대상선정	시스템 중 컨테이너로 옮길 대상을 검토	• Windows 전용 시스템은 Linux로 전환 후 컨테이너화 • 무거운 Legacy WAS는 가벼운 오픈소스 WAS 전환 고려 • DB는 추천하지 않음(성능이 Critical 하지 않은 경우만 사용) • 배포 WAR 별 컨테이너로 구성 • 세션 클러스터링이 필요없는 애플리케이션이 확장시 더 효율적
Base Image선택	Official WAS 이미지 확인	• WAS 벤더에서 제공하는 이미지가 있는지 우선 확인
WEB/WAS 연동 아키텍처	WAS IP가 동적으로 변경되며, Scale Out을 고려하여 Web 서버에서 WAS IP를 지정하는 연결방식은 사용할 수 없음	• OpenShift 환경에서는 WEB은 제외하고 WAS로만 서비스 하는 단순한 아키텍처가 효율적임
내부 인터페이스 주소	OpenShift 내부에 인터페이스 대상 서버가 있을 경우 대상 IP가 동적으로 변경되기 때문에 연동방법 확인 필요	• OpenShift의 Service명 또는 환경변수를 이용하여 해결 가능
세션클러스터링	IP가 임의로 변경되기 때문에 IP 지정방식의 클러스터링은 사용할 수 없음	• 세션 클러스터링 방식은 WAS 마다 다르기 때문에 벤더에 확인 필요함 • JBoss EAP는 세션 클러스터링 가능(JGroups DNS_PING, KUBE_PING사용) • 세션을 Data Grid에 저장하는 방식 추천
애플리케이션 설정값	변경이 필요한 값은 이미지에 넣지 않고 별도로 관리(환경변수, 파일등)	• 변경 필요한 대상 확인 • 변경 값 적용 방식 확인(애플리케이션 설정값, Java 옵션, 환경변수 등)
애플리케이션 소스	소스에서 Hostname이나 IP에 기반한 로직이 있으면 수정해야 함	• localhost나 환경변수를 사용

애플리케이션 Containerize시 고려사항 (2/2)

openmaru APM

구분	내용	비고
보안	• root 유저를 사용하는 경우, ssh client 사용하는 경우, Host의 자원에 접근하는 경우 보안에 취약할 수 있음	• 설정 변경하여 사용가능
3rd Party Library	• 애플리케이션의 라이선스가 IP나 Hostname기반일 경우 변경 필요(벤더 확인) • C로 작성된 so 파일을 사용하는 경우 해당 Image에 사용가능한 버전으로 변경 필요 • IP, Hostname 기반으로 라이선스를 사용 하는 상용 S/W는 Any IP 기반으로 교환	• 사용하는 3rd Party 라이브러리의 종류 확인 • 해당 벤더에 문의
대용량 컨텐츠	• 대용량 컨텐츠를 이미지로 만들 경우 • Build / 배포 속도가 매우 느려짐	• 공유 볼륨에 대용량 컨텐츠를 분리하여야 함 • 공유 볼륨을 PV(Persistent Volume)로 컨테이너에 마운트하여 사용
Logging	• Standard Out으로 출력되는 로그는 오픈 시프트에서 확인가능	• 파일로 저장되는 로그는 기존과 동일하게 파일로 출력 • 로그가 저장되는 디렉터리를 PV(Persistent Volume)로 연결하여 사용 가능
Batch 형태 애플리케이션	• 클러스터링이 고려되지 않는 Batch 형태 의 애플리케이션은 POD 확장시 여러 번 실행될 수 있음	• WAS의 서비스 로직과 Batch 업무 로직을 별도의 POD로 분리 • Crontab으로 처리하는 부분은 OpenShift Cron Job으로 대체 가능

제품 / 서비스에 관한 문의

- 콜 센터 : 02-469-5426 (휴대폰 : 010-2243-3394)
- 전자 메일 : sales@opennaru.com