Devops란?

CI/CD 이해 및 기초 예시 / 지속적인 통합 / 자동화

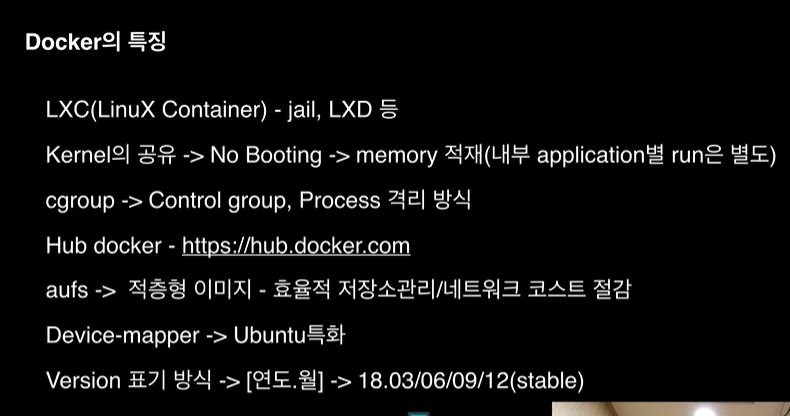
CI는 지속적 통합(Continuous Integration)  
CD란 지속적 배포(Continuous Deploy 또는 Delivery)

CI = 빌드 및 테스트 자동화

CD = 배포 자동화

컨티뉴 인터그래이션 / delivery 지속적인 배포

CI – 프로젝트 여러 개발자들이 개발 후 배포단위 어떻게 할지, 통합 관점에서 충돌나지 않게 작업하고, scoping하고 빌드 대상 어떤걸 할건지 (Jenkins – triggering 처리, scm(소스컨트롤) - git)



Linux container – LXC / 도커 컨테이너가 올라가는 서버의 커널을 공유함.

VM – 머신안에 가상의 머신이 하나 더 올라감 / 커널의 역할 – 시스템 저널을 넣으면 시스템이 부팅되고 os에 있는 커널이 해당 메모리나 디스크나 네트워크 등 채크 / 문제없으면 메모리에 기억해서 🡪 하나의 부팅과정

VM – 부팅을 또 하는 개념 / 부팅된 머신 위에 또 하나의 머신을 올리기 때문

도커 – 별도의 공간을 부팅과정 없이 만들어둠.

컨테니어 – 기존에 만들어진 환경을 그대로 쓰고, 커널도 그대로 씀.

Cgroup –

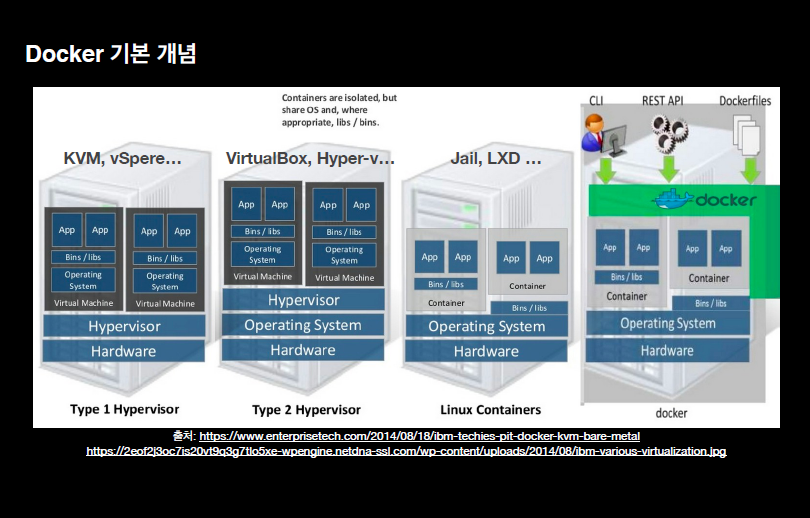
도커는 프로세스 형태로 관리해야함 / 하나의 도커 – 하나의 어플리케이션 / 하나의 프로세서 형태로 돌기 때문에 -> 이 관점이 Cgroup에서 나왔다?

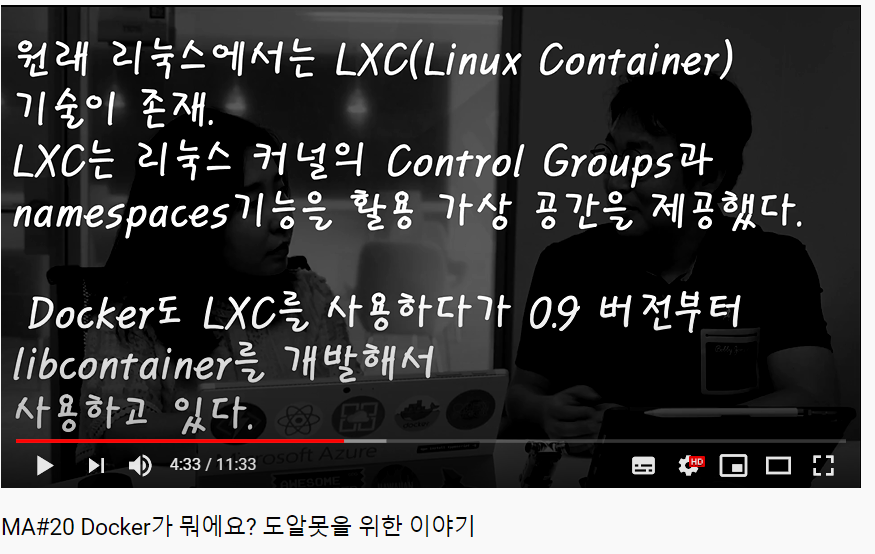
기존 LXC는 이미지를 직접 만들어야 했다.

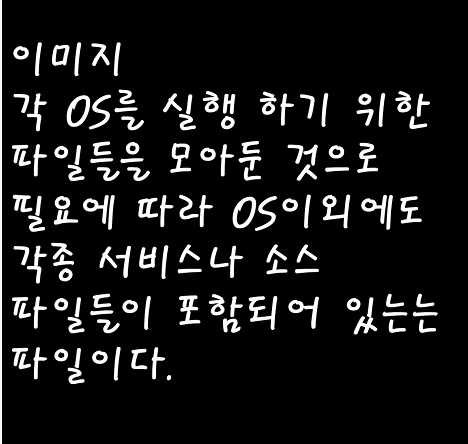
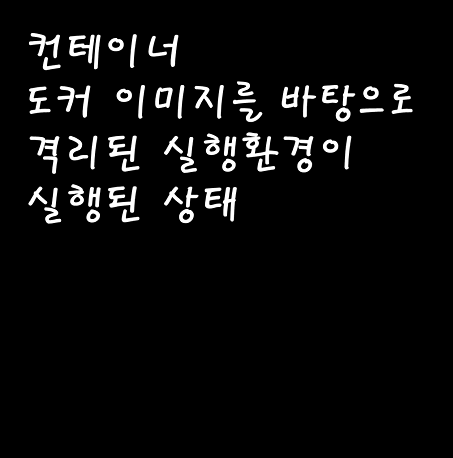
허브처럼 다른 사람이 만들어둔 이미지를 공유받아 사용할 수 있음

Aufs – 우분투에서 사용하는 파일 시스템 / 적층형 이미지 – 효율적 저장소관리 / 네트워크 코스트 절감

우분투에 특화되어있다.







CI = 빌드 및 테스트 자동화 / CD = 배포 자동화

**도커란?**

-LXC(리눅스 컨테이너 형태) / 컨테이너 기반의 오픈소스 가상화 플랫폼

-애플리케이션 실행환경을 이미지로 저장해두어 컨테이너(가상 공간) 단위로 독립적으로 실행 가능함.

-실행환경만 독립적으로 사용함. / 다른 머신에서도 같은 환경 설정 가능

- 커널을 공유/ booting 없음

**컨테이너 VS VM** - 기존에 만들어진 환경을 커널을 공유해서 사용함.

**Aufs** – 앞에 베이스 이미지가 올라가있고, 새로 이미지가 올라갈 때 기존 베이스 이미지와 겹치는 부분을 제외하고 올라감. / 적층형 이미지

도커 **이미지**

-각 os 실행을 위한 파일을 모아둔 것, os이외에 각종 서비스나 소스 파일들이 포함되어 있음. = 프로비저닝 작업

도커 **컨테이너**

-도커 이미지를 바탕으로 격리된 실행환경이 실행된 상태

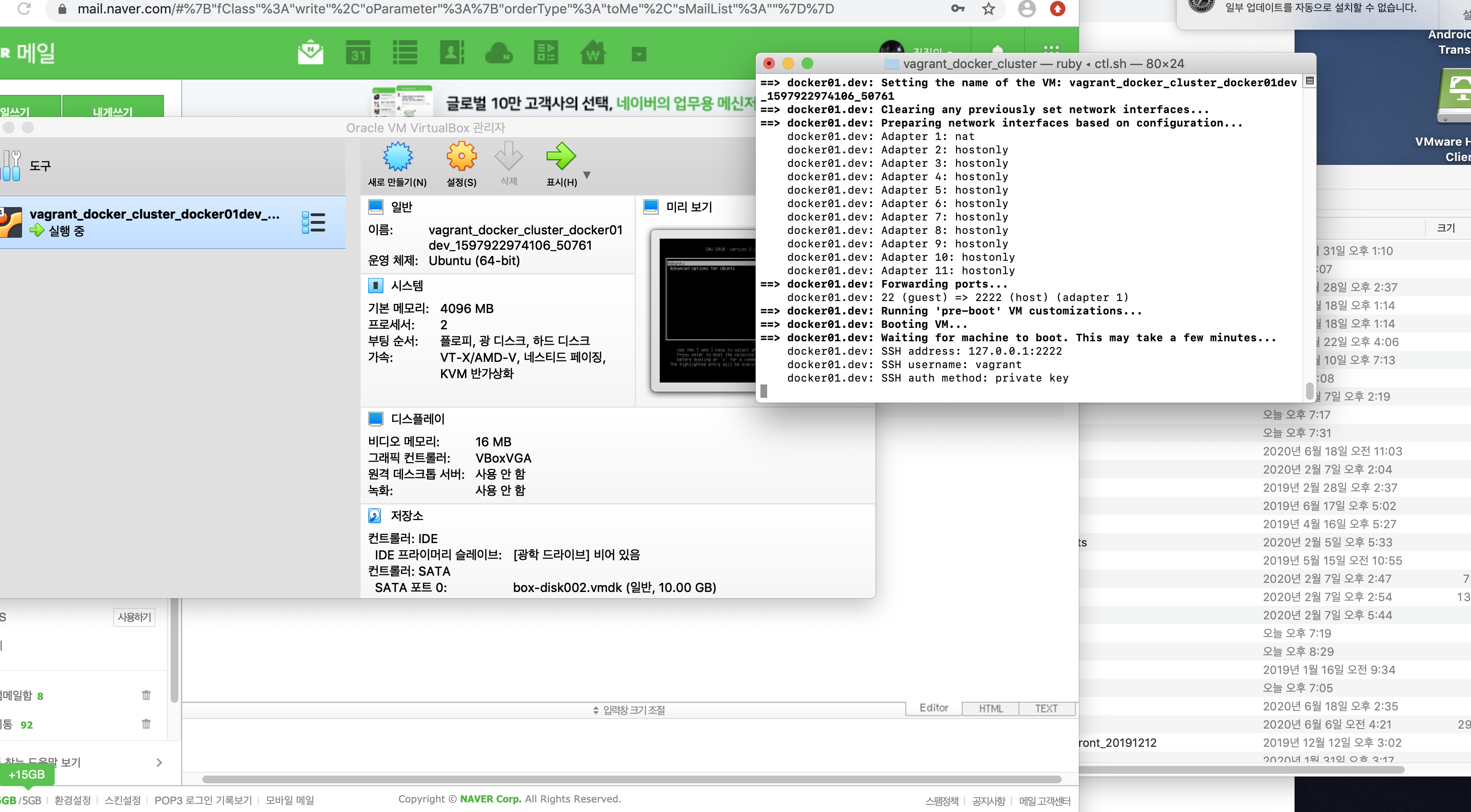
-각 애플리케이션과 종속물이 운영체제 리소스의 분리된 세그먼트를 사용하는 방식

-이미지로부터 부팅 및 실행되며, 이미지는 기본적으로 변경 불가능

***쿠버네티스*** *- 컨테이너 기반의 가상화 환경을 운영 관리하는 핵심 기술*

도커 개념 공부 : <https://www.youtube.com/watch?v=tPjpcsgxgWc>

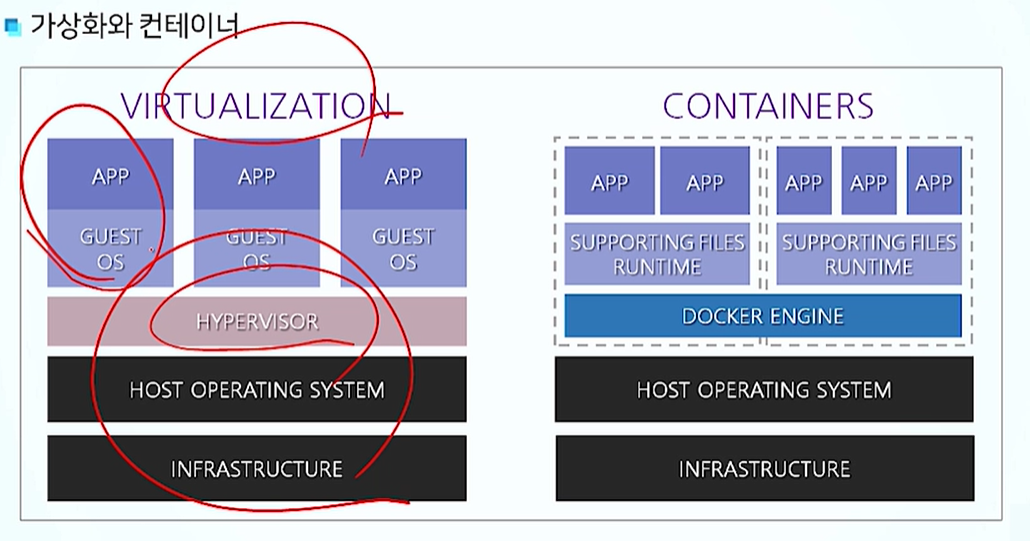
+)MSA // <https://www.youtube.com/watch?v=UsWrvBaQOA4>



애플리케이션을 작은 서비스 단위로 개발하는 방법 -> msa

* 각각 자체 프로세스에서 통신
* 대부분 http 기반 api 통신
* 완전 자동화된 배포머신을 통해 독립배포
* 중앙 집중식 관리 최소화

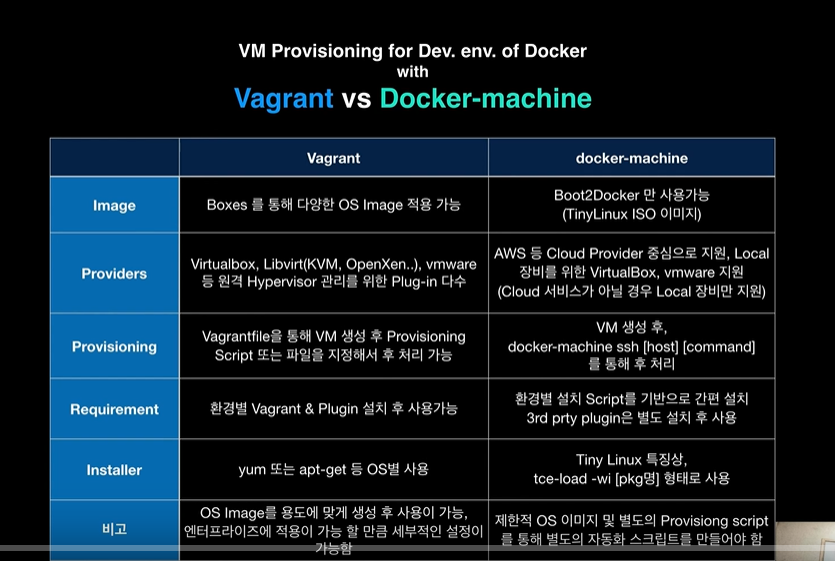
장점 : 장애 격리와 복구가 쉽다. / 비용 효율적으로 증설 가능 / 서비스 개선 속도 증가, 빠른 배포 가능 / 생산성 향상, 코드양 적음 / 신기술 도입 쉬움 / polyglot 적용가능. 서비스에 최적화된 개발 언어, dB선택 쉬움.

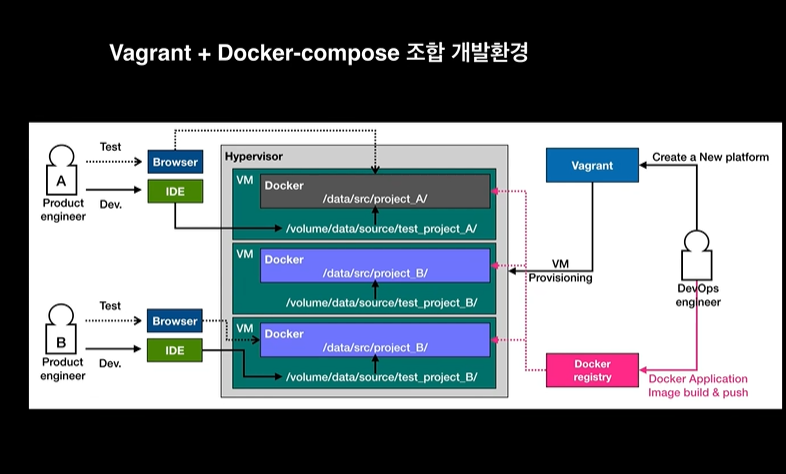


가상컴퓨팅 – 한 물리적 컴퓨터 안에 각각 os를 돌리는 가상 컴퓨터들이 물리적자원을 분할해서 사용함.

프로비저닝 - 서버의 환경을 어떤 애플리케이션이 실행가능한 상태로 준비하는 과정을 의미합니다.

vm과의 차이 : vm가상머신 ( 머신안에 가상머신 하나더 올라감) – 기존의 OS를 똑같이 구현하기 위해 -> 커널이 하는 역할을 함.





Hypervisor – 버츄얼 박스

안에 vm여러 개를 띄워둠 - vm안에 도커 컨테이너를 띄워두고

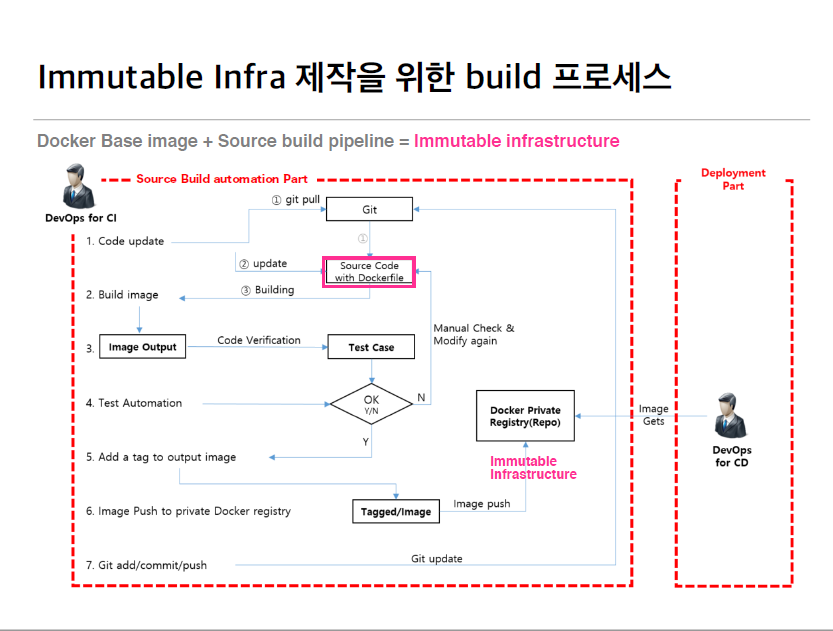
클러스터링으로 엮어둘 수 있음.

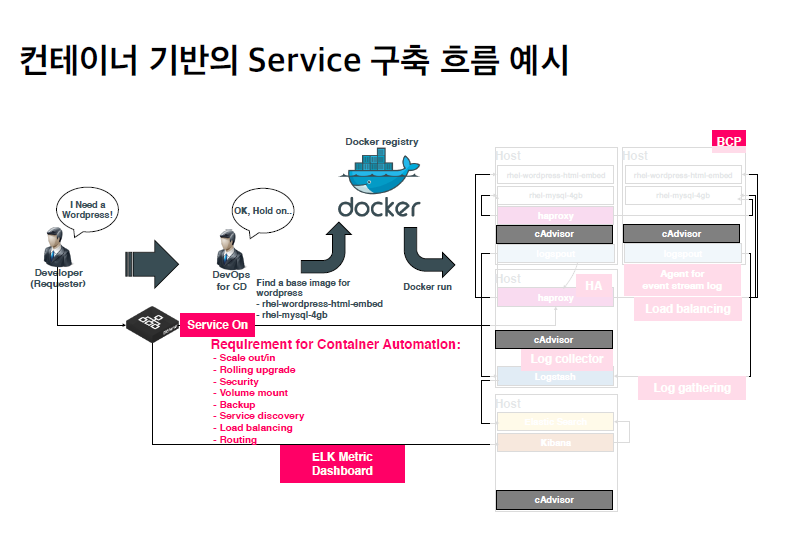
//

베이그런트 – vm만들고 or 도커 머신을이용해서 vm만들고

* 그 위에 도커 이미지를 올리는

도커 / msa / 쿠버네티스 차이





CI/CD 왜 해야 하는가?

-배포 업무 성숙도 향상

🡪 무중단 배포 > 지속적인 배포 > 빠른 개발 빠른 출시

* DEV = stage = production > 충분한 테스트 > 배포에 따른 문제 최소화
* Human error 예방

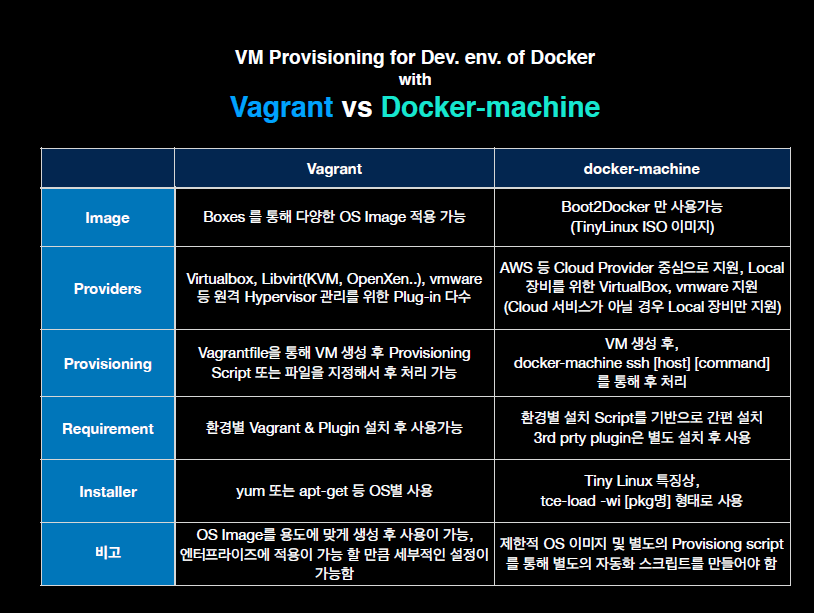
-전략적 배포 기법

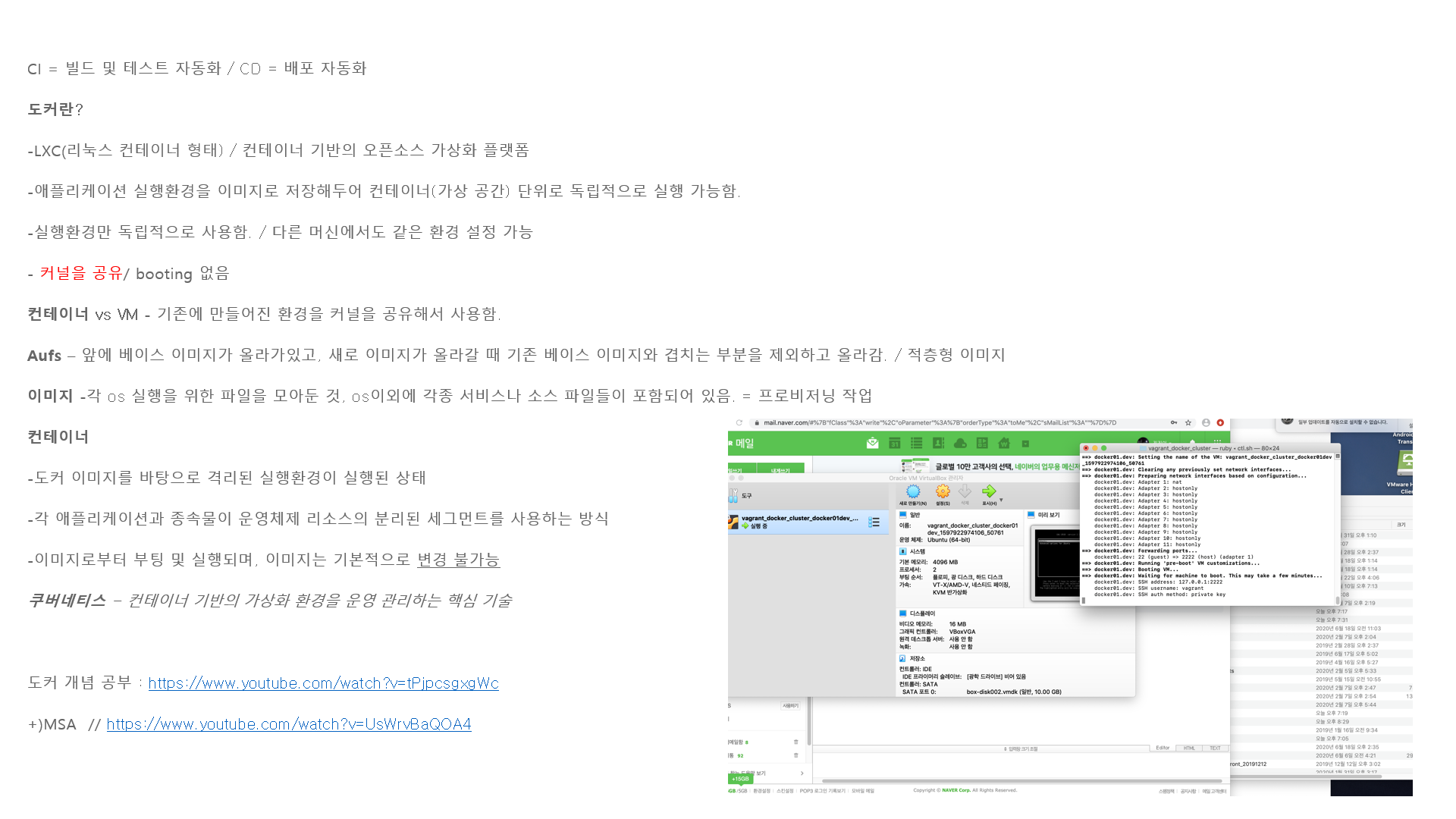
-> canary release -> soft-landing

->Blue/green deployment

->AB Test -> UX -> Service 만족도 향상

도커 실습 환경구축





개발자가 개발시, 운영자의 업무사이에 있는 업무들 때문에 배포에 시간이 걸린다.

해당 업무 분담도 애매하고 권한과 관련되어서 비효율적이다.

이런 비효율을 개선하기 위해 Devops라는 개념이 생겨났다.

CI / CD

CI : 지속적 통합, 형상관리솔루션으로 컨드롤된 소스들을 자동으로 기존 소스와 통합(젠킨스)

CD : 기존 소스와 통합된 archifact들을 자동으로 배포해주는것.

Docker : LXD기반, 기존 가상화 솔루션과 다르게 한 프로세스의 실행환경을 이미지로 저장해 컨테이너 단위로 묶는다.

해당 이미지는 환경까지 되어있기때문에 배포에 대해 편리해진다.

빠른 개발 빠른 배포를 위한 툴

베이그런트를 이용해 프로비저닝을 할 수 있다.

프로비저닝 : 사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요 시 시스템을 즉시 사용할 수 있는 상태로 미리 준비해 두는 것

도커

서버 설정에 있어서 다 환경이 다르고 버젼 관리가 어렵다는 문제점을 해결하기 위해서 도커가 등장하게 되었다

컨테이너 기반의 오픈소스 가상화 플랫폼

컨테이너 - 격리된 공간에서 프로세스가 동작하는 기술, 다양한 os위에 application이 올려져 있는 것

이미지 - 컨테이너 실행에 필요한 파일과 설정값

장점

기존 가상 머신보다 가볍고 빠르다

개발에 있어 더 자유롭다

보안성

쿠버네티스

모니터 컨테이너 상태 관리

웹 니즈에 알맞게 컨테이너 생성 및 삭제

컨테이너 신규 버전 업데이트

msa

애플리케이션을 작은 서비스 단위로 개발하는 방법

자체 프로세스에서 실행

http 기반 api 통신

비즈니스 기능을 중심으로 구축

자동화된 배포머신을 통해 독립 배포

장점

대용량 분산 환경에 적합

각 서비스에 알맞은 기술 스택 적용 가능

유연한 배포

재사용성 확장성

단점

장애추적 모니터링 어려움

여러 서비스에 걸쳐져 있을때 테스팅, 트랜잭션이 복잡

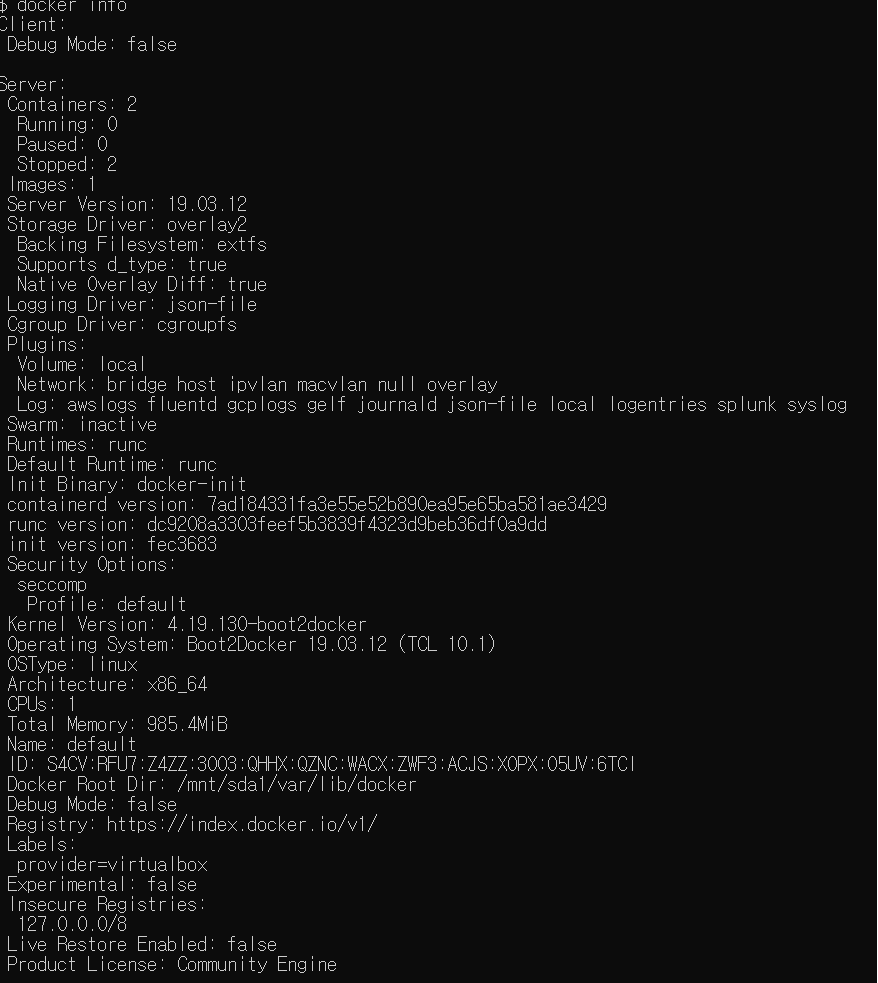
20200831

Docker toolbox 명령어 정리 on windows

Docker –version : 도커 버전 확인



Docker info : 도커 버전 정보 확인



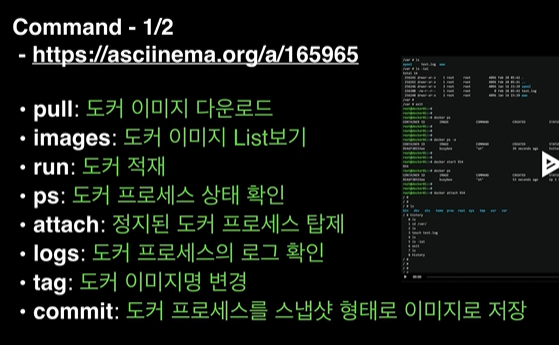
Ex) docker run hello-world

Hello-world 라는 이미지가 로컬에 없다면, 도커 허브에서 내려받아 해당 이미지를 컨테이너로 동작

Docker ps // 현재 돌아가고 있는 컨테이너 확인

Docker images // 현재 도커 머신에 설치된 도커 이미지 확인

허브에서 OS 이미지를 다운로드 받을 수 있음



Commit – 이미지 관리를 다시 생산할 수 없게됨

Docker pull busybox 도커 이미지 다운로드 도커허브에서 / buxy box 라는

Docker images – 도커 이미지 리스트 확인

Docker run -it busybox sh – 도커 비지 박스라는 이미지의 sh(명령어 쉘) 을 실행

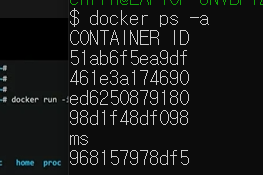
Touch test.log – test.log 파일 생성

Exit – 나오기 / 컨테이너에서 나오고 메모리에서 detached

Docker ps – 실행중인 이미지리스트 확인

Docker ps -a 과거에 실행되었었던 이미지까지 이력 확인 /exit 해서 멈춘것까지

Docker start 51a – 실행할 컨테이너 id 앞자리 3개만 적어도 실행됨



Docker attach 51a – attach 시 적재된 내용으로 들어감

쉘 내부에서 history – 과거에 실행했던 명령어 이력 확인

Ctl + p, ctl +q will be suspend node / ctrl p q 하면 exit로 컨테이너 사라지는게 아님 도커 컨테이너가 대기상태로 돌아감 / 이미지가 삭제되지 않고 남아 있음

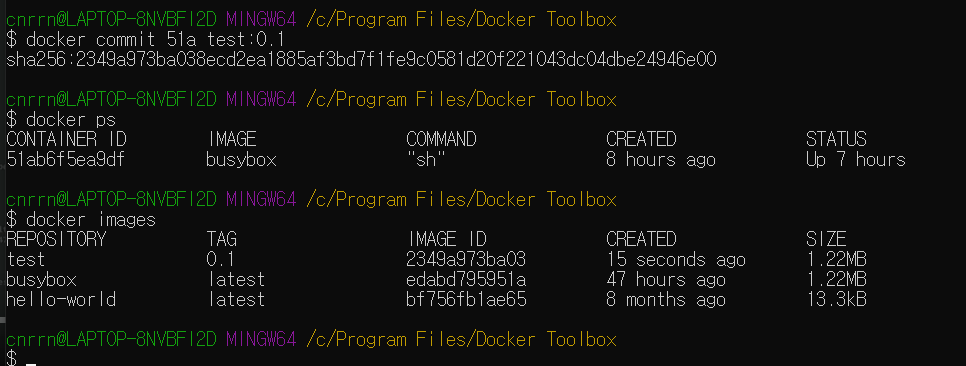
도커 시작시 – docker start / attach 로 들어감

도커 종료시 – exit / detached

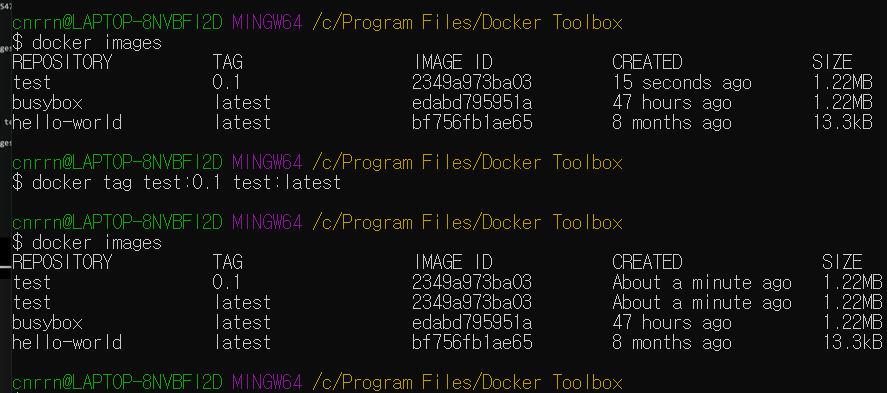
도커 대기상태로 종료 – ctrl + p + q

Docker commit 51a test:0.1 – 위에서 입력했던 상태로 새로운 이미지를 만듬

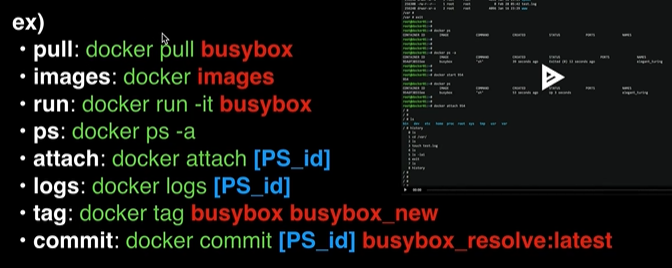
(스냅샷을 떠서 새로운 이미지로 만듬)

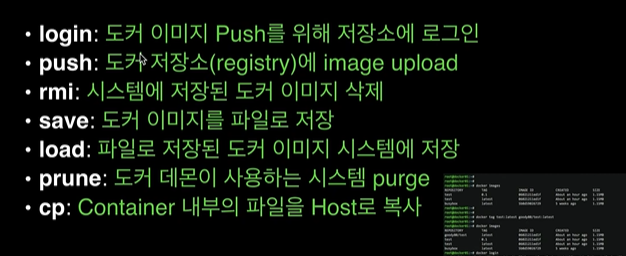


Docker tag test:0.1 test:latest – 도커 이미지명 변경



Commit – 도커 개발을 위한 테스트 용





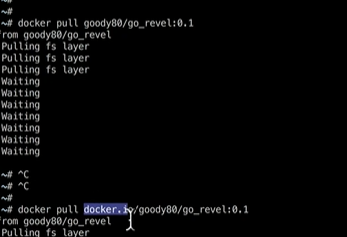
Docker login – 허브도커, private registry 에 로그인

Rm – 프로세스 리무브 / rmi – 도커 이미지 리무브

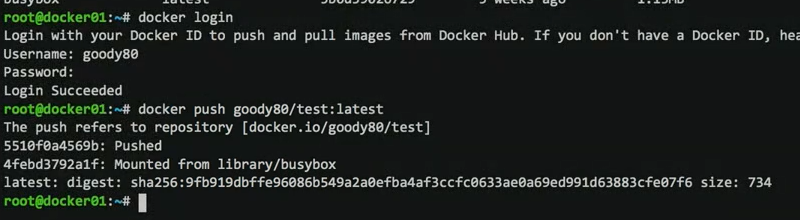
Prune – 퍼지 기능 / 불필요한거 삭제

Push – login 후에 url을 적어 푸시 함 – 도커 허브에 푸시

이미지 – uri 포함함



Push 올릴때에는 url 작성하여 푸시

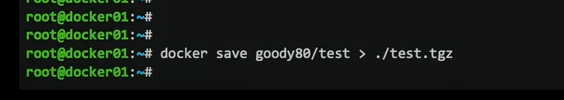


도커는 aufs 기반 적층형 이미지 /

Busybox(이미 있는 이미지는 mount되어 있기에 올리지 않음) 기준으로 만들어진 새이미지가 올라감

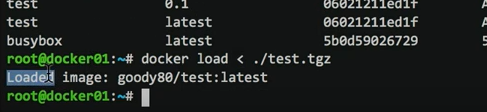
Docker rmi goody80/test – 이미지 지움

Rmi / pull / push



만들어둔 이미지를 바이너리 파일로 저장

Docker save goody90/test > ./test.tgz

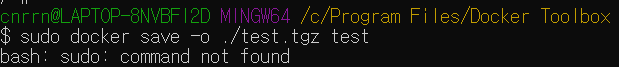


Docker load < ./test.tgz – 이미지 로드하여 적재

**>Docker save 에러 – permission denied**



1.sudo 명령어 사용하려 했으나 sudo 못 씀



2. 도커 머신 ssh default 로 돌림

Docker-machine ssh default



3. sudo docker save -o test.tgz test:latest

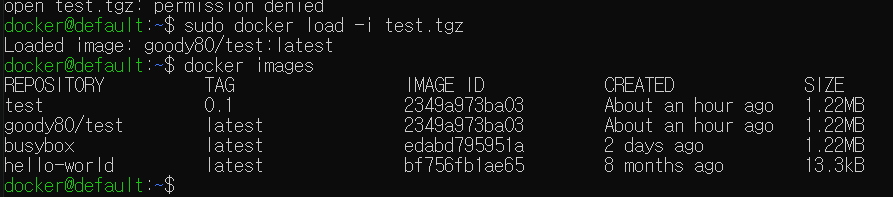
Sudo 명령어 사용하여 저장



4. docker rmi 로 삭제후 – 이미지 삭제

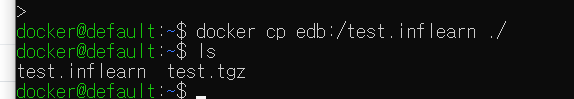


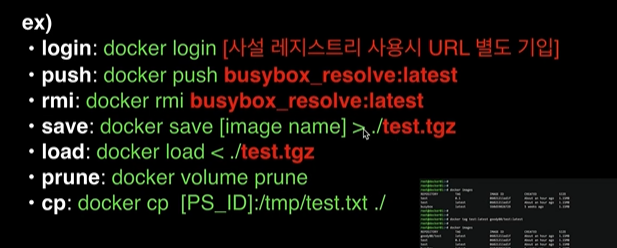
5. 로드 sudo 사용 – sudo docker load -I test.tgz



망분리 환경에서 검증된 이미지를 파일단위로 관리하기 위해

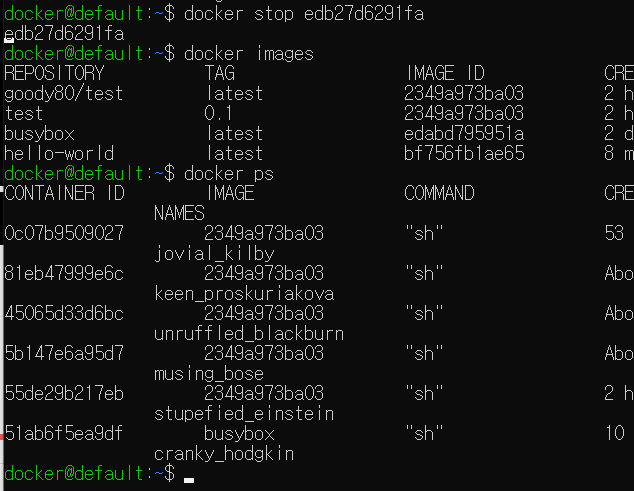
Docker cp 컨테이너아이디: 파일경로 호스트경로 - 컨테이너 내부의 파일을 HOST로 복사 / 특정 경로로 보내기



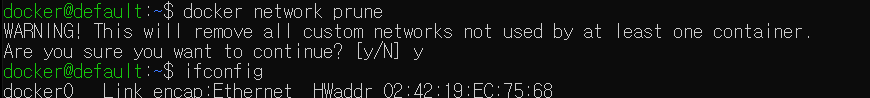


>, < 방향 잘 보기

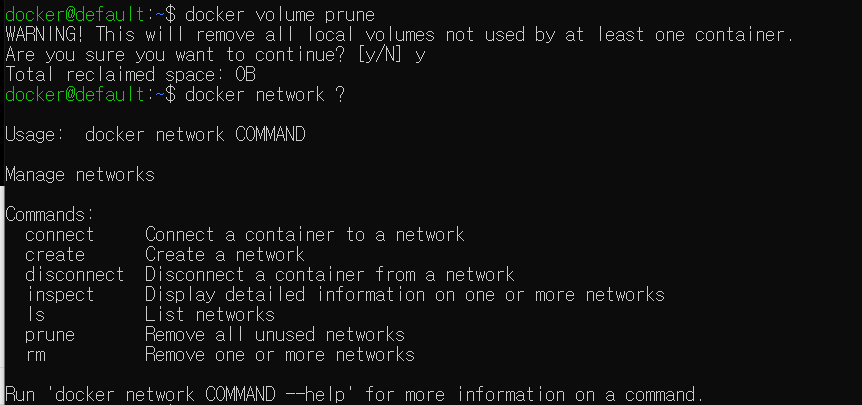
이미지 내리기 stop



Docker network prune – 불필요한 네트워크 삭제

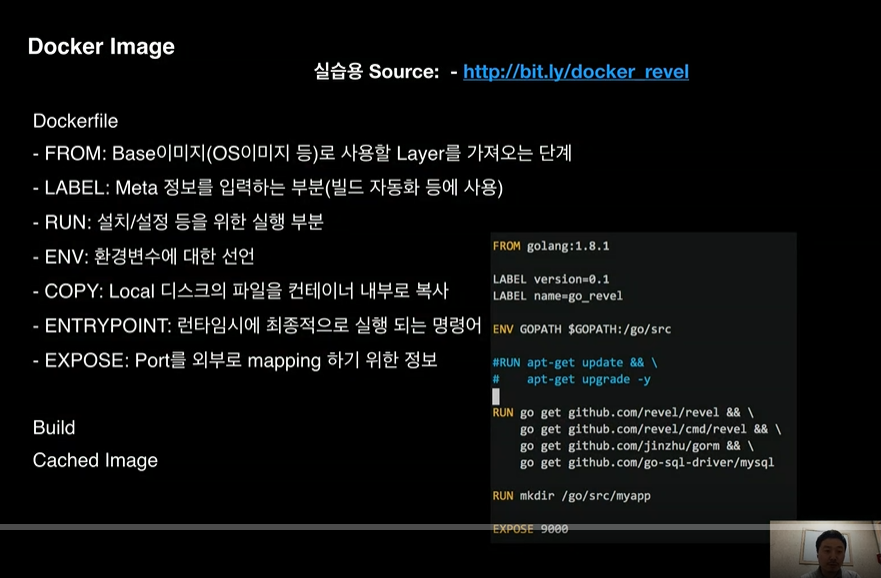


Docker volume prune – 볼륨도 삭제 가능



Docker volume prune / docker network prune – 필요없는 것을 지울 수 있음

주요사용법 2/2 – 이미지 만들기



Dockerfile 명세를 기준으로 이미지를 만듬

From – 가장 기반이 되는 Base 이미지(OS이미지 등)으로 사용할 Layer를 가져오는 단계

Label – Meta 정보를 입력하는 부분 (빌드 자동화 등에 사용)

RUN – 설치 / 설정 등을 위한 실행 부분 // 이미지 적층

ENV – 환경변수에 대한 선언

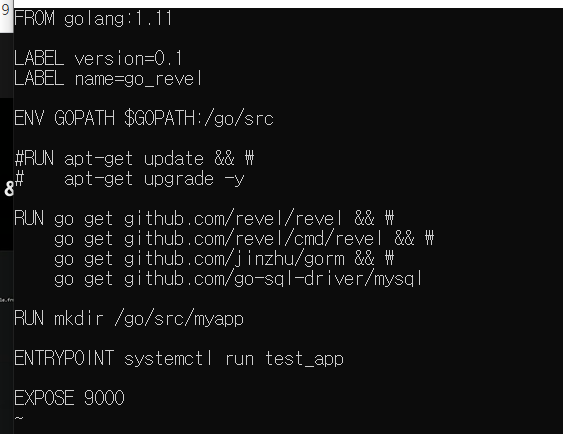
COPY – Local 디스크의 파일을 컨테이너 내부로 복사

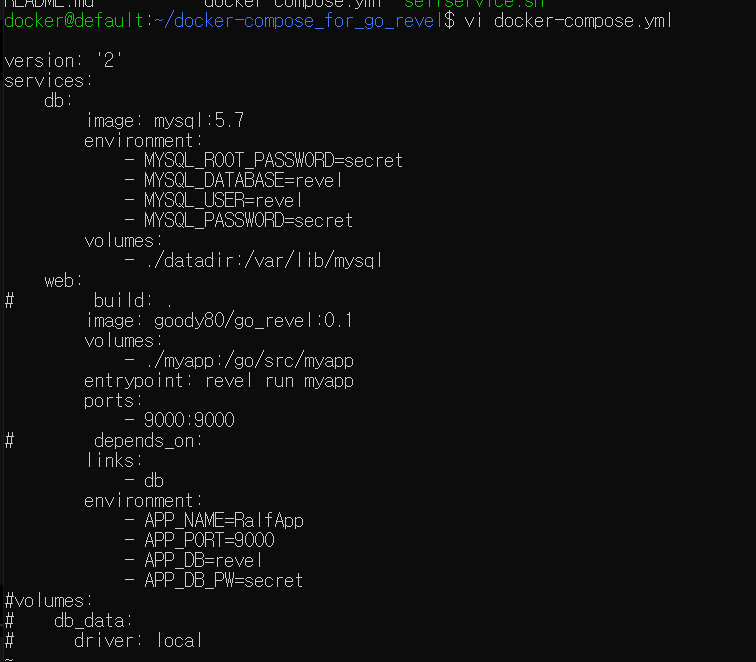
EntryPoint: 런타임시에 최종적으로 실행되는 명령어 / 도커 running 시 맨 마지막에 실행됨

Expose – Port를 외부로 mapping 하기 위한 정보

Build

Cached Image





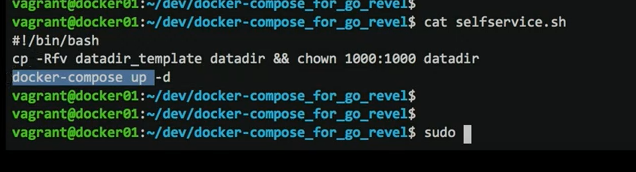
DB 와 웹을 컨테이너 기반으로 올림

이미지 밑에 goody80 밑에 revel을 가져오는데 dockerfile에 적힌 설치명세 기반으로 이미지 빌드

Volume 경로를 따라 volume 마운트가 걸림

디비를 바라봄, 우선순위 지정

환경설정을 이미지에서 미리 설정해둠



Docker-compose up : 런타임 모드 foreground 모드로 들어오는데, -d를 붙이면 background 모드로 들어옴

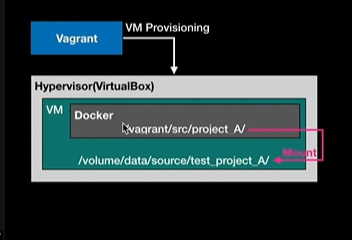
Sudo ./selfservice.sh

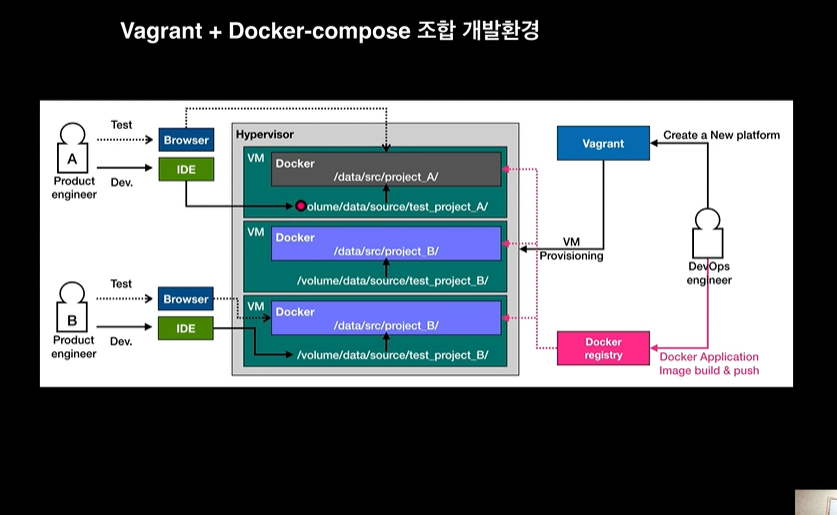
미리 만들어놓은 테이블을 디비 경로에 부어주는 작업 -> mysql pulling 하여 **적층형 방식으로** 이미지 받아옴 -> go\_revel 이미지를 만듬

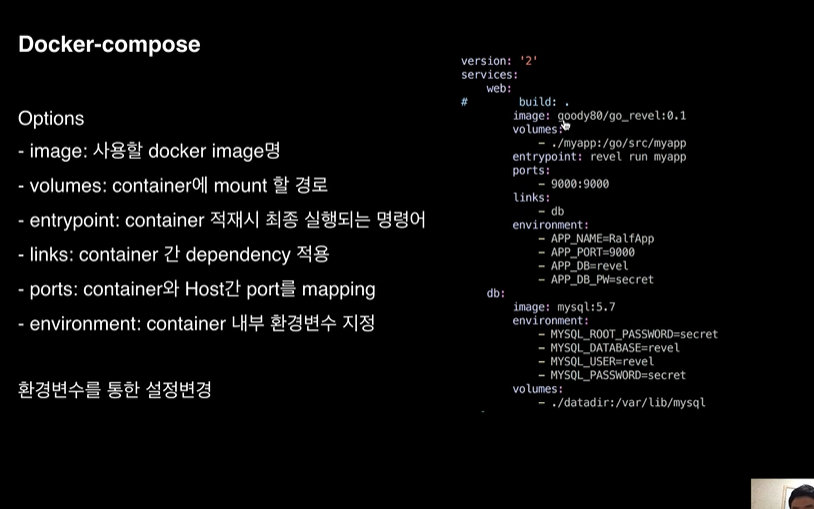
Docker build -t test ./

현재 파일에 있는 도커파일을 참고하겠다.

**주요사용법 15분 언저리 문제 – 맥북에서 돌려보기**







Docker- shared storage 성능, 네트워크 관련 호스트-포트 취약점 / 어플리케이션마다 사용할 포트가 바뀔수도 있음

도커 컨테이너 오케스트레이터– 쿠버네티스, 랜쳐 -> 포트 자동 관리

**Docker compose**는 yaml 파일로 여러 개의 도커컨테이너의 정의를 작성하여 한 번에 많은 컨테이너들을 작동시키고 관리할 수 있는 툴

**Compose란 ? (툴이다!!!)**

복수 개의 컨테이너를 실행시키는 도커 애플리케이션이 정의를 하기 위한 **툴**입니다.

Compose를 사용하면 YAML 파일을 사용하여 애플리케이션의 서비스를 구성할 수 있습니다.

그런 다음 single command를 사용하여 구성에서 모든 서비스를 만들고 시작합니다.

Compose는 모든 환경(workflow에서 production, staging, development, testing 및 CI 워크플로우)을 포함합니다.

이 글을 통해 각각의 공통 사례들을 알 수 있습니다.

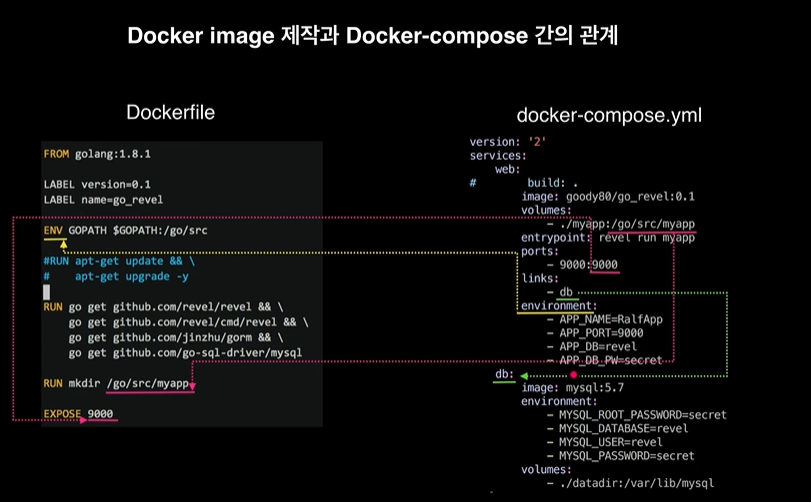
기본적으로 3스텝의 프로세스를 사용합니다.

1. 앱의 환경을 정의하여 어디에서나 재사용할 수 있는 Dockerfile을 정의합니다.

2. docker-compose.yml 에서 앱을 구성할 수 있는 서비스를 정의합니다. 그래서 단 하나의 환경ㅇ서 실행할 수 있게 합니다.

3. docker-compose up 명령어를 실행합니다. 그리고 Compose를 시작시키고 전체의 앱을 실행시킵니다

<https://scarlett-dev.gitbook.io/all/docker/untitled>

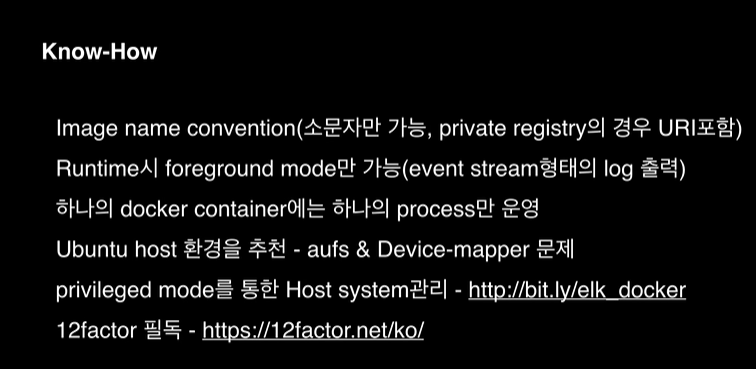


도커파일 – 도커컴포즈 관계

도커 볼륨은 이미지 파일에 연결해둔 run 경로에 마운트됨.

환경변수 지정

심화학습 1/2



노하우

-이미지 네이밍 규칙- 소문자만 가능, private registry 의 경우 uri 포함

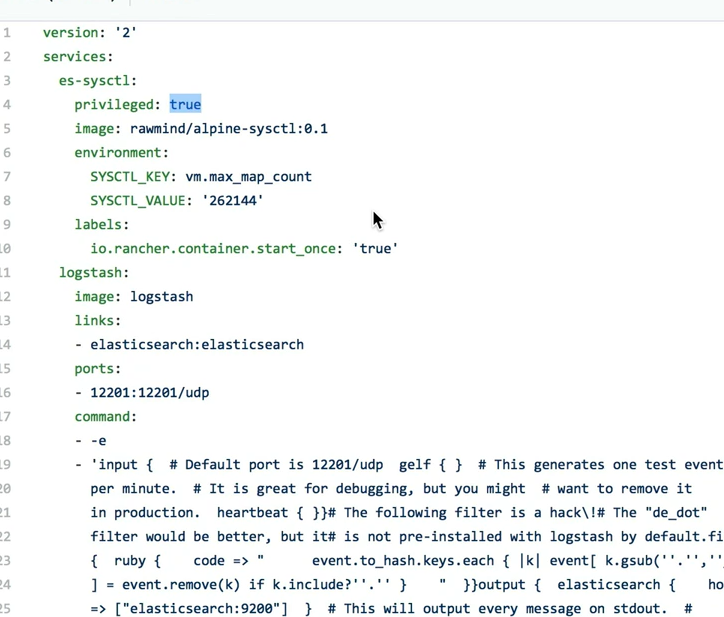
-runtime 시 foreground mode만 가능 (event stream 형태의 log 출력)

-하나의 docker container에는 하나의 process 만 운영

-Ubuntu host 환경을 추천 – aufs & Device-mapper 문제

-Privileged mode를 통한 host system 관리 <http://bit.ly/elk_docker>

-12factor 필독 – <https://12factor.net/ko/> - cloud native 기반



Privileged : 컨테이너가 올라가는 상위 호스트에 설정을 넣기 위함

심화학습 2/2

Rancher -자바 기반, tomcat 도 있음

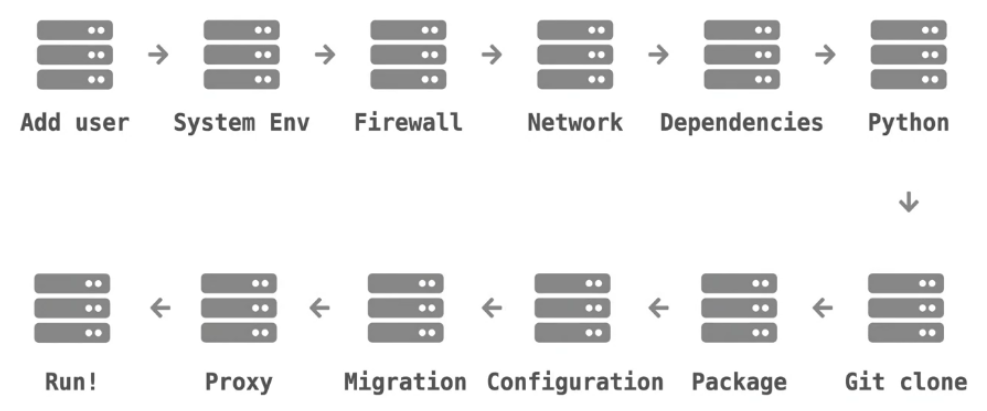
**인프런 강의 – 초보를 위한 도커 안내서**

섹션0. 도커란 무엇인가?

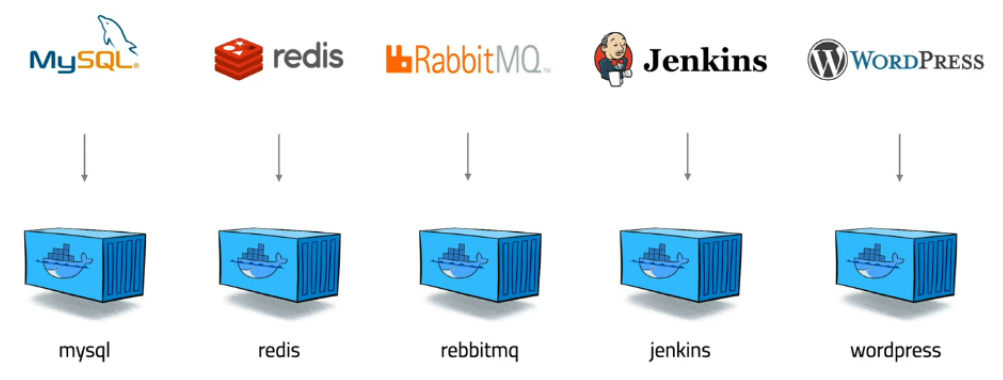
서버를 관리 한다는 것?

서버관리 방식의 변화

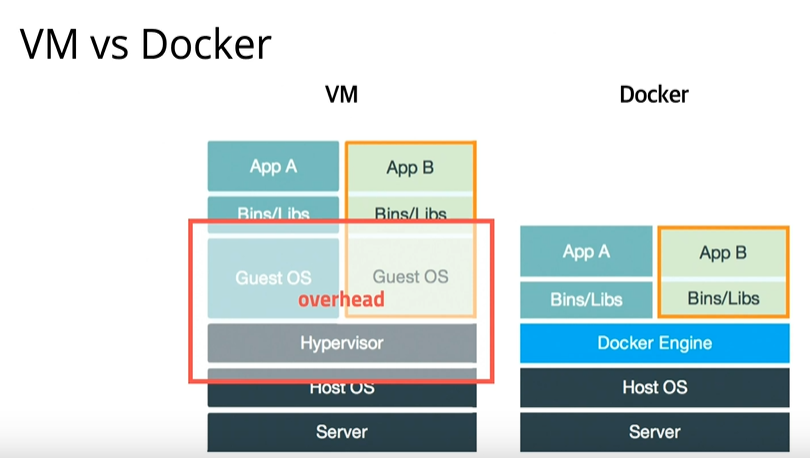
>전통적인 서버관리 방식 – 이전엔 서버 관리자가 한땀한땀



도커를 활용한 서버관리 방식



도커는 컨테이너 기반의 오픈소스 가상화 플랫폼



도커의 특징 - 확장성/이식성

- 도커가 설치되어 있다면 어디서든 컨테이너를 실행할 수 있음

- 특정 회사나 서비스에 종속적이지 않음

- 쉽게 개발서버를 만들 수 있고 테스트서버 생성도 간편함

도커의 특징 – 표준성

- 도커를 사용하지 않는 경우 ruby, nodejs, go, php로 만든 서비스들의 배포 방식은 제각각 다름

- 컨테이너라는 표준으로 서버를 배포하므로 모든 서비스들의 배포과정이 동일해짐

- capistrano? fabric? ftp? 바이바이~ 👋

도커의 특징 – 이미지

- 이미지에서 컨테이너를 생성하기 때문에 반드시 이미지를 만드는 과정이 필요

- Dockerfile을 이용하여 이미지를 만들고 처음부터 재현 가능

- 빌드 서버에서 이미지를 만들면 해당 이미지를 이미지 저장소에 저장하고 운영서버에서 이미지를 불러옴

도커의 특징 – 설정관리

- 설정은 보통 환경변수로 제어함

- MYSQL\_PASS=password와 같이 컨테이너를 띄울때 환경변수를 같이 지정

- 하나의 이미지가 환경변수에 따라 동적으로 설정파일을 생성하도록 만들어져야 함

쿠버네티스

여러대의 서버에 여러 개의 서비스 관리하기 쉽게

스케줄링

- 컨테이너를 적당한 서버에 배포해 주는 작업

- 여러 대의 서버 중 가장 할일 없는 서버에 배포하거나 그냥 차례대로 배포 또는 아예 랜덤하게 배포

- 컨테이너 개수를 여러 개로 늘리면 적당히 나눠서 배포하고 서버가 죽으면 실행 중 이던 컨테이너를 다른 서버에 띄워줌

클러스터링

- 여러 개의 서버를 하나의 서버처럼 사용

- 작게는 몇 개 안 되는 서버부터 많게는 수천 대의 서버를 하나의 클러스터로

- 여기저기 흩어져 있는 컨테이너도 가상 네트워크를 이용하여 마치 같은 서버에 있는 것처럼 쉽게 통신

서비스 디스커버리

- 서비스를 찾아주는 기능

- 클러스터 환경에서 컨테이너는 어느 서버에 생성될지 알 수 없고 다른 서버로 이동할 수도 있음

- 따라서 컨테이너와 통신을 하기 위해서 어느 서버에서 실행중인지 알아야 하고 컨테이너가 생성되고 중지될 때 어딘가에 IP와 Port같은 정보를 업데이트해줘야 함

- 키-벨류 스토리지에 정보를 저장할 수도 있고 내부 DNS 서버를 이용

20200916 도커 설치부터 실행까지