



1

2

3

4

5

6

7

8

9

10

11

12

13

{ hello cloud world; }

카카오테크 부트캠프 : 클라우드 편

카카오 AI플랫폼
리더 홍석용



강사 소개



홍석용 (데니스, dennis.hong)

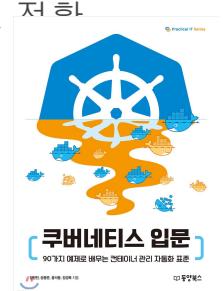
경력

- 2024, 카카오 AI 플랫폼 리더
- 2020, 카카오 클라우드네이티브 플랫폼셀 리더
- 2016, 카카오 합류, 클라우드 플랫폼 개발
- 2013, LG CNS 신입 공채, 클라우드 플랫폼 개발

주요 활동

- LG 전자 및 그룹사 클라우드 플랫폼 개발
- 카카오 클라우드 쿠버네티스 서비스 개발 리딩
- 카카오톡, 카카오택시 등 전사 서비스 클라우드 저하
- ‘쿠버네티스 입문’ 공동 저자
- 카카오 사내 AI 개발 공모전 1위 : AI Agent
- Kakao AI Platform(KAP) 개발 리딩
- 카카오 AI 교육자문 TF, 사내/외 강의

*그 외 더 궁금하신 분은 구글이나 유튜브에 '카카오 홍석용' 검색

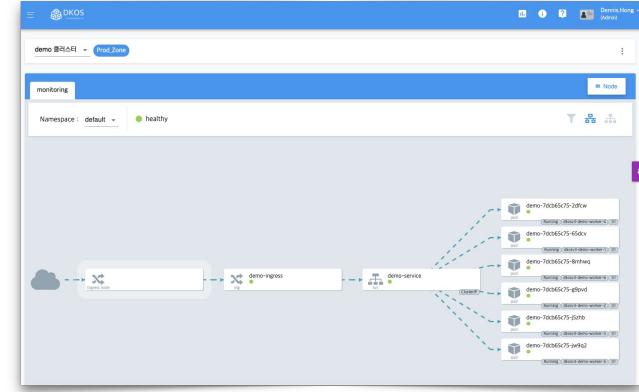
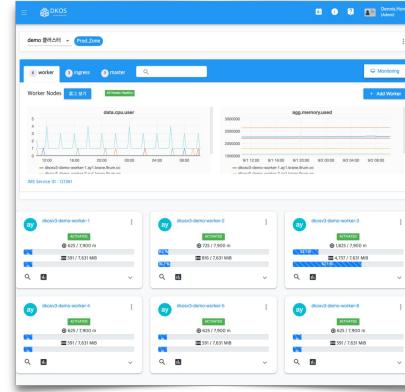




클라우드 플랫폼 개발



DKOS 개발&운영
(Kubernetes as a Service)



1M+
Containers

120k+
Servers

7k+
Clusters



관련 서비스



Kargo

(Cloud Native CI/CD)



D2hub (Private Container Image Registry)

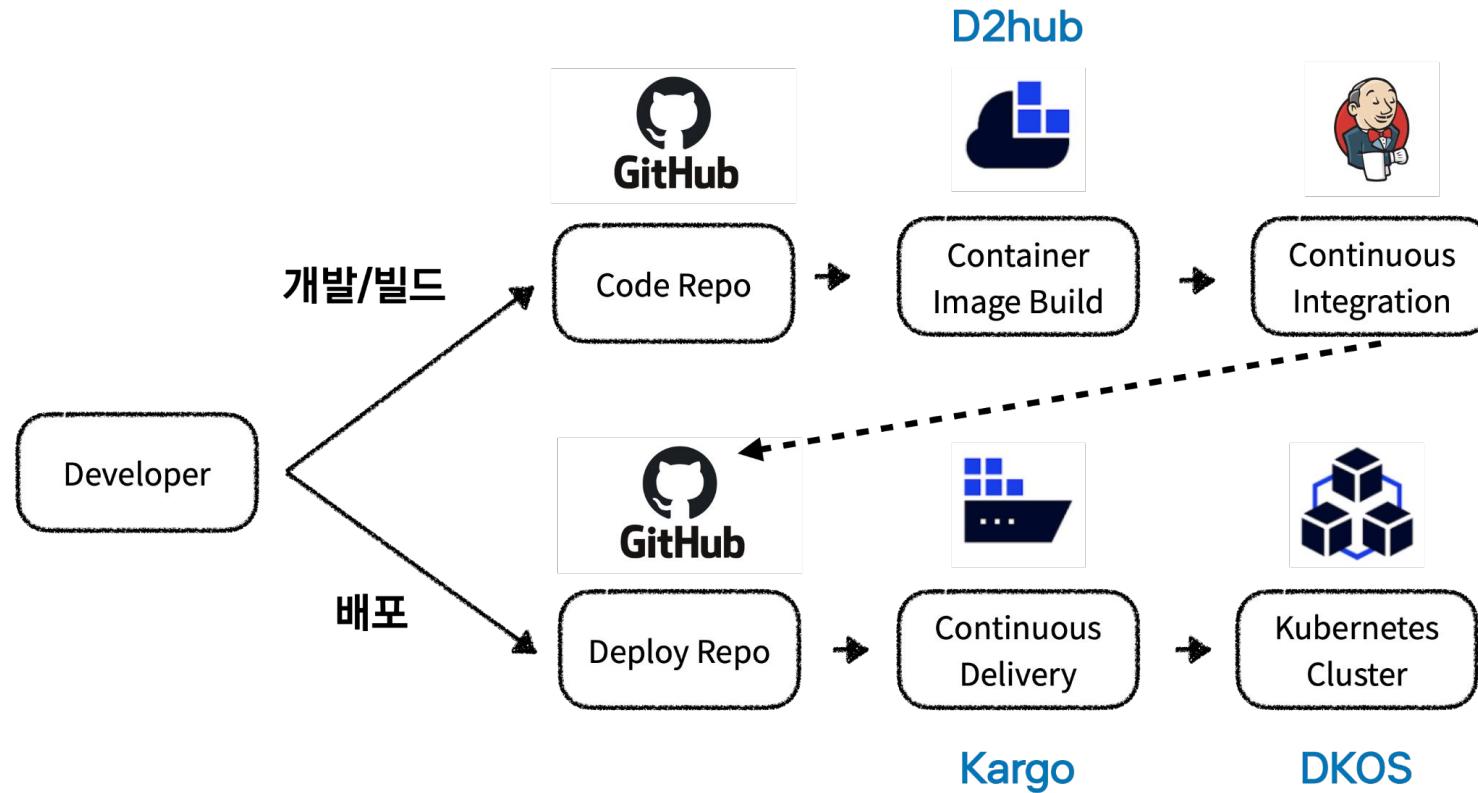
The screenshot shows the Kakao 9num Cloud Platform interface. At the top, there are tabs for Main, Admin, Guide, and API. Below the header, a breadcrumb navigation shows 'Kakao > main > 项目 > dksv3 > dksv3prod > dksv3-core'. The main content area displays the 'dksv3-prod/dksv3-core' service configuration. It includes sections for Overview, Network, Docker, Phase, and Metrics. A detailed diagram shows the service architecture with components like 'dksv3-api-spider' and 'dksv3-api-pod'. Below the diagram, a 'Build Rules' section lists three entries:

ID	Name	Git Ref Type	Pattern	Docker tag	Build Context	Dockerfile	Action	Active
42100	d120	branch	master	latest	/	/Dockerfile	<button>Build</button>	<input checked="" type="checkbox"/>
42101	d395	tag	*v*	v0.0	/	/Dockerfile	<button>Build</button>	<input checked="" type="checkbox"/>
42102	d2	tag	*v2-0.0.0*	v2.0.0	/	/Dockerfile	<button>Build</button>	<input checked="" type="checkbox"/>

At the bottom, a log viewer shows the build process for d120, starting with compressing artifacts and ending with picking revision 1008.



Cloud Native CI/CD





Cloud Native CI/CD

github repo 를 만들고 개발한 code push

```
application.py
1 from flask import Flask
2 import sys
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello():
7     return "Hello goorm!"
8
9 @app.route("/dennis/")
10 def dennis():
11     a = 1
12     b = 2
13     return str(a + b)
14
15 if __name__ == "__main__":
16     app.run(host="0.0.0.0", port=80)
17
18
19
```

```
Serving Flask app "application" (lazy loading)
Environment: production
Use a production WSGI server instead.
* Environment: production
  * Not using Werkzeug's debug mode. Do not use it in a production deployment.
  * Run with gunicorn://0.0.0.0:80/ [Press CTRL+C to quit] 172.17.0.1 -- [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 404 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET /dennis/ HTTP/1.1" 200 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET /dennis/ HTTP/1.1" 404 -
172.17.0.1 - - [04/feb/2022 08:42:15] "GET /dennis/ HTTP/1.1" 200 -
```



github.com/dennis-hong/goorm-test

dennis-hong / goorm-test Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

총 총 add .gitignore 33a291 6 minutes ago 2 commits

vscode add .gitignore 6 minutes ago

.gitignore add .gitignore 6 minutes ago

README.md test 17 minutes ago

application.py test 17 minutes ago

goorm.manifest add .gitignore 6 minutes ago

README.md

任何人都可以开发

goormIDE

Welcome to goormIDE!

goormIDE is a powerful cloud IDE service to maximize productivity for developers and teams.

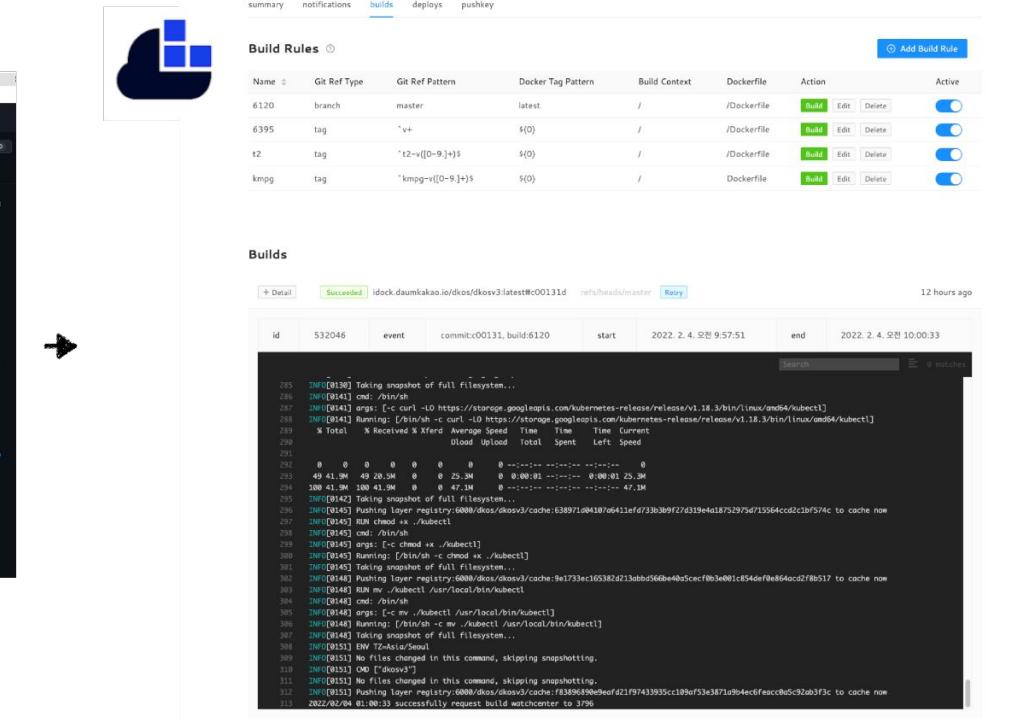
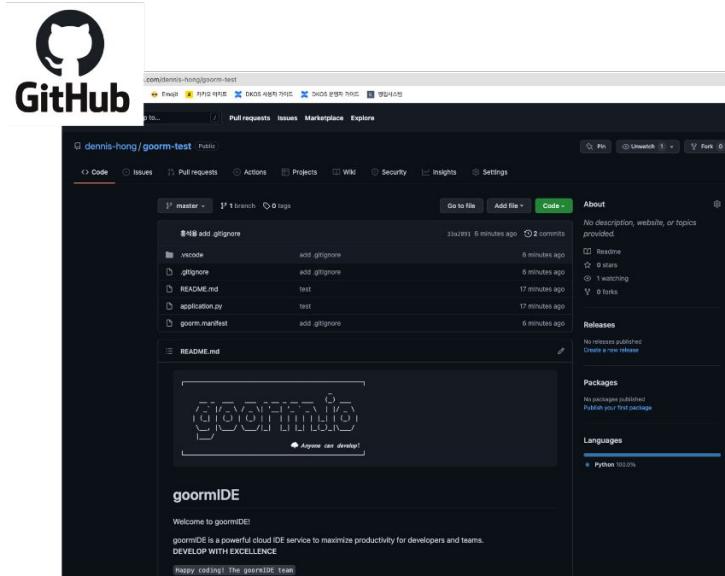
DEVELOP WITH EXCELLENCE

happy coding! The goormIDE team

Three colored circles are arranged horizontally. The first circle is red, the second is yellow, and the third is green.

Cloud Native CI/CD

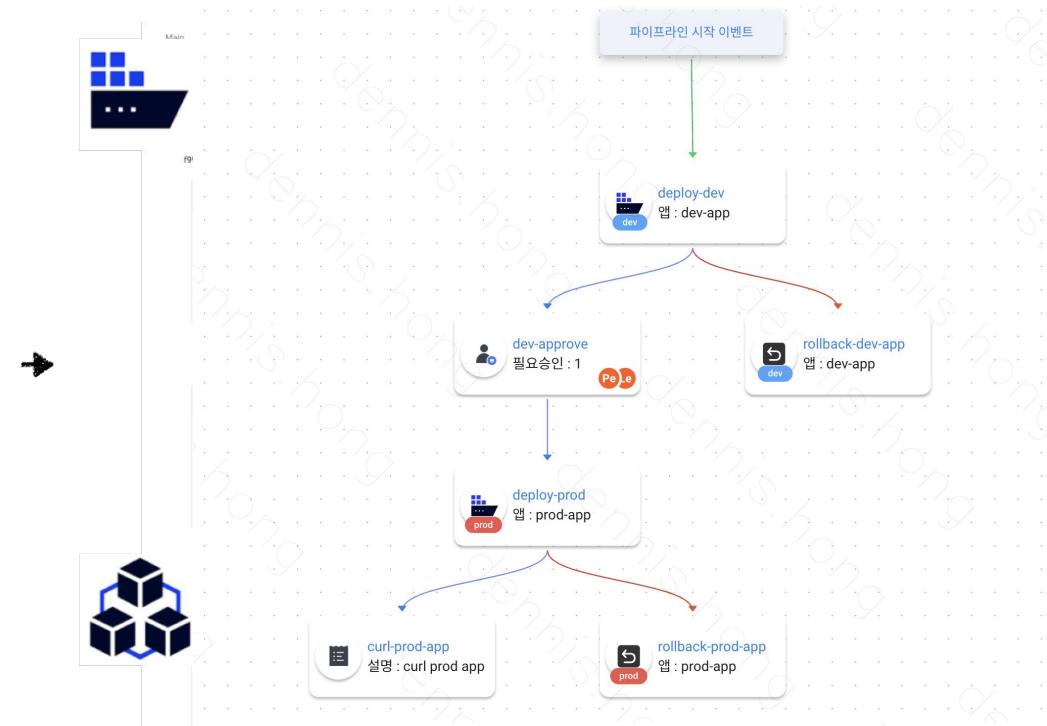
github repo 의 변경 이벤트를 받은 d2hub 가 자동 이미지 빌드





Cloud Native CI/CD

d2hub에서 이미지 빌드가 끝난 이벤트를 받아서 kargo에서 Test 및 CI/CD 파이프라인 동작.





1

2

3

4

5

6

7

8

9

10

11

12

13

{ what is cloud; }

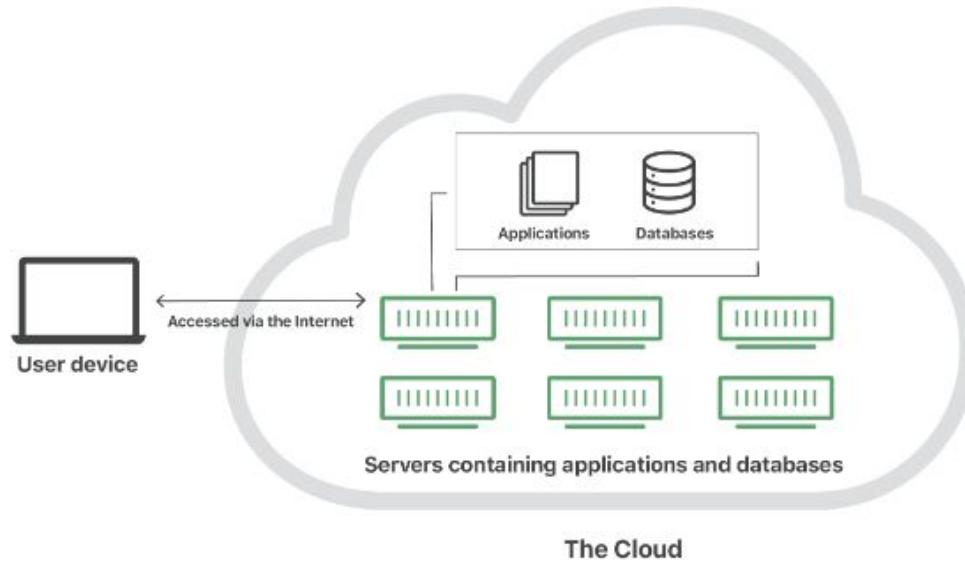
클라우드가 뭐에요?



클라우드 란?

구름(Cloud)이라는 단어가 말해주듯, 인터넷 통신망 어딘가에서 구름에 싸여 보이지 않는 컴퓨팅 자원(CPU, 메모리, 디스크 등)을 원하는 대로 가져다 쓰고 사용한 만큼 비용을 지불.

서버를 직접 구매할 때 고려해야 할 전력, 위치, 서버 세팅, 확장성을 고민하지 않고 서비스 운영에만 집중





하지만, 클라우드의 단점도 있음

- 2018.11.22 : 아마존웹서비스(AWS) 장애..쿠팡·배달의 민족 등 한때 접속불가, 헤럴드경제.
- 2019.03.13 : 구글, 이메일·클라우드 서비스 일시 장애 발생, SBS 뉴스.
- 2020.11.26 : AWS 클라우드 장애, 기업 고객들 '타격', 아이뉴스24.
- 2021.12.08 : AWS, 美 동부 장애로 통신 대란, 전자신문.
- 2022.02.07 : AWS 클라우드 20여분간 장애..배민 등 한때 먹통, 이데일리.
- 2023.01.28 : [글로벌] MS 클라우드 '애저' 7시간 먹통..."전세계 클라우드 불신론 우려, TechM.
- 2024.07.19 : '전 세계 IT 대란'...MS 클라우드 장애로 방송·통신·금융 서비스 차질

<매거진> IT 기술 기사 목록

[한경비즈니스](#)

'전세계 IT 대란'...MS 클라우드 장애로 방송·통신·금융 서비스 차질

미국 마이크로소프트(MS)의 클라우드 서비스에서 장애가 발생하면서 미국과 유럽, 일본 등 전 세계 곳곳과 한국에서도 항공편 운항이 중단되고 방송과 통신, 금융 서...

2024.07.19

MBC | 2024.07.20
MS 클라우드 장애 'IT 대란'..항공·통신 마비

세계일보 | 2024.07.19
MS '클라우드 서비스 장애'... 세계 각국 서 항공·방송·통신 대란

아오
방송 등

MBC 언론사 라
MS 클라우드 장애로 국내 항공·게임업계 피해..국내 기업·공공기관 영향 없어

미국 마이크로소프트, MS의 클라우드 서비스 장애가 발생하면서, 일부 저비용항공사 발권과 예약 시스템, 국내 온라인 게임에 잇따라 장애가 발생했습니다. 인천공...

2024.07.19

SBS 뉴스 디자인 컨텐츠 시사 재데크 라이프 기자
주요뉴스 기사는 제한된 권역에 적용됩니다. 마지막면 맥스
종합뉴스 나스닥 12000.015

제보

구글, 이메일·클라우드 서비스 일시 장애 발생..."3시간만 해결"

마일 전송이나 파일 다운로드 오류 발생

김태원 기자 | 입력 2023.03.13 20:09 수정 2023.03.13 21:15

인터넷

[글로벌] MS 클라우드 '애저' 7시간 먹통..."전세계 클라우드 불신론 우려"

2018.11.22 11:21

Amazon Web Services(AWS) 장애..쿠팡·배달의 민족 등 한때 접속불가

[헤럴드경제=김상수·정태일·정윤경 기자] 쿠팡, 마켓컬리, 달고마켓, 카카오스타트, 배달의 민족, 암호화폐 거래소 업비트, 코인원 등 국내 웹사이트와 애플리케이션(앱) 등의 접속이 한 때 원활하지 않았다. 이들 웹사이트는 '502 Bad Gateway'라는 에러코드가 뜨며 접속이 차단되거나 끊겼다.

웹사이트 접속 장애를 겪은 업체들은 글로벌 클라우드 서비스 아마존웹서비스를 사용한다. 때문에 업계에서는 아마존웹서비스의 국내 네트워크에 장애가 발생한 것으로 추정하고 있다.

마이
관계자
화자





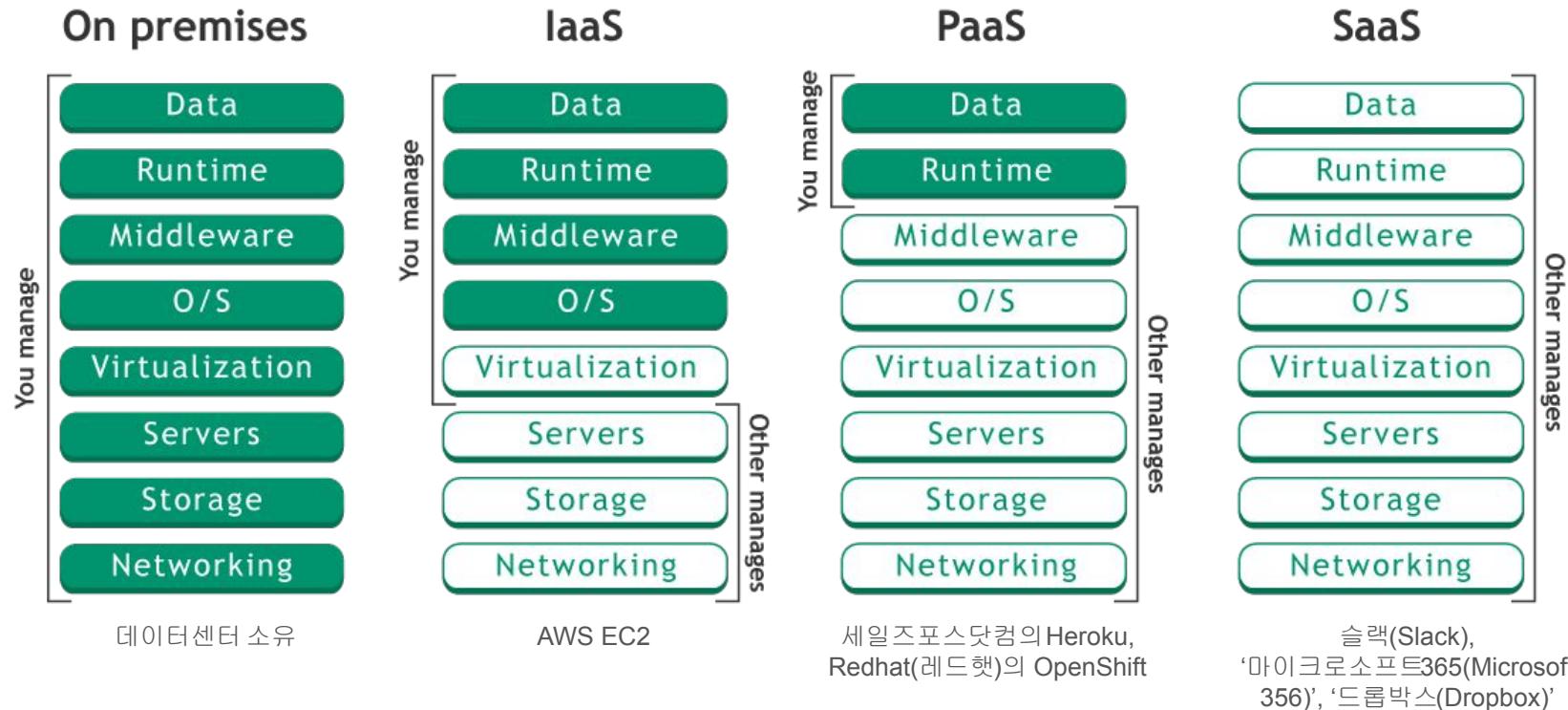

하지만, 클라우드의 단점도 있음

- 1) 장애 종속성: 클라우드 제공업체의 장애 또는 중단으로 인한 영향
- 2) 보안 위험: 조직 외부에 저장된 애플리케이션과 데이터, 클라우드 제공업체의 높은 보안이 필요함.

잘 알려진 글로벌 클라우드 서비스도 장애가 발생할 수 있으므로 멀티 클라우드 전략도 검토하는 추세. 제공업체마다 보안, 안정성, 가격이 달라 성공적인 클라우드 도입을 위해서는 신중한 검토와 계획이 필요.

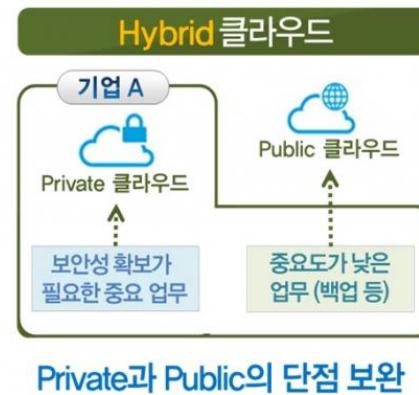
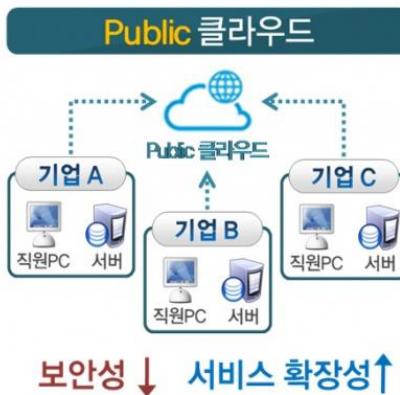


클라우드 란?



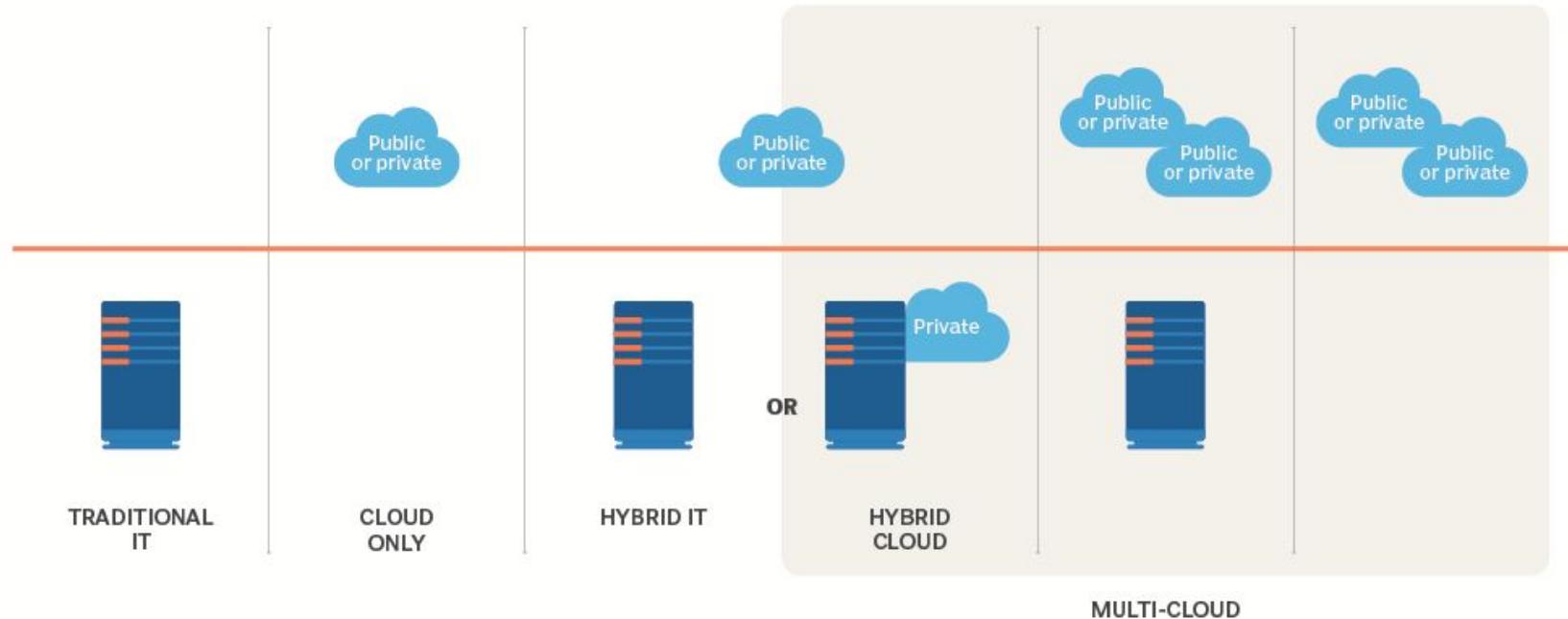


클라우드 란?





클라우드 란?



클라우드 네이티브 애플리케이션 개발

이란?
what is cloud native

클라우드 네이티브 란?

클라우드의 이점을 최대한 활용할 수 있도록 애플리케이션을 구축하고 실행,
배포하는 방식

클라우드 네이티브의 목표 : 변화하는 비즈니스 요구 사항에 빠르게 적응할 수 있는 유연하고 가용성이 높으며 확장 가능한
소프트웨어 제공

클라우드 네이티브로의 변화

[소프트웨어 아키텍처]
모놀리틱 ⇨ 서비스중심 ⇨
マイ크로서비스

[일하는 방식]
워터폴 ⇨ 애자일 ⇨ 데브옵스

[개발 방식]
수동 빌드/배포 ⇨
CI/CD

[인프라]
물리서버 ⇨ 가상서버 ⇨
컨테이너

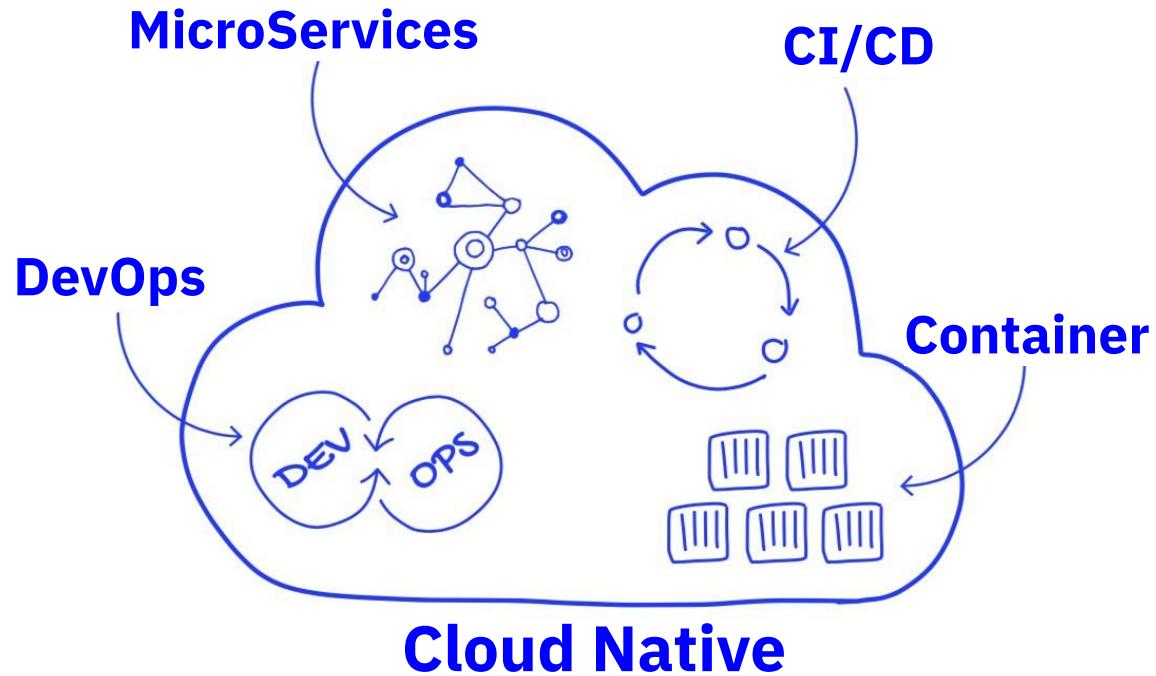
비즈니스 요구 사항

변화하는 불확실한 시장
상황에 빠르고 효율적으로
대응해야 함

= 빠른 릴리즈와 유연한 변경
필요

클라우드 네이티브 란?

클라우드의 4가지 특성



CI/CD 란?

CI/CD = 지속적 통합(Continuous Integration, CI) + 지속적 배포(Continuous Deployment, CD)
코드 변경 사항이 자동으로 빌드, 테스트, 통합되어 서비스 환경에 배포되는 과정이 자동화된
개발 방식

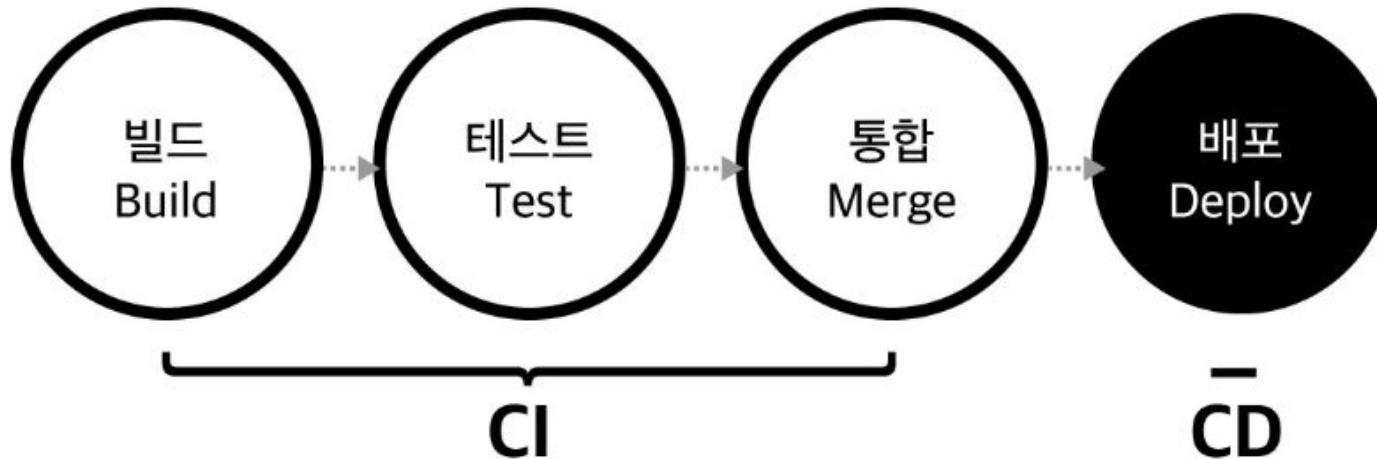


Jenkins



GitHub Actions

CI/CD 파이프라인 자동화



CI

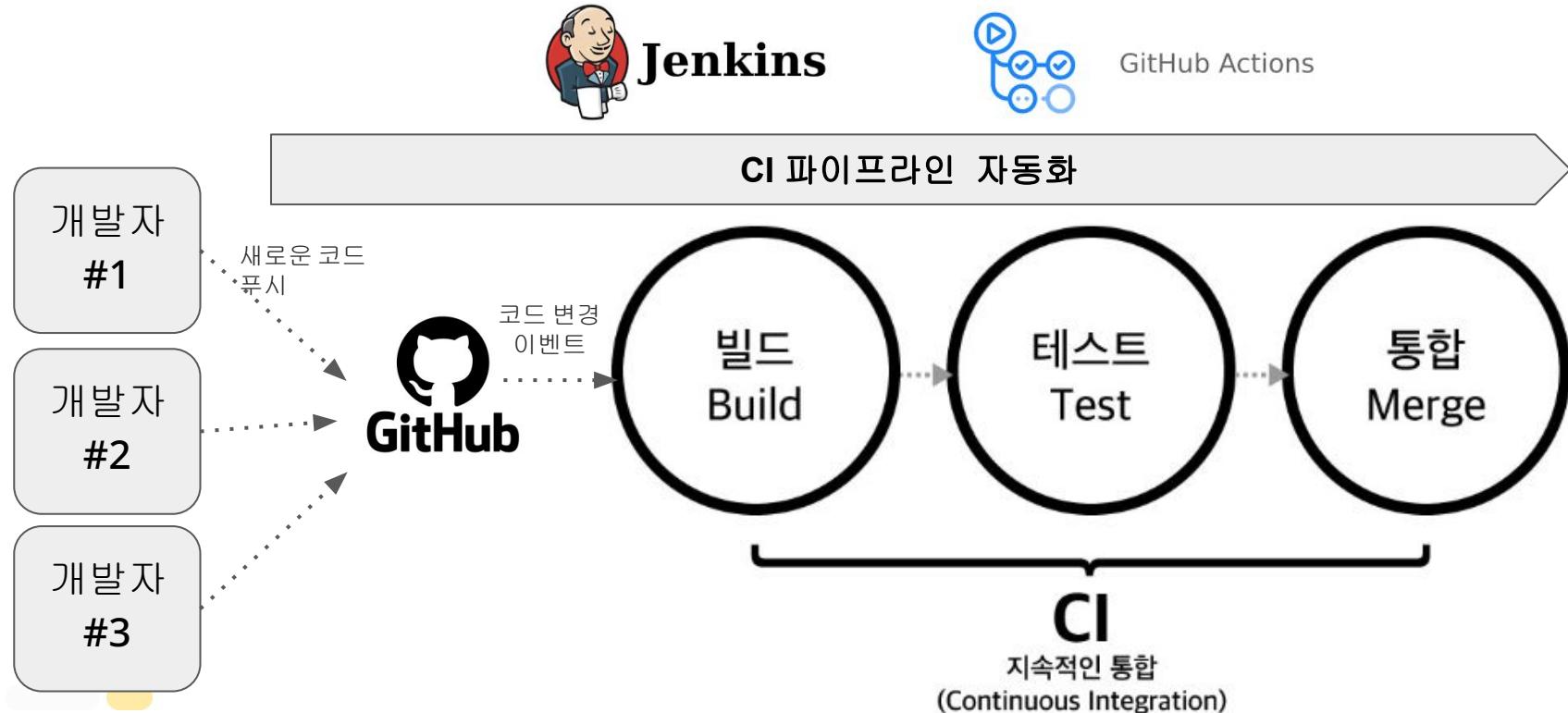
지속적인 통합
(Continuous Integration)

CD

지속적인 전달/배포
(Continuous Delivery/Deployment)

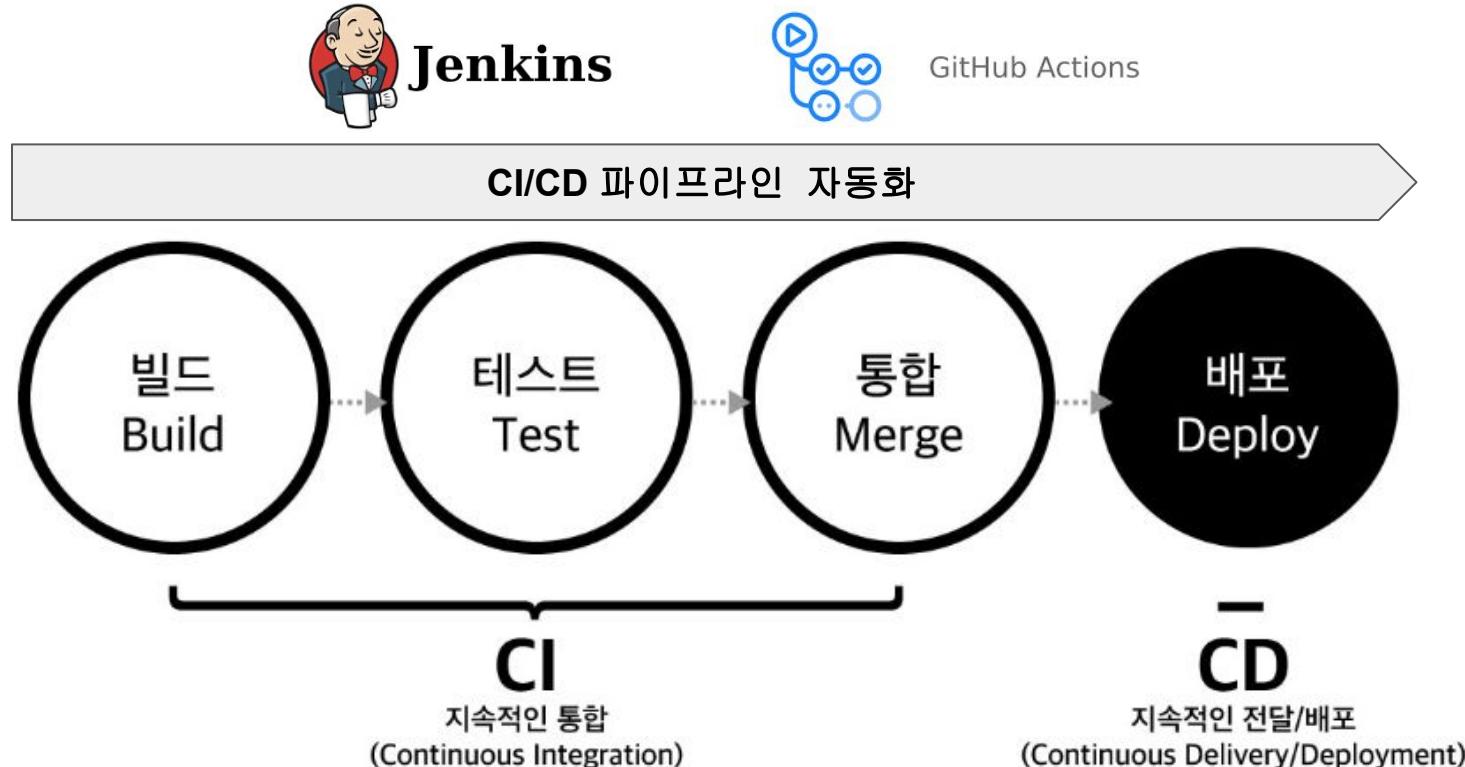
CI (Continuous Integration) 란?

CI : 빌드/테스트/통합의 자동화



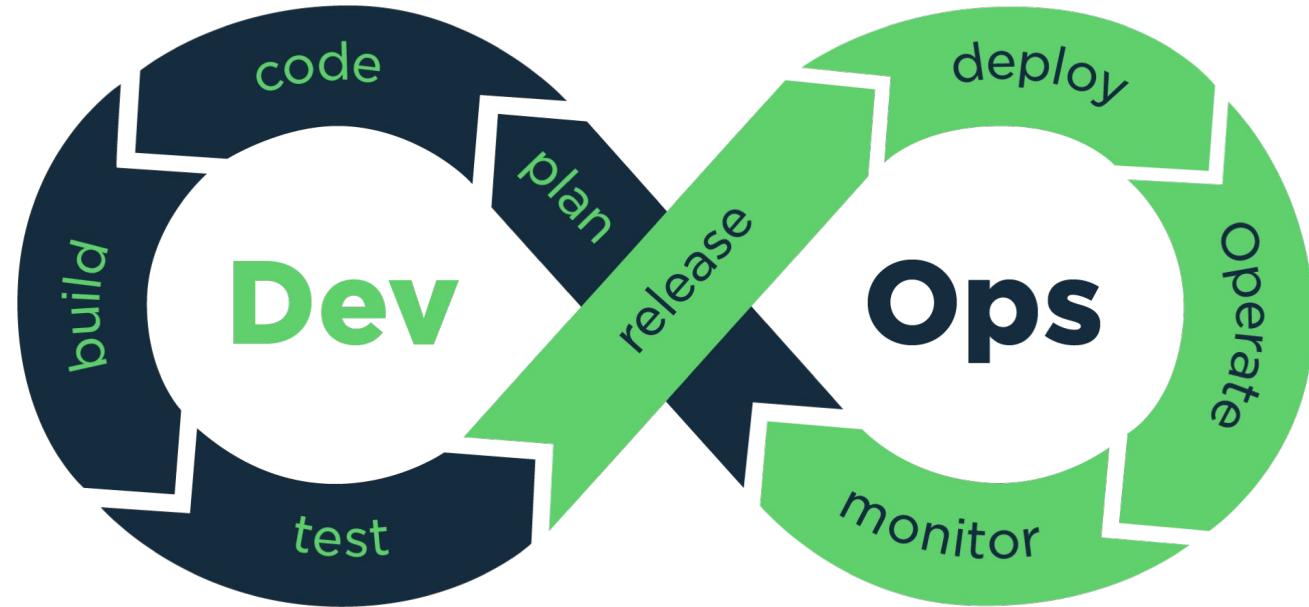
■ CD (Continuous Deployment) 란?

CD : 배포의 자동화



DevOps 란?

DevOps는 소프트웨어 개발(Dev)과 운영(Ops)을 유기적으로 통합하여 개발 라이프 사이클을 단축하는 동시에 비즈니스 목표를 달성할 수 있도록 빈번한 업데이트를 제공하는 것을 목표로 하는 소프트웨어 엔지니어링 문화



DevOps 의 핵심 원칙

협업 및 커뮤니케이션

개발자, 테스터, 운영 팀 간의 정기적인 커뮤니케이션과 조정을 통해 모든 사람이 변경 사항, 위험 및 개선 기회를 인지. 부서 간의 사일로를 허물고 응집력 있는 팀으로 협력하여 소프트웨어 업데이트를 더 빠르고 효율적으로 제공.

자동화

예를 들어 코드 빌드 및 테스트, 애플리케이션 배포, 인프라 관리와 같은 작업 자동화. 자동화를 통해 소프트웨어 업데이트를 더 빠르고 더 높은 품질로 제공. 가능한 한 많은 작업을 자동화하여 인적 오류를 줄이고 효율성을 개선하며 일관되고 반복 가능한 결과를 보장하여 프로세스 속도를 높이는 것.

지속적인 개선 및 낭비 최소화

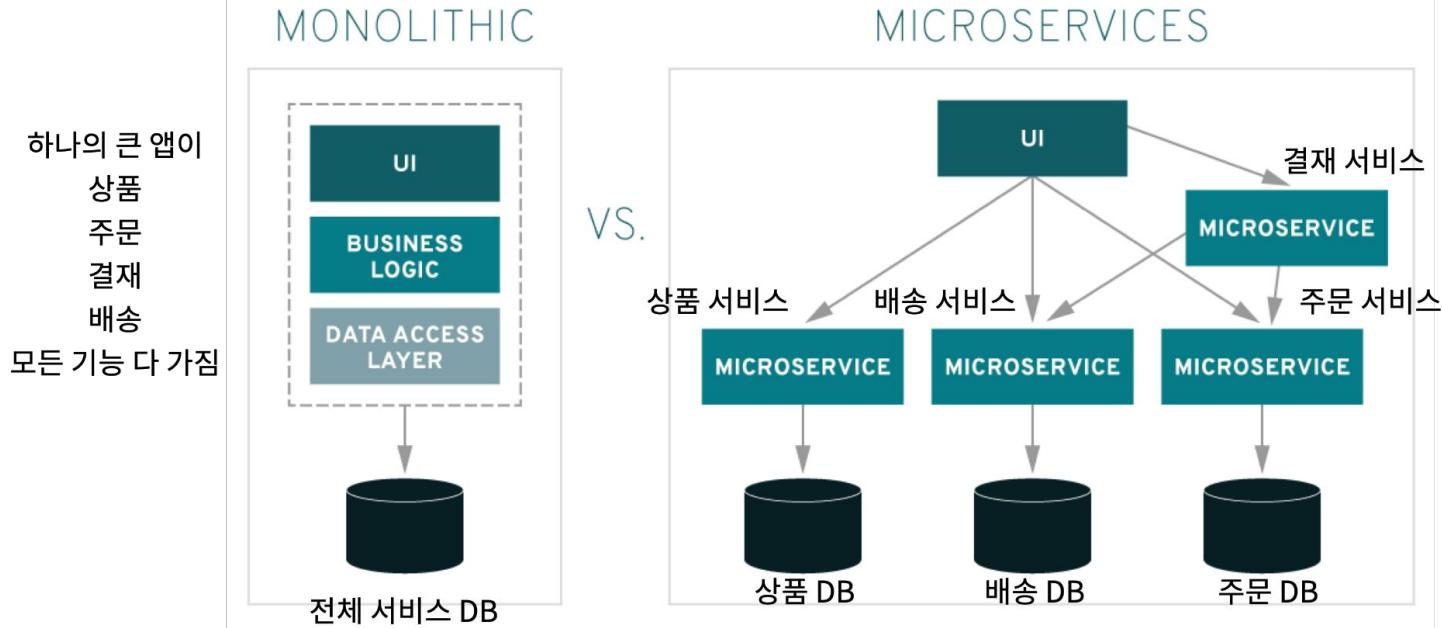
데이터와 피드백을 사용하여 프로세스와 시스템을 지속적으로 평가하고 효율성을 개선하고 낭비를 최소화. 이러한 활동은 실험, 데이터 분석 및 피드백을 통해 이루어지며, 이를 통해 개선이 필요한 영역과 프로세스와 시스템을 최적화할 수 있는 기회를 파악

짧은 피드백 루프를 통해 사용자 요구 사항에 집중

정기적으로 사용자와 함께 변경 사항을 테스트 및 검증하고 사용자의 피드백을 개발 프로세스에 통합하여 고객의 요구를 충족하는 소프트웨어를 만들고 사용자 피드백을 기반으로 지속적으로 개선

マイクロ서비스 アーキテクチャ(MSA) 란?

대규모 애플리케이션이 API를 통해 서로 통신하는 작고 독립적인 서비스 모음으로 구성되는
소프트웨어 아키텍처

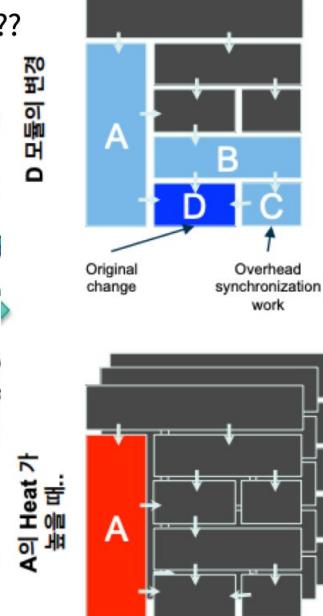
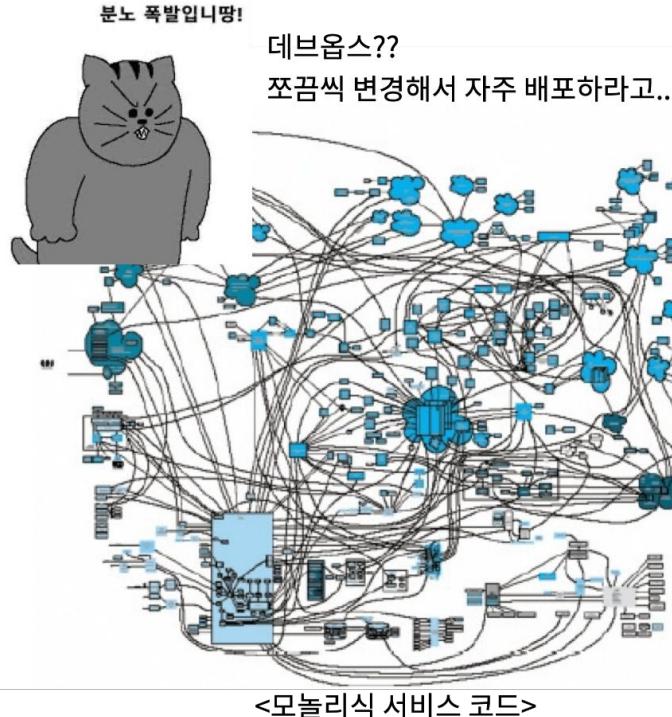


소프트웨어 애플리케이션을 독립적으로 배치 가능한
작은 서비스 조합(suite)으로 설계하는 방식



マイクロサービスアーキテクチャ(MSA)란?

모놀리식 아키텍처에서의 데브옵스 : 분노 폭발



모놀리틱 구조의 한계

D모듈 변경 시 A,B,C의 추가 테스트 필요

D모듈의 작은 변화도 전체 App의 배포를 요구

D모듈만 테스트하고는 릴리즈 할 수 없음

A,B,C,D모들이 동시에 변경할 경우 엄청난 부가적인 작업이 필요

결론적으로 매우 느린 릴리즈 사이클이 형성

A 모듈이 집중적으로 많이 사용되는 경우에도 어쩔 수 없이 전체 App을 Scale 해야 함
(메모리, Disk 등의 비효율성)

A 모듈이 문제가 생기면 장애가 나머지 모듈에도 영향을 줄 수 밖에 없음

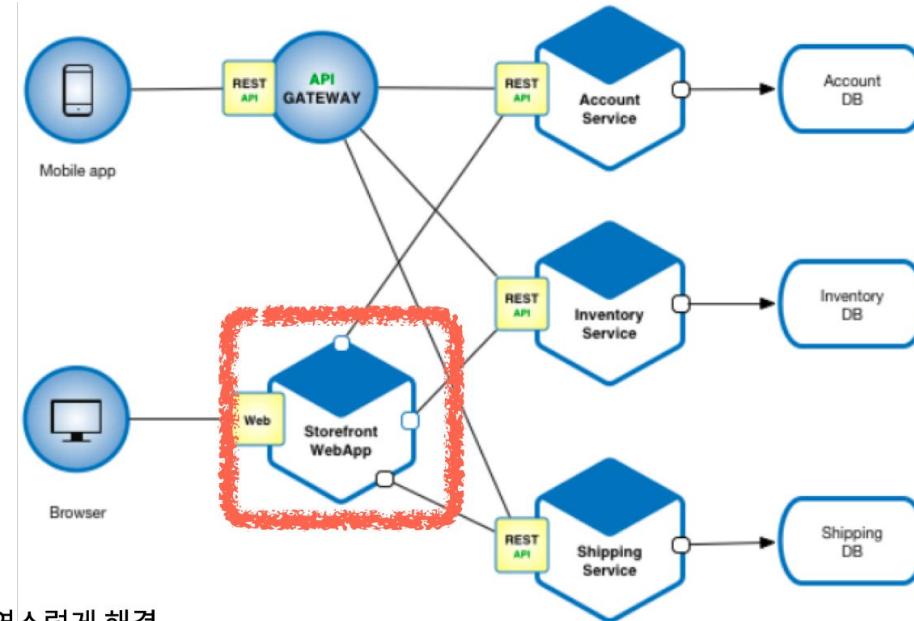


マイクロサービス アーキテクチャ(MSA) は?

マイクロサービス アーキテクチャ에서의 데브옵스 : 좋습니당!



이 작은 애플리케이션
코드만 바꿔서
배포하면 되겠네



쪼개서 집중하면 많은 문제가 자연스럽게 해결

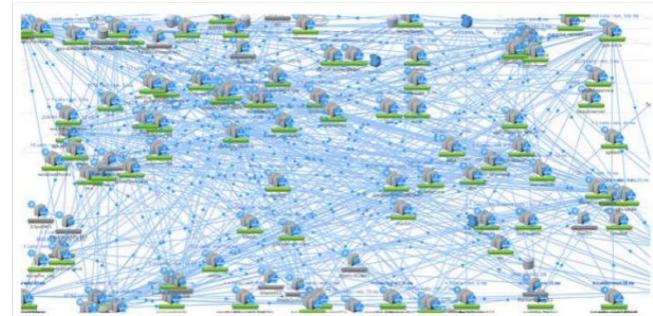
새로운 기술의 도입도, 고가용성과 무중단에 대한 고민도, 장애가 났을 때의 대처도 모두 비교할 수 없게 수월
소규모 팀이 주도적으로 서비스를 발전시키는 것을 용이



マイクロサービス ア키텍처(MSA) 란?

하지만, 마이크로서비스도 장점만 있는 것은 아님.

- 분산된 수 백, 수 천 개의 작은 앱을 관리해야 함
- 앱 별 다양한 런타임 환경, 성능 간섭
- 잦은 빌드/배포, 설정 관리 복잡
- 트래픽 분산 관리, 서비스 디스커버리 등



절망입니땅...



컨테이너(Container) 란?

컨테이너는 실행 환경과 성능을 격리, 표준화.

애플리케이션을 OS 환경까지 가상화해서 패키징

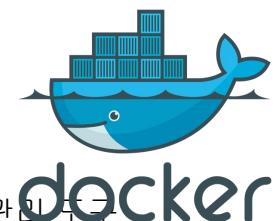
컨테이너는 OS 환경까지 패키징되었기 때문에 **Linux, Windows, Mac**, 가상 머신, 베어메탈, 개발자의 컴퓨터, 데이터 센터, 온프레미스 환경, 퍼블릭 클라우드 등 사실상 어느 환경에서나 구동되므로 개발 및 배포가 쉬워짐

애플리케이션의 성능 격리

컨테이너는 **CPU, 메모리, 스토리지, 네트워크 리소스**를 OS 수준에서 가상화하여 개발자에게 기타 애플리케이션으로부터 논리적으로 격리된 OS 샌드박스 환경을 제공합니다.

도커(Docker)

컨테이너를 실행시키거나 컨테이너 이미지로 빌드하는 과정에서 가장 많이 사용하는 컨테이너 관리 도구

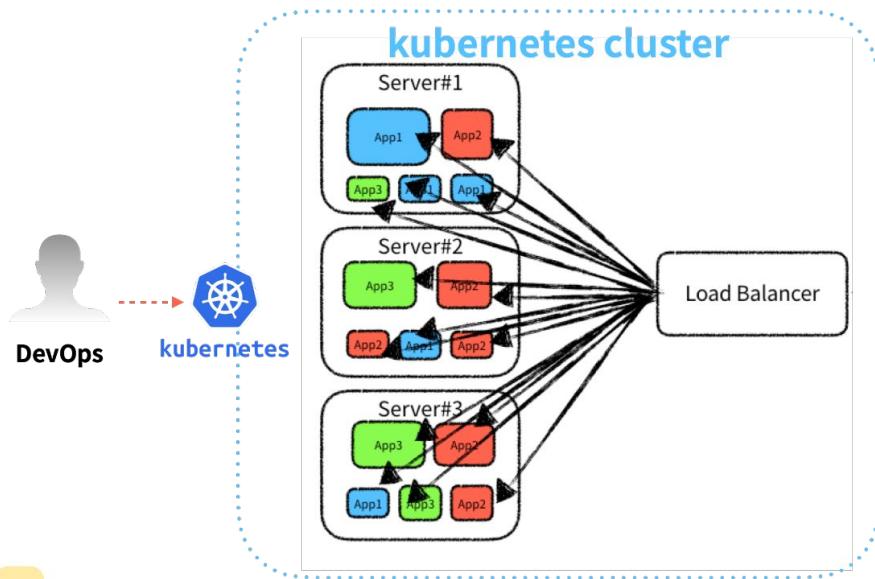


컨테이너(Container) 란?

마이크로서비스의 단점을 극복하는 기술 : 컨테이너와 쿠버네티스

- 분산된 수 백, 수 천 개의 작은 앱을 관리해야 함
- 앱 별 다양한 런타임 환경, 성능 간섭
- 짧은 빌드/배포, 설정 관리 복잡
- 트래픽 분산 관리, 서비스 디스커버리 등

→ Container : 다양한 App 환경을 OS 까지 Image화 배포, 성능 격리
→ Container Orchestration : 사실상의 표준 kubernetes



kubernetes가
컨테이너 단위의 어플리케이션과
로드밸런서 연결을
자동으로 제어하며
서비스 중단 없는 배포를 자동화.

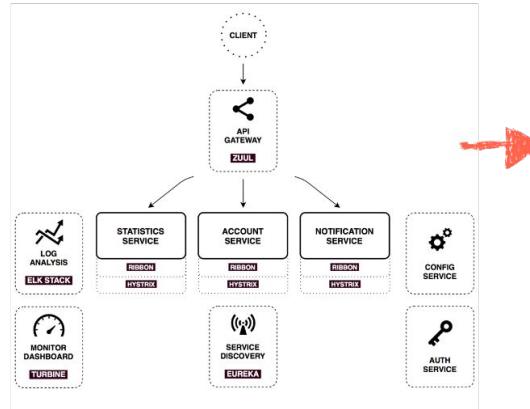


클라우드 기술로 MSA의 단점 극복

マイクロ서비스의 복잡성을 클라우드 환경에서 해결

before cloud : 마이크로서비스를 구현하기 위한 인프라를 어플리케이션에서 전부 구현

after cloud : 비지니스 로직을 제외한 인프라 역할은 클라우드로 이관



Microservices Concern	Spring Cloud & Netflix OSS	Kubernetes
Configuration Management	Config Server, Consul, Netflix Archaius	Kubernetes ConfigMap & Secrets
Service Discovery	Netflix Eureka, Hashicorp Consul	Kubernetes Service & Ingress Resources
Load Balancing	Netflix Ribbon	Kubernetes Service
API Gateway	Netflix Zuul	Kubernetes Service & Ingress Resources
Service Security	Spring Cloud Security	-
Centralized Logging	ELK Stack (Logstash)	EFK Stack (Fluentd)
Centralized Metrics	Netflix Spectator & Atlas	Heapster, Prometheus, Grafana
Distributed Tracing	Spring Cloud Sleuth, Zipkin	OpenTracing, Zipkin
Resilience & Fault Tolerance	Netflix Hystrix, Turbine & Ribbon	Kubernetes Health Check & resource isolation
Auto Scaling & Self Healing	-	Kubernetes Health Check, Self Healing, Autoscaling
Packaging, Deployment & Scheduling	Spring Boot	Docker/Rkt, Kubernetes Scheduler & Deployment
Job Management	Spring Batch	Kubernetes Jobs & Scheduled Jobs
Singleton Application	Spring Cloud Cluster	Kubernetes Pods

java : spring cloud framework

cloud : kubernetes

■ 현대 비즈니스 환경에서 살아남기 위한 경쟁력, 클라우드 네이티브

클라우드의 이점을 최대한 활용할 수 있도록 애플리케이션을 구축하고 실행,
배포하는 방식

클라우드 네이티브의 목표 : 변화하는 비즈니스 요구 사항에 빠르게 적응할 수 있는 유연하고 가용성이 높으며 확장 가능한
소프트웨어 제공

클라우드 네이티브로의 변화

[소프트웨어 아키텍처]
모놀리틱 ⇨ 서비스중심 ⇨
マイ크로서비스

[일하는 방식]
워터폴 ⇨ 애자일 ⇨ 데브옵스

[개발 방식]
수동 빌드/배포 ⇨
CI/CD

[인프라]
물리서버 ⇨ 가상서버 ⇨
컨테이너

비즈니스 요구 사항

변화하는 불확실한 시장
상황에 빠르고 효율적으로
대응해야 함

= 빠른 릴리즈와 유연한 변경
필요

현대 비즈니스 환경에서 살아남기 위한 경쟁력, 클라우드 네이티브

여러분이 최근 핫한 **ChatGPT** 같은 AI 기술을 자사 서비스에 빠르게 도입해야 하는 회사의 개발자라고 가정해 봅시다.

- 1) 만약 애플리케이션이 MSA 구조가 아니라 거대한 모놀리틱 구조이거나,
- 2) 컨테이너화되어 클라우드 환경에 올라가 있지 않고 물리 서버 OS에 모두 직접 설치되어 있다면?
- 3) CI/CD 파이프라인이 자동화되어 있지 않고 빌드, 테스트, 배포 과정이 모두 수작업으로 이루어진다면?
- 4) 개발 조직과 운영, 테스트 조직이 서로 소통하지 않고 독립적으로 일을 한다면?

자사 서비스에 AI 기술을 빠르게 도입하고 다양하게 시도와 실험을 하며 빠르게 적응할 수 있을까요?

클라우드 네이티브 애플리케이션 개발 환경을 갖춘 경쟁사와 경쟁에서 승리할 수 있을까요?

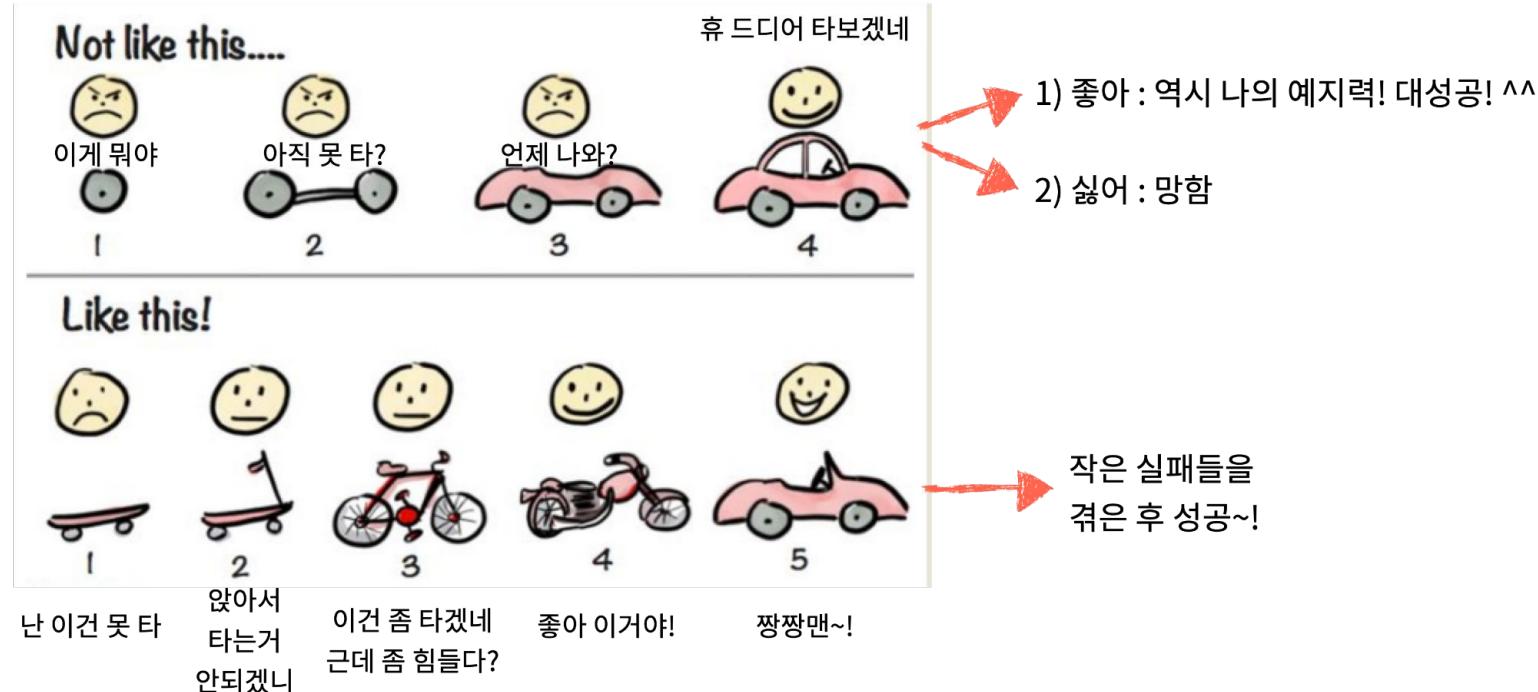
이러한 상황을 상상해 보면, 이것은 매우 어려운 일이 될 것입니다.



비지니스 요구사항과 소프트웨어 개발 방법론의 진화

1) 복잡하고 불확실한 비지니스 환경과 시장 니즈

뭐가 담인지 모름. 일단 작게 만들어서 빠르게 출시하고 시장 반응을 보고 적절히 맞춰감(애자일)

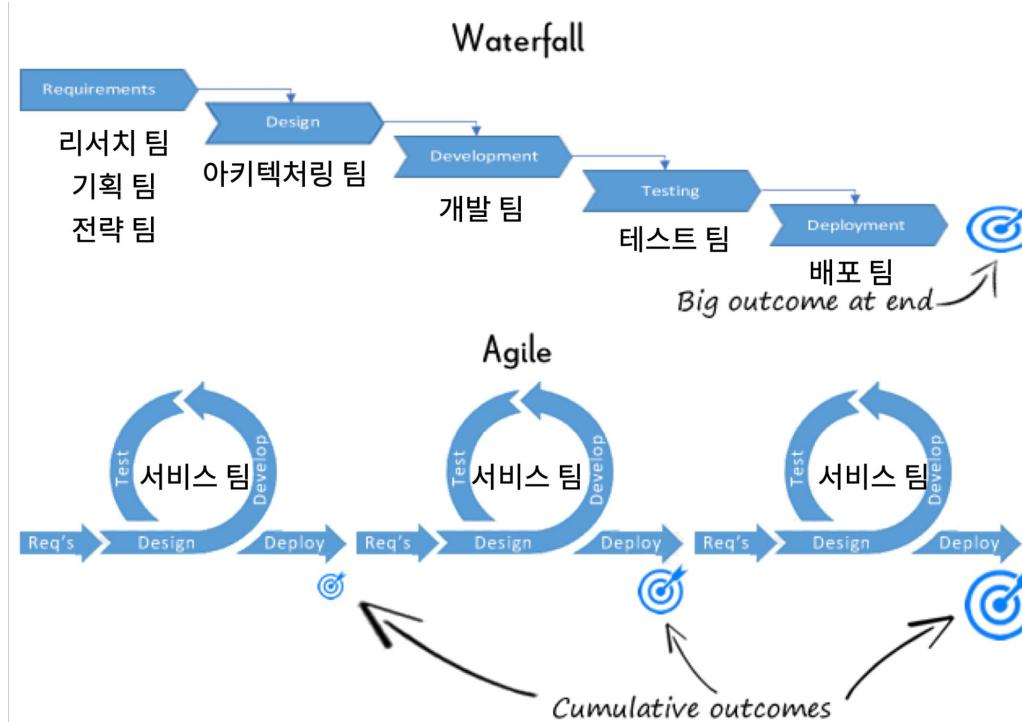




비지니스 요구사항과 소프트웨어 개발 방법론의 진화

2) 빠르게 변화하는 비지니스 환경과 시장 니즈

유연한 변경과 빠른 전달이 필요함(time to market)





비지니스 요구사항과 소프트웨어 개발 방법론의 진화

비지니스 요구사항

빠르고 유연한 변경
빈번한 배포 필요

데브옵스(DevOps)

일하는 방식과 문화, 도구를 바꾸자~

워터폴 -> 애자일 -> 데브옵스

マイ크로서비스(Microservice)

소프트웨어 아키텍처도 민첩한 대응을 할 수 있게 바꾸자~

모놀리틱 -> 서비스지향 -> 마이크로서비스

클라우드(Cloud)

인프라를 유연하고 확장 가능하게 바꾸자~

물리 서버 -> 가상 머신 -> 컨테이너(도커) -> 오케스트레이터(쿠버네티스)



1

2

3

4

5

6

7

8

9

10

11

12

13

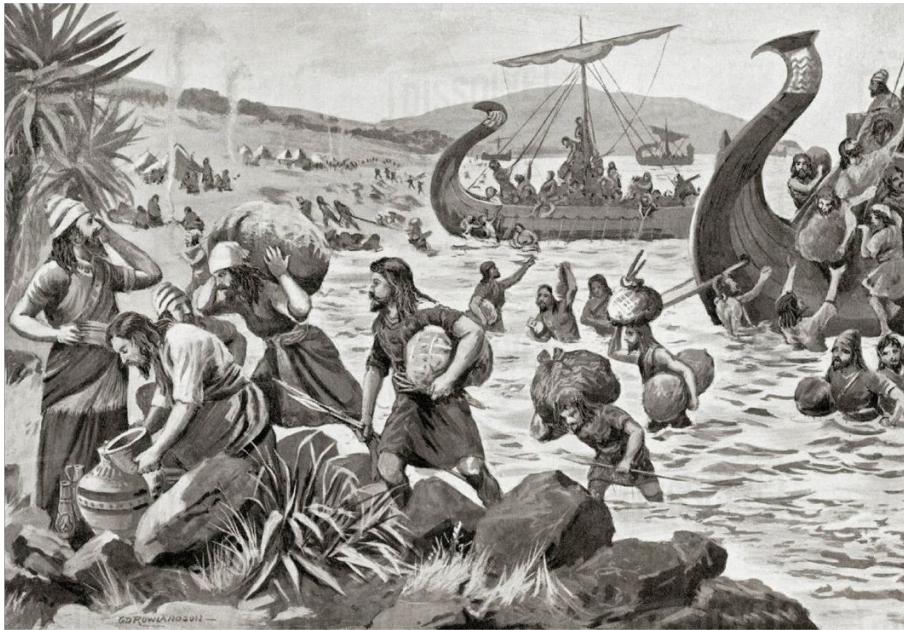
{ container; }

그림으로 설명하는 베이직 컨셉



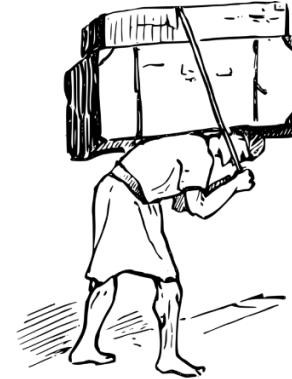
컨테이너란?

국제 물류에서의 컨테이너 혁명



고대 지중해의 거상 페니키아 상인

- 인류는 3500년 동안 화물을 일일이 사람의 힘으로 배에싣고 내리는 수작업에 의존
- 전체 운송시간의 3분의 2가 선적과 하역에 소요



Docker (부두노동자)



컨테이너란?

국제 물류에서의 컨테이너 혁명



1956년, 시랜드 사는 컨테이너 방식을 통해 화물 운송비를 무려 97% 절감

시간과 비용의 획기적 절감이 가능해지자 항만과 해운사들은 빠르게 컨테이너 방식을 받아들였고 불과 15년 만에 수 천 년을 이어온 해상무역의 모습이 변화

컨테이너의 빠른 확산에는 시랜드가 관련 특허를 무상으로 공개하고 여러 관련 기관들이 참여와 활용을 촉진하는 플랫폼을 만든 점이 크게 기여

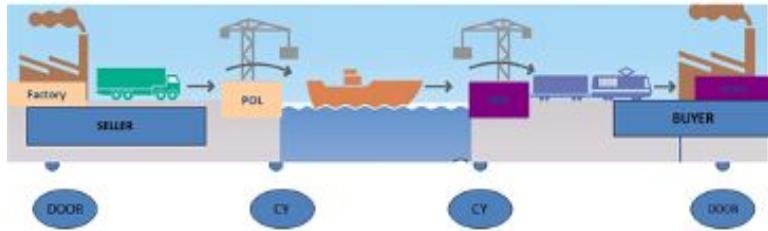
현대 사회에서 석유와 가스처럼 특수 선박이 필요한 경우는 제외하고 사실상 거의 모든 화물이 컨테이너에 담긴 채 바다, 철도, 도로로 운송

컨테이너화를 이끈 '시랜드' 창립자 말콤 맥린



컨테이너란?

컨테이너 박스의 등장으로 인해 서플라이체인의 각 단계에서 운송비가 10%씩 절감
=> 사업의 성패를 가르는 차이가 된다.



컨테이너가 없던 시절, 해상 운송비의 약 50%는 인건비.

제각각 모양과 무게가 다른 화물들

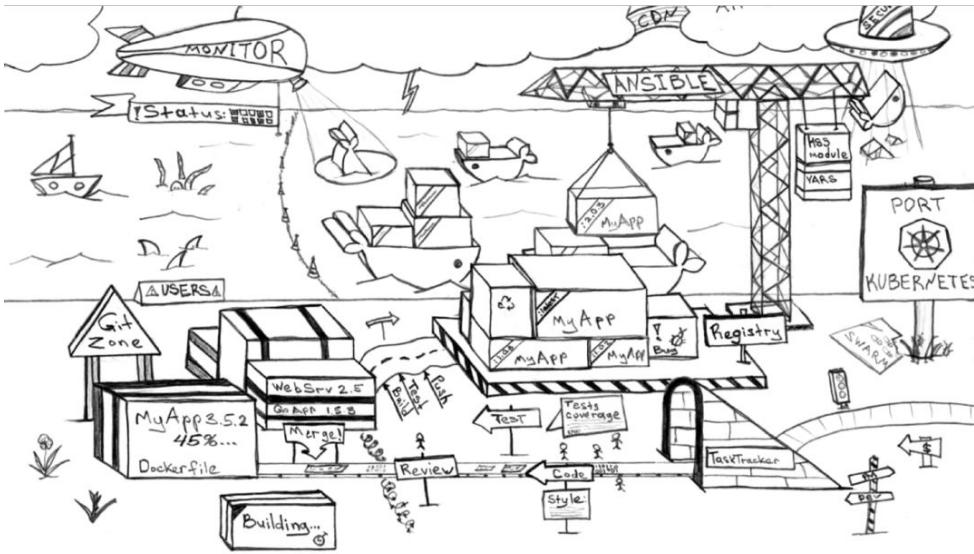
항구 노동자(Docker) 수백 명이 달라붙어도 최소 며칠 ~ 몇 주 소요. 화물의 분실과 파손도 빈번.

태평양을 건너는 비용 < 항구에서 짐을 싣고 내리는 비용



컨테이너란?

IT 환경에서 컨테이너와 오케스트레이터의 혁명



배 = 서버

컨테이너 화물 = 컨테이너로 패키징된 애플리케이션

선적 기증기 = 쿠버네티스

많은 서버에 컨테이너를 자동 배포

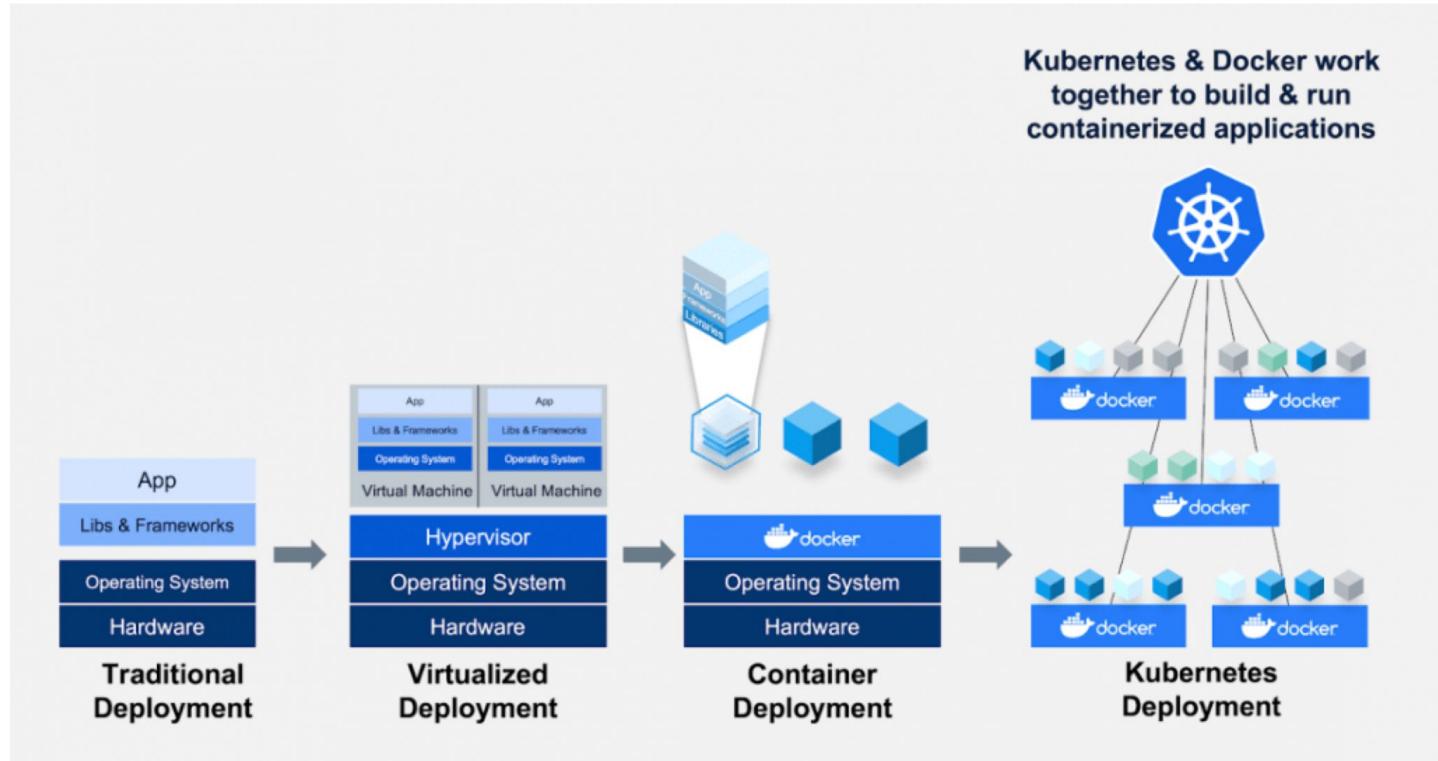
특히 다양한 종류의 많은 서비스들을 관리해야하는
マイ크로서비스에 효과적

***도커** = 대표적인 리눅스 컨테이너 관리 도구



컨테이너란?

IT 환경에서 컨테이너와 오케스트레이터의 혁명





컨테이너란?

전사적으로 개발 / 빌드 / 테스트 / 배포 / 운영 각 체인에서 10% 씩만 효율을 끌어올려도 큰 효과.

개발 / 빌드 / 테스트 환경을 그대로 운영 환경에 옮길 수 있음.





컨테이너란?

컨테이너는 **실행 환경과 성능**을 격리, 표준화

애플리케이션을 OS 환경까지 가상화해서 패키징

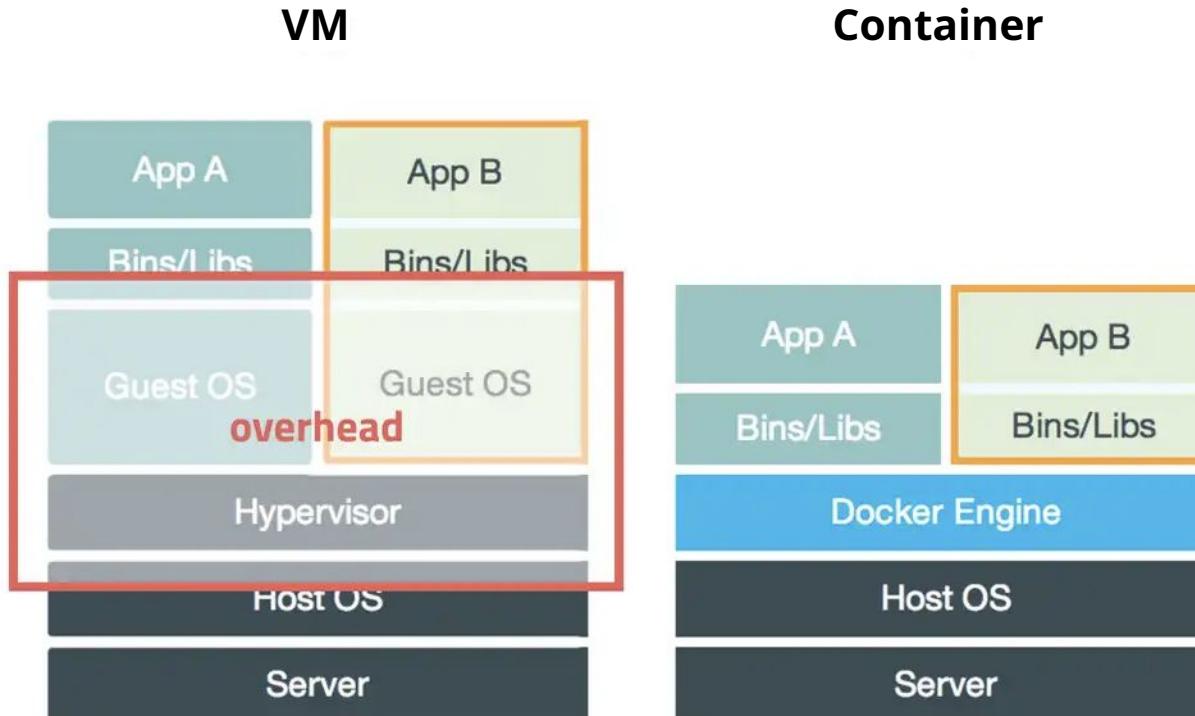
컨테이너는 OS 환경까지 패키징되었기 때문에 Linux, Windows, Mac, 가상 머신, 베어메탈, 개발자의 컴퓨터, 데이터 센터, 온프레미스 환경, 퍼블릭 클라우드 등 사실상 어느 환경에서나 구동되므로 개발 및 배포가 쉬워짐

애플리케이션의 성능 격리

컨테이너는 CPU, 메모리, 스토리지, 네트워크 리소스를 OS 수준에서 가상화하여 개발자에게 기타 애플리케이션으로부터 논리적으로 격리된 OS 샌드박스 환경을 제공합니다.

컨테이너란?

하나의 호스트에서 컨테이너 이미지를 각각 격리된 별도의 프로세스로 실행시켜주는 경량 가상화 기술





컨테이너란?

물리서버에서 직접 동작 시키는 것과 성능 차이가 거의 없음.

	성능 측정 도구	호스트	Docker
CPU	sysbench	1	0.9945
메모리 쓰기	sysbench	1	0.9826
메모리 읽기	sysbench	1	1.0025
디스크 I/O	dd	1	0.9811
네트워크	iperf	1	0.9626



컨테이너란?

컨테이너만 있다면 어디서든 OS를 가리지 않고 실행시킬 수 있다.

MacBook에서 CentOS 리눅스 머신을 띄워보자

docker run -it centos bash

```
~ at  monitoring-demo-context took 4m 28s
→ docker run -it centos bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
Digest: sha256:5528e8b1b1719d34604c87e11dc1c0a201
Status: Downloaded newer image for centos:latest
[root@537854e01a49 /]#
```

노트북 MacOS
에서

centos 리눅스 가상 머신을 띄워서 접속

cat /etc/*release

```
[root@537854e01a49 /]# cat /etc/*release
```

```
CentOS Linux release 8.3.2011
NAME="CentOS Linux"
VERSION="8"
```

리눅스 CentOS
컨테이너 실행

```
PS C:\Users\mathi> .\docker.exe run -it ubuntu
root@c656f87bde2c:/>
24C
25C
26C
27C
28C
29C
30C
31C
32C
33C
34C
35C
36C
37C
38C
39C
40C
41C
42C
43C
44C
45C
46C
47C
48C
49C
50C
51C
52C
53C
54C
55C
56C
57C
58C
59C
60C
61C
62C
63C
64C
65C
66C
67C
68C
69C
70C
71C
72C
73C
74C
75C
76C
77C
78C
79C
80C
81C
82C
83C
84C
85C
86C
87C
88C
89C
90C
91C
92C
93C
94C
95C
96C
97C
98C
99C
100C
101C
102C
103C
104C
105C
106C
107C
108C
109C
110C
111C
112C
113C
114C
115C
116C
117C
118C
119C
120C
121C
122C
123C
124C
125C
126C
127C
128C
129C
130C
131C
132C
133C
134C
135C
136C
137C
138C
139C
140C
141C
142C
143C
144C
145C
146C
147C
148C
149C
150C
151C
152C
153C
154C
155C
156C
157C
158C
159C
160C
161C
162C
163C
164C
165C
166C
167C
168C
169C
170C
171C
172C
173C
174C
175C
176C
177C
178C
179C
180C
181C
182C
183C
184C
185C
186C
187C
188C
189C
190C
191C
192C
193C
194C
195C
196C
197C
198C
199C
200C
201C
202C
203C
204C
205C
206C
207C
208C
209C
210C
211C
212C
213C
214C
215C
216C
217C
218C
219C
220C
221C
222C
223C
224C
225C
226C
227C
228C
229C
230C
231C
232C
233C
234C
235C
236C
237C
238C
239C
240C
241C
242C
243C
244C
245C
246C
247C
248C
249C
250C
251C
252C
253C
254C
255C
256C
257C
258C
259C
259C
260C
261C
262C
263C
264C
265C
266C
267C
268C
269C
270C
271C
272C
273C
274C
275C
276C
277C
278C
279C
280C
281C
282C
283C
284C
285C
286C
287C
288C
289C
289C
290C
291C
292C
293C
294C
295C
296C
297C
298C
299C
300C
301C
302C
303C
304C
305C
306C
307C
308C
309C
309C
310C
311C
312C
313C
314C
315C
316C
317C
318C
319C
319C
320C
321C
322C
323C
324C
325C
326C
327C
328C
329C
329C
330C
331C
332C
333C
334C
335C
336C
337C
338C
339C
339C
340C
341C
342C
343C
344C
345C
346C
347C
348C
349C
349C
350C
351C
352C
353C
354C
355C
356C
357C
358C
359C
359C
360C
361C
362C
363C
364C
365C
366C
367C
368C
369C
369C
370C
371C
372C
373C
374C
375C
375C
376C
377C
378C
379C
379C
380C
381C
382C
383C
384C
385C
386C
387C
388C
389C
389C
390C
391C
392C
393C
394C
395C
396C
397C
398C
399C
400C
401C
402C
403C
404C
405C
406C
407C
408C
409C
409C
410C
411C
412C
413C
414C
415C
416C
417C
418C
419C
419C
420C
421C
422C
423C
424C
425C
426C
427C
428C
429C
429C
430C
431C
432C
433C
434C
435C
436C
437C
438C
438C
439C
440C
441C
442C
443C
444C
445C
446C
447C
448C
449C
449C
450C
451C
452C
453C
454C
455C
456C
457C
458C
459C
459C
460C
461C
462C
463C
464C
465C
466C
467C
468C
469C
469C
470C
471C
472C
473C
474C
475C
476C
477C
478C
478C
479C
480C
481C
482C
483C
484C
485C
486C
487C
488C
489C
489C
490C
491C
492C
493C
494C
495C
496C
497C
498C
499C
500C
501C
502C
503C
504C
505C
506C
507C
508C
509C
509C
510C
511C
512C
513C
514C
515C
516C
517C
518C
519C
519C
520C
521C
522C
523C
524C
525C
526C
527C
528C
529C
529C
530C
531C
532C
533C
534C
535C
536C
537C
537C
538C
539C
539C
540C
541C
542C
543C
544C
545C
546C
547C
548C
548C
549C
550C
551C
552C
553C
554C
555C
556C
557C
558C
558C
559C
560C
561C
562C
563C
564C
565C
566C
567C
568C
569C
569C
570C
571C
572C
573C
574C
575C
576C
577C
578C
578C
579C
580C
581C
582C
583C
584C
585C
586C
587C
588C
589C
589C
590C
591C
592C
593C
594C
595C
596C
597C
598C
599C
600C
601C
602C
603C
604C
605C
606C
607C
608C
609C
609C
610C
611C
612C
613C
614C
615C
616C
617C
618C
618C
619C
620C
621C
622C
623C
624C
625C
626C
627C
628C
629C
629C
630C
631C
632C
633C
634C
635C
636C
637C
638C
638C
639C
640C
641C
642C
643C
644C
645C
646C
647C
648C
649C
649C
650C
651C
652C
653C
654C
655C
656C
657C
658C
659C
659C
660C
661C
662C
663C
664C
665C
666C
667C
668C
669C
669C
670C
671C
672C
673C
674C
675C
676C
677C
678C
678C
679C
680C
681C
682C
683C
684C
685C
686C
687C
688C
689C
689C
690C
691C
692C
693C
694C
695C
696C
697C
698C
698C
699C
700C
701C
702C
703C
704C
705C
706C
707C
708C
709C
709C
710C
711C
712C
713C
714C
715C
716C
717C
718C
718C
719C
720C
721C
722C
723C
724C
725C
726C
727C
728C
729C
729C
730C
731C
732C
733C
734C
735C
736C
737C
738C
738C
739C
740C
741C
742C
743C
744C
745C
746C
747C
748C
749C
749C
750C
751C
752C
753C
754C
755C
756C
757C
758C
759C
759C
760C
761C
762C
763C
764C
765C
766C
767C
768C
769C
769C
770C
771C
772C
773C
774C
775C
776C
777C
778C
778C
779C
780C
781C
782C
783C
784C
785C
786C
787C
788C
789C
789C
790C
791C
792C
793C
794C
795C
796C
797C
798C
798C
799C
800C
801C
802C
803C
804C
805C
806C
807C
808C
809C
809C
810C
811C
812C
813C
814C
815C
816C
817C
818C
818C
819C
820C
821C
822C
823C
824C
825C
826C
827C
828C
829C
829C
830C
831C
832C
833C
834C
835C
836C
837C
838C
838C
839C
840C
841C
842C
843C
844C
845C
846C
847C
848C
849C
849C
850C
851C
852C
853C
854C
855C
856C
857C
858C
859C
859C
860C
861C
862C
863C
864C
865C
866C
867C
868C
869C
869C
870C
871C
872C
873C
874C
875C
876C
877C
878C
878C
879C
880C
881C
882C
883C
884C
885C
886C
887C
888C
888C
889C
890C
891C
892C
893C
894C
895C
896C
897C
898C
898C
899C
900C
901C
902C
903C
904C
905C
906C
907C
908C
909C
909C
910C
911C
912C
913C
914C
915C
916C
917C
918C
918C
919C
920C
921C
922C
923C
924C
925C
926C
927C
928C
929C
929C
930C
931C
932C
933C
934C
935C
936C
937C
938C
938C
939C
940C
941C
942C
943C
944C
945C
946C
947C
948C
949C
949C
950C
951C
952C
953C
954C
955C
956C
957C
958C
959C
959C
960C
961C
962C
963C
964C
965C
966C
967C
968C
969C
969C
970C
971C
972C
973C
974C
975C
976C
977C
978C
978C
979C
980C
981C
982C
983C
984C
985C
986C
987C
988C
988C
989C
990C
991C
992C
993C
994C
995C
996C
997C
998C
999C
1000C
```

노트북 Window
에서

리눅스 Ubuntu
컨테이너 실행도 가능!

'하드웨어'를 '소프트웨어'로 구현

컨테이너는 가상 머신이 아닙니다.
'프로세스'입니다.

컨테이너는 격리된 환경에서 실행된 프로세스



컨테이너의 정체를 파헤쳐 본다.

운영 체제 관점에서 보자면,

컨테이너 = 프로세스



Memory

컨테이너 이미지 = 실행파일

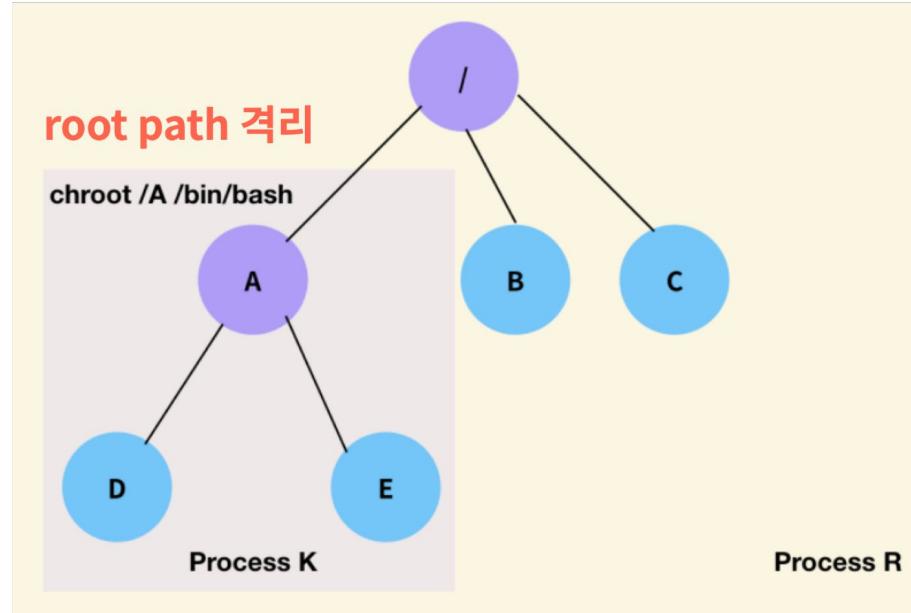


Program



컨테이너의 정체를 파헤쳐 본다.

운영 체제 관점에서 보면, 컨테이너 이미지 = 실행파일 / 컨테이너 = 프로세스
단지, 실행되는 root path 와 리소스, 권한이 격리되어 있다.



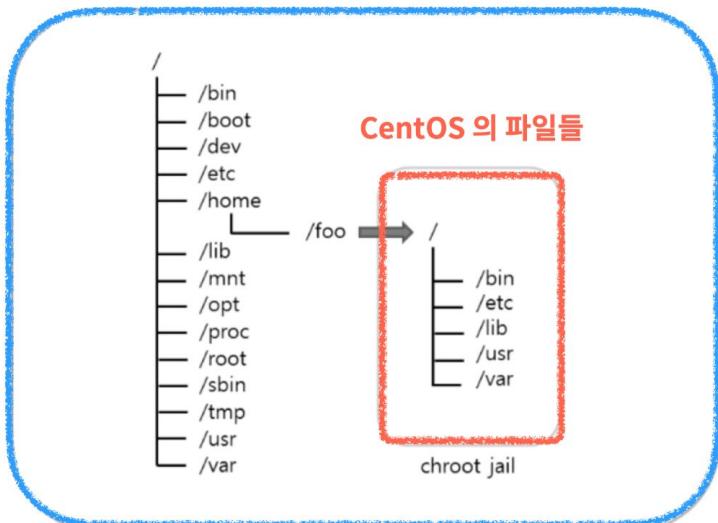


컨테이너의 정체를 파헤쳐 본다.

파일시스템의 가상화/격리

chroot

Ubuntu OS 가 설치된 서버



/var/lib/docker/overlay2

안쪽에

- 1) 컨테이너 이미지가 가지고 있는 파일들
- 2) 사용자가 컨테이너 실행후 변경한 파일들
- 3) 1)과 2) 를 조합한 파일들을 볼 수 있음

또는 /proc/\${PID}/root/ 에서도..

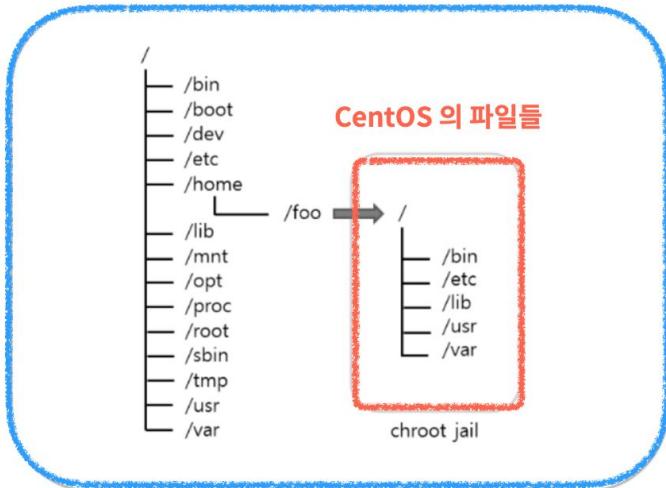


컨테이너의 정체를 파헤쳐 본다.

chroot로 컨테이너 환경 같은 프로세스를 만들어보자

chroot

Ubuntu OS 가 설치된 서버



```
chroot ${new_root_path} ${run_program}
```

```
chroot /tmp/new_root /bin/bash
```

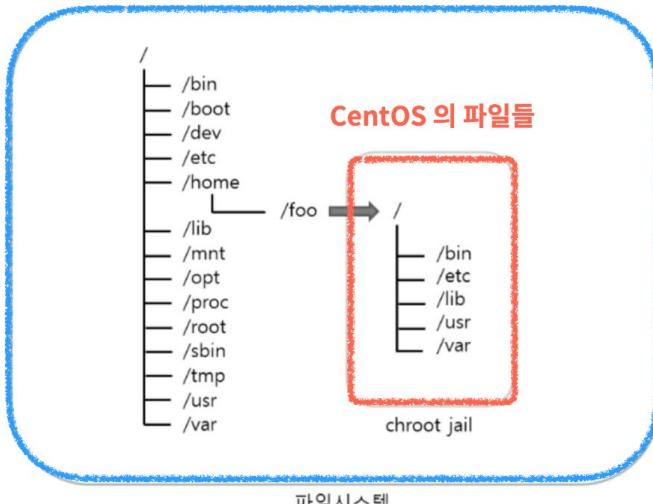


컨테이너의 정체를 파헤쳐 본다.

어디서 많이 보던... docker run ?

chroot

Ubuntu OS 가 설치된 서버



chroot \${루트패스} \${실행파일}

```
chroot /home/deploy/new_root /bin/bash
```

docker run \${이미지} \${실행파일}

```
docker run -it ubuntu:18.04 /bin/bash
```

*사실 container는 chroot를 쓰지는 않고 pivot_root라는 시스템콜을 씁니다.



컨테이너의 정체를 파헤쳐 본다.

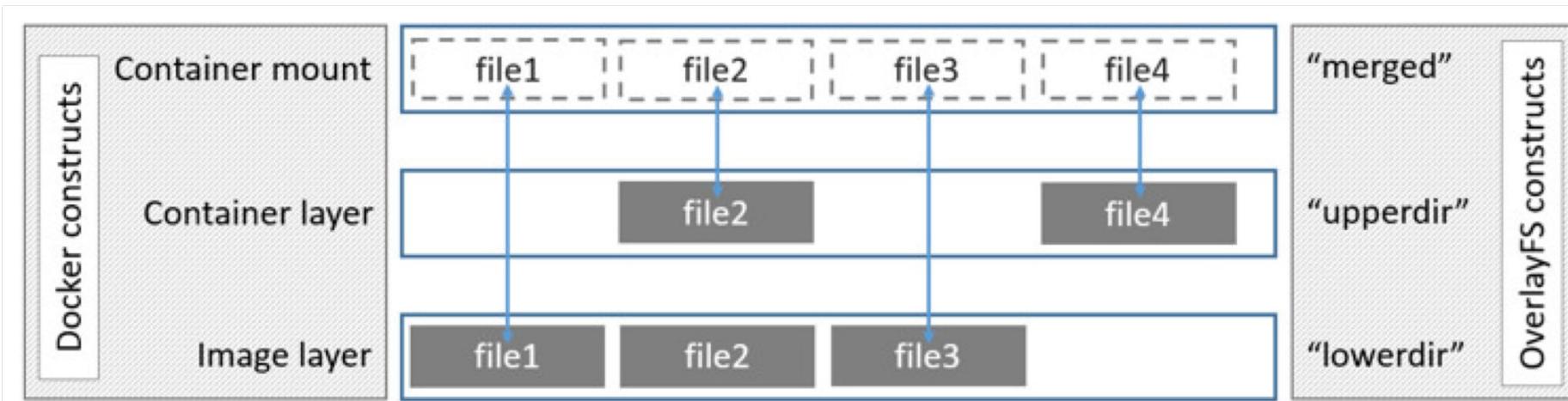
실습 :

<https://dennis.k8s.kr/2>



컨테이너의 정체를 파헤쳐 본다.

Docker로 띄운 경우는 어떨까? **overlay2 driver**





컨테이너의 정체를 파헤쳐 본다.

파일시스템의 가상화/격리 : MergedDir 에 가면 컨테이너 파일시스템의 실체를 볼 수 있다.

경로 찾는 법

docker inspect \${ID} |grep Dir

LowerDir : 컨테이너 이미지 원본

UpperDir : 컨테이너 실행 후 변경분

MergedDir : 둘 다 합쳐서 보이는 결과 -> /var/lib/docker/overlay2/320f~~~~~18fb/**merged**

```
root@dkosv3-monitoring-demo-worker-1:~# docker inspect e78b24ab440f |grep Dir
    "LowerDir": "/var/lib/docker/overlay2/320f18fa2afaebc99cf781f732af7de697bbd6900d426a92c7e0adff5f587d1b-init/diff",
    "/var/lib/docker/overlay2/27729249b75f6f8399a8c3d000f28c20710d3420cc06f5074fc0e5e892e8552/diff",
        "MergedDir": "/var/lib/docker/overlay2/320f18fa2afaebc99cf781f732af7de697bbd6900d426a92c7e0adff5f587d1b/merged",
        "UpperDir": "/var/lib/docker/overlay2/320f18fa2afaebc99cf781f732af7de697bbd6900d426a92c7e0adff5f587d1b/diff",
        "WorkDir": "/var/lib/docker/overlay2/320f18fa2afaebc99cf781f732af7de697bbd6900d426a92c7e0adff5f587d1b/work"
```

또는 /proc/\${PID}/root/ 에서도.. 볼 수 있음..

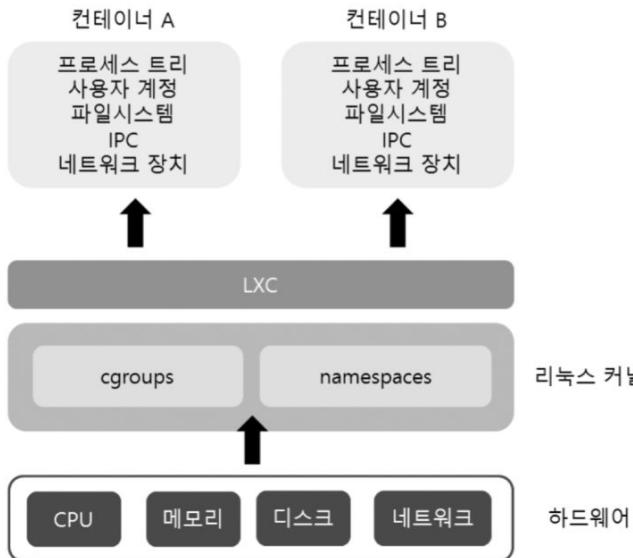
docker inspect \${ID} |grep Pid



컨테이너의 정체를 파헤쳐 본다.

컨테이너는 cgroups와 namespace를 결합하여 애플리케이션을 위한 고립된 환경을 제공

LXC(LinuX Container)



LXC (LinuX Containers)는 단일 컨트롤 호스트 상에서 여러 개의 격리된 리눅스 시스템 (컨테이너)들을 실행하기 위한 OS 레벨 가상화 방법

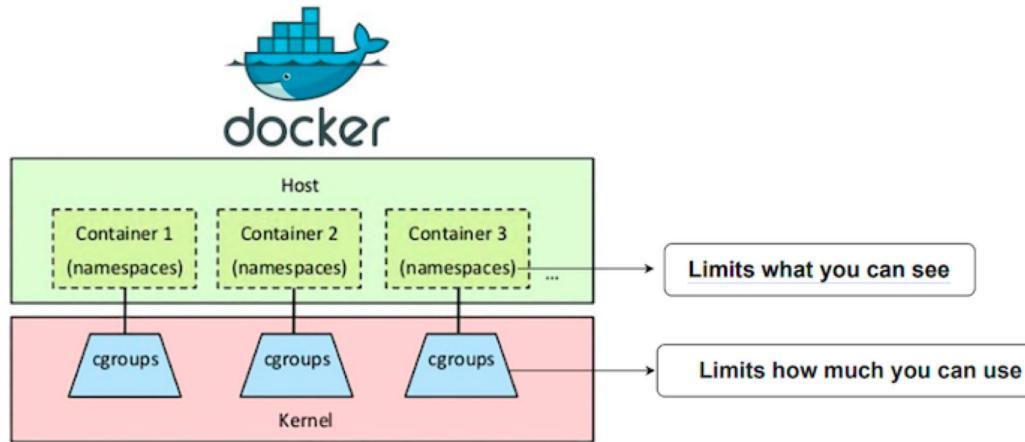
운영 체제로 보면 **컨테이너 이미지는 실행파일, 컨테이너는 프로세스**

리눅스 커널은

- 1) cgroups를 통해 프로세스별 자원 할당을 관리
(CPU, 메모리, 블록 I/O, 네트워크 등)
- 2) namespace 를 통해 장치를 격리해서 관리
(프로세스 트리, 네트워크, 사용자 ID, 마운트된 파일 시스템 등)



컨테이너의 정체를 파헤쳐 본다.



- namespace vs cgroup
 - cgroup은 해당 프로세스가 쓸 수 있는 사용량을 제한한다.
 - namespace는 해당 프로세스가 볼 수 있는 범위를 제한한다.

https://velog.io/@whattsup_kim/Linux-Kernel-%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A5%BC-%EC%9C%84%ED%95%9C-%EB%A6%AC%EB%88%85%EC%8A%A4%EC%9D%98-%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4-%EA%B2%A9%EB%A6%AC-%EA%B8%B0%EB%8A%A5-cgroup-namespace-union-mount

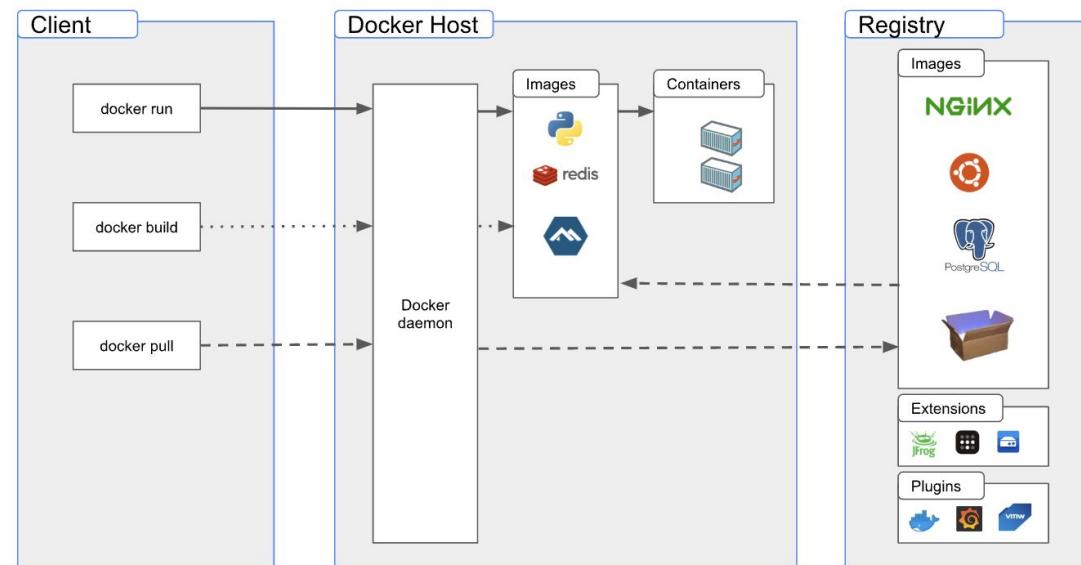
도커란?

도커 : Linux, MacOS, Windows에서 동일한 방법으로 컨테이너를 실행시키고 관리할 수 있는 방법을 제공

도커 파일(Dockerfile) : 컨테이너 이미지를 자동으로 빌드할 수 있도록 지원하는 스크립트

도커 허브(Dock

저장소의 역할





컨테이너의 정체를 파헤쳐 본다.

Namespace 확인 실습

/proc/[PID]/ns

```
root@dkosv3-test-dennis22-worker-g978:/proc/8911/ns# ll
total 0
dr-x--x--x 2 root root 0 Nov 30 21:19 .
dr-xr-xr-x 9 root root 0 Sep 19 12:00 ..
lrwxrwxrwx 1 root root 0 Nov 30 21:27 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0 Nov 30 21:19 ipc -> 'ipc:[4026532215]'
lrwxrwxrwx 1 root root 0 Nov 30 21:19 mnt -> 'mnt:[4026532230]'
lrwxrwxrwx 1 root root 0 Nov 30 21:19 net -> 'net:[4026531993]'
lrwxrwxrwx 1 root root 0 Nov 30 21:19 pid -> 'pid:[4026532231]'
lrwxrwxrwx 1 root root 0 Nov 30 21:27 pid_for_children -> 'pid:[4026532231]'
lrwxrwxrwx 1 root root 0 Nov 30 21:27 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Nov 30 21:27 uts -> 'uts:[4026531838]'
```



컨테이너의 정체를 파헤쳐 본다.

PID Namespace 확인 실습

unshare는 특정 프로세스를 실행할 때 특정 네임스페이스를 분리해주는 기능

```
sudo unshare --fork --pid --mount-proc bash  
ps aux
```

```
$ cd /var/run/docker/runtime-runc/moby/  
$ ls  
622eb0c2acac4d6c9304e57dcc8ea83fd3ec31020ce124e281942dc44ee30725
```

```
lsns -t pid -p 1
```

*참고

<https://www.44bits.io/ko/post/is-docker-container-a-virtual-machine-or-a-process>

<https://tech.ssut.me/what-even-is-a-container/>



컨테이너의 정체를 파헤쳐 본다.

UTS Namespace 확인 실습

<https://www.44bits.io/ko/post/container-network-1-uts-namespace>



컨테이너의 정체를 파헤쳐 본다.

NS(마운트) Namespace 확인 실습

<https://milhouse93.tistory.com/85>



컨테이너의 정체를 파헤쳐 본다.

성능의 가상화/격리 : 컨테이너는 호스트 머신 관점에선 그냥 프로세스이다.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: test
  labels:
    app: test
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
        - image: mdock.daumkakao.io/centos:centos8
          name: test
          command: ["/bin/bash","-c","while true; do sleep 1000; done"]
          imagePullPolicy: IfNotPresent
```

컨테이너 실행시 시작시킨 프로세스 sleep 을
호스트 서버에서 찾아보자.



컨테이너의 정체를 파헤쳐 본다.

성능의 가상화/격리 : 컨테이너는 호스트 머신 관점에선 그냥 프로세스이다.

[호스트 머신]에서는 sleep PID 가... 12106 로 보인다.

```
root@dkosv3-monitoring-demo-worker-1:~# ps -ef |grep sleep
root      2436 21016  0 22:59 pts/0    00:00:00 grep --color=auto sleep
root      12106 12081  0 21:55 ?        00:00:00 /bin/bash -c while true; do sleep 1000; done
```

[컨테이너안]에서는 init 프로세스가 별도로 없고 PID 1번이 sleep 이다. = sleep 이 종료되면? 컨테이너도 종료.

```
[root@test-78c598fdbd-4h8w2 ~]# ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root      1      0  0 12:55 ?        00:00:00 /bin/bash -c while true; do sleep 1000; done
```



컨테이너의 정체를 파헤쳐 본다.

성능의 가상화/격리 : 컨테이너는 호스트 머신 관점에선 그냥 프로세스이다.

```
[root@test-78c598fdbd-4h8w2 /]# sleep 1000 & sleep 1000 & sleep 1000 & sleep 1000 &
[4] 135
[5] 136
[6] 137
[7] 138
[root@test-78c598fdbd-4h8w2 /]# ps -ef
UID      PID  PPID  C STIME TTY      TIME CMD
root        1     0  0 12:55 ?    00:00:00 /bin/bash -c while true; do sleep 1000; done
root      102     1  0 14:02 ?    00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      117     0  0 14:09 pts/0   00:00:00 bash
root      131    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      132    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      133    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      135    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      136    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      137    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      138    117  0 14:10 pts/0   00:00:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep 1000
root      139    117  0 14:10 pts/0   00:00:00 ps -ef
[root@test-78c598fdbd-4h8w2 /]#
```

```
root@dkosv3-monitoring-demo-worker-1:/proc/12106/root# pstree -p | grep sleep
|           |-containerd-shim(12081)-+--bash(5969)-+-sleep(6197)
|           |                   |           |           |-sleep(6198)
|           |                   |           |           |-sleep(6199)
|           |                   |           |           |-sleep(6215)
|           |                   |           |           |-sleep(6216)
|           |                   |           |           |-sleep(6217)
|           |                   |           |           `--sleep(6218)
|           |                   |
|           |                   |-bash(12106)--sleep(8948)
```

컨테이너 안에서 추가로 프로세스를 실행시키면,

호스트 머신에서도 그대로 자식 프로세스로 보인다. (PID 는 격리되어 있어서 다르게 보인다.)



컨테이너의 정체를 파헤쳐 본다.

그럼 호스트 머신에서 pod 의 PID Namespace 에 접근할 수 있겠네?

Usage:

```
nsenter [options] <program> [<argument>...]
```

Run a program with namespaces of other processes.

Options:

- t, --target <pid> target process to get namespaces from
- m, --mount[=<file>] enter mount namespace
- u, --uts[=<file>] enter UTS namespace (hostname etc)
- i, --ipc[=<file>] enter System V IPC namespace
- n, --net[=<file>] enter network namespace
- p, --pid[=<file>] enter pid namespace
- U, --user[=<file>] enter user namespace
- S, --setuid <uid> set uid in entered namespace
- G, --setgid <gid> set gid in entered namespace
 - preserve-credentials do not touch uids or gids
- r, --root[=<dir>] set the root directory
- w, --wd[=<dir>] set the working directory
- F, --no-fork do not fork before exec'ing <program>
- Z, --follow-context set SELinux context according to --target PID



컨테이너의 정체를 파헤쳐 본다.

Network Namespace 확인 실습

bash 나 netstat, ping 등 트러블 슈팅을 위한 tool 이 안깔려있는 컨테이너의 네트워크 점검 꿀팁

```
nsenter -t ${PID} -n ${실행할명령}
```

인터페이스 확인

```
nsenter -t 11938 -n ip a
```

핑 확인

```
nsenter -t 11938 -n ping
```

소켓 상태 확인

```
nsenter -t 11938 -n netstat
```

패킷 덤프

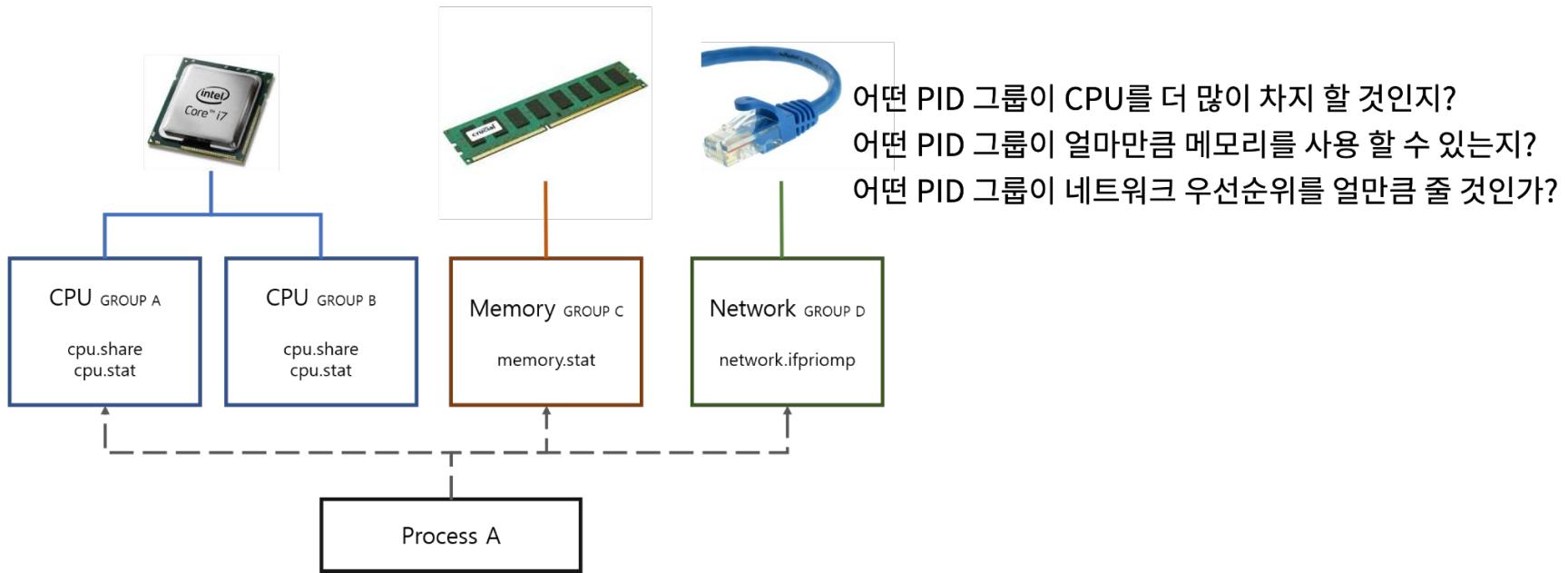
```
nsenter -t 11938 -n tcpdump -peni any "arp"
```

*PID 알아내기 : docker inspect -f '{{.State.Pid}}' \${컨테이너 아이디}



컨테이너의 정체를 파헤쳐 본다.

성능의 가상화/격리 : cgroup 은 PID 별로 하드웨어 리소스를 할당 한다.





컨테이너 사용 팁

1. docker 컨테이너는 실행이 끝나도 자동 삭제 되지 않는다. 용량도 먹는다.

```
$ docker run hello-world
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

좋아! 돌고 있는 컨테이너가 없군 내 디스크는 완전 깨끗하겠지?

\$ docker ps -a -s (컨테이너별 사용 디스크 용량 확인 옵션 -s) 해보면 컨테이너 시체들이 즐비함.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	SIZE
70623b3f5610	hello-world	"/hello"	Less than a second ago	Exited (0) 2 seconds ago		pedantic_ride	0B (virtual 1.85kB)
5724d2448c64	hello-world	"/hello"	3 seconds ago	Exited (0) 5 seconds ago		nifty_curi	0B (virtual 1.85kB)
99c8912979fc	hello-world	"/hello"	4 seconds ago	Exited (0) 6 seconds ago		admiring_reamon	0B (virtual 1.85kB)
74a0db250fb3	hello-world	"/hello"	5 seconds ago	Exited (0) 7 seconds ago		pedantic_jang	0B (virtual 1.85kB)
369ad0c84e6560	hello-world	"/hello"	About a minute ago	Exited (0) About a minute ago		practical_snyder	0B (virtual 1.85kB)
78028a30298e	hello-world	"/hello"	About a minute ago	Exited (0) About a minute ago		clever_goodall	0B (virtual 1.85kB)
5100cce22060	idock.doumekoko.io/nlp.nbc/nbc-latest	"bin/sh -c '/bin/bash'"	3 weeks ago	Exited (0) 3 weeks ago		competent_kelom	0B (virtual 4.05GB)
b6df095ec0f7	idock.doumekoko.io/d2hub/docker-builder:latest	"dockerd-entrypoint..."	3 months ago	Exited (255) 3 months ago	2375/tcp, 0.0.0.0:3000->3000/tcp	d2hubdennis_builder_1	0B (virtual 111MB)
78cd87ca84	idock.doumekoko.io/d2hub/registryv2:2.6.0	"/usr/bin/supervisord"	3 months ago	Exited (255) 3 months ago	5000/tcp, 0.0.0.0:5000->5000/tcp	d2hubdennis_index_registry_1	1.81kB (virtual 81MB)
cc952692549a	idock.doumekoko.io/d2hub/registry:2.6.0	"/entrypoint.sh /etc..."	3 months ago	Exited (2) 3 months ago		d2hubdennis_opl_registry_1	0B (virtual 33.2MB)
e3c2bcc30816	idock.doumekoko.io/d2hub/docker-auth:latest	"/auth_server --v=2 ..."	3 months ago	Exited (1) 3 months ago		d2hubdennis_auth_server_1	1.62kB (virtual 11.6MB)
f0f0855ca0c9	idock.doumekoko.io/d2hub/registryv2:2.6.0	"bin/truen"	3 months ago	Exited (0) 3 months ago		d2hubdennis_data_registry_1	0B (virtual 33.2MB)
c838c565ac5e	mysql:latest	"docker-entrypoint.s..."	9 months ago	Exited (255) 2 weeks ago	0.0.0.0:3306->3306/tcp	dkosv2dennis_mysql_1	0B (virtual 407MB)
fc563d82632d	redis:latest	"docker-entrypoint.s..."	9 months ago	Exited (255) 2 weeks ago	0.0.0.0:6379->6379/tcp	dkosv2dennis_redis_1	0B (virtual 106MB)

\$ docker run --rm my-docker (--rm 옵션주고 실행시키면 종료시 자동 삭제됨)



컨테이너 사용 팁

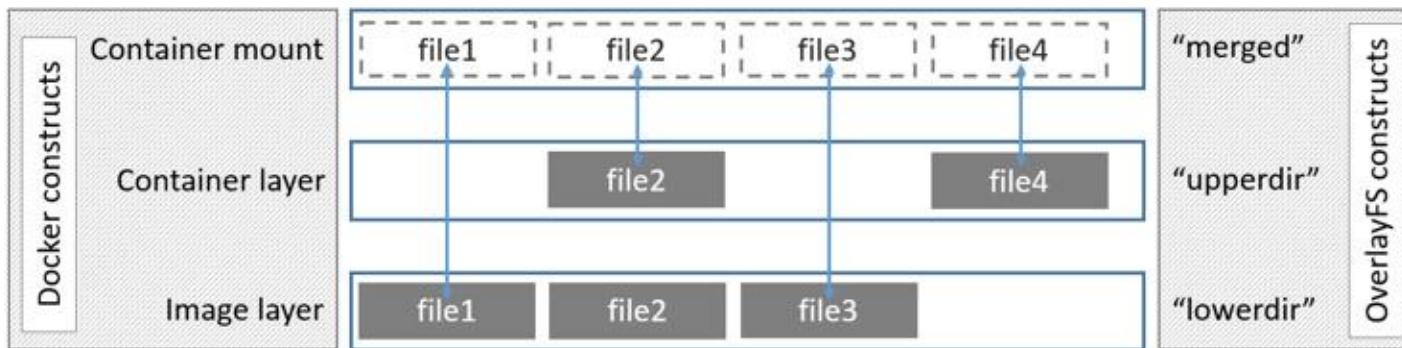
2. 컨테이너가 사용하는 파일들은 어디 있는가?

```
docker run -it ubuntu /bin/bash
```

```
echo "hello" > hello.txt
```

docker inspect c5dca4804501 하면 관련 정보 조회 가능.

/var/lib/docker/containers/{container_ID}/hostname 안에 관련 파일들이 다 있음



<https://docs.docker.com/storage/storagedriver/overlayfs-driver/>



컨테이너 사용 팁

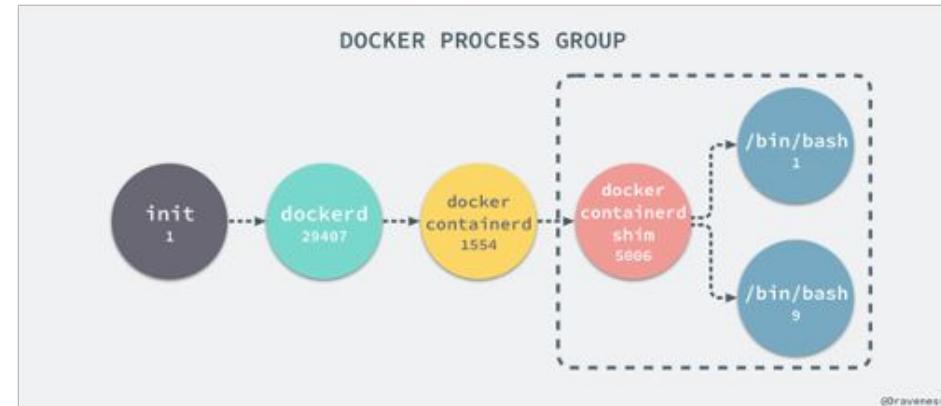
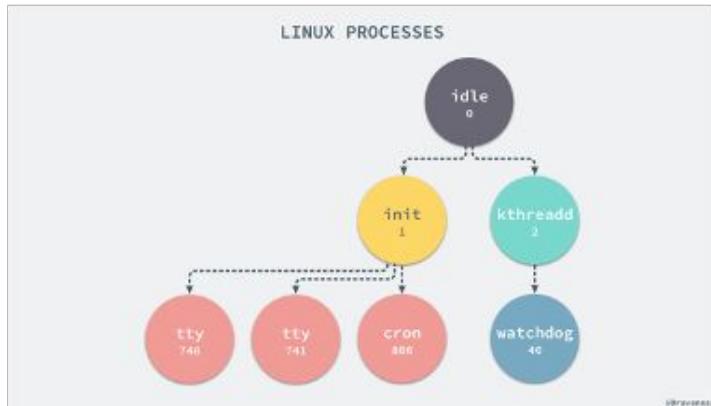
3. docker container 실행에 관하여..

docker run 했을 당시 실행한 cmd가 init 프로세스가 된다.

따라서 해당 cmd가 종료되면 컨테이너도 종료된다.

컨테이너 하나에 여러개 프로세스 올리면 문제점 -> init 프로세스가 종료되면 다 죽는다.

최소 실행시에는 run , 기존 돌고 있는 컨테이너에 추가 실행은 exec , 입력 필요한 터미널 실행시에는 -it

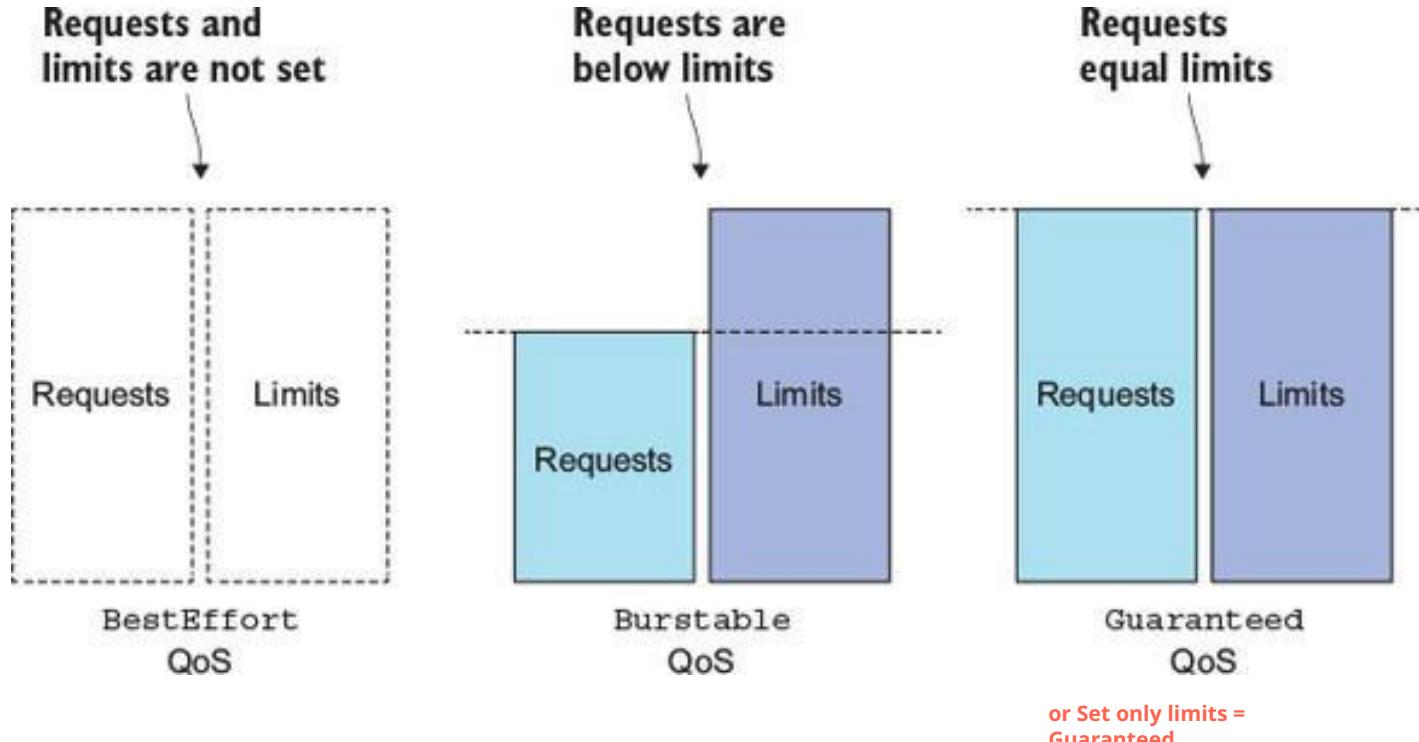


kubernetes에서의 Resource 관리

k8s 자원 할당의 정체를 파헤쳐

보다,

k8s에서 Pod Resource Request / Limit 설정에 따른 QoS
관리



k8s 자원 할당의 정체를 파헤쳐 본다.

Resource Request / Limit 설정 안주고 배포하면 어떻게 되나? best-effort

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cpu-stresser
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-stresser
  template:
    metadata:
      labels:
        app: cpu-stresser
    spec:
      containers:
        - name: cpustress
          image: containerstack/cpustress
          resources: {}
```

k8s 자원 할당의 정체를 파헤쳐

본다

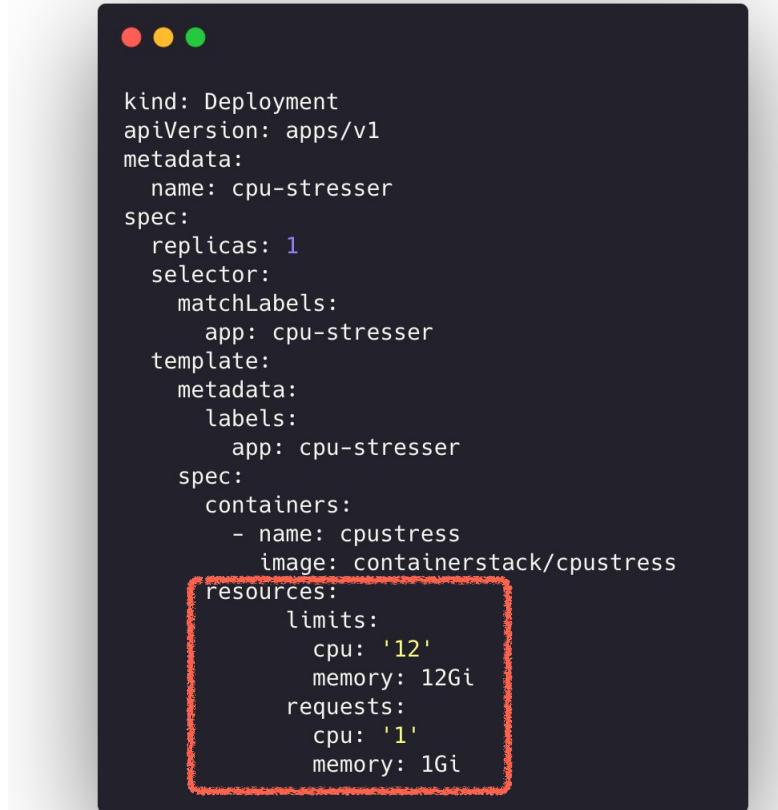
Resource Request / Limit 설정 없으면 **best-effort** : 최소

설정

	구분	k8s	docker	cgroups (linux)
cpu	min	spec.containers[]. resource. requests.cpu 설정 없음	--cpu-shares 2 (최소)	cpu.shares 2
	max	spec.containers[]. resource. limit.cpu 설정 없음	--cpu-period 100000 (고정) --cpu-quota 0 (제한없음)	cpu.cfs_period_us 100000 cpu.cfs_quota_us -1
memory	min	spec.containers[]. resource. requests.ram 설정 없음	--memory-reservation 0(미사용)	memory.soft_limit_in_bytes 9223372036854771712 (제한 없음, 항상 고정값)
	max	spec.containers[]. resource. limit.ram 설정 없음	--memory 0	memory.limit_in_bytes 9223372036854771712 (제한없음)

k8s 자원 할당의 정체를 파헤쳐 본다.

Resource Request < Limit 설정 주고 배포하면 어떻게 되나? burstable



```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cpu-stresser
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-stresser
  template:
    metadata:
      labels:
        app: cpu-stresser
    spec:
      containers:
        - name: cpustress
          image: containerstack/cpustress
      resources:
        limits:
          cpu: '12'
          memory: 12Gi
        requests:
          cpu: '1'
          memory: 1Gi
```

k8s 자원 할당의 정체를 파헤쳐

본다.

Resource Request < Limit 설정 주고 배포하면 어떻게
되나?

CPU 최소/최대 설정, Memory 는 최대 제한만.

	구분	k8s	docker	cgroups (linux)
cpu	min	spec.containers[]. resource. requests.cpu 2	--cpu-shares 2048	cpu.shares 2048
	max	spec.containers[]. resource. limit.cpu 12	--cpu-period 100000 (고정) --cpu-quota 1200000	cpu.cfs_period_us 100000 cpu.cfs_quota_us 1200000
memory	min	spec.containers[]. resource. requests.ram 2 Gi	--memory-reservation 0(미사용)	memory.soft_limit_in_bytes 9223372036854771712 (제한 없음, 항상 고정값)
	max	spec.containers[]. resource. limit.ram 12 Gi	--memory 12884901888	memory.limit_in_bytes 12884901888 (12G 제한)

k8s 자원 할당의 정체를 파헤쳐 본다

Resource Request = Limit 설정 주고 배포하면 어떻게 되나? Guaranteed



```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: cpu-stresser
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cpu-stresser
  template:
    metadata:
      labels:
        app: cpu-stresser
    spec:
      containers:
        - name: cpustress
          image: containerstack/cpustress
      resources:
        limits:
          cpu: '12'
          memory: 12Gi
        requests:
          cpu: '12'
          memory: 12Gi
```

k8s 자원 할당의 정체를 파헤쳐

본다.

Resource Request = Limit 설정 주고 배포하면 어떻게
되나?

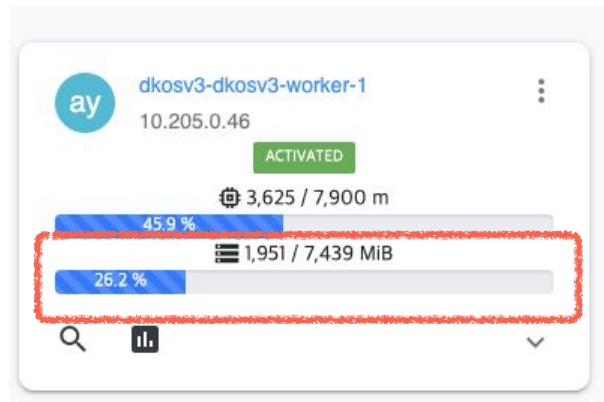
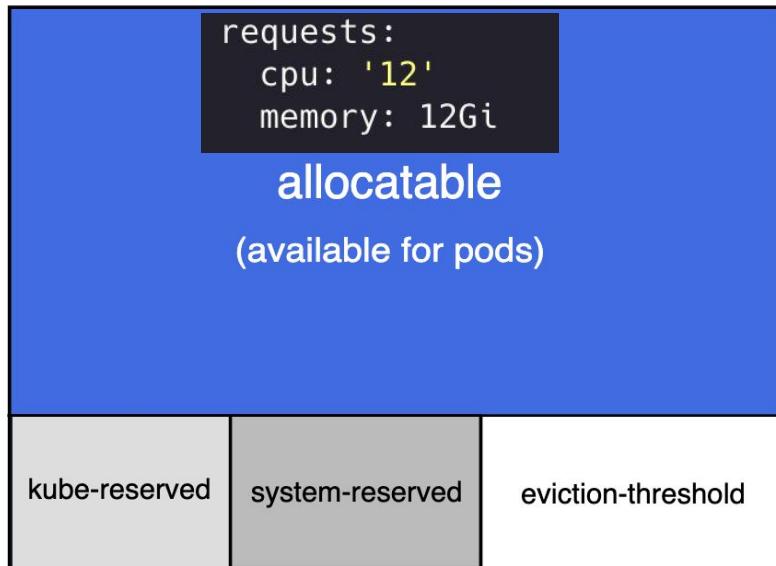
CPU 최소(가중치)/최대 제한 설정. Memory는 최대
제한만.

	구분	k8s	docker	cgroups (linux)
cpu	min	spec.containers[]. resource. requests.cpu 11	--cpu-shares 11264	cpu.shares 11264
	max	spec.containers[]. resource. limit.cpu 11	--cpu-period 100000 (고정) --cpu-quota 1100000	cpu.cfs_period_us 100000 cpu.cfs_quota_us 1100000
memory	min	spec.containers[]. resource. requests.ram 11 Gi	--memory-reservation 0(미사용)	memory.soft_limit_in_bytes 9223372036854771712 (제한 없음, 항상 고정값)
	max	spec.containers[]. resource. limit.ram 11 Gi	--memory 11811160064	memory.limit_in_bytes 11811160064 (11G 제한)

오잉? 그럼 Memory 는 최소 보장은 어떻게 ?

pod의 memory request 와 node allocatable 보고 스케줄링해서 자원이 (아마)
있을거야

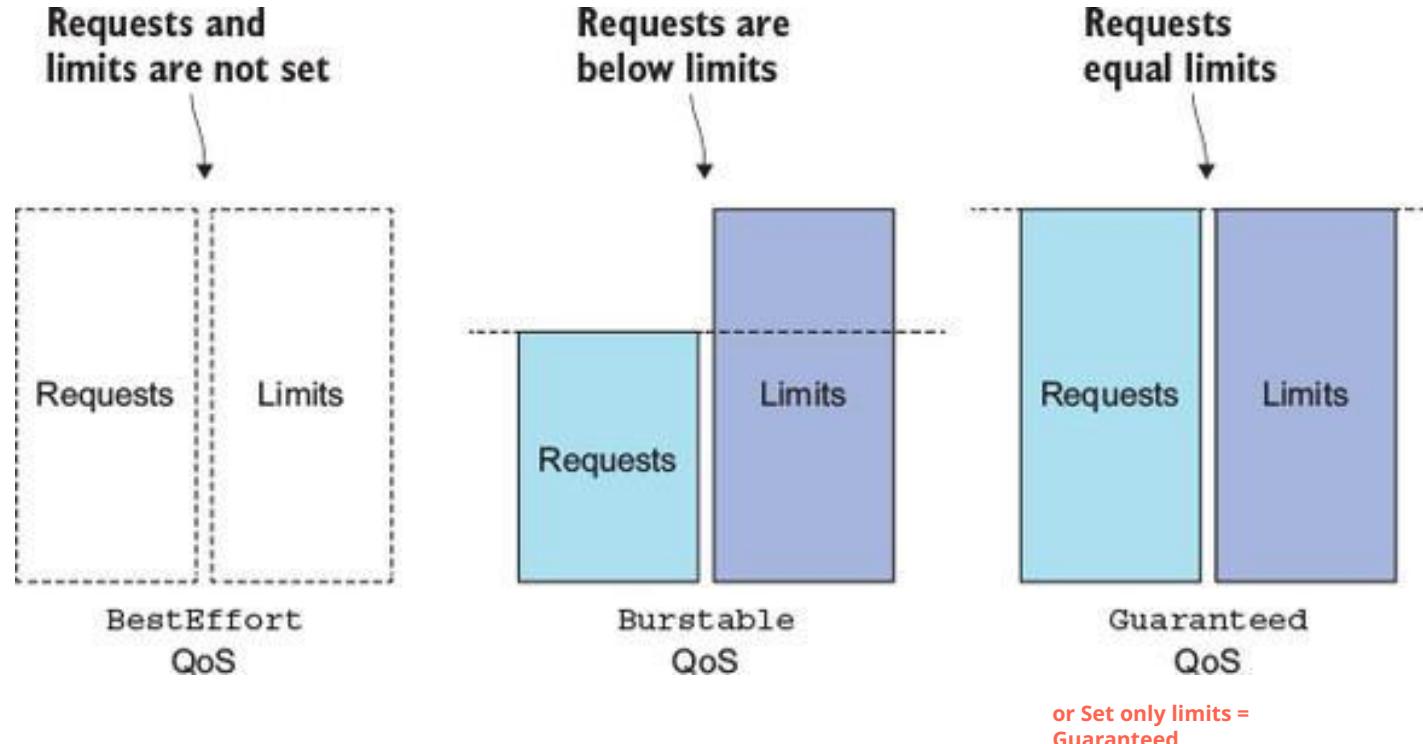
Node Capacity



오잉? 그럼 Memory 는 최소 보장은

어떻게?

k8s가 Pod Memory Request / Limit 설정에 따른 계급을 매겨 우선 순위를 정해준다.



오잉? 그럼 Memory 는 최소 보장은 어떻게...? cgroup 관리도 QoS Class 에 의한 계급

```
root@dkosv3-d2hub-pg-worker-1:/sys/fs/cgroup/memory/kubepods# ll
total 0
drwxr-xr-x 1 root root 0 Jan 15 09:04 .
drwxr-xr-x 1 root root 0 Jan 15 08:43 ..
drwxr-xr-x 1 root root 0 Jan 15 08:43 besteffort/
drwxr-xr-x 1 root root 0 Jan 15 09:04 burstable/
drwxr-xr-x 1 root root 0 Jan 15 08:43 cgroup.clone_children
drwxr-xr-x 1 root root 0 Jan 15 08:43 cgroup.event_control
drwxr-xr-x 1 root root 0 Jan 15 08:43 cgroup.procs
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.failcnt
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.force_empty
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.failcnt
drwxr-xr-x 1 root root 0 Jan 15 09:46 memory.kmem.limit_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.max_usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.slabinfo
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.tcp.failcnt
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.tcp.limit_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.tcp.max_usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.tcp.usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.kmem.usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 09:47 memory.limit_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.max_usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.memsw.failcnt
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.memsw.limit_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.memsw.max_usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.memsw.usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.move_charge_at_immigrate
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.numa_stat
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.oom_control
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.pressure_level
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.soft_limit_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.stat
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.swappiness
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.usage_in_bytes
drwxr-xr-x 1 root root 0 Jan 15 08:43 memory.use_hierarchy
drwxr-xr-x 1 root root 0 Jan 15 08:43 notify_on_release
drwxr-xr-x 1 root root 0 Jan 15 09:08 podcb245ee8-6fa4-4309-8539-57c5141d9cc0/
drwxr-xr-x 1 root root 0 Jan 15 08:43 tasks
```

천민
(bestEffort)

농민
(burstable)

천룡인
(Guaranteed)

```
root@dkosv3-d2hub-pg-worker-1:/sys/fs/cgroup/memory/kubepods# tree -d
.
+-- besteffort
|   +-- pod158db4a4-1e4e-42d8-b970-196a0ef5a8ec
|       +-- 6b07e5cd2f7842662e2f663d38b6b7b55bd03ccb0c20039aea970532dbc108
|           +-- 900ef4575f36a199e4142f43ab174e953b8fc1fb73986fe558db64976bf1a004
|               +-- cb945431899f094a2c32f50a49c842316e8e81607efaa70ee68c0fb1c73fc719
|                   +-- podb4db6ab1-d1bc-4f26-b1b9-413844401f2b
|                       +-- 0272ed9b1998b629934fe7460cef1b4e9475a1dbc68624a14faad957cf27d91c
|                           +-- a9375fce8ce039e072cedc28c22fc79647d9d297332db643819867f53dbe37
|                               +-- bc0d3023278a1292d0c800fd6427c96fb53f32600ab374dfbf2f0db37e77a39
|                                   +-- podb71fd44-d7dc-42fc-bb4f-cae15b3fecf2
|                                       +-- 18cd18f4bee4a4fa74d609f71fea0843fa13ee2d6f12b990a18fc2ded3c437b6
|                                           +-- 4014b3d6d97f6ad5ff5ce082171c137d1dc8efbf2f690cfb50662d6da5fa53d17
|                                               +-- podcd91327-c0e-4574-a9ab-90ca41d9b29f
|                                                   +-- 3be27b460d90982a10378ead8c184d6685b31b5ddbcad2cf34b2c25badfaac0d
|                                                       +-- af513d5a6f352b422810215d505645966592371dcc093a388aba43e4fb69186
|                                               +-- podeb79ad23-7538-4d58-b649-152be5b60135
|                                                   +-- 200f18eca77d0460e7fce0177709c47e52335cf3494ddf8b1f7b4a8c4401e53
|                                                       +-- 8fbfd6a8219e96d5d86f0c7b5454929c7a86e562a3613e82f17c9bfd46252f6ea
+-- burstable
    +-- pod261711ab-83f9-4d3e-abdb-5988e9d5873b
        +-- 4d001567f42za8261100dce6b808cf43c42b9f01e7048209d9ca3e9e3c274093
            +-- 6338e564a964f15df5e53eb4ef2058c1504d6650d9ef5f4f27d5f3e4f99f3be
        +-- pod39bbaaa8-85bf-492b-ac7e-90497c31e9c
            +-- bab18020eed0523992fa0f876e2f6b6592e924b05e36067f5879f7ebf1672154
                +-- c7f2d3488115f76c238dd48740a02c64970a004ef3b143ccalc1f4758d3ecab1
        +-- pod48081aa7e323a8bac09ed7b54898e50d
            +-- 8ae06ac2d066e191ba68e1d82e09788108b1ca7c102a623c4481a06dbfc
                +-- c274f490c4d8f8ca5207666f5d3d2847831f41c4bfe5ea6b27986579e38113d5
        +-- podad9618d9-894f-41ea-ab8c-8eba2bf30fe9
            +-- 3095f689240316c34e69754206ebd1be5d167c4d1bb58c807f785891a949b33
                +-- 5a45d11bee2bf4e2083c01457d29dbd5a0455e1ab8139e6ad53d42fc409319
        +-- pode520a428-0530-46ee-ae0c-fccf5ecb4af6
            +-- 1945b38f1aea2ac970eab5edf705a78316a2bf6eb843d8d9a19f79af6f9d9015a
                +-- 4e4c7b8c749260a89e5c8047e08ed1ea23c5f39a7043040a4656d93b8e7bd
        +-- podf9d7af5b-cfe9-4a26-bda9-553f4db2098e
            +-- c33c166ab240c51e6c06627e6a7f424481b1a0f222073fcf4ec3d70f750
                +-- c39d14b23a727f139e8bab9888c44e1ea0d28044ac2672d3ed70ccb5c794f8
        +-- podcb245ee8-6fa4-4309-8539-57c5141d9cc0
            +-- 50827f2225d78cf288fd2a812ebddc0d9399f91e87d71d4612e23fd3cce5f2
                +-- d21929980ab12f6b789252cf244a0f7cdc38fa34e7366c4726e6177b8f6e77a
```

오잉? 그럼 Memory 는 최소 보장은

어떻게..?
메모리 없어..? 우선 순위 낮은애를 먼저 죽여서 보장해
줄께... ^^

kernel OOM killer with k8s

1) Limit 이상 메모리 사용하면 죽임

2) oom_score + oom_score_adj 높은 순서대로 죽임

Container의 Process의 oom_score 값과

Kubernetes가 QoS Class에 따라 oom_score_adj 부여
함이 높을수록 메모리 부족시 OOM Killer에게 먼저 죽는다.

oom_score + oom_score_adj = 0 (영생)

oom_score + oom_score_adj = 1~999 (높은 순서대로 사망)

oom_score + oom_score_adj= 1000 이상 (즉사)

Pod QoS	oom_score_adj
Guaranteed	-998
Burstable	$\min(\max(2, 1000 - (1000 * \text{memoryRequestBytes}) / \text{machineMemoryCapacityBytes}), 999)$
BestEffort	1000

[표 1] Pod의 QoS에 따른 oom_score_adj 값



oom_score + oom_score_adj = 1000
人脉..

Pod 끼리 싸우면 누가 이기나?

결론!

Memory = 계급(QoS Class) 높은 애가 이김.

기본 적으로 Request 만 설정해서 자원을
효율적으로 사용.

중요한 애는 Limit 도 설정 해서 높은 계급을 줌

OOM 이슈 시 디버깅 절차

- 1) 해당 pod 의 memory limit 을 초과하였는가?
- 2) Node 에 메모리가 부족한가? -> request 현실화..
- 3) request / limit 설정을 통해 충분한 계급을
보장해줬는가?
(request 값이 아무리 높아도 limit 없으면 계급 낮아서 먼저
죽음)



Pod 끼리 싸우면 누가 이기나?

결론!

CPU=가중치(Request) 높은 애가 이김.

Request 만 설정하는게 가장 자원 효율
좋음.

확실히 성능을 놀려주고 싶은 애만 Limit

설정

16 core test

CONTAINER ID	NAME	CPU %
39de12c2c82b	k8s_cpustress_cpu-stresser-guaranteed-9d749f9c8-fw4ss_default_df0b06c0-51e6-437b-8785-f5942f55fb90_0	1128.21%
2fcbb50e1ece	k8s_cpustress_cpu-stresser-bustable-bb6c7755-t2lnl_default_1eacc8af-902a-459f-a1a0-8ed483488cf2_0	478.64%
d6de4518a54b	k8s_cpustress_cpu-stresser-best-effort-59679c956f-445wf_default_cbefcb27-ee8c-4f87-88af-b946a162bdः_0	0.40%

Pod 끼리 싸우면 누가 이기나?

결론!

CPU=가중치(Request) 높은 애가 이김.

Request 만 설정하는게 가장 자원 효율 좋음.

확실히 성능을 높려주고 싶은 애만 Limit 설정

(Request milicpu / Node Total milicpu) * 1024 = shares

Contaier A : (1500 / 2000) * 1024 = 768

Contaier B : (500 / 2000) * 1024 = 256

CPU 성능 부족 이슈 시 디버깅 절차

1) CPU limit 설정이 있다면 해당 Pod 가 limit 만큼 사용하는가?

2) CPU limit 설정이 없다면 Node 의 전체 CPU 를 사용하는가?

3) 경합 우선 순위에서 가중치 싸움에서 밀리는가?



k8s 자원 할당의 정체를 파헤쳐

본다 cgroup 설정 찾아가는 방법

```
root@dkosv3-d2hub-pg-worker-1:~# cat /proc/${PID}/cgroup
12:rdma:/  

11:blkio:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
10:perf_event:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
1
9:pids:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
8:freezer:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
7:cpuset:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
6:memory:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
5:hugetlb:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
4:net_cls,net_prio:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
3:devices:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
2:cpu,cpuacct:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
1
1:name=systemd:/kubepods/besteffort/pod6323f508-0a43-4ddb-9990-f44f5a0d56a1/f67e8067234437ceccf37c8f15049fe23eee0b77c0571f4bd34a5c7d80ca3331
0::/system.slice/containerd.service
```

```
cat /sys/fs/cgroup/cpu/${위에경로}/cpu.shares
```

```
cat /sys/fs/cgroup/memory/${위에경로}/memory.limit_in_bytes
```

k8s 자원 할당의 정체를 파헤쳐

본다 도커 자원 할당 찾기

```
docker inspect --format '{{.HostConfig.CpuShares}} {{.HostConfig.CpuQuota}} {{.HostConfig.CpuPeriod}}' 5cc990f730b5  
docker inspect --format '{{.HostConfig.Memory}}' 5cc990f730b5
```

```
docker inspect -f '{{.State.Pid}}' 5cc990f730b5
```



1

2

3

4

5

6

7

8

9

10

11

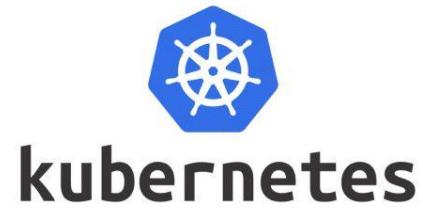
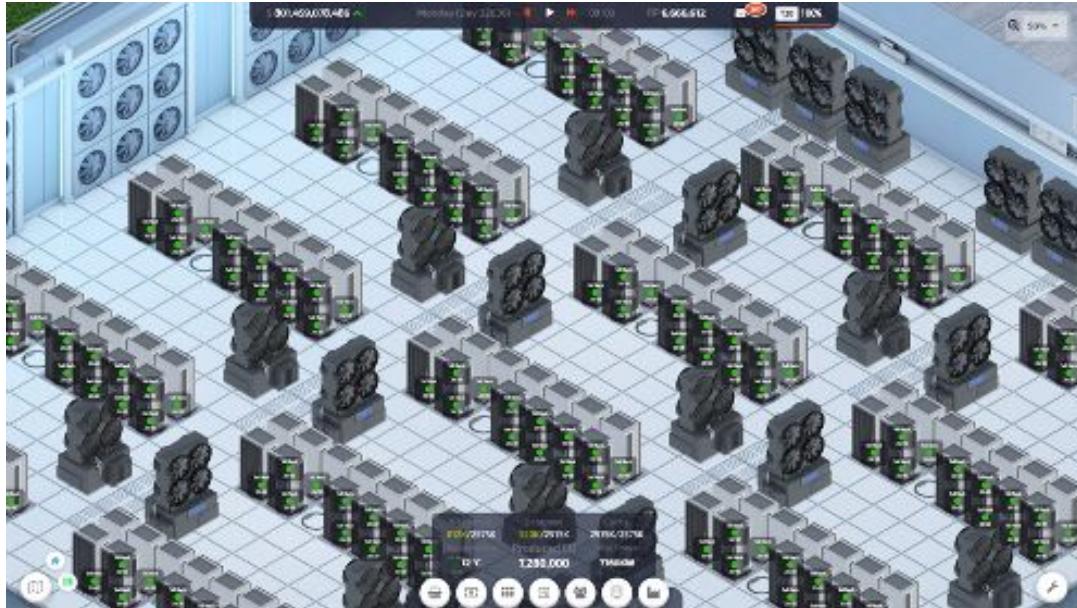
12

13

{ hello kubernetes; }

컨테이너 오케스트레이터란 무엇이고 왜 필요한가?

서버 1대에서는 도커만으로 충분, 하지만 관리해야 할 서버가 수 천대라면? 컨테이너 오케스트레이터, 쿠버네티스!





Kubernetes 란?

대규모 분산 서버 환경에서 Container 관리를 자동화하는 유연하고 확장성 있는 오픈소스 플랫폼

- 쿠버네티스란 명칭은 키잡이(helmsman)이나 파일럿을 뜻하는 그리스어에서 유래
- k와 s 사이에 스펠링이 8개라서 k8s라고 불리기도 함. ex) i18n
- 컨테이너 배포, 자동 복구, 확장 등 컨테이너화된 애플리케이션의 관리를 위한 오픈소스 시스템
- 컨테이너 오케스트레이터의 사실상의 표준 기술. 크고, 빠르게 성장하는 생태계를 가짐
 - Google Cloud GKE(Google **Kubernetes** Engine)
 - AWS Elastic Container Service(ECS) -> EKS(Elastic **Kubernetes** Service)
 - MS Azure Container Service(ACS) -> AKS(Azure **Kubernetes** Service)

-> 쿠버네티스 애플리케이션은 노트북에서부터 전세계 클라우드에서도 그대로 동작 가능



kubernetes



Kubernetes의 주요 기능

수 천대의 서버(클러스터)에 분산되어 있는 대규모 컨테이너와 자원을 선언한대로 관리해준다.

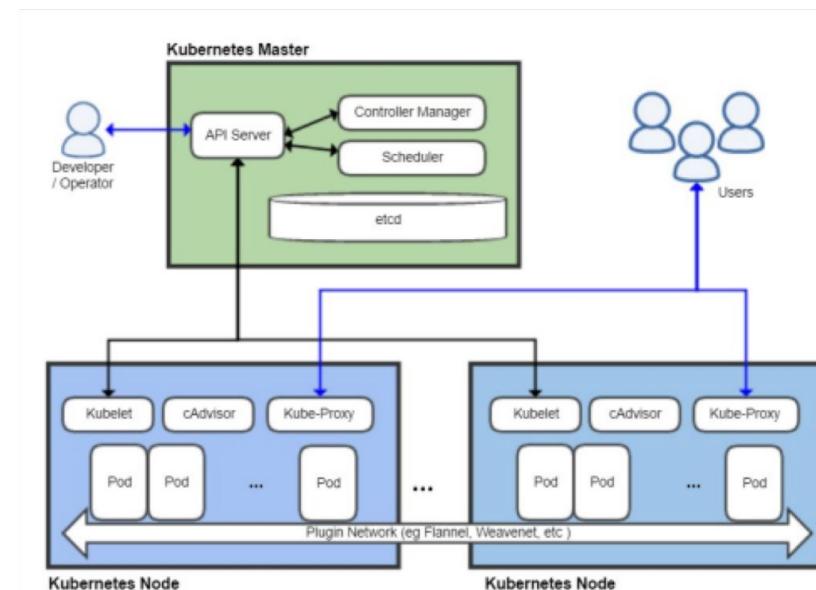
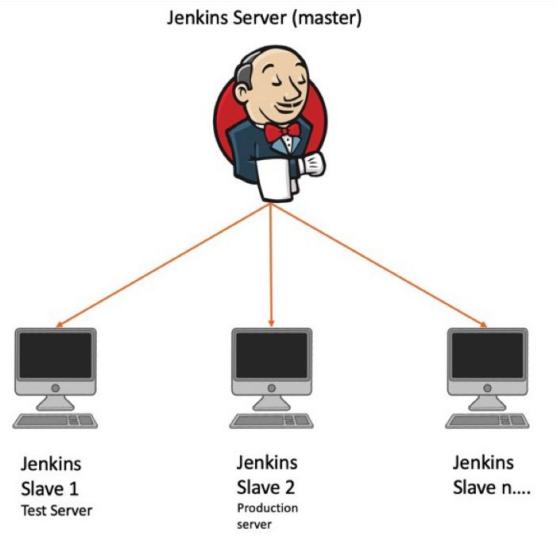
- 스케줄링 : 다수의 컨테이너를 다수의 호스트(클러스터)에 적절하게 분산 실행
- 상태 관리 : 원하는 상태(desired state)로 실행상태를 유지. (오토 페일오버 등)
- 배포 / 스케일링 : 서비스 중단 없이 자동화된 배포를 통해 업데이트하거나 스케일을 확장/축소
- 서비스 디스커버리 : 새로 뜨거나 옮겨간 컨테이너에 접근할 수 있는 방법을 제공
- Resource 연결 관리 : 컨테이너의 라이프 사이클에 맞춰 관련 자원을 연결 (설정 / 네트워크 / 스토리지)



Kubernetes 동작 원리

쿠버네티스의 아키텍처 : Master & Slave 아키텍처

어디서 많이 보던거.. 젠킨스...?!



쿠버네티스의 아키텍처

Master Node에서 동작하는 컴포넌트

kube-apiserver : 모든 요청을 받고, etcd에 저장

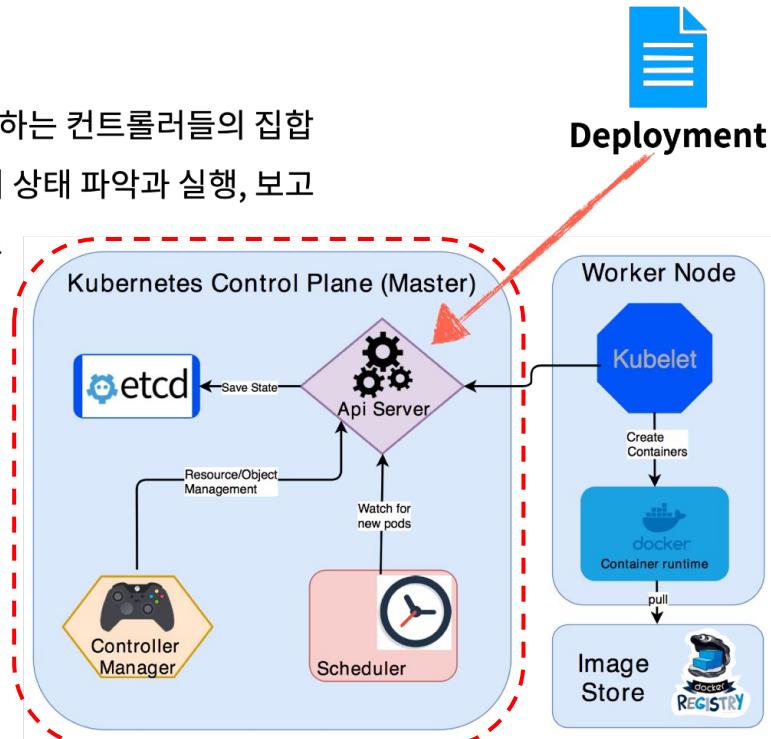
kube-controller-manager : 실제 Resource 관리를 수행하는 컨트롤러들의 집합

kubelet : Worker Node의 에이전트로서 Node와 Pod의 상태 파악과 실행, 보고

etcd : 모든 오브젝트의 상태를 저장하는 key/value 저장소

kube-scheduler : 새 pod의 생성을 관찰하여 스케줄링

마스터
노드



쿠버네티스의 아키텍처

Worker Node에서 동작하는 컴포넌트

kubelet : 워커 노드에 설치되는 에이전트.

kube-apiserver에 연결되어

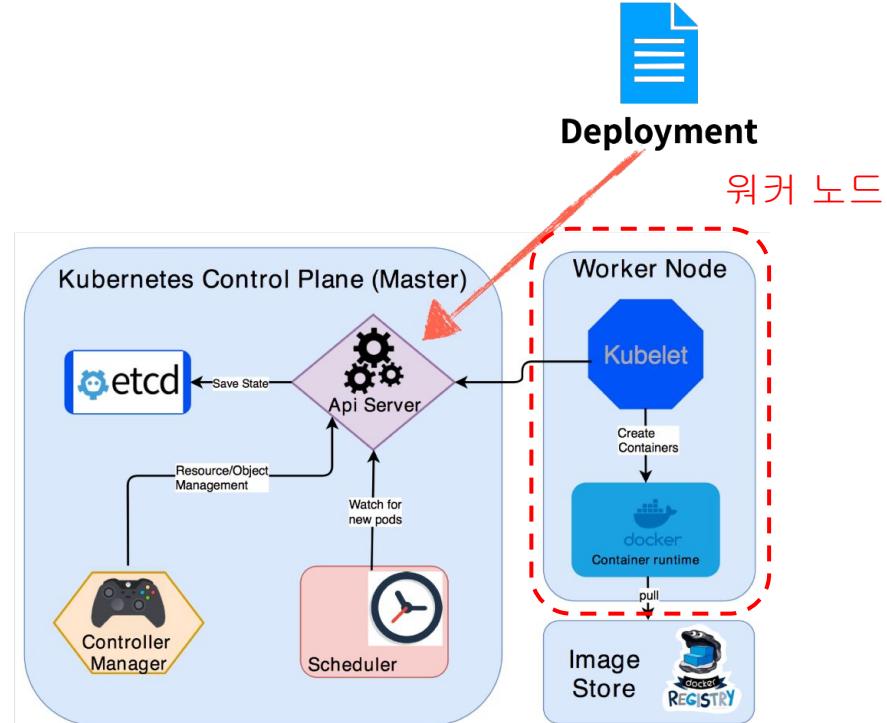
마스터의 명령을 받아

컨테이너를 실행 및 관리.

kube-proxy : Service 트래픽을 Pod로 연결

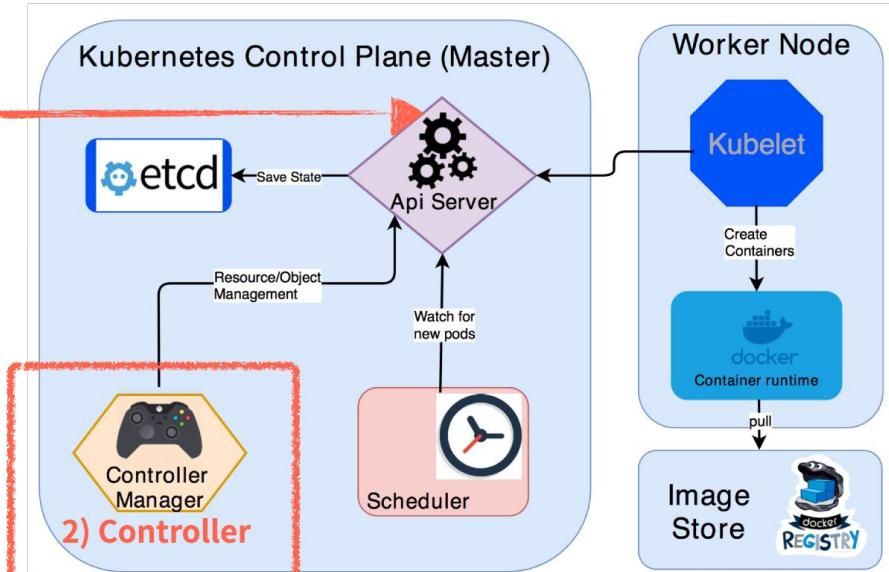
container-runtime : 컨테이너의 실행 환경

ex) docker, containerd



쿠버네티스의 아키텍처

Declarative API (선언형 API) = Object 와 Controller



쿠버네티스의 핵심 동작 원리

api Version

오브젝트 스키마의 버전. 컨트롤러는 버전별 호환성을 지원하기 위해 여러가지 버전을 처리해야합니다. 이때 이 정보를 참조하여 버전별로 알맞을 동작을 수행합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  ...상세 내용...
spec:
  ...상세 내용...
status:
  ...상세 내용...
```

kind

말대로 오브젝트의 종류. 개별 쿠버네티스 컨트롤러는 본인이 처리해야할 타입의 오브젝트의 변화를 항상 감시하고 있습니다. 예를 들어 **Deployment** 컨트롤러는 etcd에 저장된 데이터들 중에서 kind 가 Deployment 인 오브젝트의 업데이트에 대한 변화 이벤트만 항상 감시하고 있다가 적절한 동작을 실행합니다.

metadata

해당 리소스 오브젝트에 대한 메타데이터를 저장하는 필드입니다. 해당 리소스에 대한 이름과 네임스페이스 정보 등이 관리되며 쿠버네티스에서 리소스 이름과 네임스페이스의 조합은 항상 유니크합니다. 따라서, 쿠버네티스 클러스터 내의 리소스를 구별할 때는 오브젝트의 종류와 이름, 네임스페이스의 조합을 통해 구별할 수 있습니다.

spec

사용자가 원하는 리소스의 상태.

status

쿠버네티스가 조사한 현재 리소스의 실제 상태.

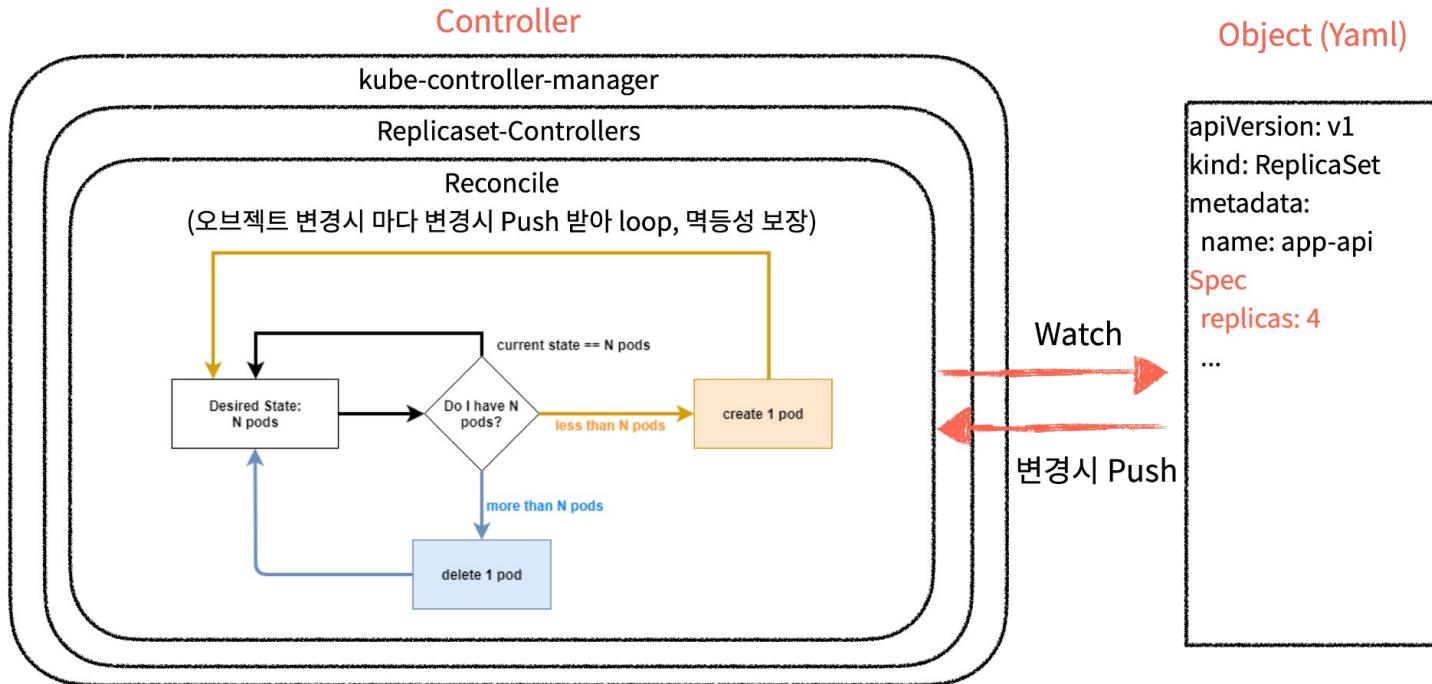
쿠버네티스의 아키텍처

쿠버네티스 Deployment 오브젝트 Watch 실습

```
curl -s -X GET 'https://$MASTER_NODE_IP]:6443/apis/apps/v1/deployments?watch=true' \
--header "Authorization: Bearer $TOKEN" \
--insecure | jq .
```

쿠버네티스의 아키텍처

Controller가 제출된 Object에 Desired State (=Spec) 와 Current State 를 비교, Resource 관리





DKOS(Datacenter of Kakao OS) 소개

KaaS = Kubernetes as a Service

카카오 개발자는 사내 프라이빗 클라우드에 스스로 쿠버네티스 클러스터를 생성해서 사용합니다.

#개인/조직/서비스별

#개별클러스터제공

#학습/개발/서비스용도

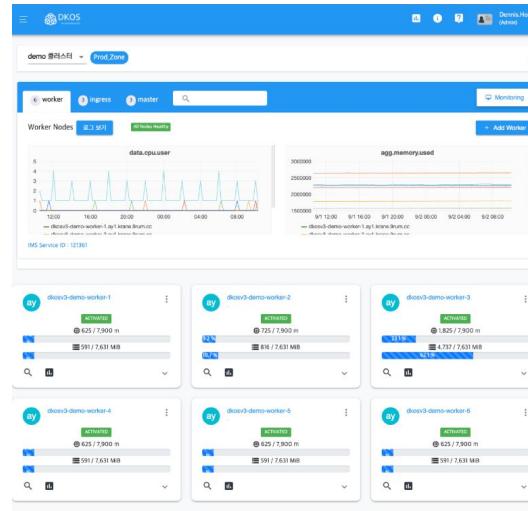
#프라이빗클라우드

#셀프서비스

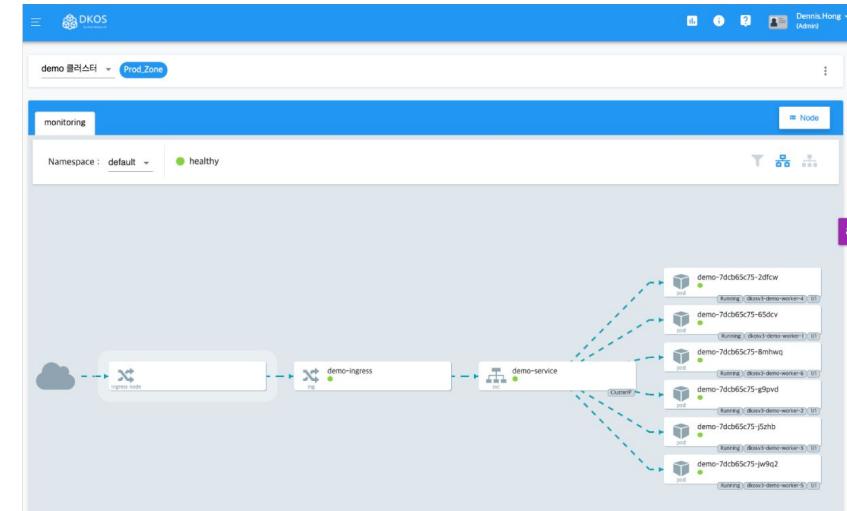
#수천개의클러스터

#어느새오픈7주년

수 만 대의 서버에서



수십 만 개의 컨테이너를 관리



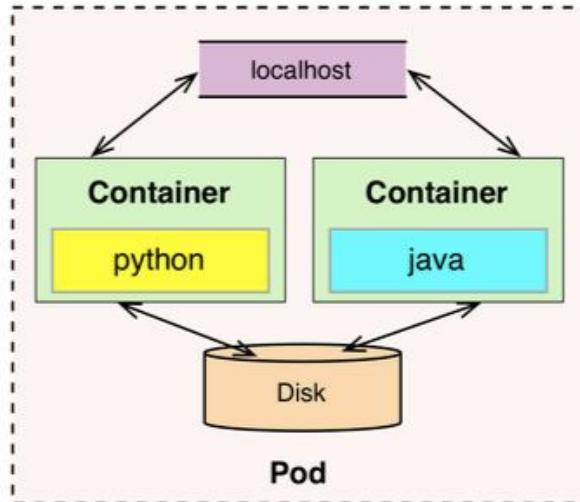


쿠버네티스의 주요 오브젝트

1. **Pod:** 쿠버네티스의 기본 배포 단위로 하나 이상의 컨테이너를 포함하는 컨테이너 그룹입니다.
2. **ReplicaSet:** Pod 의 복제본을 생성하고 관리합니다.
3. **Deployment:** ReplicaSet 을 관리하며, Pod 의 롤링 업데이트나 롤백을 수행합니다.
4. **Service:** Pod 을 연결하고 서비스 디스커버리를 제공합니다.
5. **Ingress:** 클러스터 외부에서 Service 에 접근할 수 있도록 하는 API 오브젝트입니다.
6. **ConfigMap:** 애플리케이션의 환경 설정 정보를 저장합니다.
7. **Secret:** 애플리케이션의 보안 정보를 저장합니다.
8. **PersistentVolume:** Pod 에서 사용할 수 있는 스토리지를 관리합니다.
9. **StatefulSet:** Pod 을 개별적인 ID 와 함께 관리합니다.
10. **DaemonSet:** 모든 노드에 Pod 을 실행하고 관리합니다.
11. **CronJob:** 스케줄링된 작업을 관리합니다.
12. **Job:** 일회성 작업을 관리합니다.

파드 (Pod)

파드는 언제나 노드 상에서 동작 (노드에서 실행되는 컨테이너로서 동작)
하나의 노드는 여러 개의 파드를 가질 수 있음



Pod as the deployment and management unit

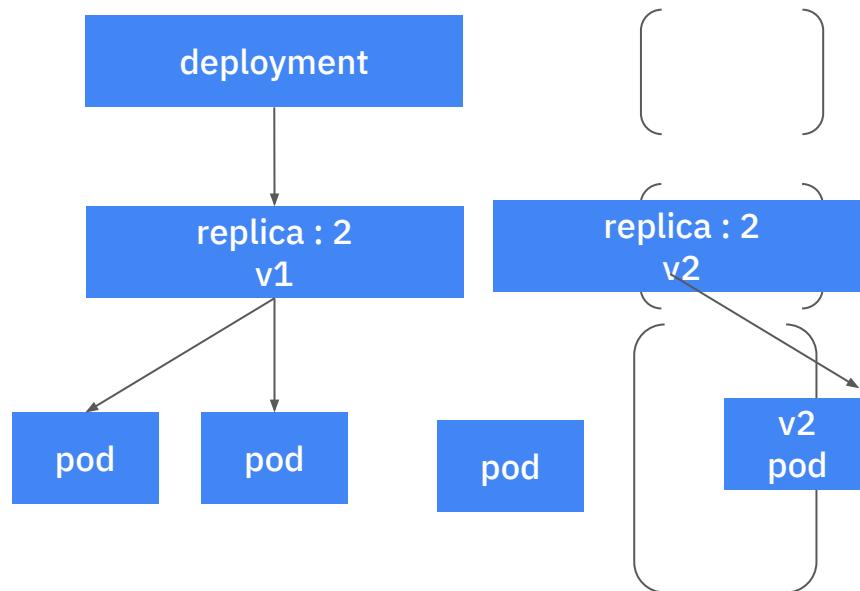
Pod 추상화를 통한 장점

- pause 컨테이너를 통해 namespace를 공유함으로서 **Pod** 내부의 컨테이너들은 **Pod** 외부와 격리된 컨테이너의 장점을 유지하면서, **Pod** 내부에서는 네트워크와 스토리지 자원을 공유.
- 단일 컨테이너를 단일 목적으로 분리하기 더 쉬워짐



디플로이먼트 (Deployment)

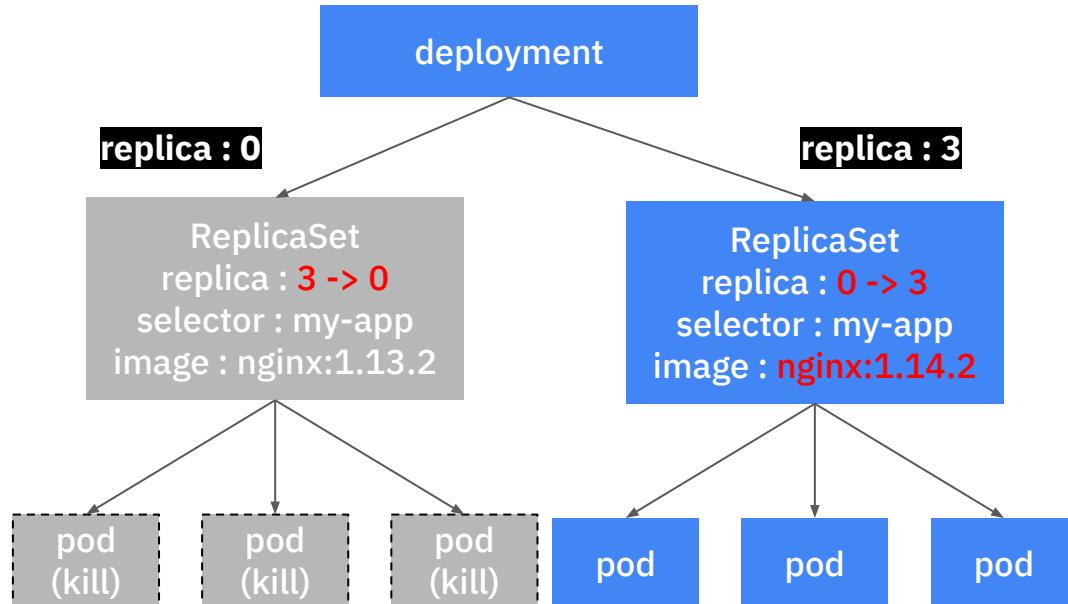
애플리케이션 인스턴스를 생성하고 업데이트 하는 역할 담당





디플로이먼트 (Deployment)

애플리케이션 인스턴스를 생성하고 업데이트 하는 역할 담당



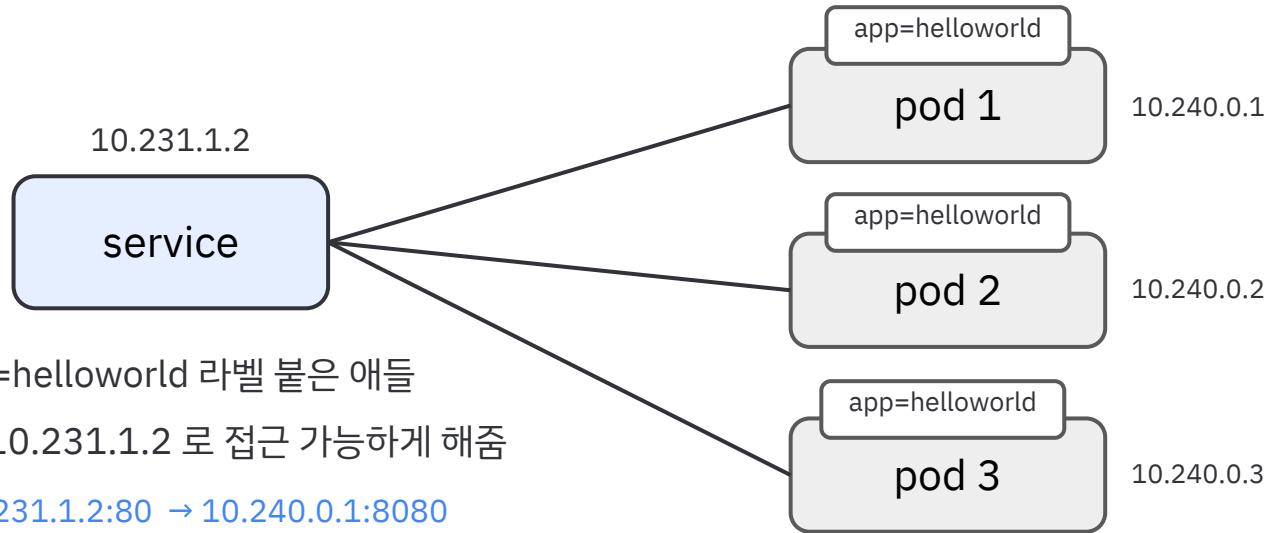
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx!1.14.2
          ports:
            - containerPort: 80
```

버전 변경!
= 신규 배포

■ 서비스 (Service)

pod 는 언제든 죽을 수 있는 존재! → **pod IP** 를 그대로 사용하기 어려움

이런 짧은 생명 주기의 **pod IP** 를 사용하지 않고, 고정된 단일 엔드포인트를 제공하자! → **Service**
여러개의 **pod** 를 하나의 엔드포인트 접근할 수 있도록 해줌



서비스 (Service)

서비스의 4가지 Type

- **ClusterIP (default)** : 이것이 디폴트 타입입니다. **ClusterIP**는 클러스터 내부에서만 사용 가능하며, 클러스터 내 **Pod**나 **Node**에서만 접근할 수 있습니다.
- **NodePort** : 이것은 클러스터의 모든 노드에 특정 포트를 해당 서비스로의 연결로 노출시키는 방법입니다. 예를 들어 **NodePort** 를 선언하고 **35000** 번 포트를 할당 받았다면 클러스터 내 모든 노드의 **35000** 번 포트로 접근하면
- 연결된 **Pod** 로 로드밸런싱해줍니다.
- **LoadBalancer** : 클라우드 서비스 제공업체의 로드밸런서를 이용하여 외부에서 접근할 수 있는 **External IP** 를 부여받을 수 있습니다.
- **External Name** : **External Name**은 외부 도메인 주소를 쿠버네티스 클러스터 안에서 고정된 서비스의 네임으로 사용하기 위해서 사용.

■ 서비스 (Service)

서비스의 IP는 서비스의 **Name** 이 자동으로 클러스터 **DNS**에 등록되므로 클러스터 내부에서는 서비스 **Name** 만으로 고정된 엔드포인트의 통신이 가능함.

서비스를 생성하면 생기는 단일 엔드포인트인 **IP**는 **2가지**로 나눌 수 있음

- **Cluster IP** : 클러스터 내부에서만 사용 가능 (**pod / node** 안에서만 가능)
- **External IP** : 클러스터 외부에서 사용 가능 → **LoadBalancer Type** 서비스

클러스터 클러스터
내부에서만 외부에서도
접근가능 접근가능

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	20d

인그레스 (Ingress)

인그레스는 클러스터 내부 서비스에 대한 외부 접근을 관리하는 오브젝트

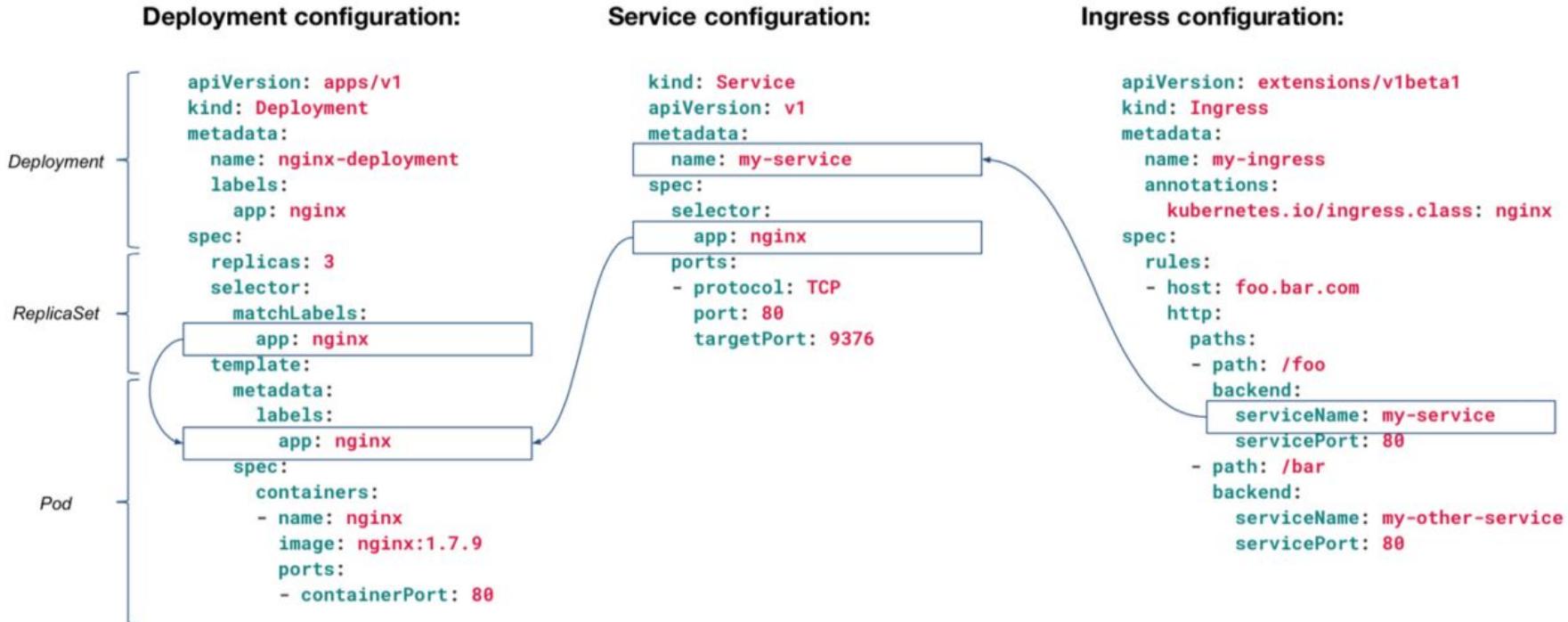
L7 Proxy 를 쿠버네티스 네이티브한 리소스로 관리할 수 있게 하자! → **Ingress**

- 다양한 라우팅 룰 설정 가능
- SSL 인증서 적용
- Rate Limit
- ...

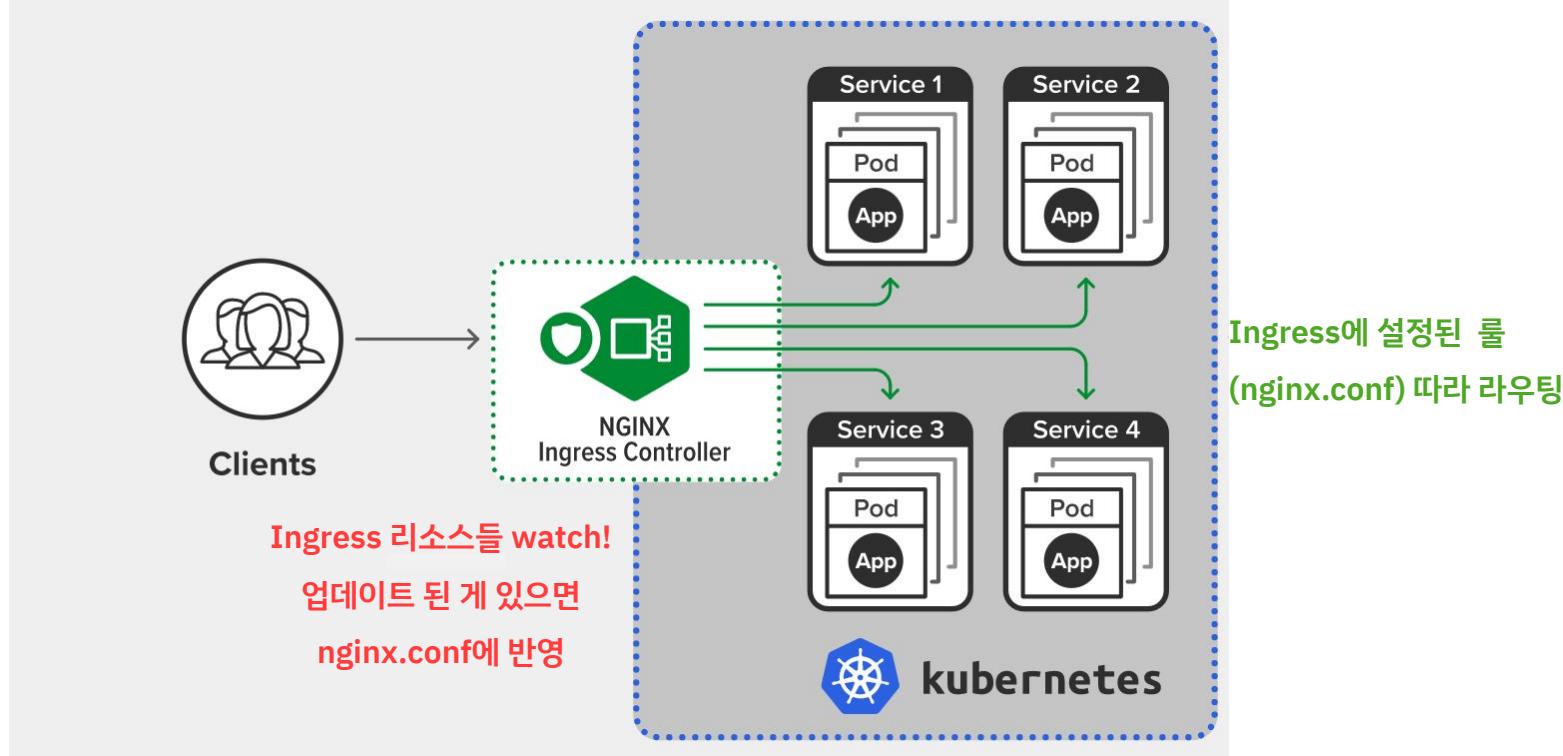
반드시 “인그레스 컨트롤러” 도 같이 배포해 주어야 동작함 (ex. ingress nginx controller)

- 프록시별 여러가지 구현체가 있음 (카카오 DKOS 는 nginx ingress controller 사용)

서비스 배포와 노출의 핵심, Deployment, Service, Ingress 간의 관계



쿠버네티스에서의 서비스 트래픽의 흐름



카카오의 마이크로서비스 (MSA)

MSA 가 최신 트렌드 아키텍처라고 항상 옳거나 모든 문제를 해결하는 만능기는 아님.

서비스 별, 조직 별 특성을 반영하여 상황과 요구사항에 맞는 다양한 아키텍처 패턴 사용.

서비스 성장에 따른 지속적인 아키텍처 개선

모놀리틱 아키텍처

초기 실험적 서비스는 작게 시작

적절한 규모의 분산 아키텍처

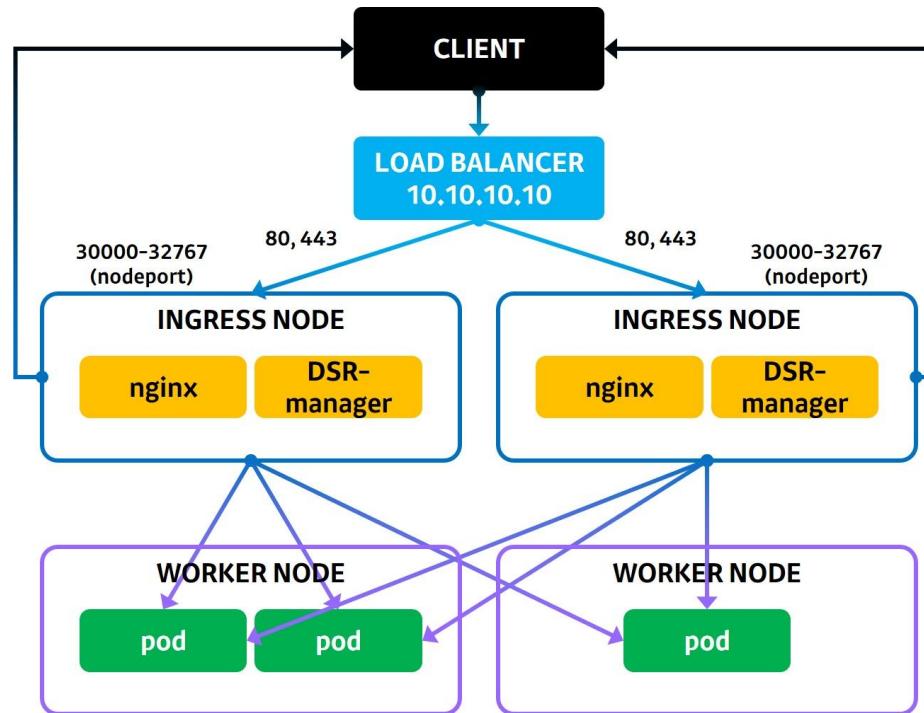
서비스 성장에 따른 시스템 요구 사항과 조직 규모에 따라서 분산

마이크로 서비스 아키텍처

대규모 서비스 일수록, 다양한 조직이 관여할 수록 분산

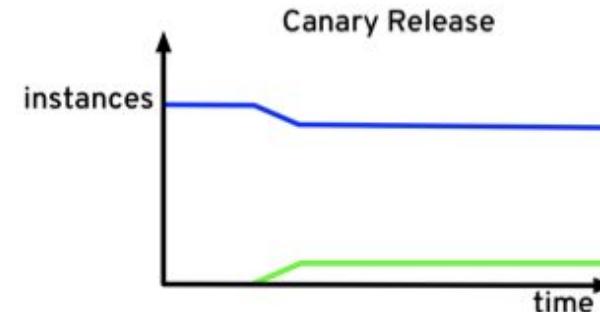
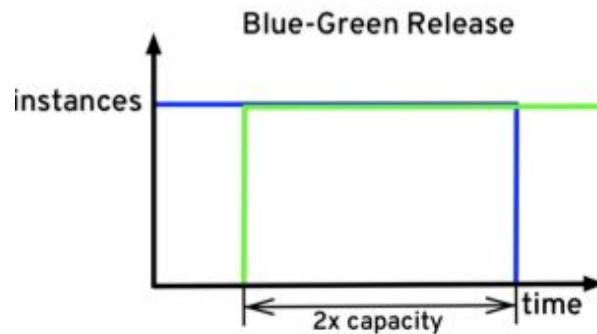
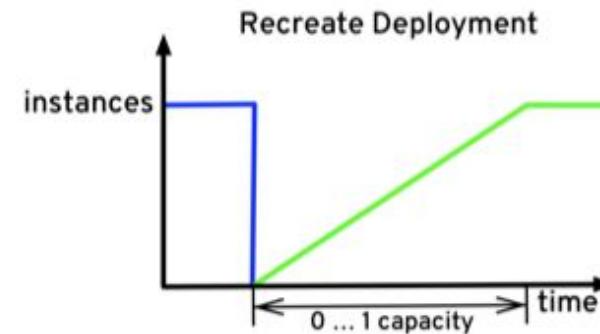
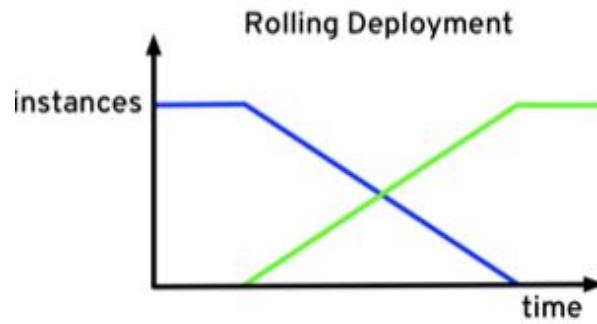


DKOS 서비스 구조





쿠버네티스를 이용한 배포 전략



Deployment and release strategies

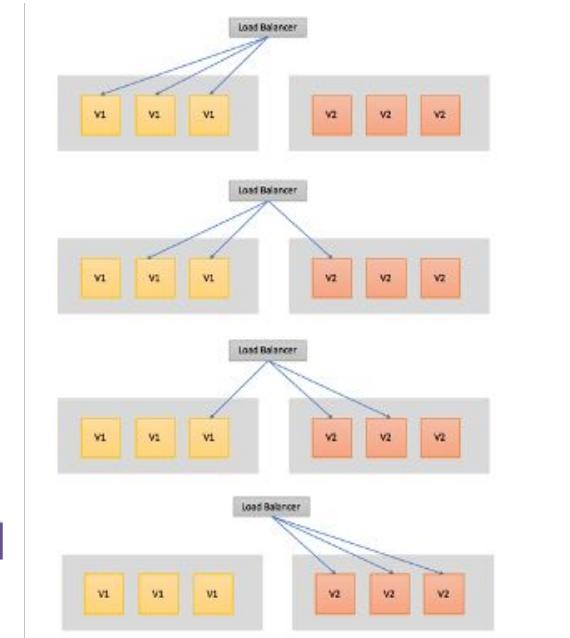
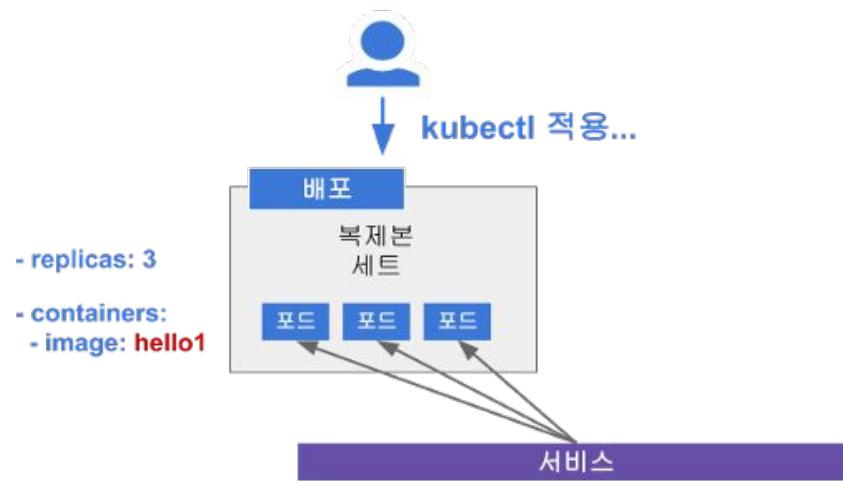


쿠버네티스를 이용한 배포 전략

1) Rolling update

배포가 새 버전으로 업데이트되면 새로운 복제본 세트가 생성되고 기존 복제본 세트의 복제본이 줄어들면서 새 복제본 세트의 복제본 수가 서서히 늘어납니다. -> n개 씩 천천히 신규버전 pod로 바꿔나감.

start / pause / resume / rollback 기능 제공.

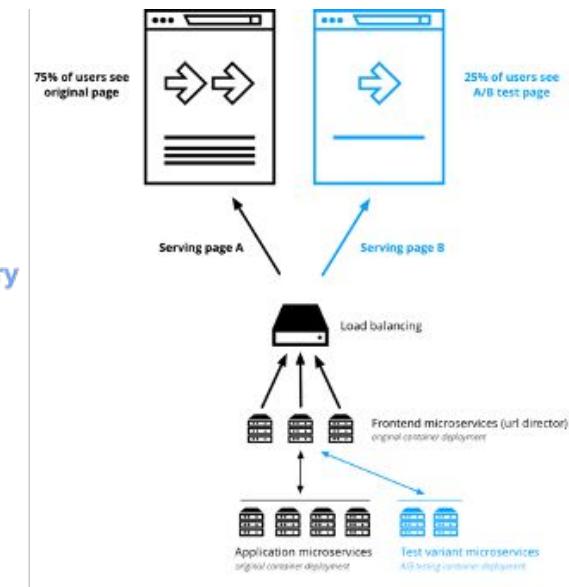
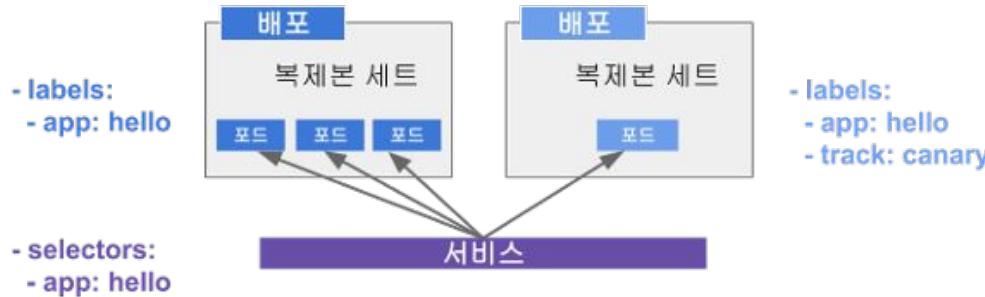




쿠버네티스를 이용한 배포 전략

2) Canary deployment

카나리 배포는 새 버전의 별도 배포와 안정적인 일반 배포 및 카나리 배포를 타겟팅하는 서비스로 이루어집니다.



*프로덕션의 카나리 배포 - 세션 연관성(sessionAffinity) 설정

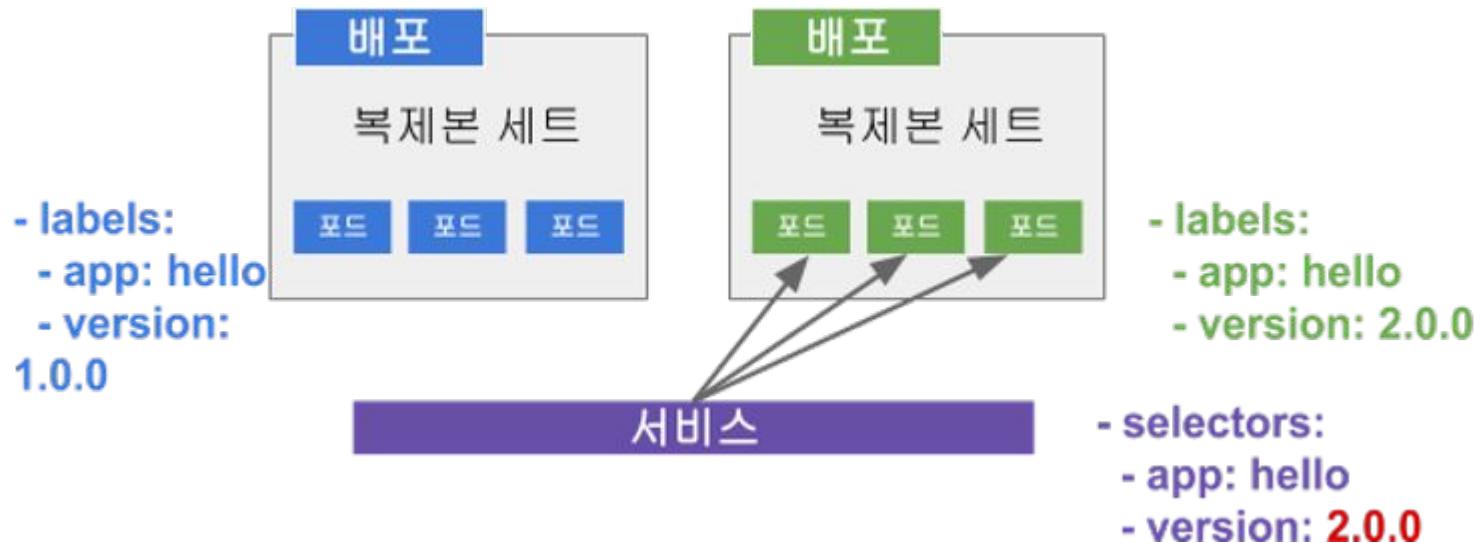
IP 주소가 동일한 모든 클라이언트의 요청이 동일한 버전의 애플리케이션으로 전송됩니다.



쿠버네티스를 이용한 배포 전략

3) Blue-Green deploy

Blue 버전과 Green 버전 둘다 띄워놓은 상태에서 한번에 서비스 엔드포인트를 스위칭.
동시에 띄워놓고 진행하기 때문에 2배의 자원이 필요함.



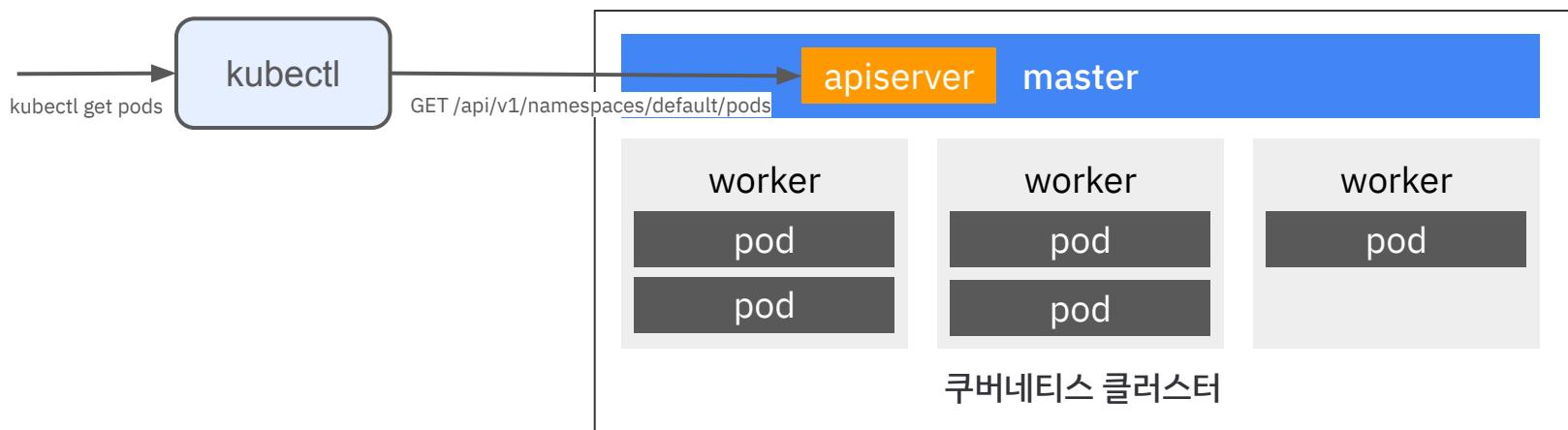
kubernetes 실습

kubernetes 리소스 알아보기



kubectl

- 쿠버네티스 클러스터에 여러 리소스들을 생성하고 관리할 수 있는 CLI 툴
- (CLI 커맨드로 k8s API를 호출할 수 있도록 해주는 툴)
- 완만한 사용법은 요기에 <https://kubernetes.io/ko/docs/reference/kubectl/cheatsheet/>





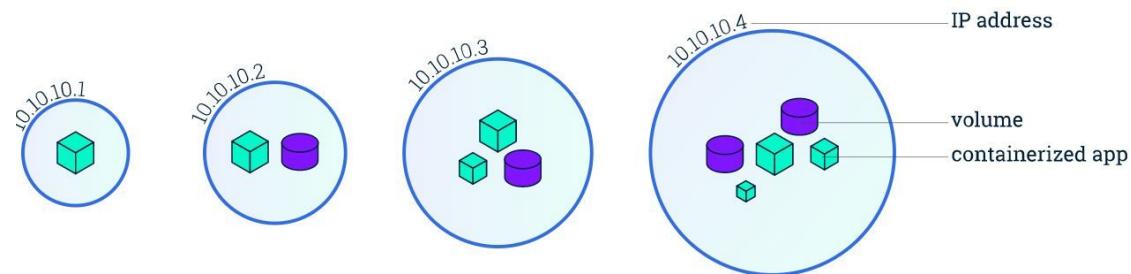
파드 (Pod)

사용자 애플리케이션!

하나 또는 여러 개 애플리케이션 컨테이너의 그룹을 나타내는 k8s 의 추상적인 개념

쿠버네티스에서 생성/관리/배포 가능한 가장 작은 컴퓨팅 단위

```
● ● ●  
apiVersion: v1  
kind: Pod  
metadata:  
  name: nginx  
spec:  
  containers:  
    - name: nginx  
      image: nginx:1.14.2  
      ports:  
        - containerPort: 80
```

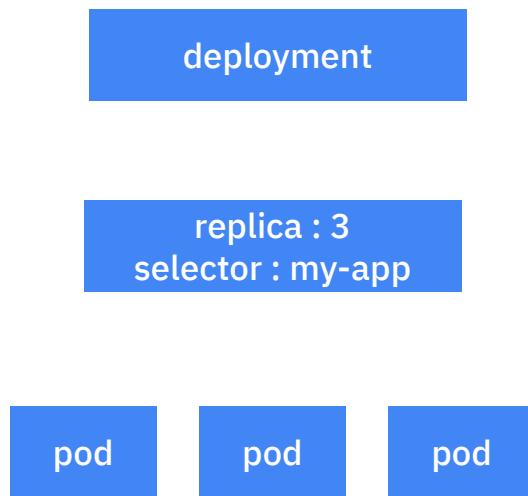


* 쿠버네티스 상에서는 리소스 스펙을 yaml 형태로 정의



디플로이먼트 (Deployment)

pod 찍어내는 템플릿 (어떤 앱인지 / 몇개를 띄울 것인지 / hc 어떻게 할지 / 업데이트 방법 등등)

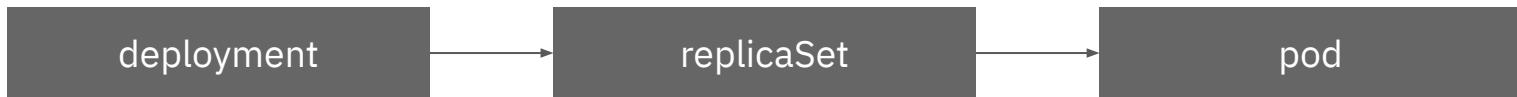


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```



디플로이먼트 (Deployment)

Deployment 생성시 ReplicaSet / Pod 가 자동으로 생성 됨



ReplicaSet?

항상 “지정한 파드 개수” 만큼 실행될 수 있도록 관리 해주는 역할

Ex. replica=5 설정시 파드 1개가 죽을 경우 다시 신규 파드1개를 생성하여 5개가 유지되도록 해줌

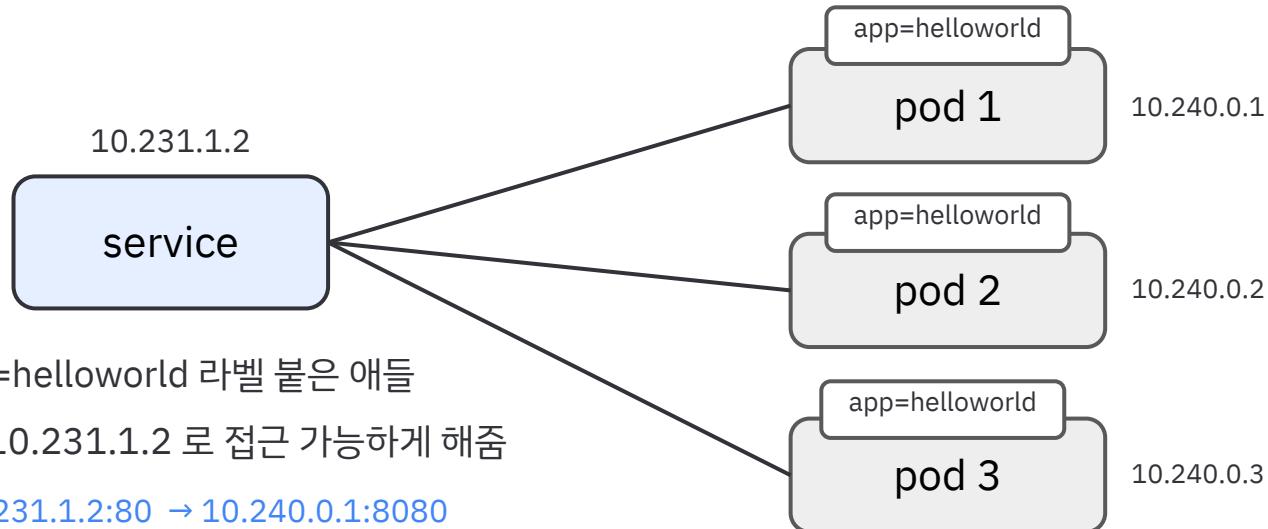


서비스 (Service)

pod 는 언제든 죽을 수 있는 존재! → pod IP 를 그대로 사용하기 어려움

이런 짧은 생명 주기의 pod IP 를 사용하지 않고, 고정된 단일 엔드포인트를 제공하자! → Service

여러개의 pod 를 하나의 엔드포인트 접근할 수 있도록 해줌

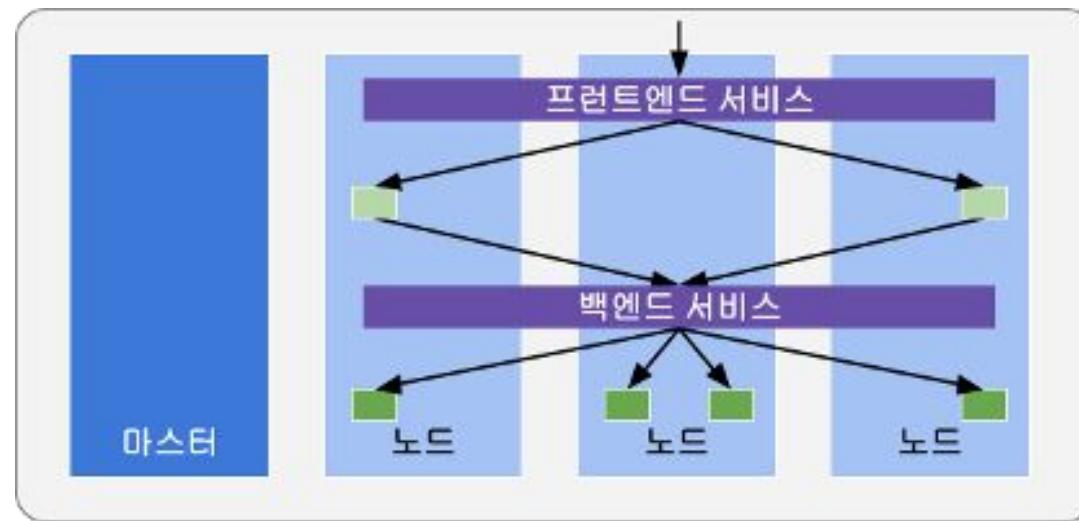




서비스 (Service)

Service Layer 추상화의 장점

Application Stack Layer 간에 Service 라는 추상화 계층을 통해 부하 분산 및 네트워크 접근이 편리해짐





서비스 (Service)

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy # label
spec:
  containers:
    - name: nginx
      image: nginx:stable
      ports:
        - containerPort: 80
          name: http-web-svc
```

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector: 어떤 pod 들을 묶을 것이냐 (label 기반)
  app.kubernetes.io/name: proxy # selector
  ports:
    - name: name-of-service-port
      protocol: TCP 어떤 포트를 노출할 것이냐
      port: 80
      targetPort: 8080
```



서비스 (Service)

4가지 타입이 존재

- ClusterIP (default)
- NodePort
- LoadBalancer
- ExternalName

서비스의 IP를 2가지로 나눌 수 있음

- Cluster IP : 클러스터 내부에서만 사용 가능 (pod / node 안에서만 가능)
- External IP : 클러스터 외부에서 사용 가능 → LoadBalancer Type 서비스



인그레스 (Ingress)

인그레스는 클러스터 내부 서비스에 대한 외부 접근을 관리하는 오브젝트

L7 Proxy 를 쿠버네티스 네이티브한 리소스로 관리할 수 있게 하자! → Ingress

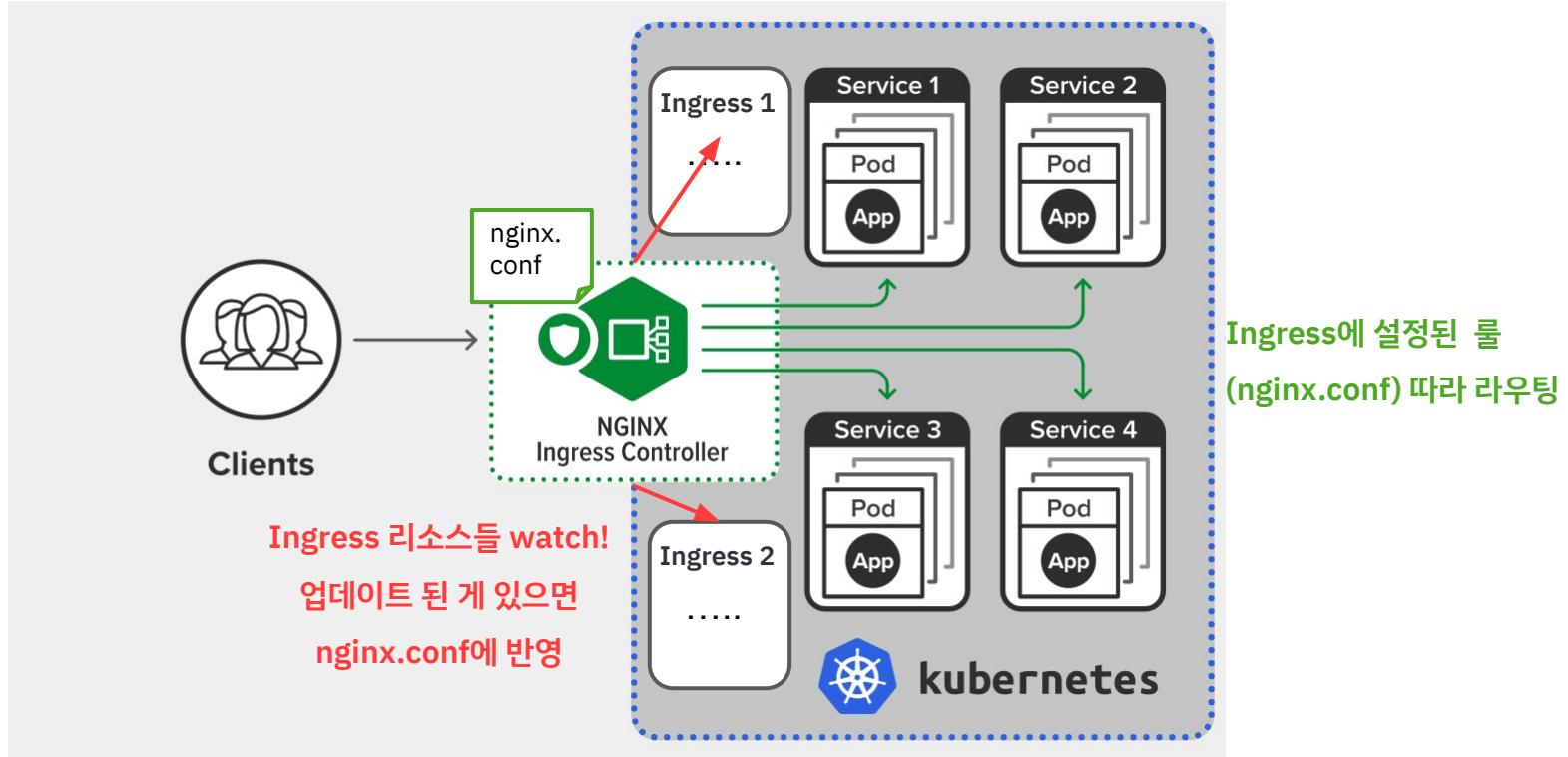
- 다양한 라우팅 룰 설정 가능
- SSL 인증서 적용
- Rate Limit
- ...

반드시 “인그레스 컨트롤러” 도 같이 배포해 주어야 동작함 (ex. ingress nginx controller)

- 프록시별 여러가지 구현체가 있음 (DKOS 는 nginx ingress controller 사용)

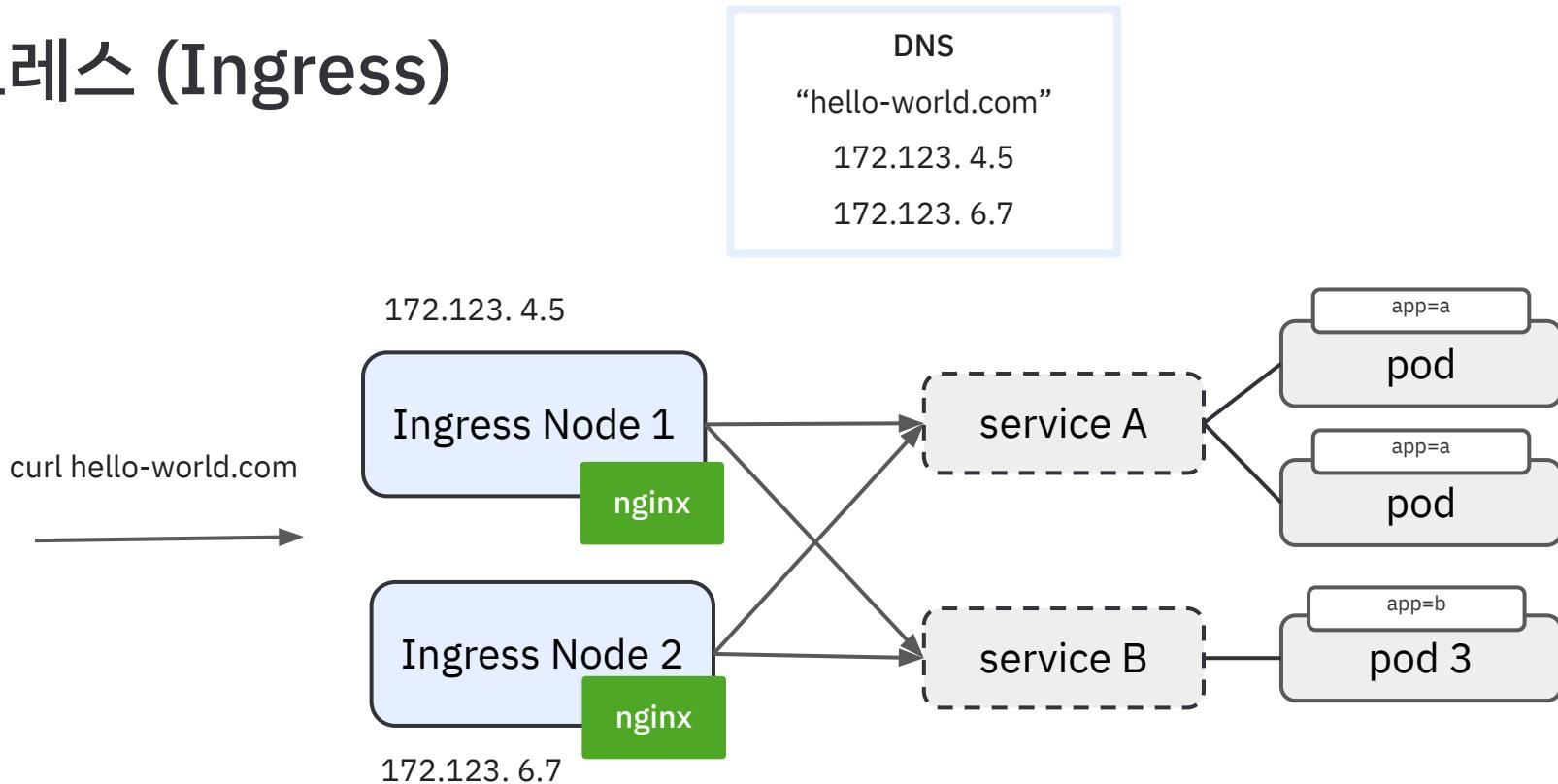


인그레스 (Ingress)



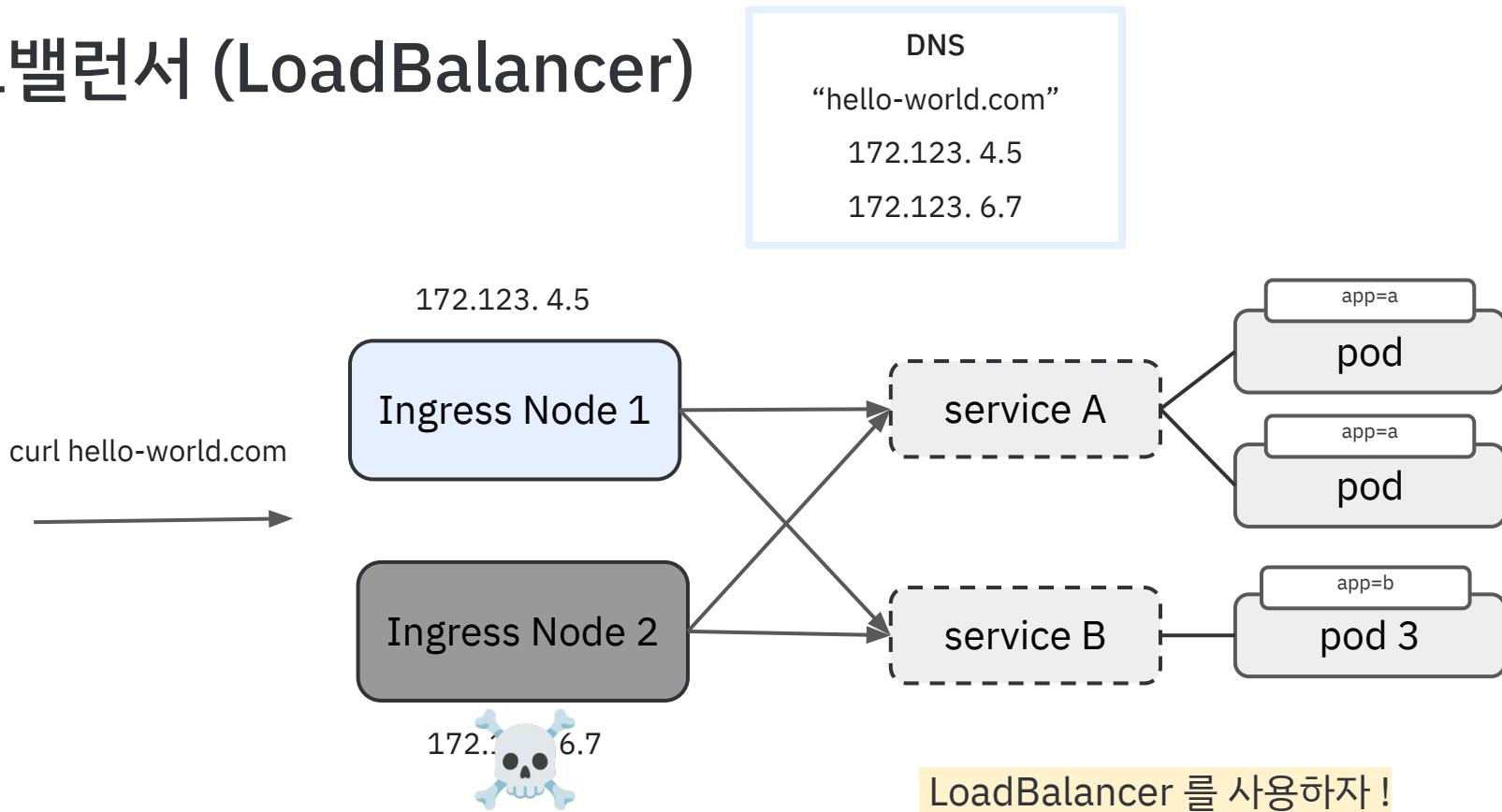


인그레스 (Ingress)





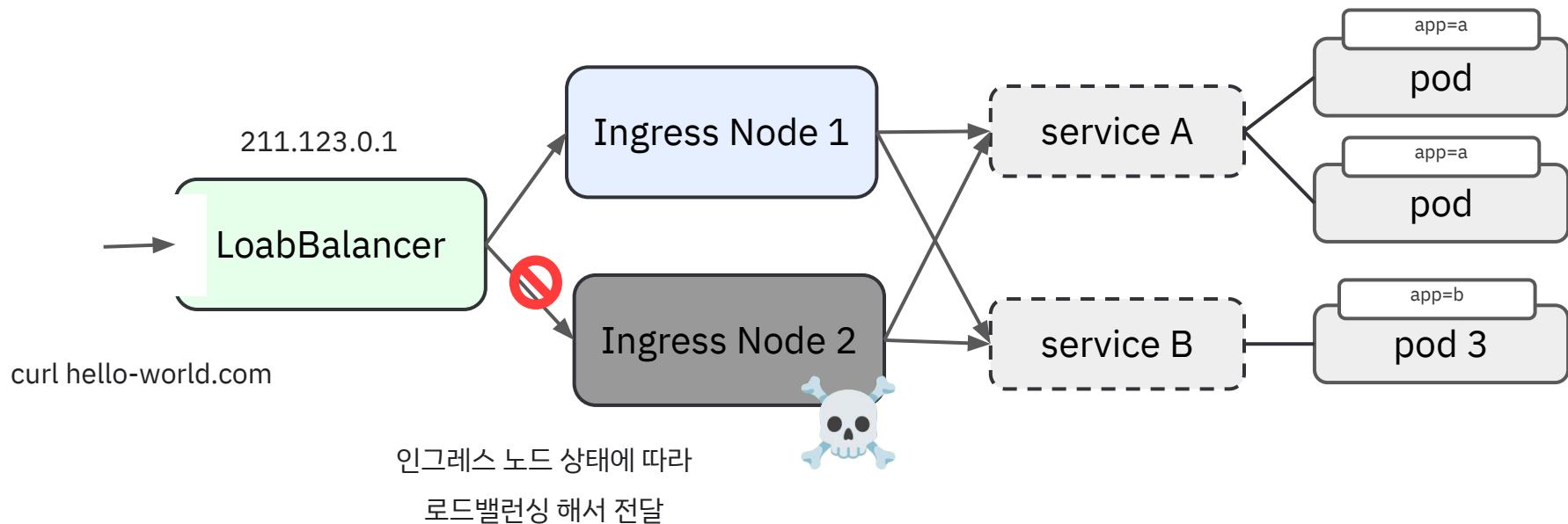
로드밸런서 (LoadBalancer)





로드밸런서 (LoadBalancer)

DNS
“hello-world.com”
211.123.0.1





kubectl cheatsheet

노드 조회

```
kubectl get node  
kubectl get node -o wide
```

더 자세한 것들은 요기애~

<https://kubernetes.io/ko/docs/reference/kubectl/cheatsheet/>

리소스 생성

```
kubectl apply -f {filename}.yaml
```

리소스 삭제

```
kubectl delete -f {filename}.yaml  
kubectl delete {resourceKind} {resourceName}
```

리소스 조회

```
kubectl get all
```

특정 리소스 상세 조회

```
kubectl get {resourceKind} {resourceName} -o wide|yaml|json  
kubectl describe {resourceKind} {resourceName}
```

pod 에서 명령어 수행

```
kubectl exec {podName} -- {command} # 커맨드 한번 수행  
kubectl exec -it {podName} -- {command} # 바로 종료되지 않고 유지
```

pod 로그 조회

```
kubectl logs -f {podName}
```



kubectl GUI

맥북 사용자라면 kubernetes GUI 클라이언트 중, [Lens](#)를 많이 사용할텐데, 2023년 1월 2일부터 기업 유저는 Pro 구독 (\$199/1년)을 해야 사용 가능하다.

오픈소스 버전의 Lens 소스코드는 MIT 라이센스로 깃허브에 공개되어 있어 직접 빌드해서 써도 되고, 빌드한 바이너리를 올려둔 OpenLens 리파지토리에서 받아서 써도 된다.

- Source: <https://github.com/lensapp/lens>
- Binary Repo: <https://github.com/MuhammedKalkan/OpenLens>



1

2

3

4

5

6

7

8

9

10

11

12

13

{ Designing Cloud Native Application; }

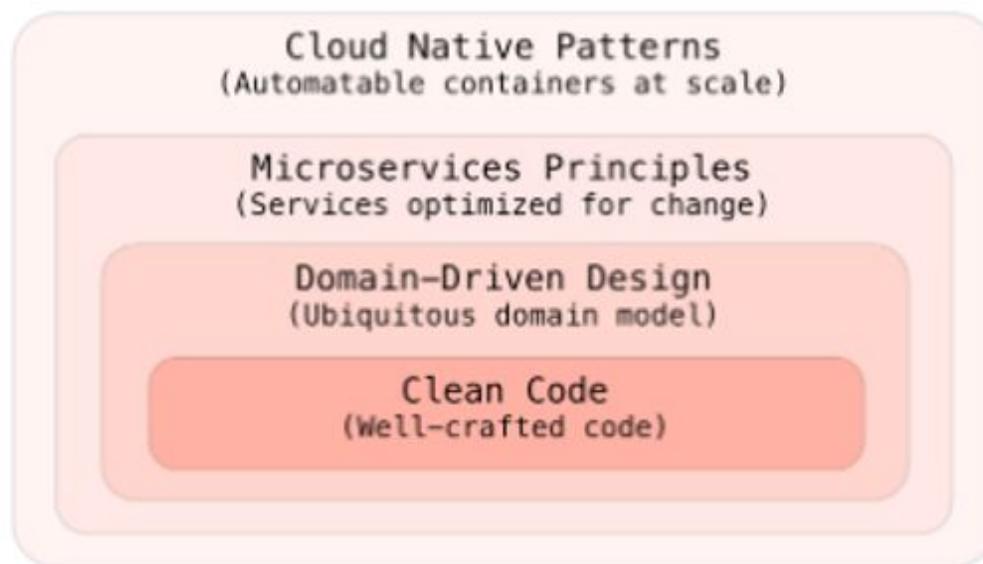
클라우드 네이티브 애플리케이션 설계



Cloud Native Pattern 이란 ?

Clean Code + DDD + MSA 다 배웠더니 Cloud Native 도 알아야 되는 세상이

oL o



대규모 컨테이너 마이크로 서비스를 자동화하기 위한 패턴

분산 응용 프로그램을 설계. 확장성, 탄력성 고려

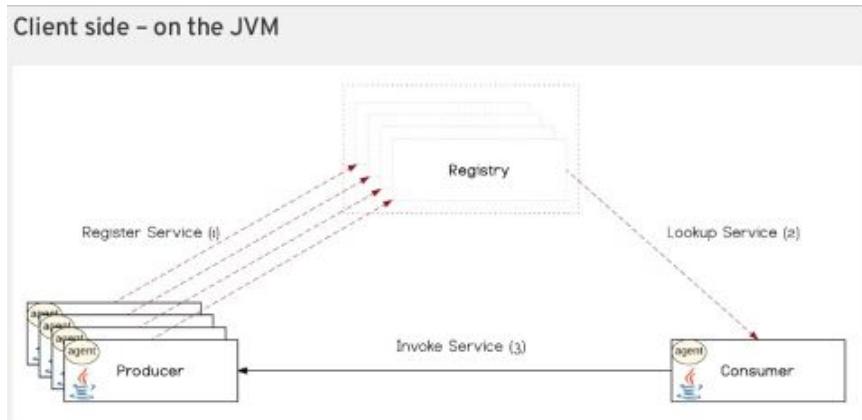
실제 세계에 가까운 비즈니스 관점에서 소프트웨어 설계

깨끗한 코드 작성, 자동화 된 테스트, 리팩토링, 코드 품질

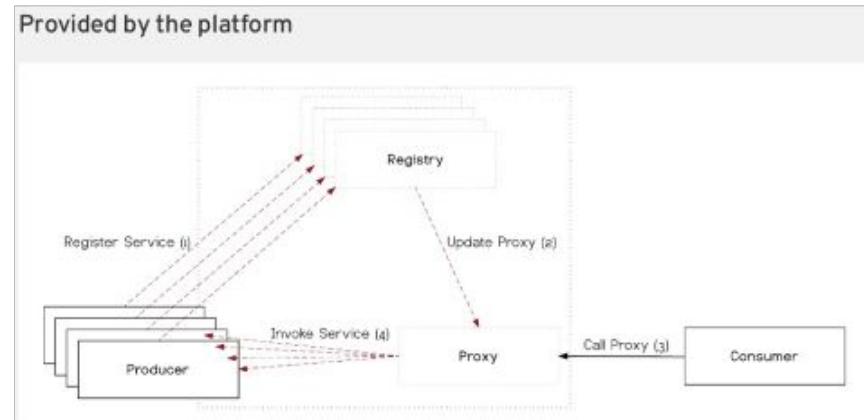


Cloud Native Pattern 이란 ?

マイクロ 서비스 ア키텍처



클라우드 네이티브 아키텍처

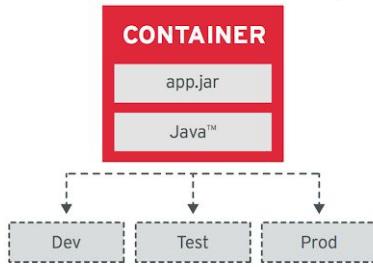


<- 오케스트레이터의 매니지먼트 사이클과
어플리케이션이 유기적으로 연동되게
해주는 설계가 필요.



Principles of Container-based Application Design

Image Immutability Principle



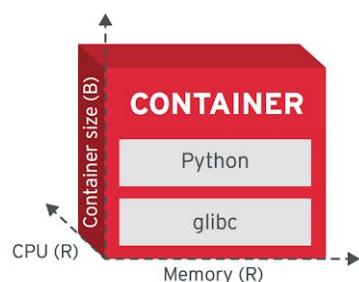
High Observability Principle



Process Disposability Principle



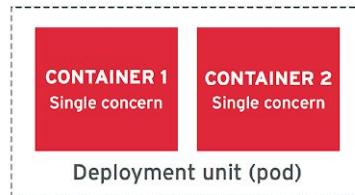
Runtime Confinement Principle



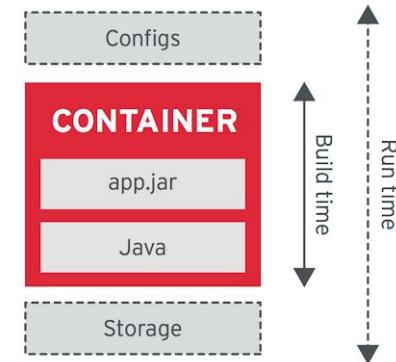
Lifecycle Conformance Principle



Single Concern Principle



Self-Containment Principle

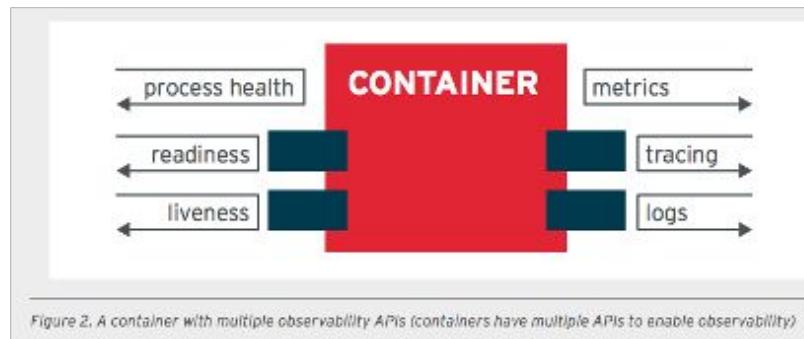




Principles of Container-based Application Design

1. Log는 stdout / stderr 로 뿌려 주세요.

- Docker 에 log를 관리해주는 기능이 있음.
>> File로 쌓아줌. Rotate도 해줌. json 파싱도 해줌, docker CLI로 log 확인도 됨
(awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog.. 등등 플러그인 제공)
- Kubernetes / Mesos 도 이 기능을 활용한 기능들을 제공함.
- Container 안에서 File로 로그를 쌓게되면 컨테이너 종료/삭제시 Log 유실됨.
- Log Size 커지면 Disk Full 나서 해당 호스트 서버가 죽는 참사가 일어날 수도 있음.
(같은 호스트에 돌고 있던 다른 컨테이너들은 의문의 1페..)
- 반드시 파일로 쌓아야 할 경우엔 volume attach 해서 쌓아주세요.

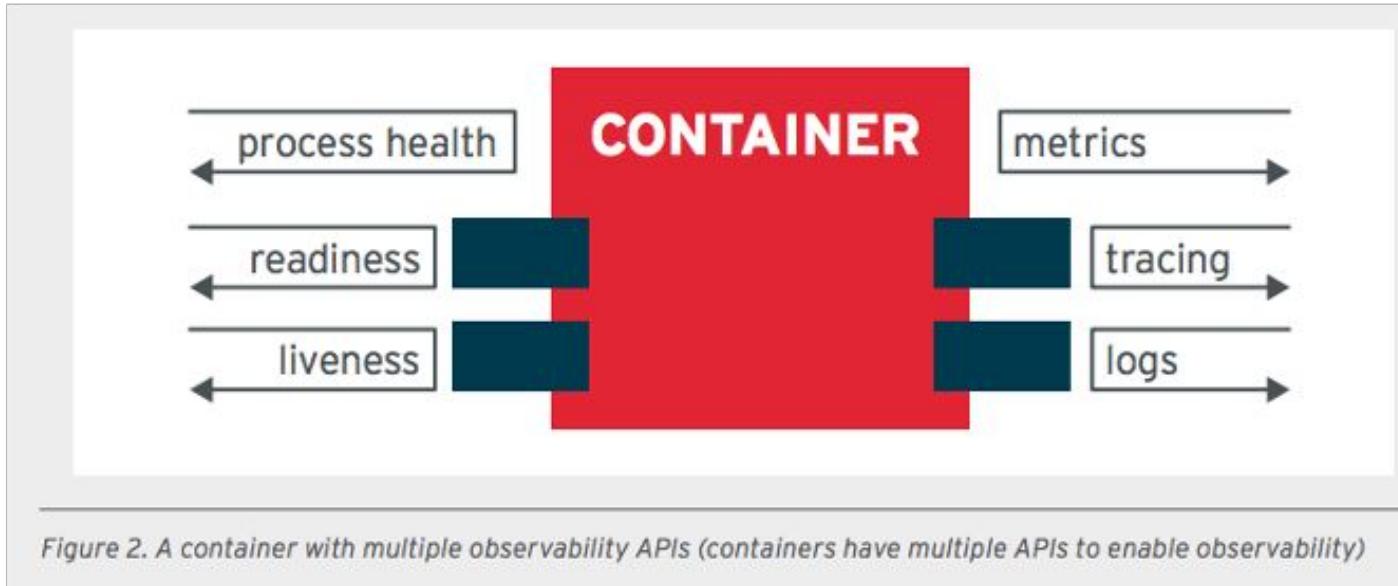




Principles of Container-based Application Design

2. 컨테이너 헬스체크와 트래픽을 보내도 되는지 확인 할 수 있는 API를 제공해주세요.

- liveness : fail 하면 컨테이너가 비정상으로 인지하고 죽임. 없으면 auto fail over 를 못함
- readiness : ok 하면 컨테이너가 서비스 준비가 된 것으로 인지하고 트래픽을 흘려보내기 시작함. (k8s)



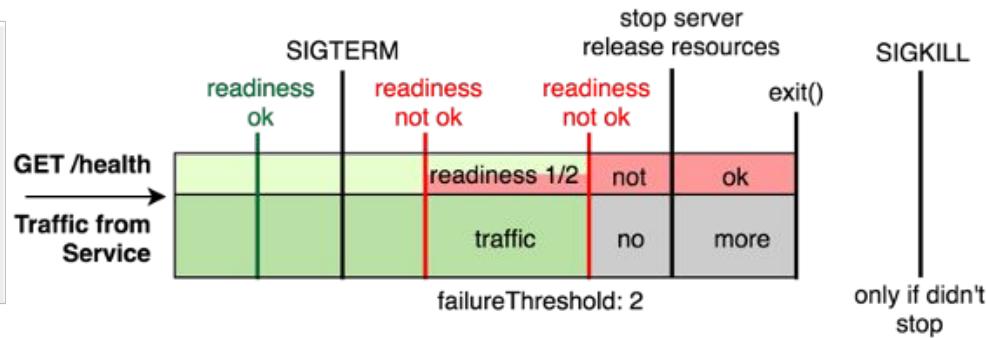
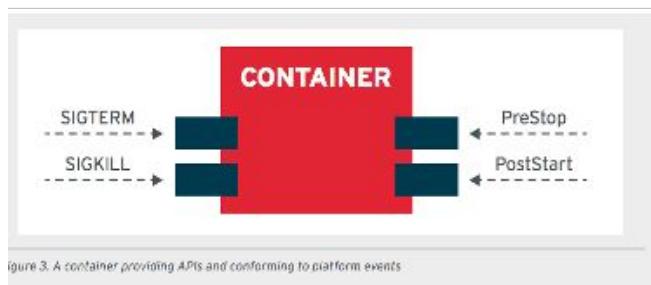


Principles of Container-based Application Design

3. 컨테이너에는 플랫폼에서 오는 이벤트에 반응하여 준수하는 방법이 있어야 합니다.

- app 이 SIGTERM 을 받았을 때 처리하던 task를 마무리하고 종료하도록 처리해주세요.
- 무중단 배포를 위한 필수 사항.
- Docker 가 컨테이너를 죽일 때 SIGTERM 을 먼저 주고 SIGKILL 을 통해 죽인다.
- 이때 sigterm SIGTERM 대한 처리가 없으면 앱은 처리하던 task를 내팽개치고 죽어버림.
- 그외에도 kubernetes 에서는 PreStop, PostStart 액션을 지정할 수 있음

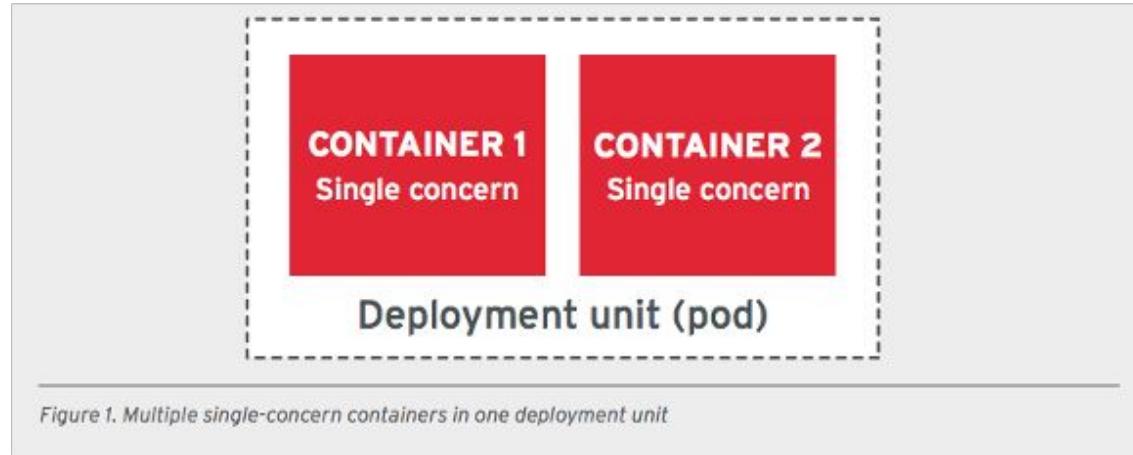
예시 : <https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/>



Principles of Container-based Application Design

4. 컨테이너 1개에는 가급적 단일 목적 1개 프로세스만 띄워주세요.

- init 프로세스가 종료되면 나머지도 다 죽음.
- 프로세스 별 헬스체크도 어려움.
- 프로세스별 / 목적별 배포가 분리 될 수 있음.
- SINGLE CONCERN PRINCIPLE 대로 설계하면 컨테이너 이미지 재사용이 높아짐.
- 운영이 편리하고 성능 병목이나 이슈 파악이 용이함.

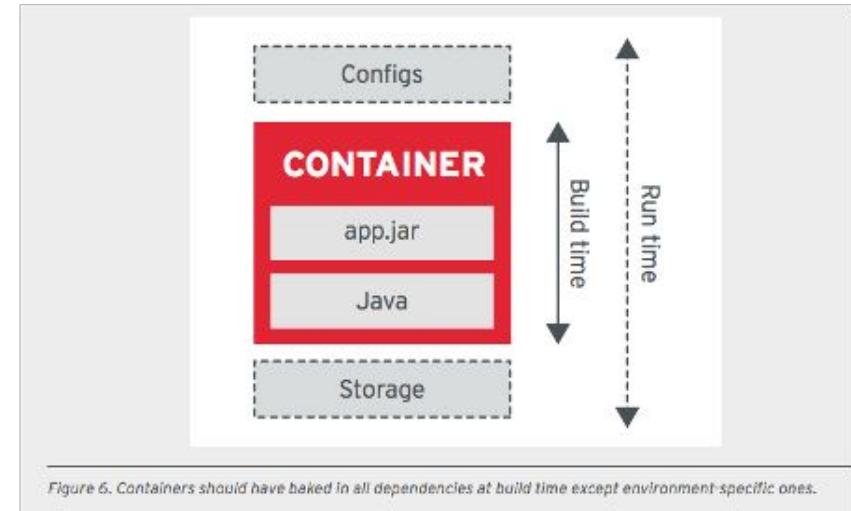
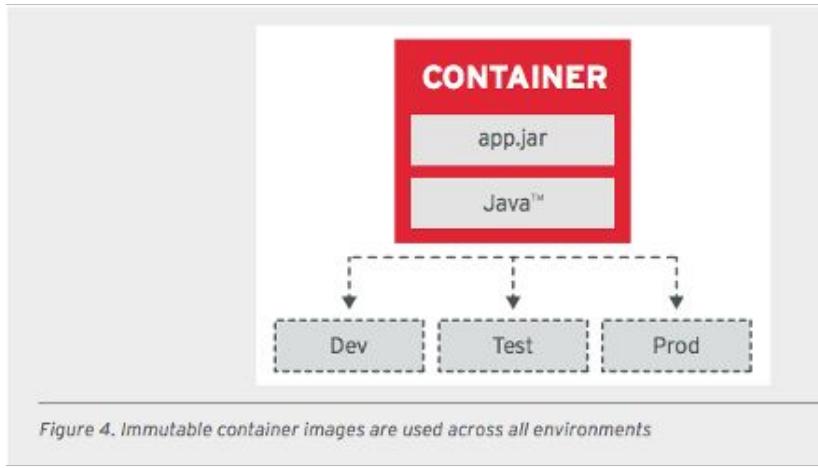


Principles of Container-based Application Design

5. 모든 배포 환경에 사용되는 이미지는 동일하게.

build time에선 이미지에 담고 **run time**에 필요한건 **configMap**, **Storage**, **Secret**에 담아주세요.

- prod 환경과의 유사성을 최대한 보장.
- 동적인 설정이나 정보는 **configMap**, **Storage**, **Secret**으로 전달.

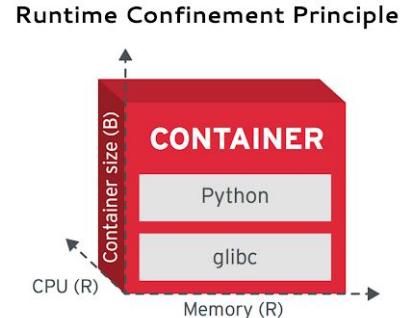




Principles of Container-based Application Design

6. 적절한 Resource Request / Limit 설정을 반드시 확인해주세요.

- 설정하지 않으면 최소값으로 설정되어 가장 후순위 프로세스가 됨.
- 호스트 서버에 남는 자원을 사용해서 동작하기 때문에 실서비스 전에는 문제가 없어보임
- 하지만 서버에 부하를 받기 시작하면 다른 프로세스에 밀려 느려지거나 OOM 으로 가장 먼저 정리됨.





Principles of Container-based Application Design

7. 컨테이너화된 애플리케이션은 가능한 한 임시적이어야 하며 언제든지 다른 컨테이너 인스턴스로 대체될 준비가 되어 있어야 합니다.

- 개별 컨테이너는 세션 정보 등 상태(**state**)를 가지면 안됩니다.(외부화 / 분산)
- 컨테이너 애플리케이션은 언제든 종료되고 다시 시작 될 수 있다는 가정하에 개발해야합니다.
- 애플리케이션의 시작과 종료가 빨라야 합니다.

Process Disposability Principle





Principles of Container-based Application Design

1. Single concern principle (SCP) : container 1개에는 1개의 process 만 띄워주세요.
2. High observability principle (HOP) : app readiness / health check API 제공, log를 stdout, stderr로 뿌려주세요 .
3. Life-cycle conformance principle (LCP) : SIGTERM 처리 필요, PreStop, PostStart 활용 .
4. Image immutability principle (IIP) : dev/test/prod 에 모두 같은 이미지 사용. conf 만 다르게.
5. Process disposability principle (PDP) : 상태를 외부화하거나 분산시켜야하며 응용 프로그램을 시작하고 종료하는 것이 빨라야합니다.
6. Self-containment principle (S-CP) : 모든 디펜던시는 빌드타임에 이미지에 담고, 환경에 따라 다른 정보는 런타임 시에 configMap, Storage에 담아주세요 .
7. Runtime confinement principle (RCP) : 컨테이너 app이 필요한 CPU / MEM / Disk 에 대해 고려되어야 함.



1

2

3

4

5

6

7

8

9

10

11

12

13

{ Designing K8s Cluster; }

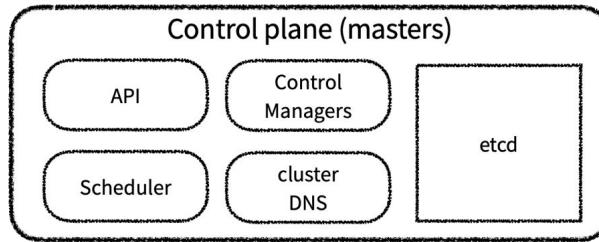
쿠버네티스 클러스터 설계



하나의 큰 쿠버네티스 클러스터, 엔터프라이즈에 적합할까?

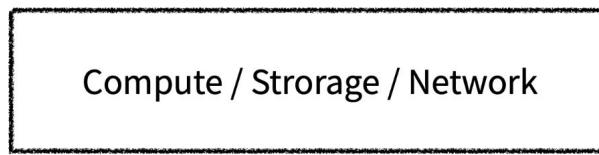
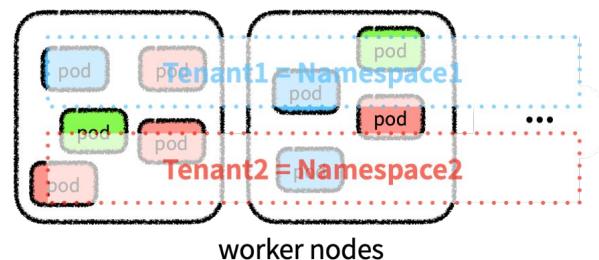
One Big Cluster

Soft Multi-Tenancy : ops affects all tenant



1) Control plane 을 공유

- 모두 동일한 쿠버네티스 버전/설정이 적용
- 모든 Pod가 하나의 cluster DNS 를 사용
- 모든 User가 하나의 Control plane 을 공유해서 사용



Cloud Infrastructure



Kubernetes 클러스터를 고장내는 다양한 방법들..

Case1) 시작하자마자 종료되는 Pod를 배포한다.

- 앱 시작 과정에서 환경 변수나 외부 API 호출을 통한 설정 등을 읽어오는데 실패하면 그대로 종료하게 구현된 앱에서 발생.
- 컨테이너 실행 command 나 script 가 잘못되어 아무 것도 하지 않고 종료되는 경우 발생.



Type	Reason	Age	From	Message
Normal	NodeHasSufficientDisk	57m (x86 over 15h)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeHasSufficientDisk
Normal	NodeNotReady	52m (x49 over 15h)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeNotReady
Normal	NodeReady	46m (x73 over 15h)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeReady
Normal	Starting	43m	kubelet, dkosv3-destroy-test-worker-1	Starting kubelet.
Warning	SystemOOM	43m (x7 over 43m)	kubelet, dkosv3-destroy-test-worker-1	System OOM encountered
Normal	NodeHasSufficientPID	43m	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeHasSufficientPID
Normal	NodeAllocatableEnforced	42m	kubelet, dkosv3-destroy-test-worker-1	Updated Node Allocatable limit across pods
Warning	ContainerGCFailed	4m55s	kubelet, dkosv3-destroy-test-worker-1	rpc error: code = DeadlineExceeded desc = context deadline exceeded
Normal	NodeNotReady	4m54s (x6 over 43m)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeNotReady
Normal	NodeHasSufficientMemory	4m54s (x8 over 43m)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeHasSufficientMemory
Normal	NodeHasSufficientDisk	4m54s (x8 over 43m)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeHasSufficientDisk
Normal	NodeHasNoDiskPressure	4m54s (x8 over 43m)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeHasNoDiskPressure
Normal	NodeReady	3m18s (x2 over 41m)	kubelet, dkosv3-destroy-test-worker-1	Node dkosv3-destroy-test-worker-1 status is now: NodeReady

Worker Node All Kill					
NAME	STATUS	ROLES	AGE	VERSION	
dkosv3-destroy-test-master-1	Ready	master	15h	v1.11.5	
dkosv3-destroy-test-worker-1	NotReady	node	15h	v1.11.5	
dkosv3-destroy-test-worker-2	NotReady	node	15h	v1.11.5	
dkosv3-destroy-test-worker-3	NotReady	node	15h	v1.11.5	
dkosv3-destroy-test-worker-4	NotReady	node	15h	v1.11.5	

```

apiVersion: apps/v1
kind: Deployment
...
spec:
replicas: 1000
spec:
containers:
- name: death
image: ubuntu
imagePullPolicy: Always
command: ['echo', 'hello sidney']
  
```

클러스터를 불지옥으로 보내는 티켓

*imagePullPolicy: always 옵션

까지 뒀다면 container image

저장소도 함께 지옥으로 보낼 수 있다.



Kubernetes 클러스터를 고장내는 다양한 방법들..

Case2) History Limits 설정없이 CronJob을 걸어 둔다.

무한히 누적되는 SUCCESSFUL Job

NAME	READY	DESIRABLE	SUCCESSFUL	AGE
deploy-kubernetes-127996-1	1	0	248d	
deploy-kubernetes-128241-1	1	0	248d	
deploy-kubernetes-128297-1	1	0	248d	
deploy-kubernetes-128584-1	1	0	246d	
deploy-kubernetes-128585-1	1	0	246d	
deploy-kubernetes-128591-1	1	0	246d	
deploy-kubernetes-128614-1	1	0	245d	
deploy-kubernetes-128700-1	1	0	237d	
deploy-kubernetes-128705-1	1	0	236d	
deploy-kubernetes-128707-1	1	0	236d	
deploy-kubernetes-128712-1	1	0	236d	
deploy-kubernetes-128747-1	1	0	236d	
deploy-kubernetes-128748-1	1	0	236d	
deploy-kubernetes-128750-1	1	0	236d	
deploy-kubernetes-128757-1	1	0	236d	
deploy-kubernetes-128771-1	1	0	236d	
deploy-kubernetes-128773-1	1	0	236d	
deploy-kubernetes-128798-1	1	0	236d	
deploy-kubernetes-131714-1	1	0	233d	
deploy-kubernetes-131731-1	1	0	233d	
deploy-kubernetes-131844-1	1	0	233d	
deploy-kubernetes-131873-1	1	0	233d	
deploy-kubernetes-132078-1	1	0	230d	
deploy-kubernetes-132512-1	1	0	228d	
deploy-kubernetes-132531-1	1	0	228d	
deploy-kubernetes-132532-1	1	0	228d	
deploy-kubernetes-132608-1	1	0	228d	
deploy-kubernetes-132625-1	1	0	228d	
deploy-kubernetes-132627-1	1	0	228d	
deploy-kubernetes-132628-1	1	0	228d	
deploy-kubernetes-132629-1	1	0	228d	
deploy-kubernetes-132631-1	1	0	228d	
deploy-kubernetes-132632-1	1	0	228d	
deploy-kubernetes-132634-1	1	0	228d	
deploy-kubernetes-132635-1	1	0	228d	
deploy-kubernetes-132642-1	1	0	228d	
deploy-kubernetes-132712-1	1	0	228d	
deploy-kubernetes-132723-1	1	0	228d	
deploy-kubernetes-132724-1	1	0	228d	
deploy-kubernetes-132745-1	1	0	228d	
deploy-kubernetes-132746-1	1	0	228d	
deploy-kubernetes-132750-1	1	0	228d	
deploy-kubernetes-132751-1	1	0	228d	
deploy-kubernetes-132758-1	1	0	228d	
deploy-kubernetes-132759-1	1	0	228d	
deploy-kubernetes-132802-1	1	0	228d	
deploy-kubernetes-132927-1	1	0	227d	

무한히 누적되는 Completed Pod

NAME	READY	STATUS	RESTARTS	AGE
deploy-kubernetes-128934-1-6sbbq	0/1	Completed	0	33d
deploy-kubernetes-128963-1-jhoms	0/1	Completed	0	33d
deploy-kubernetes-128969-1-c5zlw	0/1	Completed	0	33d
deploy-kubernetes-128996-1-7v8vl	0/1	Completed	0	33d
deploy-kubernetes-128991-1-592cs	0/1	Completed	0	33d
deploy-kubernetes-128994-1-wh57h	0/1	Completed	0	33d
deploy-kubernetes-128998-1-64kb6	0/1	Completed	0	33d
deploy-kubernetes-129001-1-46m6t	0/1	Completed	0	33d
deploy-kubernetes-129001-1-1e6df	0/1	Completed	0	33d
deploy-kubernetes-129013-1-bz8fg	0/1	Completed	0	33d
deploy-kubernetes-129019-1-spjj4	0/1	Completed	0	33d
deploy-kubernetes-129020-1-a5rpk	0/1	Completed	0	33d
deploy-kubernetes-129027-1-az6qq	0/1	Completed	0	33d
deploy-kubernetes-129028-1-tmdv6	0/1	Completed	0	33d
deploy-kubernetes-129029-1-lzgcj	0/1	Completed	0	33d
deploy-kubernetes-129032-1-grl6h	0/1	Completed	0	33d
deploy-kubernetes-129035-1-mfnx4	0/1	Completed	0	33d
deploy-kubernetes-129035-1-g89v1	0/1	Completed	0	33d
deploy-kubernetes-129036-1-8rcm7	0/1	Completed	0	33d
deploy-kubernetes-129041-1-hsrtr	0/1	Completed	0	33d
deploy-kubernetes-129041-1-zppmm	0/1	Completed	0	33d
deploy-kubernetes-129050-1-btxhv	0/1	Completed	0	33d
deploy-kubernetes-129053-1-t2z4t	0/1	Completed	0	33d
deploy-kubernetes-129055-1-7zdbh	0/1	Error	0	33d
deploy-kubernetes-129055-1-16vhf	0/1	Error	0	33d
deploy-kubernetes-129055-1-nzf6s	0/1	Error	0	33d
deploy-kubernetes-129056-1-4fmwv	0/1	Completed	0	33d
deploy-kubernetes-129061-1-5rjdh	0/1	Error	0	33d
deploy-kubernetes-129061-1-668sv	0/1	Error	0	33d
deploy-kubernetes-129061-1-dv8p9	0/1	Error	0	33d
deploy-kubernetes-129061-1-g5gvv	0/1	Error	0	33d
deploy-kubernetes-129062-1-10000	0/1	Completed	0	33d
deploy-kubernetes-129064-1-8ph96	0/1	Completed	0	33d
deploy-kubernetes-129065-1-qtb84	0/1	Completed	0	33d
deploy-kubernetes-129067-1-ts2kb	0/1	Completed	0	33d
deploy-kubernetes-129068-1-tlfm6	0/1	Completed	0	33d
deploy-kubernetes-129069-1-ddm5j	0/1	Completed	0	33d
deploy-kubernetes-129079-1-12jxg	0/1	Completed	0	33d
deploy-kubernetes-129071-1-sdxpl	0/1	Completed	0	33d
deploy-kubernetes-129072-1-nlx1l	0/1	Completed	0	33d
deploy-kubernetes-129074-1-fs29j	0/1	Completed	0	33d
deploy-kubernetes-129075-1-hs57h	0/1	Completed	0	33d
deploy-kubernetes-129076-1-r5f9q	0/1	Completed	0	33d
deploy-kubernetes-129099-1-1c91j	0/1	Completed	0	33d
deploy-kubernetes-129099-1-21j8q	0/1	Completed	0	33d
deploy-kubernetes-129099-1-19wt	0/1	Completed	0	33d
deploy-kubernetes-129093-1-lsslp	0/1	Completed	0	33d
deploy-kubernetes-129094-1-k4wrl	0/1	Completed	0	33d
deploy-kubernetes-129094-1-k4wrl	0/1	Completed	0	33d

1) etcd 서서히 부하 증가, kube-system component OOM 유발

2) 장기간에 걸쳐 서서히 쌓여오다가 어느 순간 클러스터 컨트롤이 안되기 시작하더니 한순간에 모든 마스터 장애

[.spec.successfulJobsHistoryLimit](#)

[.spec.failedJobsHistoryLimit](#)

기본값은 0 (무한)

클러스터 전체에 영향을 줄 수 있을 만한 설정의 기본값을

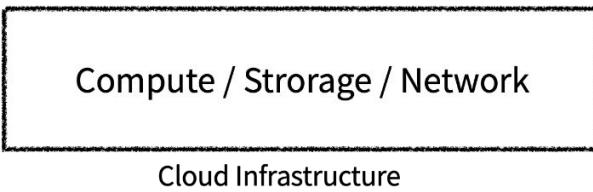
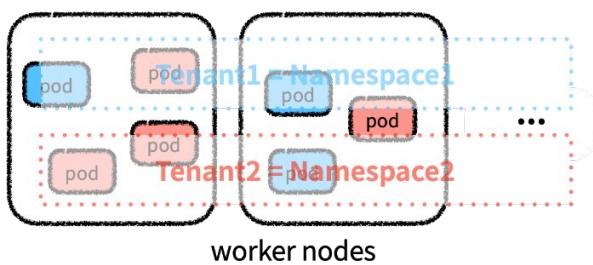
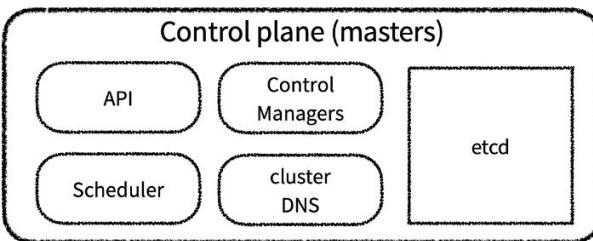
쿠버네티스가 안전하게 보장해주진 않습니다.



One big cluster, or many small ones?

One Big Cluster

Soft Multi-Tenancy : ops affects all tenant



1) Control plane 을 공유

- 모두 동일한 쿠버네티스 버전/설정이 적용
- 모든 Pod가 하나의 cluster DNS 를 사용
- 모든 User가 하나의 Control plane 을 공유해서 사용

2) Worker Node 를 공유

- k8s 의 NameSpaces 로 논리적으로 분리해서 사용
- container(cgroups) 로 성능 할당, 격리

Q) kubernetes의 NameSpaces 가 충분한 Isolation 을 제공 하는가?

1) Tenant1 이 실서비스를 하고 있는 클러스터에

Tenant2 가 신규 서비스 오픈 전 부하/성능 테스트를 한다면 정말 영향이 없는가?
-> DISK, NETWORK I/O 영향 있음, 격리를 위한 추가적인 blkio 관리 노력 필요

2) Tenant1의 Pod와 Tenant2의 Pod 간 네트워크 통신이 격리되어 있는가?

-> Pod 별 견고한 Network Policy 적용이 필요함, 보안 ACL 을 유지하기 위한 운영 노력이 필요

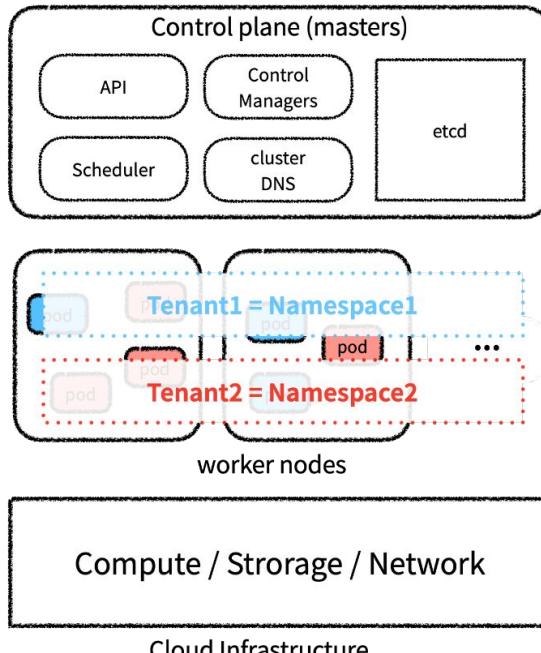


One big cluster, or many small ones?

Ensure Hard Multi-Tenancy : Multi Cluster Strategy

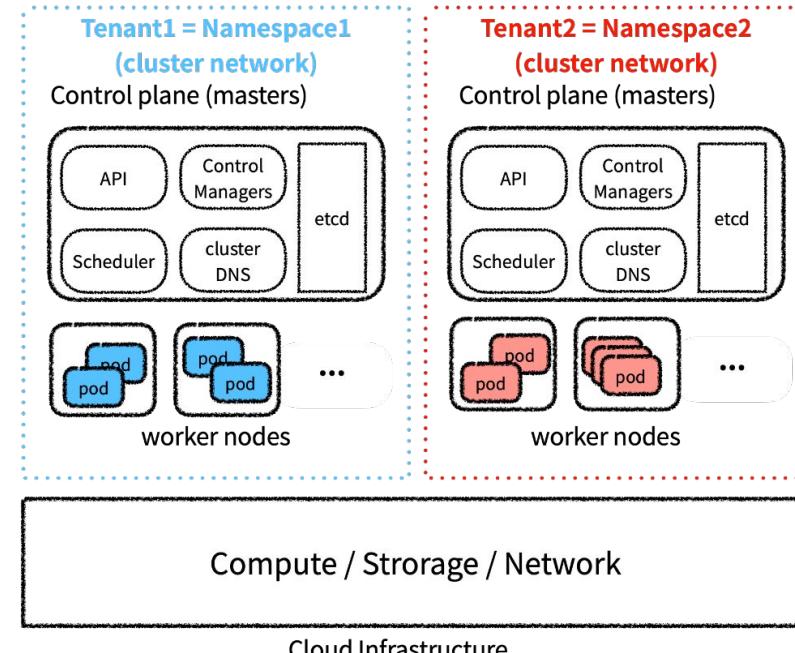
One Big Cluster

Soft Multi-Tenancy : ops affects all tenant



Many Small Ones

Hard Multi-Tenancy : blast radius limit





One big cluster, or many small ones?

Isolation 과 Security 보장은 필수 조건이기 때문에 Multi Cluster Strategy 선택

	One big cluster	Multi Cluster
Cluster Management	Easy ↑ manage single cluster	Hard ↓ manage multi cluster
Admission Control	Hard ↓ control the indiscriminate use	Easy ↑ give the Dev freedom, the way to 'DevOps'
Resource Efficiency	Good ↑ 5 masters, 5 etcd server / shared nodes	Bad ↓ per cluster 1~5 server
Cluster Stability	Bad ↓ single control plane, shared	Good ↑ many control plane, isolation
Isolation	Hard ↓	Easy ↑
Security	Hard ↓	Easy ↑



One big cluster, or many small ones?

멀티 클러스터 전략의 단점을 보완하기 위해 Kubernetes as a Service 개발

	One big cluster	Multi Cluster
Cluster Management	Easy ↑	Hard ↓
Admission Control	Hard ↓	Easy ↑
Resource Efficiency	Good ↑	Bad ↓
Cluster Stability	Bad ↓	Good ↑
Isolation	Hard ↓	Easy ↑
Security	Hard ↓	Easy ↑

멀티 클러스터 전략의 단점을 보완하자!

1) KaaS 개발로 클러스터 라이프 사이클 자동화

- 클러스터 생성/삭제, 노드 추가/삭제, 복구, 업데이트 자동화

2) 멀티 클러스터 인증 통합 및 관리 도구 개발

- kubectl login / cluster 선택 CLI 제공을 통해 멀티 클러스터 관리
- D2hub (사내 이미지 저장소)를 통해 자동 배포 대상 클러스터 선택

3) master node를 VM으로 구성하여 자원 효율 ↑

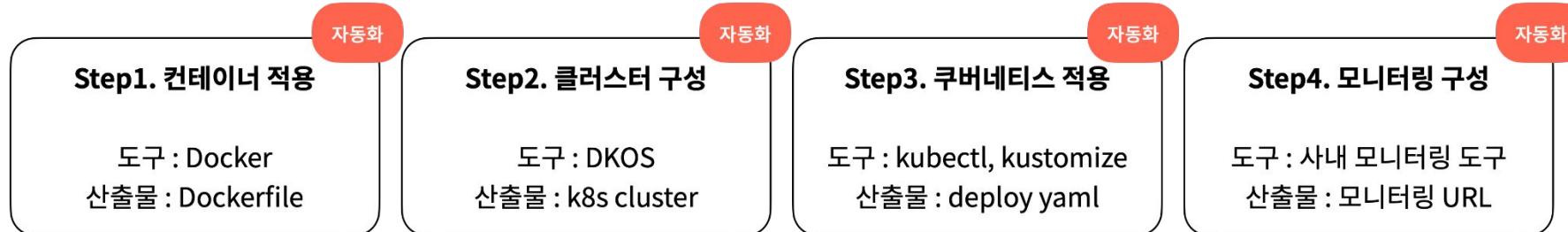
- 클러스터 규모별 권장 마스터 스펙
 - 1-50 nodes: 4 core / 4GB (VM)
 - 51-100 nodes: 8 core / 8GB (VM)
 - 101~501 nodes: 8 core / 16 GB (PM)
- 용도별 마스터 권장 갯수
 - individual / test : 1대
 - dev : 3대
 - prod : 5대

4) master node scale-up/out 기능 제공

- 작은 마스터로 시작해서 클러스터가 커짐에 따라 마스터 노드도 확장



쿠버네티스 적용 절차

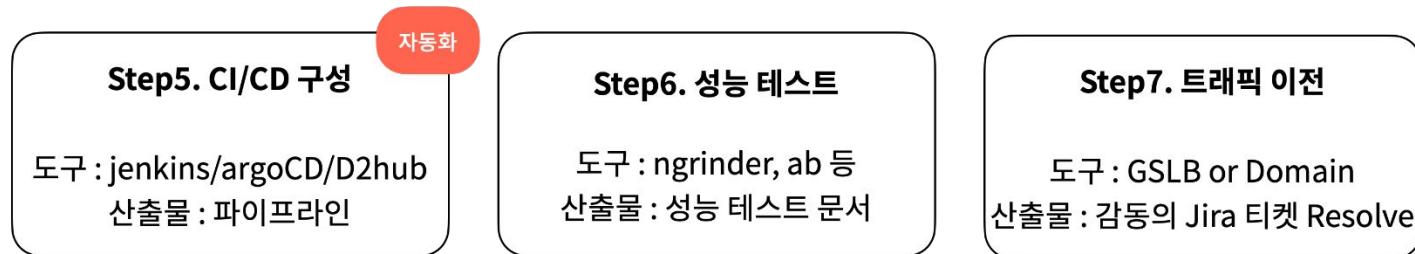


기존 jar App -> 컨테이너화

파트 공용 클러스터 구축
판교/안양 클러스터 2개로 이종화
Master(VM)/Ingress(VM) / Worker(PM)

LBaaS VIP -> Ingress -> Service -> Pods
kustomize 사용

서버 성능 : 모피어스
컨테이너 성능 : KOCOON-Prometheus
앱 로그 : 자체 logstash+logmon
ingress-nginx 로그 : KOCOON-Hermes



CI : 젠킨스 파이프라인 활용

CD : argoCD, argo-rollout 사용

GSLB 활용 점진적 트래픽 증가



ITSM 을 고려한 Kubernetes 도입

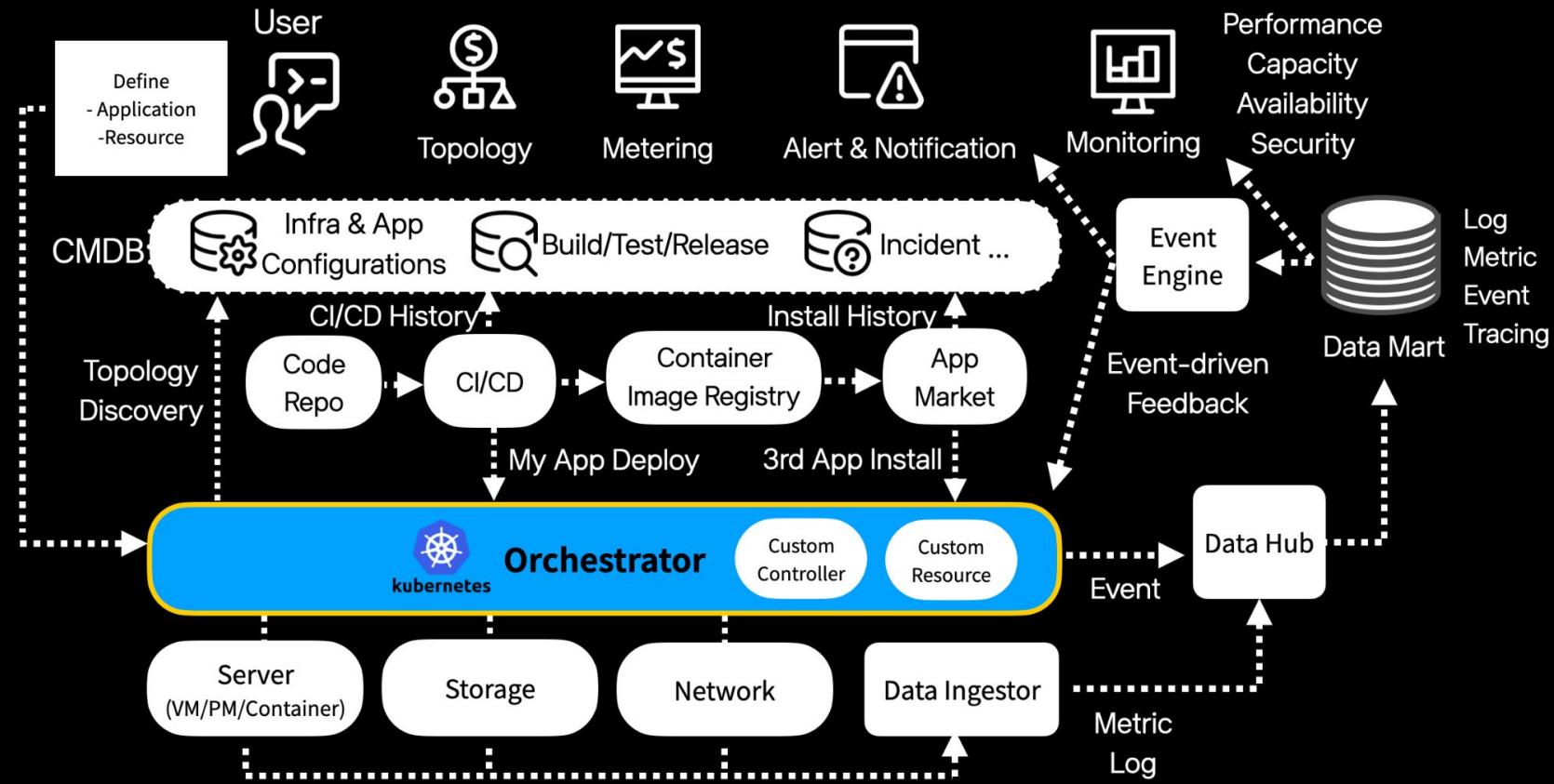
전사적 관점에서 kubernetes 의 장점을 더 잘 활용하려면?

ITSM

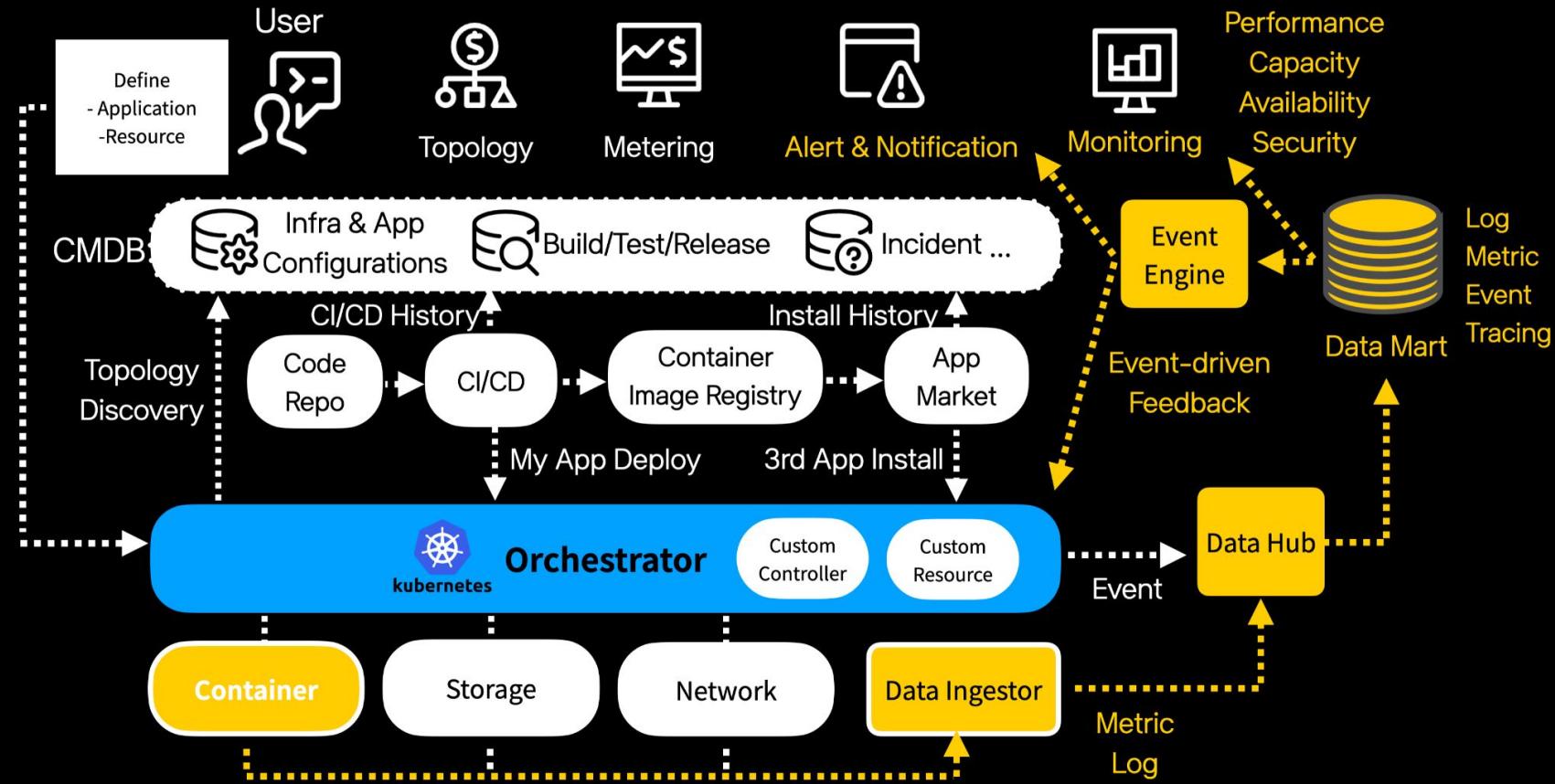
IT Service Management

IT 서비스를
계획/제공/운영/제어하는
조직에서 수행하는 활동
전체

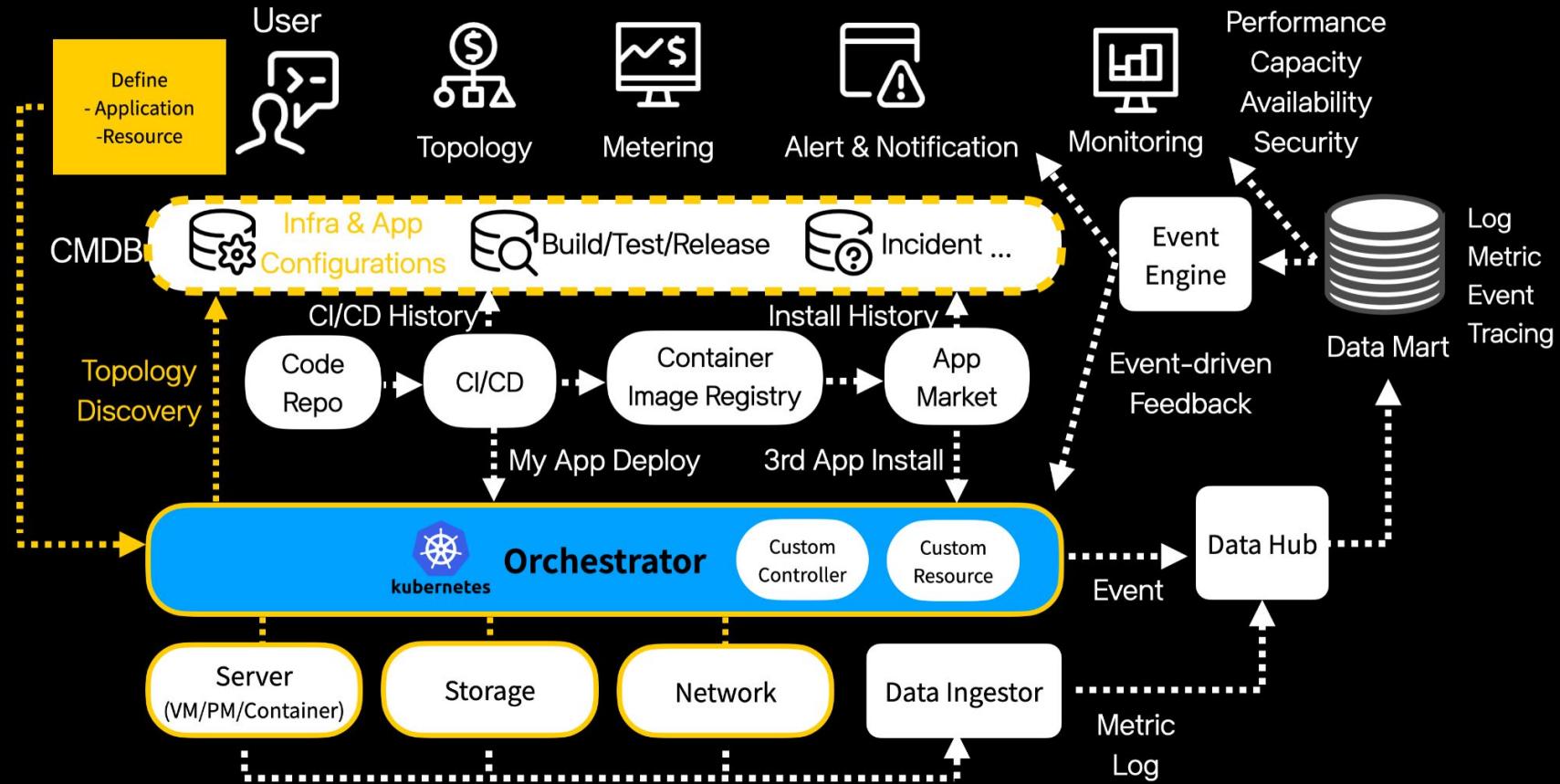
ITSM 을 고려한 Kubernetes 도입 전략



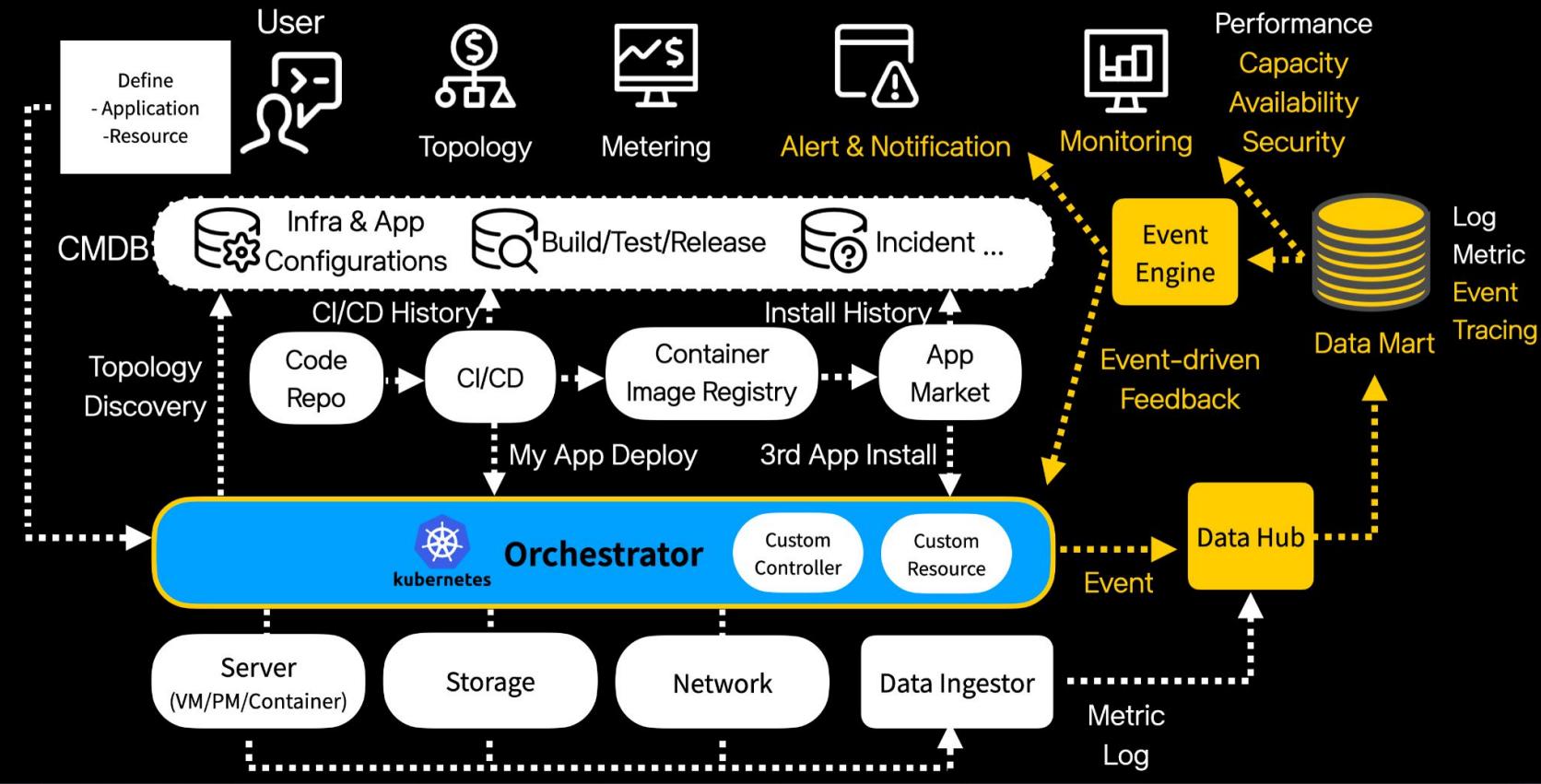
Container 의 자원에 대한 관리 필요



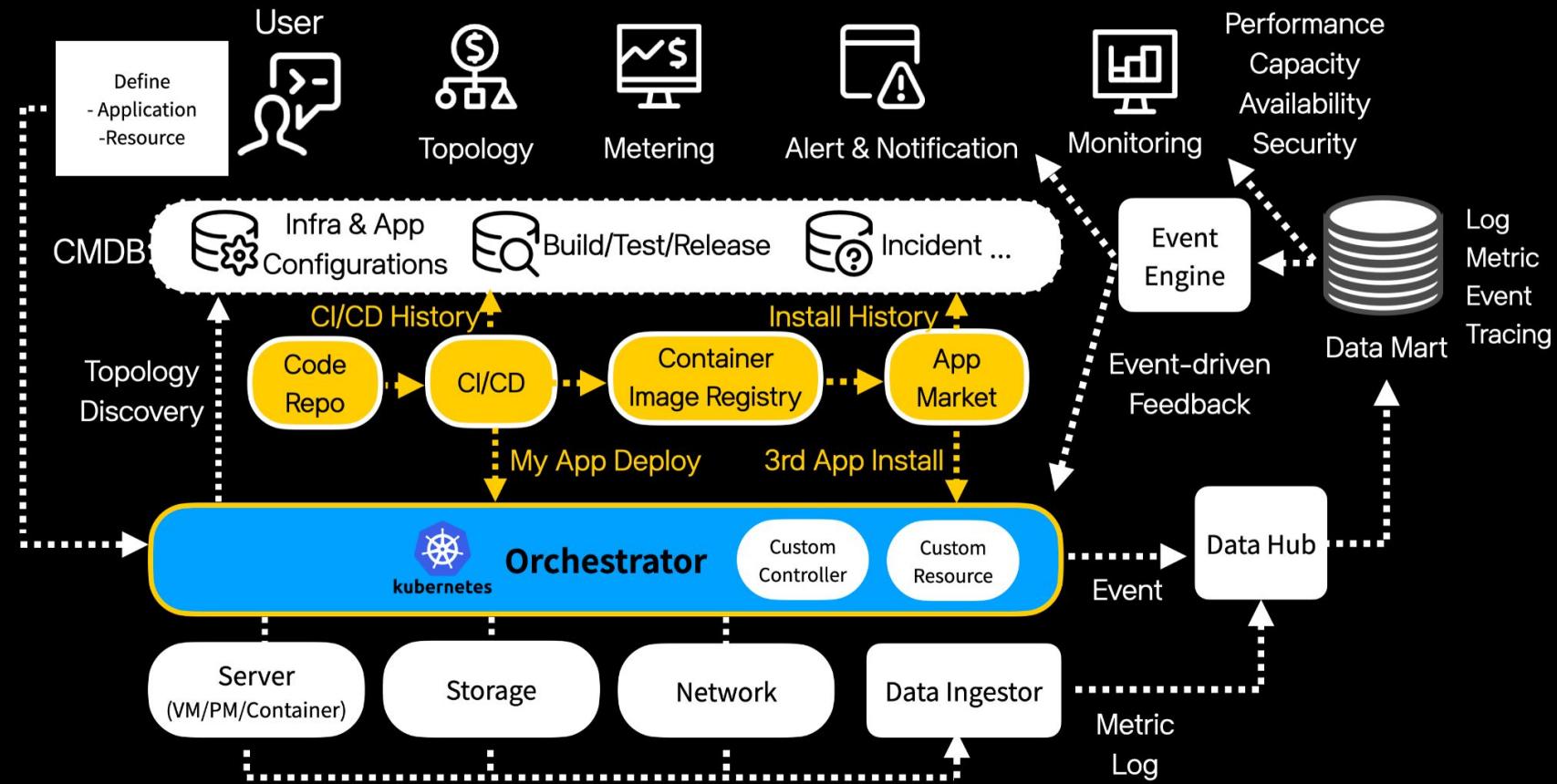
Application & Resource Topology 공유



Orchestrator Event 공유



CI/CD Pipeline 및 App install 정보 공유



ITIL & ITSM 이란?

ITSM

IT Service Management

IT 서비스를 계획/제공/운영/제어하는
조직에서 수행하는 활동 전체

ITIL

IT Infrastructure Library

ITSM 실천모델(Best Practice)
체계적 관리를 위한 지침 제공

ITSM 을 효과적으로 달성하기 위한
실천 모델을 담은 프레임워크



ITIL Service Management Lifecycle

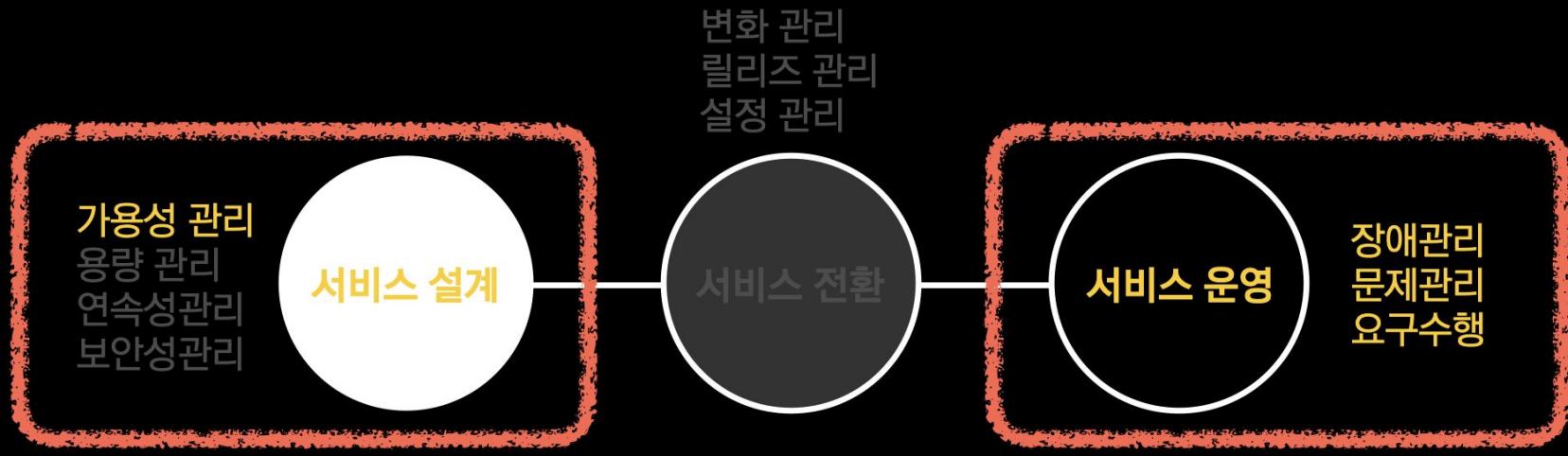


지속적 서비스 개선

서비스 리포팅
서비스 측정
서비스 레벨 관리

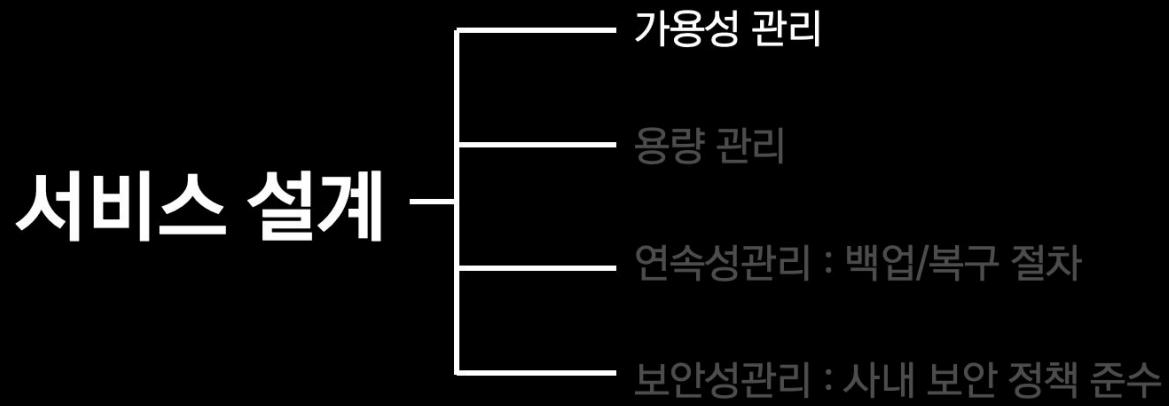


ITIL Service Management Lifecycle



지속적 서비스 개선

서비스 리포팅
서비스 측정
서비스 레벨 관리



What?

“서비스 설계에 앞서 가장 먼저해야할 일은 무엇 일까요?”

SLA

(Service Level Agreement)

서비스 수준 정하기

SLA (Service Level Agreement)의 예

가용성

99.99% 의 가용성

=1년중 53분 이하의 서비스 중단

성능

평균 API 응답시간 30ms

용량

상시 피크시의 4배 보유

Master Node 의 가용성 관리

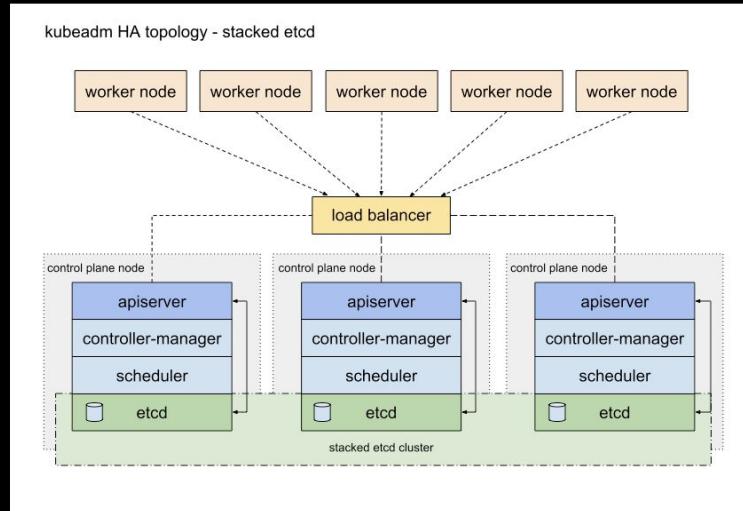
Master Node 개수	장애 감내 정도	권장 목적
1대	Master Node 1개 장애시 쿠버네티스 컨트롤러 장애	개인 테스트/학습용
3대	Master Node 1개 장애를 허용	개발 환경용
5대	Master Node 2개 장애를 허용	실 서비스용

More

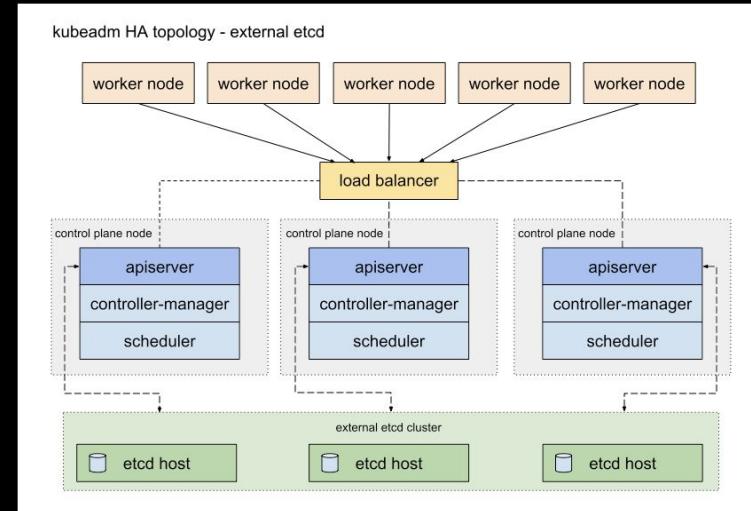


Master Node 의 가용성 관리 (More)

k8s 저장소인 etcd 를 Master Node 에 같이 설치하지 않고, 외부 분리하여 추가적인 가용성을 확보 가능



비용 효율성



고가용성

More

Node 의 가용성 관리

kubelet 이 기본적으로 Node 의 MemoryPressure, DiskPressure, PIDPressure, Ready 상태¹⁾를 모니터링

컨트롤러에 node-monitor-grace-period (기본 40 초) 기간 안에 상태를 보고하지 못하면 Unknown 상태

컨트롤러는 Node Condition이 Ready 값이 True 인 Node 에게만 Pod 를 할당해 줌

```
$ kubectl get nodes
NAME           STATUS    ROLES      AGE   VERSION
dkosv3-ingress-1  Ready    ingress   274d  v1.18.8
dkosv3-ingress-2  Ready    ingress   274d  v1.18.8
dkosv3-master-1   Ready    master    274d  v1.18.8
dkosv3-master-2   Ready    master    274d  v1.18.8
dkosv3-master-3   Ready    master    274d  v1.18.8
dkosv3-worker-1   Ready    worker   274d  v1.18.8
dkosv3-worker-2   Ready    worker   274d  v1.18.8
dkosv3-worker-3   Ready    worker   274d  v1.18.8
dkosv3-worker-4   Ready    worker   274d  v1.18.8
```

1) <https://kubernetes.io/docs/concepts/architecture/nodes/#condition>

Worker Node 의 가용성 관리 (More)

node-problem-detector¹⁾ : 더 상세한 Node Condition, Event 정보 제공

The screenshot shows the GitHub repository page for `kubernetes/node-problem-detector`. The repository has 52 stars, 1,133 forks, and 305 issues. The master branch has 498 commits. The commit history includes several pull requests and various commits from maintainers like `k8s-ci-robot` and `abansal4032`. The commits are dated from 6 days ago to 4 years ago and involve updates to builder, cmd, config, deployment, docs, pkg, test, vendor, .gitignore, .travis.yml, CHANGELOG.md, CONTRIBUTING.md, Dockerfile.in, and LICENSE files.

Commit	Message	Date
builder	Updated maintainer from copy-pasted Dockerfile to this pr...	2 years ago
cmd	Add health-check-monitor	3 months ago
config	Update docker-monitor.json	23 days ago
deployment	update system-log-monitor and image version	5 months ago
docs	enable condition update when message change for custo...	2 years ago
pkg	Add logging levels to custom plugin logs.	6 days ago
test	Check metric sanity in e2e tests	7 months ago
vendor	Add github.com/prometheus/procfs library	7 months ago
.gitignore	Add problem maker to simulate problems for e2e test	9 months ago
.travis.yml	Fix build tags manipulation in Makefile	9 months ago
CHANGELOG.md	Add CHANGELOG.md for NPD repo.	4 years ago
CONTRIBUTING.md	add CONTRIBUTING.md	2 years ago
Dockerfile.in	docker image: add health-checker binary	3 months ago
LICENSE	Initial commit	4 years ago

1) <https://github.com/kubernetes/node-problem-detector>

Worker Node 의 가용성 관리 (More)

Node 의 상태에 대한 간단한 설정만으로 모니터링 항목 추가 가능

/dev/kmsg 로그에서

```
{  
    "plugin": "kmsg",  
    "logPath": "/dev/kmsg",  
    "lookback": "5m",  
    "bufferSize": 10,  
    "source": "kernel-monitor",  
    "metricsReporting": true,  
    "conditions": [  
        {  
            "type": "KernelDeadlock",  
            "reason": "KernelHasNoDeadlock",  
            "message": "kernel has no deadlock"  
        },  
        {  
            "type": " ReadonlyFilesystem",  
            "reason": "FilesystemIsNotReadOnly",  
            "message": "Filesystem is not read-only"  
        }  
    ],  
    "rules": [  
        {  
            "type": "temporary",  
            "reason": "OOMKilling",  
            "pattern": "Kill process \\\d+ (.+) score \\\d+ or sacrifice child\\  
        },  
        {  
            "type": "permanent",  
            "reason": "FilesystemIsReadonly",  
            "pattern": "Filesystem is not read-only"  
        }  
    ]  
}
```

OOM 로그 패턴 감지

Worker Node 의 가용성 관리 (More)

Custom Node Condition 추가 : Pod 를 동작시킬 수 없는 노드의 영구적 문제 보고

Default NodeCondition

```
conditions:
- lastHeartbeatTime: "2020-09-07T06:11:00Z"
  lastTransitionTime: "2019-12-04T03:01:24Z"
  message: kubelet has sufficient memory available
  reason: KubeletHasSufficientMemory
  status: "False"
  type: MemoryPressure
- lastHeartbeatTime: "2020-09-07T06:11:00Z"
  lastTransitionTime: "2020-06-30T12:23:10Z"
  message: kubelet has no disk pressure
  reason: KubeletHasNoDiskPressure
  status: "False"
  type: DiskPressure
- lastHeartbeatTime: "2020-09-07T06:11:00Z"
  lastTransitionTime: "2019-12-04T03:01:24Z"
  message: kubelet has sufficient PID available
  reason: KubeletHasSufficientPID
  status: "False"
  type: PIDPressure
```

+ Custom NodeCondition

```
conditions:
- lastHeartbeatTime: "2019-12-04T03:02:05Z"
  lastTransitionTime: "2019-12-04T03:02:05Z"
  message: KakaoNodeManager is running on this node
  reason: KakaoNodeManagerIsUP
  status: "True"
  type: KakaoManagerUnavailable
```

Worker Node 의 가용성 관리 (More)

Custom Event 추가 : Pod 에 미치는 영향은 제한적이지만 일시적 문제 정보를 제공

+ Custom Event

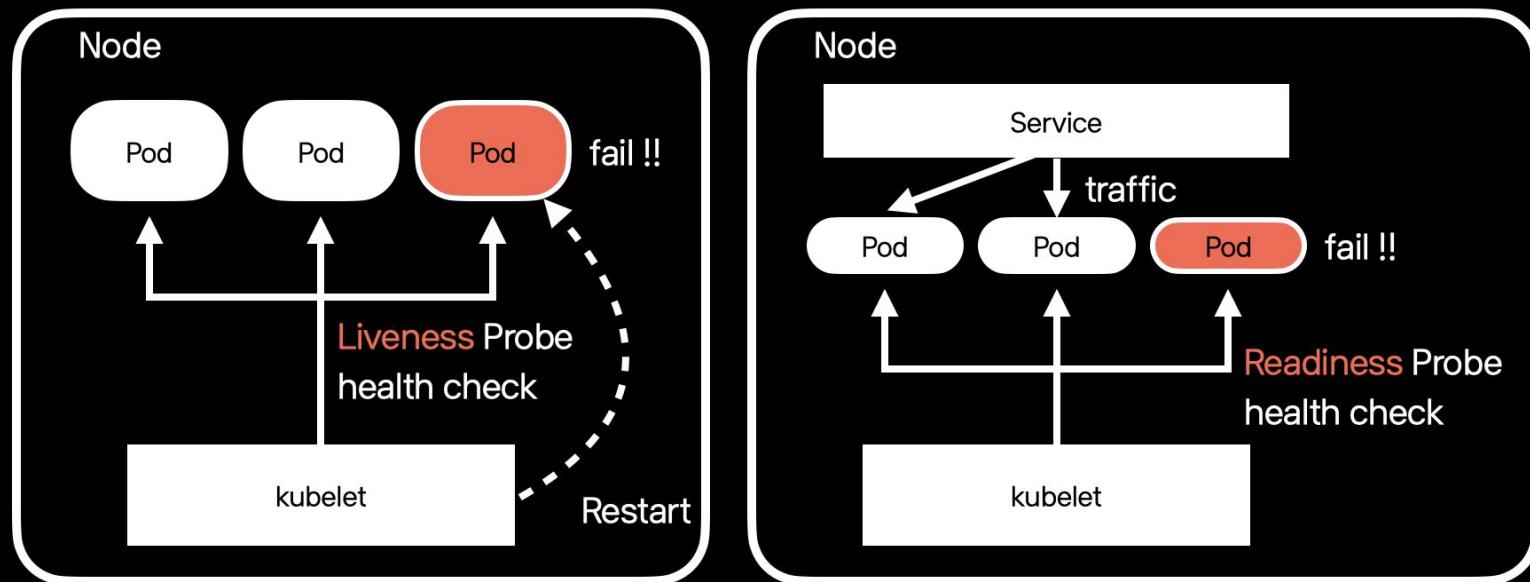
```
$ kubectl get events --field-selector type=Warning
```

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
48m	Warning	Unhealthy	node/dkosv3-worker-1	Kakao-manager failed
48m	Warning	Unhealthy	pod/kakao-controller-manager-5466db69db-j56wq	Liveness probe failed
49m	Warning	Unhealthy	pod/kakao-controller-manager-5466db69db-lp2bt	Liveness probe failed
49m	Warning	Unhealthy	pod/kakao-controller-manager-5466db69db-bwgc8	Liveness probe failed
49m	Warning	Unhealthy	pod/kakao-controller-manager-5466db69db-4xtwj	Liveness probe failed



Pod 의 가용성 관리

Pod 는 Deployment, DaemonSet, StatefulSet 을 통해 생성하여 k8s 에 의해 가용성 자동 관리

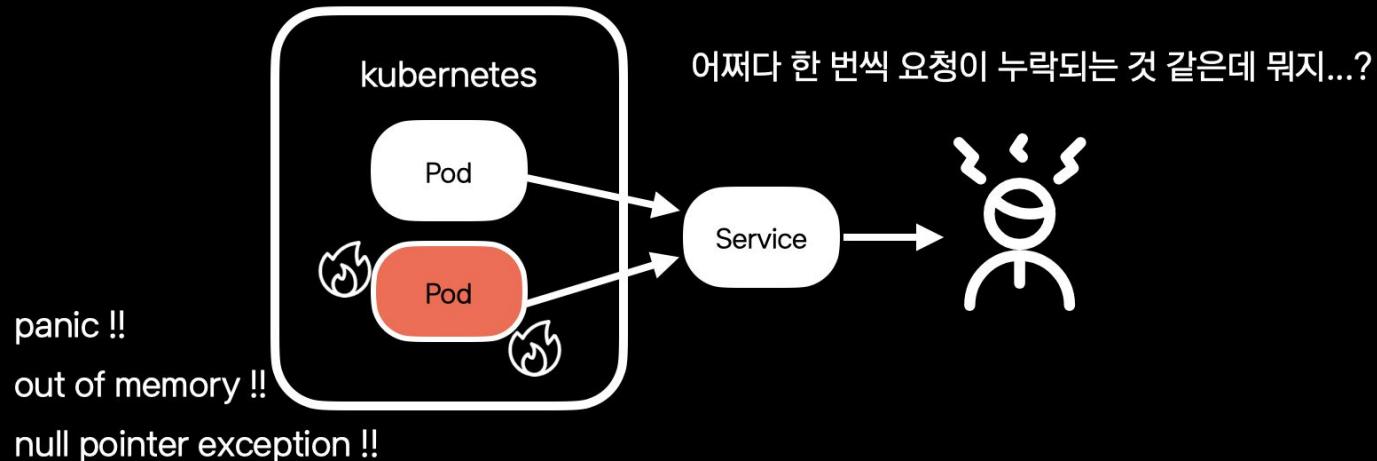




Pod 의 가용성 관리

Pod 는 Deployment, DaemonSet, StatefulSet 을 통해 생성하여 k8s 에 의해 가용성 자동 관리

문제는 사용자도 모르게 Pod 를 너무 잘 살려준다는 것!!





Pod 의 가용성 관리 (More)

kube-event-watcher 개발 : Pod 가용성에 대한 k8s 이벤트 감지하여 카카오톡이나 Slack 으로 알림



DKOSv3 - 😊 새로운 App 실행
=====
Cluster: pnu-cluster
Namespace: default
Deployment: blue
Replicas status/spec: 0 / 3
=====

DKOSv3 - 😊 배포 완료
=====
Cluster: pnu-cluster
Namespace: default
Deployment: blue
Replicas status/spec: 3 / 3
=====

DKOSv3 - 😊 컨테이너 내용 변경
=====
Cluster: pnu-cluster
Namespace: default
Deployment: green
Replicas status/spec: 3 / 3
=====

상세내용
=====
object.Image:
Before: "koogk7/product-server:1.212"
After : "koogk7/product-server:1.122"
=====

DKOSv3 - 😕 Warning0! 발생했습니다!
=====
Cluster: test-cluster
Pod: blue-8476f65f77-sgqbh
Namespace: default
Reason: Unhealthy
Msg: Liveness probe errored: rpc
error: code = Unknown desc =
container not running
(f17200e528ba5c7892d74557a5e4f0
abf2ca15895fbfc08562449129c98b1
d8d)
=====

DKOSv3 - 😊 Job 정상 성공
=====
Cluster: test-cluster
Namespace: default
JobName: cron-node-1581913680
Succeed/Completions: 6 / 6
=====

시작시간 : 2020-02-17, 13:28:08
완료시간 : 2020-02-17, 13:28:13
=====

DKOSv3 - 😕 Job 실패
=====
Cluster: test-cluster
Namespace: default
JobName: cron-node-1581909660
Succeed/Completions: 0 / 6
=====

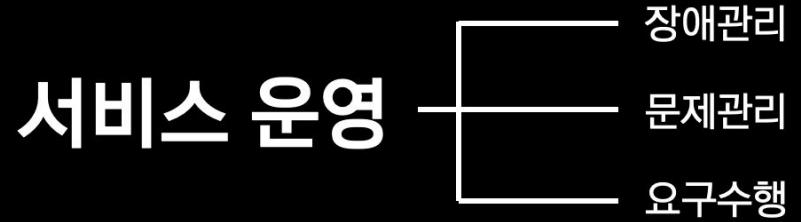
Reason: BackoffLimitExceeded
=====

신규 배포 시작/진행/완료

컨테이너 이미지 변경

헬스체크 실패, fail-over

CronJob 실행 이벤트





서비스 운영

조직 구성 / 문화 / 프로세스

DevOps / Agile / Automate First



DevOps 조직 구성

개발팀 = 운영팀

'우리가 짠 코드는 우리가 운영한다'

모든 개발자가 Weekly On-call 참여



DevOps 조직 리얼 장점

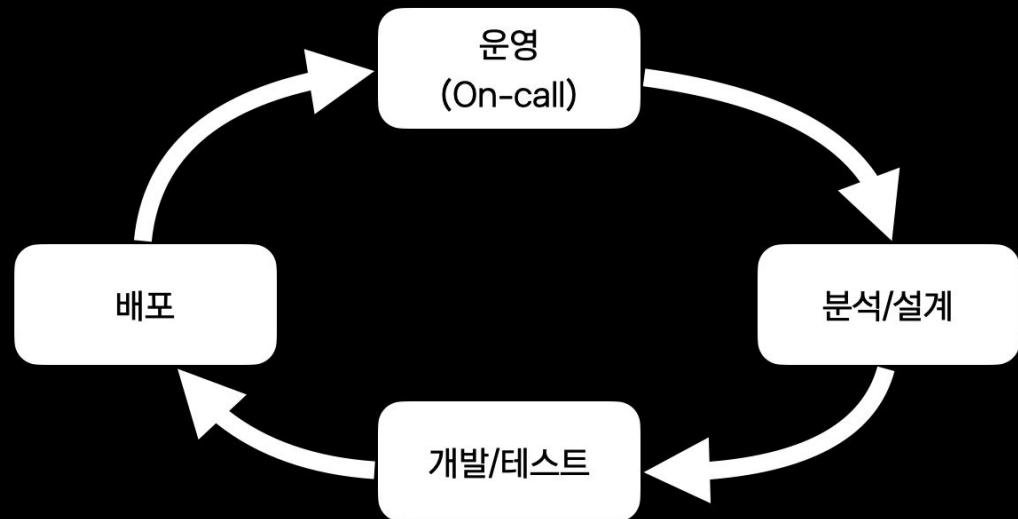
서비스 품질 UP! : 내돈내산! (내가 개발, 내가 서비스)

코드 품질 UP! : 온콜 때 살아남기 위해 운영을 고려한 코드 작성

빠른 대응 : 온콜 담당자가 코드 레벨까지 이슈 추적, 빠른 패치

Agile

On-call에서 발견된 이슈들을 다시 개발 요구사항에 반영



Automation First

운영 업무의 발견(인지)에서 해결까지 이벤트 기반 자동화

On-call 업무 분류

Proposal

신규 기능 및 기능 개선 제안
ex) 여러 클러스터를 한번에 삭제 할 수 있게 해주세요

Q&A

단순 기능 및 정책 문의
ex) 로드밸런서 상세 설정, SSL 인증서 세팅

Service Request

관리자가 해줘야할 일반적인 운영 요청
ex) 쿼터 추가, 공인 IP 부여, 클러스터 업그레이드 등

Incident

원인과 처리 절차가 알려진 기능/품질 저하 이벤트
ex) Node 생성 실패, k8s 설치 실패 등

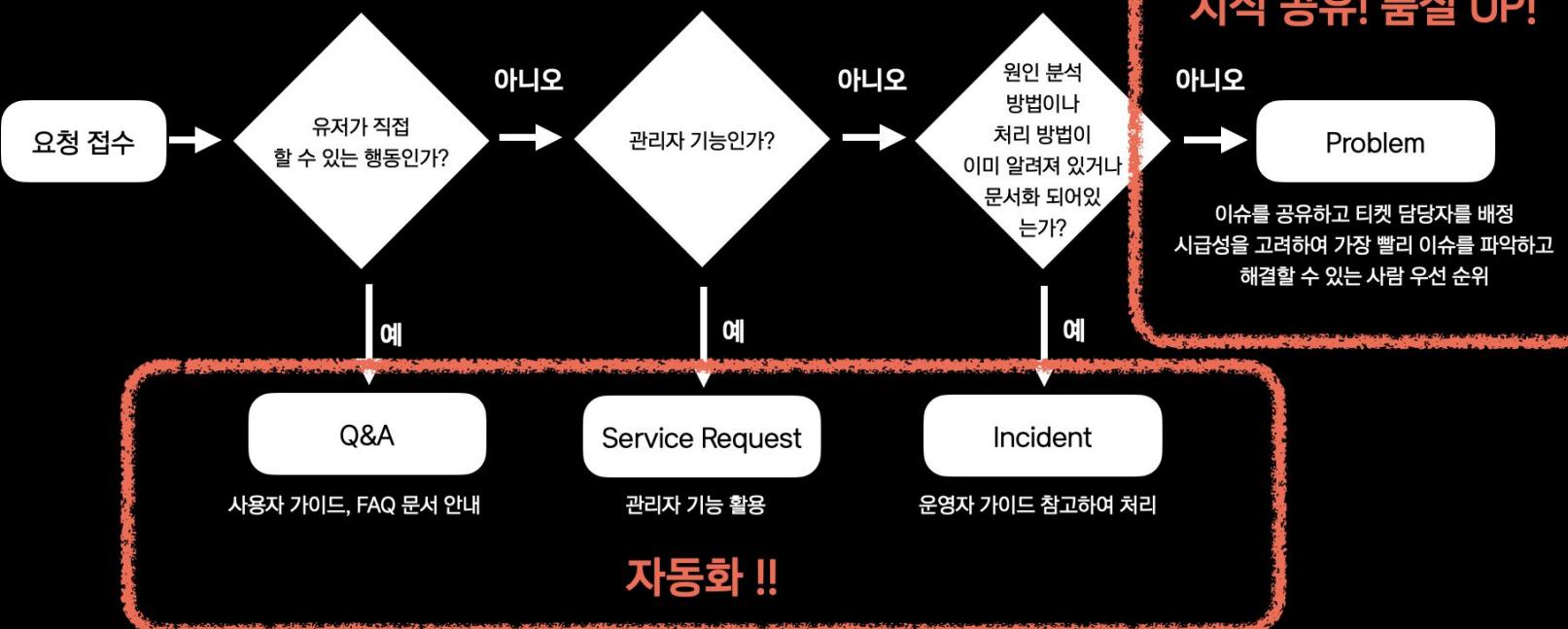
Problem

원인 분석이 필요한 플랫폼 또는 사용자 서비스 장애
ex) 원인 모를 서비스 접속 지연 또는 장애, 애러

Solution Architecture

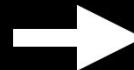
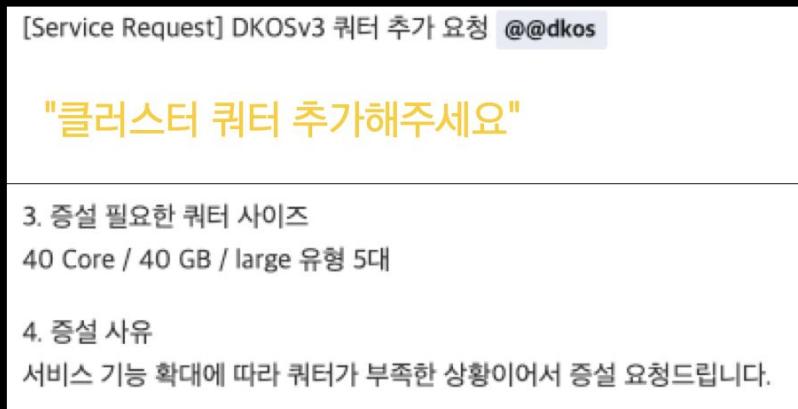
아키텍처, 가용성, 용량, 성능, 보안, 최적화 컨설팅 지원
ex) 대규모 신규 서비스 아키텍처링,
특수 요구사항 구성 등

Automation First



Automation First

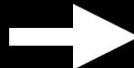
요청 자동 접수/분류 → Jira 티켓 생성 → 담당자 배정



접수 / 진행 / 완료된 요청
실시간 관리

Automation First

승인 댓글달면 자동처리



처리 결과 자동 입력

The screenshot shows a Jira board with three columns: '할 일', '진행 중', and '완료'. There are several issues listed, each with its status changed to 'Resolved'. The issues include:
1. DKOSOC-155: D2hub v2 기능 에러 및 장애 문의 (Incident)
2. DKOSOC-162: DKOSv3 기능 및 정책 문의 (Q&A)
3. DKOSOC-174: D2hub v2 기능 에러 및 장애 문의 (Incident)
4. DKOSOC-94: D2hub v2 기능 에러 및 장애 문의 (Incident)
5. DKOSOC-113: DKOSv3 기능 에러 및 장애 문의 (Incident)
6. DKOSOC-116: DKOSv3 기능 및 정책 문의 (Q&A)
7. DKOSOC-150: D2hub v2 신규 기능 및 개선 제안 (Proposal)
8. DKOSOC-160: D2hub v2 기능 및 정책 문의 (Q&A)
9. DKOSOC-145: DKOSv3 기반 서비스 설계 문의 (Solution Architecture)
10. DKOSOC-198: D2hub v2 기능 및 정책 문의 (Q&A)
11. DKOSOC-142: DKOSv3 Ingress w/ KakaoAuth 신청 (Service Request)
12. DKOSOC-143: DKOSv3 기능 에러 및 장애 문의 (Incident)
13. DKOSOC-144: DKOSv3 쿼터 추가 요청 (Service Request)

쿼터 추가 API 호출

+ 카카오톡 챗봇으로도 자동화

