

Container Orchestration with GCP (DevOps)

3rd Jan 2020, ver2.0



Copyright © 2020. Jinyoung Jang & uEngine-solutions All rights reserved.

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes ✓
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

“

DevOps

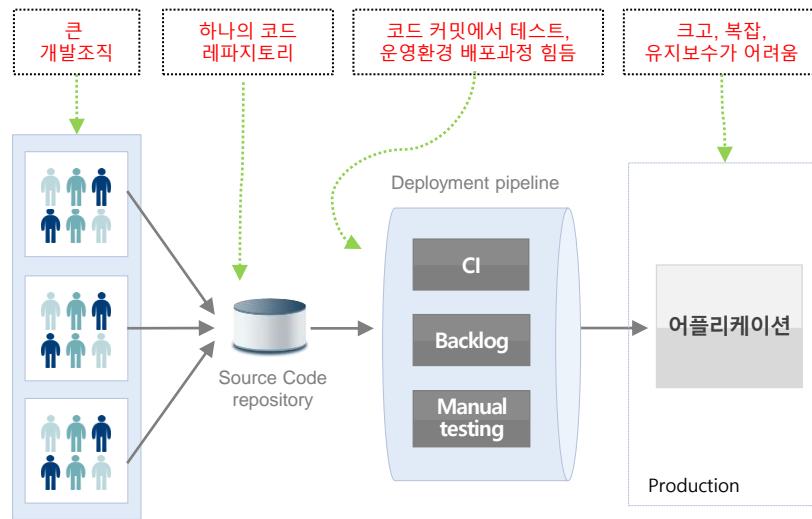
- DevOps Process and Tools
- Deploy Strategies
- Containers (Docker) and Orchestrators
- Kubernetes
- Cloud Platforms supporting Kubernetes

https://www.youtube.com/watch?v=_I94-tJlovg

Process Change : 열차말고 택시를 타라!

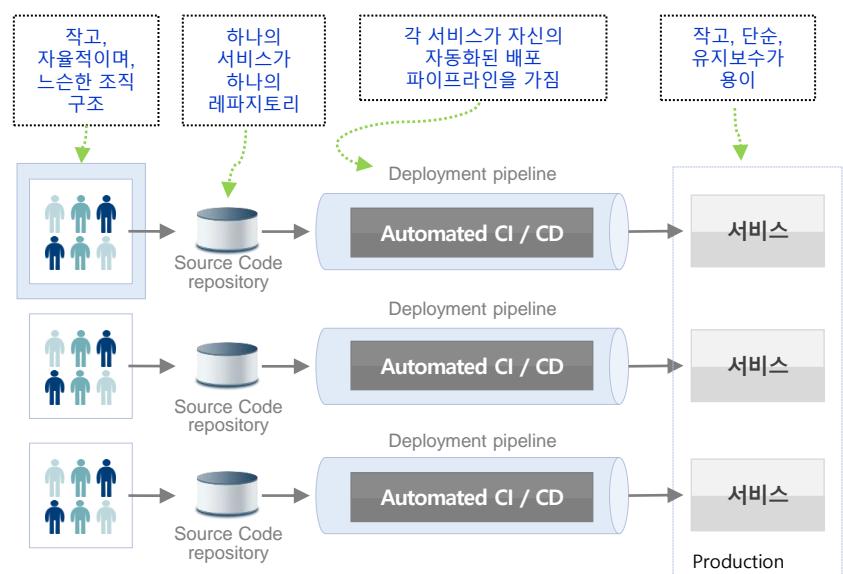
모노리식 개발 및 운영환경

- 모노리식 환경하에서는 큰 개발팀이 하나의 소스코드 레파지토리에 변경 사항을 커밋하므로 코드간 상호 의존도가 높아 신규 개발 및 변경이 어려움
- 작은 변화에도 전체를 다시 테스트/ 배포하는 구조이므로 통합 스케줄에 맞춘 파이프라인을 적용하기가 어렵고 Delivery 시간이 과다 소요



マイ크로서비스 개발 및 운영환경

- 작고 분화된 조직에서 서비스를 작은 크기로 나누어 개발하므로 해당 비즈니스 로직에만 집중하게 되어 개발 생산성 향상
- 연관된 마이크로서비스만 테스트를 수행하므로 개발/테스트/배포 일정이 대폭 축소

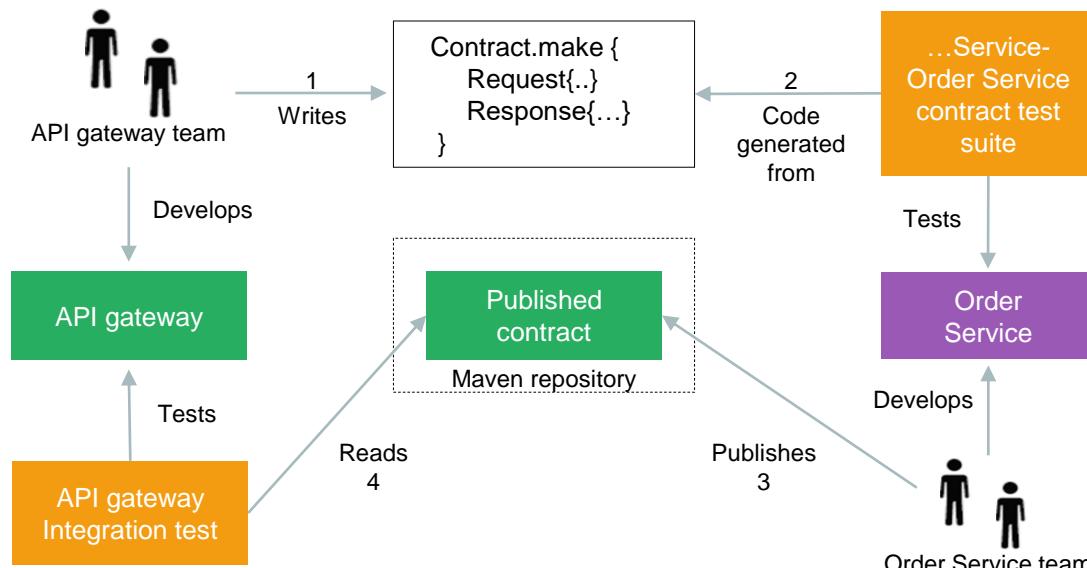


Process Change : Consumer-driven Test

MSA 서비스의 테스트

컨트랙트 테스트는 서비스 수요자가 주도하여 테스트를 작성한다.

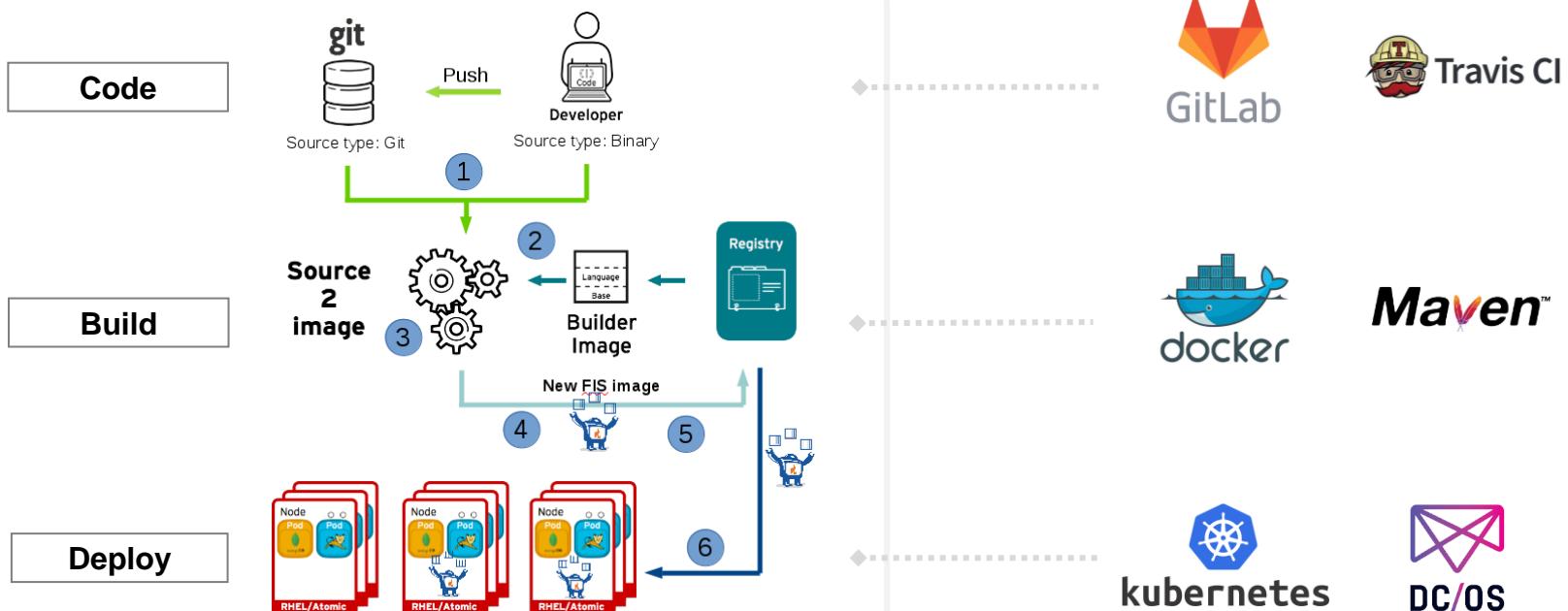
수요자가 테스트를 작성하여 제공자의 레포지토리에 Pull-Request 하면, 제공자가 이를 Accept 한후에 제공자측에서 테스트가 생성되어 테스트가 벌어진다.



특징

- MSA 테스트에는 Contract-based Test 가 특징적
- API Consumer가 먼저 테스트 작성
- API Provider가 Pull Request 수신 후 테스트는 자동으로 생성됨
- Consumer 측에 Mock 객체가 자동생성 병렬 개발이 가능해짐
- Spring Cloud Contract가 이를 제공
- Provider의 일방적인 버전업에 따른 하위 호환성 위배의 원천적 방지

DevOps toolchain



Supporting Continuous Delivery

Facebook

2. 배포 주기

- 매일 마이너 업데이트
- 메이저 업데이트 (매주 화요일 오후)



3. Deployment Pipeline

출처: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6449236>

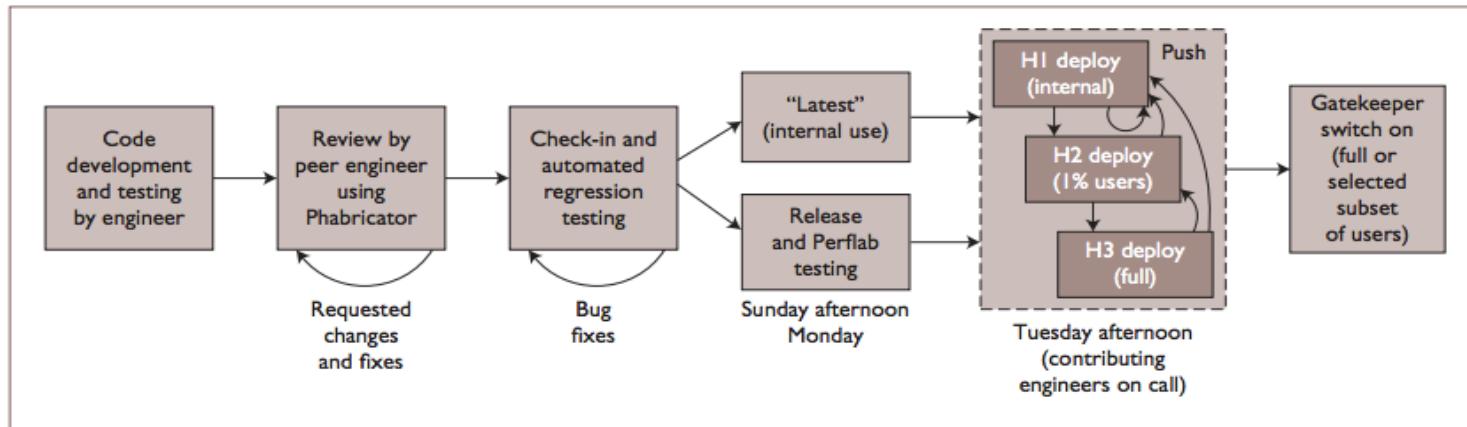


Figure 8. The Facebook deployment pipeline. Multiple controls exist over new code.

Supporting Continuous Delivery

Netflix

4. Canary를 통해서 확신 갖기

- Canary란? 실제 production 환경에서 small subset에서 새로운 코드를 돌려 보고 옛날 코드와 비교해서 새로운 코드가 어떻게 돌아가는지 보는 것
- Canary 분석 작업(HTTP status code, response time, 실행수, load avg 등이 옛날 코드랑 새로운 코드랑 비교해서 어떻게 다른지 확인하는 것)은 1000개 이상의 metric을 비교해서 100점 만점에 점수를 주고 일정 점수일 경우만 배포할 수 있음.



Canary Score

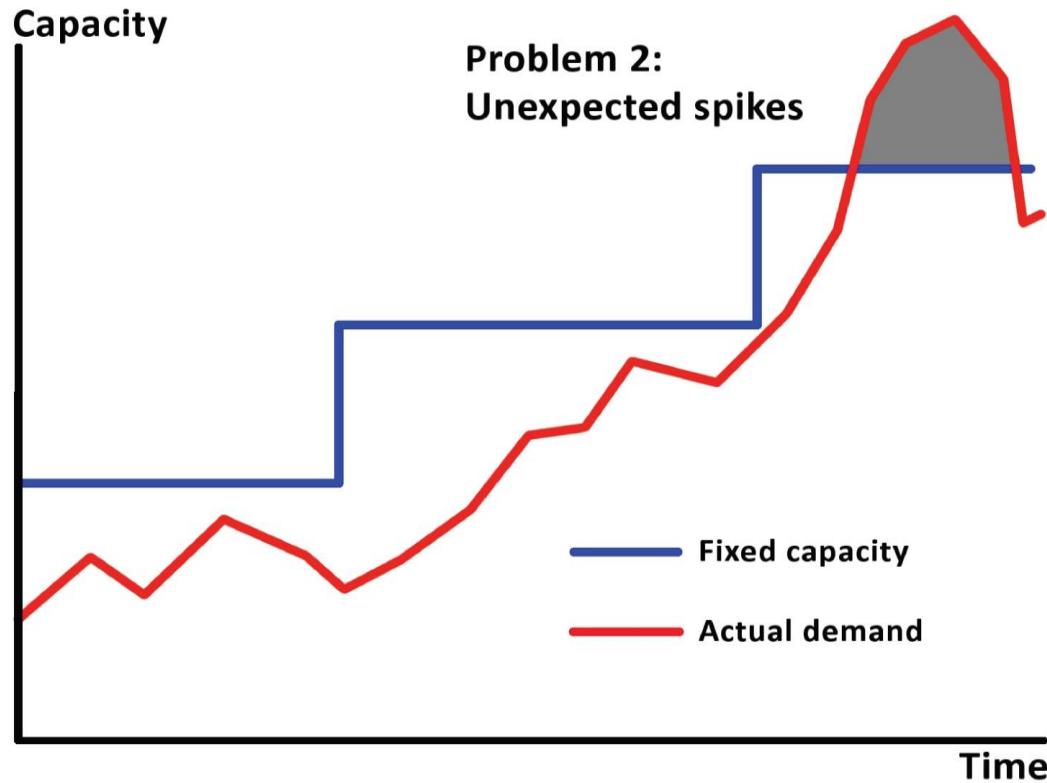
AMI Name: ami-4219582b Date: Tue Aug 13 23:31:27 UTC 2013 -1 Hours



출처: <http://techblog.netflix.com/2013/08/deploying-netflix-api.html>

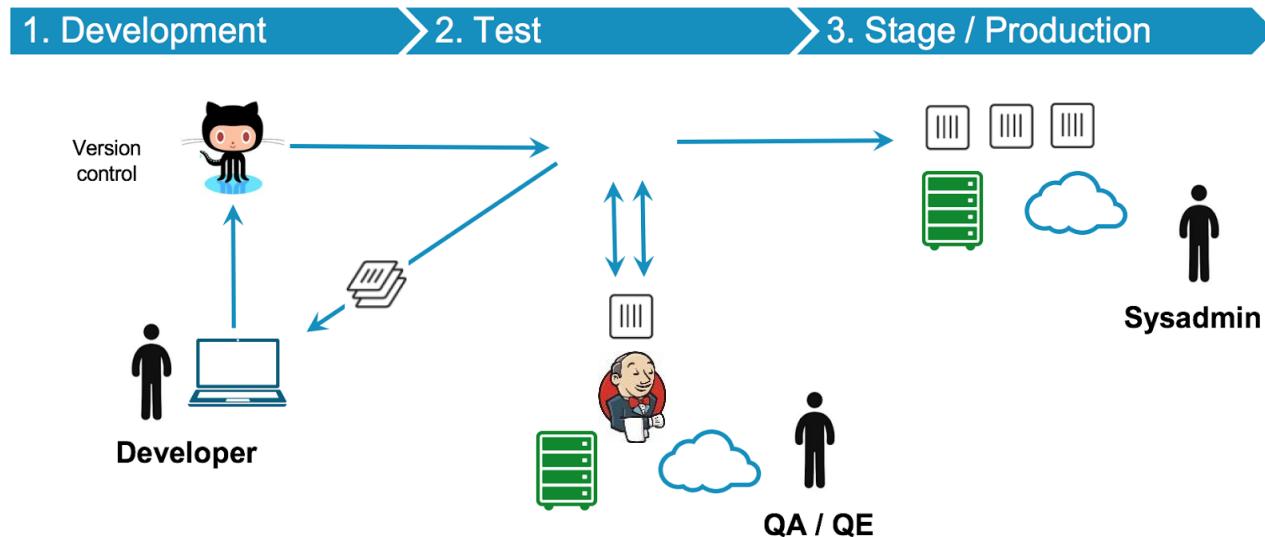
DevOps : Issues

Managing Scalability



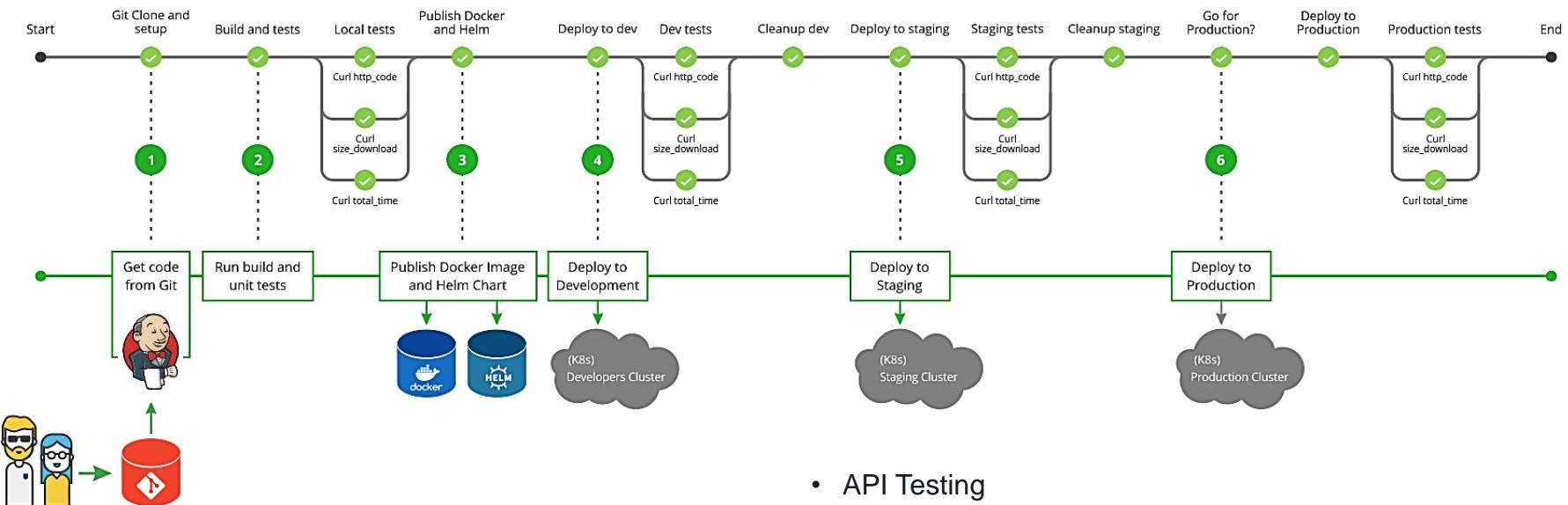
CI/CD Tools – conventional CI/CD

CI /CD Workflow



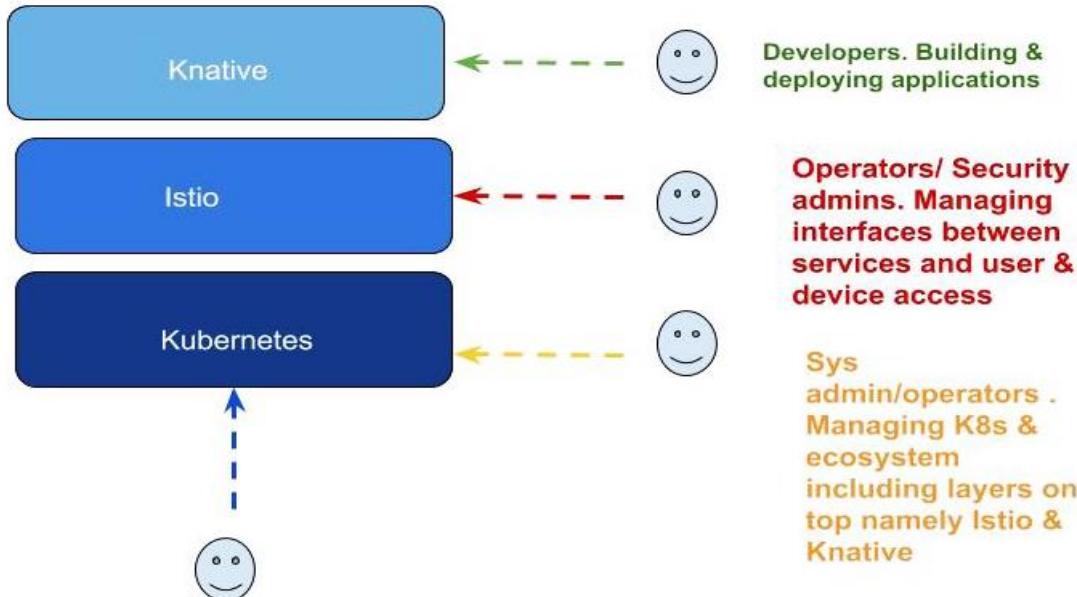
- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Tools – Container-based



- API Testing
- Blue/Green, Canary
- 도구 : Jenkins + Docker + Spinnaker + Helm + Kubernetes
→ 매우복잡

Future - Serverless



Platform providers or K8s
on-premises(sysadmin/operators)
Making K8s work and interconnect

- Infrastructure As A Code
- Application Configuration 과 Infra Configuration을 하나의 설정에 통합
- 단일 도구

배포 전략별 비교

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.



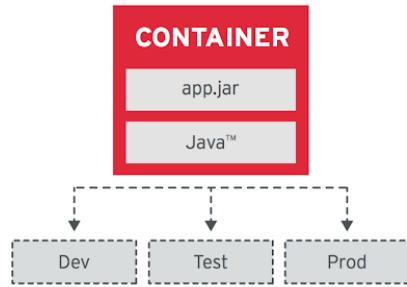
Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■ □ □	■ ■ ■	■ ■ ■	□ □ □
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■ □ □	■ ■ ■	■ □ □	■ □ □
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■ ■ ■	□ □ □	■ ■ □	■ ■ □
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■ □ □	■ □ □	■ □ □	■ ■ □
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■ □ □	■ □ □	■ □ □	■ ■ ■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

DevOps Platforms

Container	Workload Distribution Engine (Container Orchestrator)	PaaS
<ul style="list-style-type: none">Docker	<ul style="list-style-type: none">KubernetesDocker SWARM(toy)Mesos Marathon(DCOS)Cloud Foundry	<ul style="list-style-type: none">Google Cloud PlatformRedhat Open ShiftAmazon EKSIBM Bluemix
<ul style="list-style-type: none">Warden(Garden)		<ul style="list-style-type: none">HerokuGE's Predix
		<ul style="list-style-type: none">Pivotal Web Services
<ul style="list-style-type: none">Hypervisor	<ul style="list-style-type: none">CF version 1Engine yard....	<ul style="list-style-type: none">Amazon Beanstalk

Container-based Application Design

Image Immutability Principle



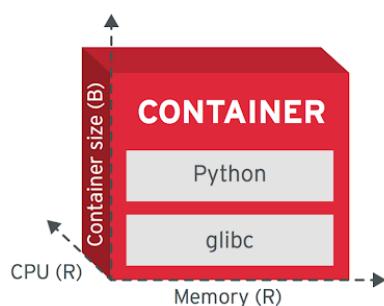
High Observability Principle



Process Disposability Principle



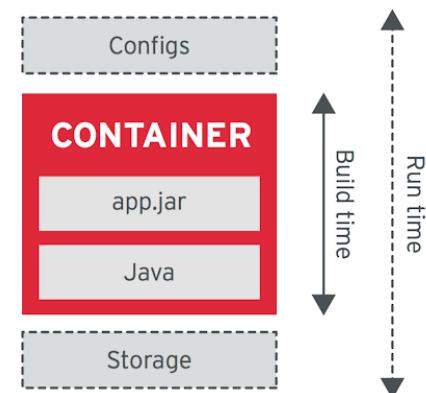
Runtime Confinement Principle



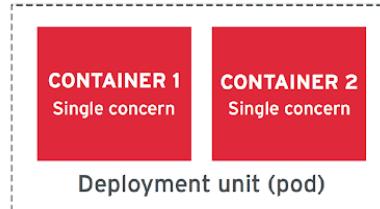
Lifecycle Conformance Principle



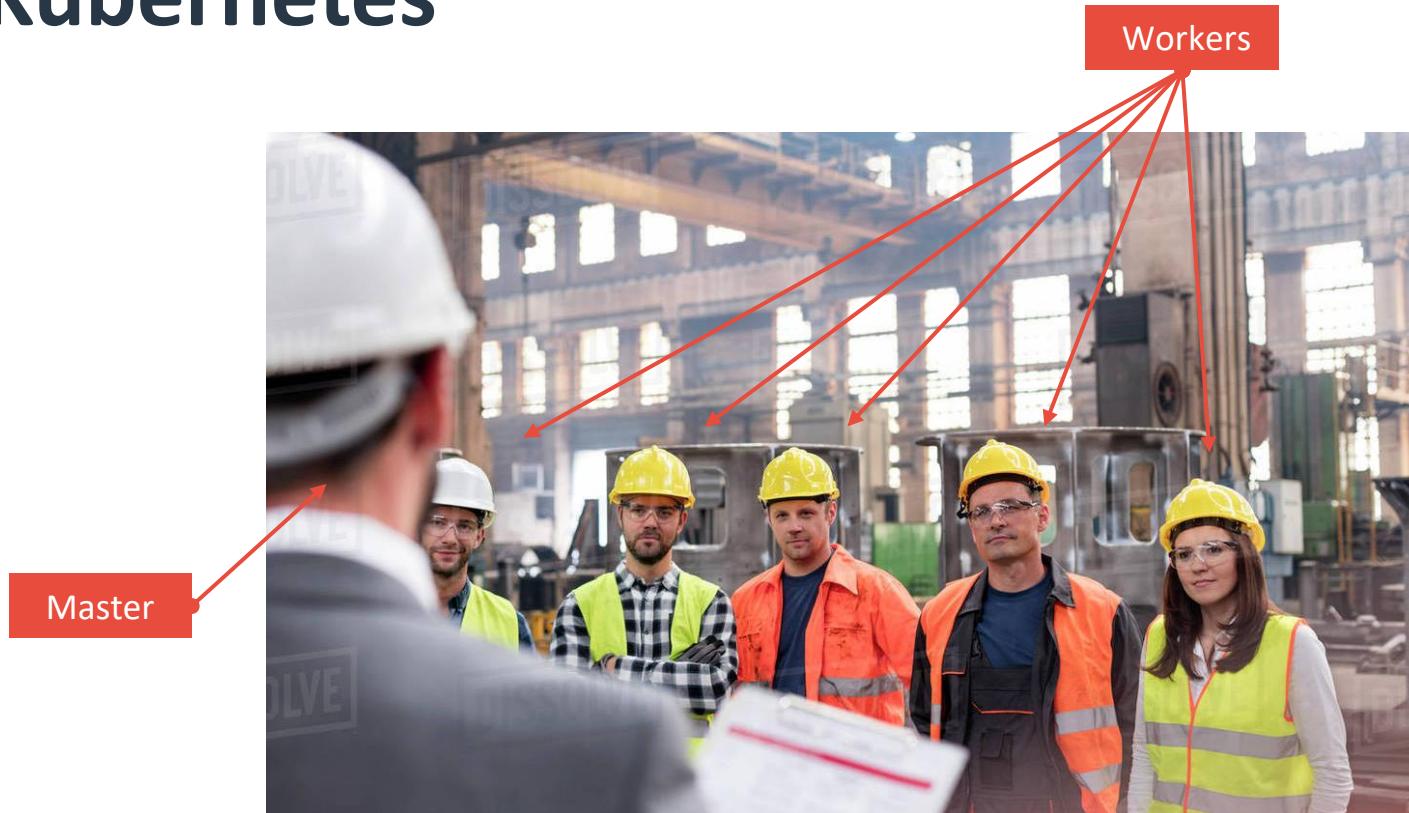
Self-Containment Principle



Single Concern Principle

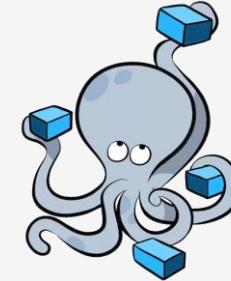


Kubernetes



Container Orchestration Features

- 컨테이너 자동 배치 및 복제
- 컨테이너 그룹에 대한 로드 밸런싱
- 컨테이너 장애 복구
- 클러스터 외부에 서비스 노출
- 컨테이너 확장 및 축소
- 컨테이너 서비스간 인터페이스를 통한 연결



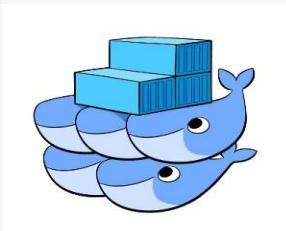
Container Orchestrators

Kubernetes



구글에서 개발, 가장 기능이 풍부하며 널리 사용되는 오케스트레이션 프레임워크
베어 메탈, VM환경, 퍼블릭 클라우드 등의 다양한 환경에서 작동하도록 설계
컨테이너의 롤링 업그레이드 지원

Docker Swarm



여러 개의 Docker 호스트를 함께 클러스터링하여 단일 가상 Docker 호스트를 생성
호스트 OS에 Agent만 설치하면 간단하게 작동하고 설정이 용이
Docker 명령어와 Compose를 그대로 사용 가능

Apache Mesos



수만 대의 물리적 시스템으로 확장할 수 있도록 설계
Hadoop, MPI, Hypertable, Spark 같은 응용 프로그램을 동적 클러스터 환경에서 리소스 공유와 분리를 통해 자원 최적화 가능
Docker 컨테이너를 적극적으로 지원

Kubernetes

“Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.”

“**Kubernetes**는 컨테이너화된 어플리케이션을 자동으로 배포, 조정, 관리할 수 있는 오픈소스 플랫폼이다.

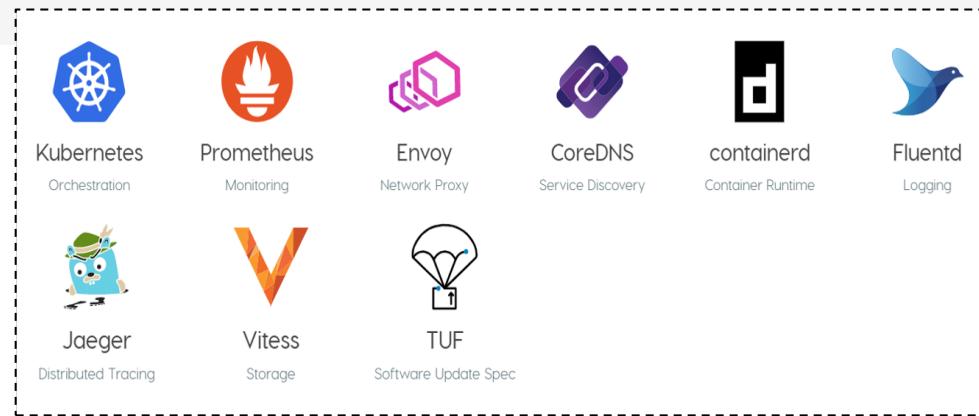
- From Kubernetes Website

Kubernetes Origin

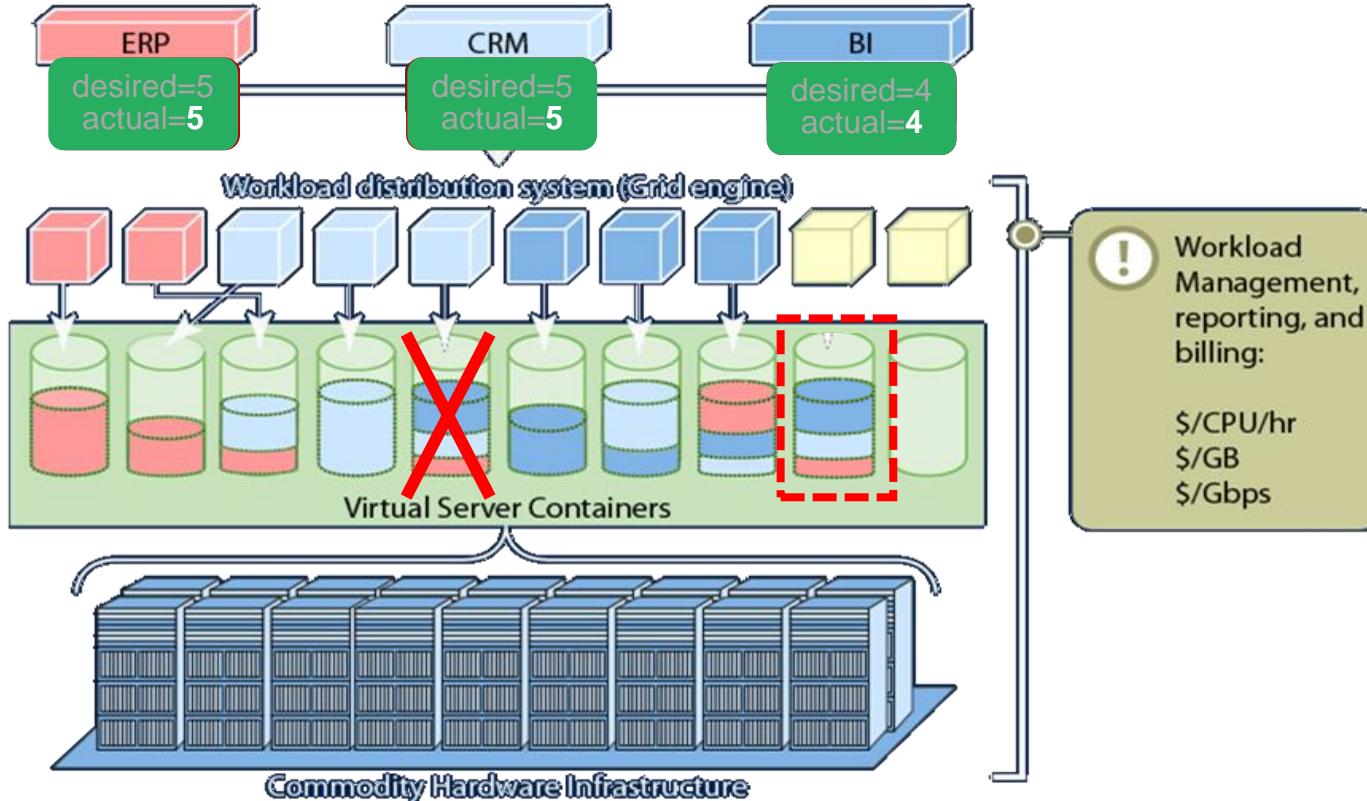
- Borg System 영향을 받아, 2014년 구글의 의해 처음 발표
- 2015년 7월 21일, v1.0 출시
- 리눅스재단과 Cloud Native Computing Foundation(CNCF) 설립
- Kubernetes를 seed 테크놀로지(seed technology)로 제공



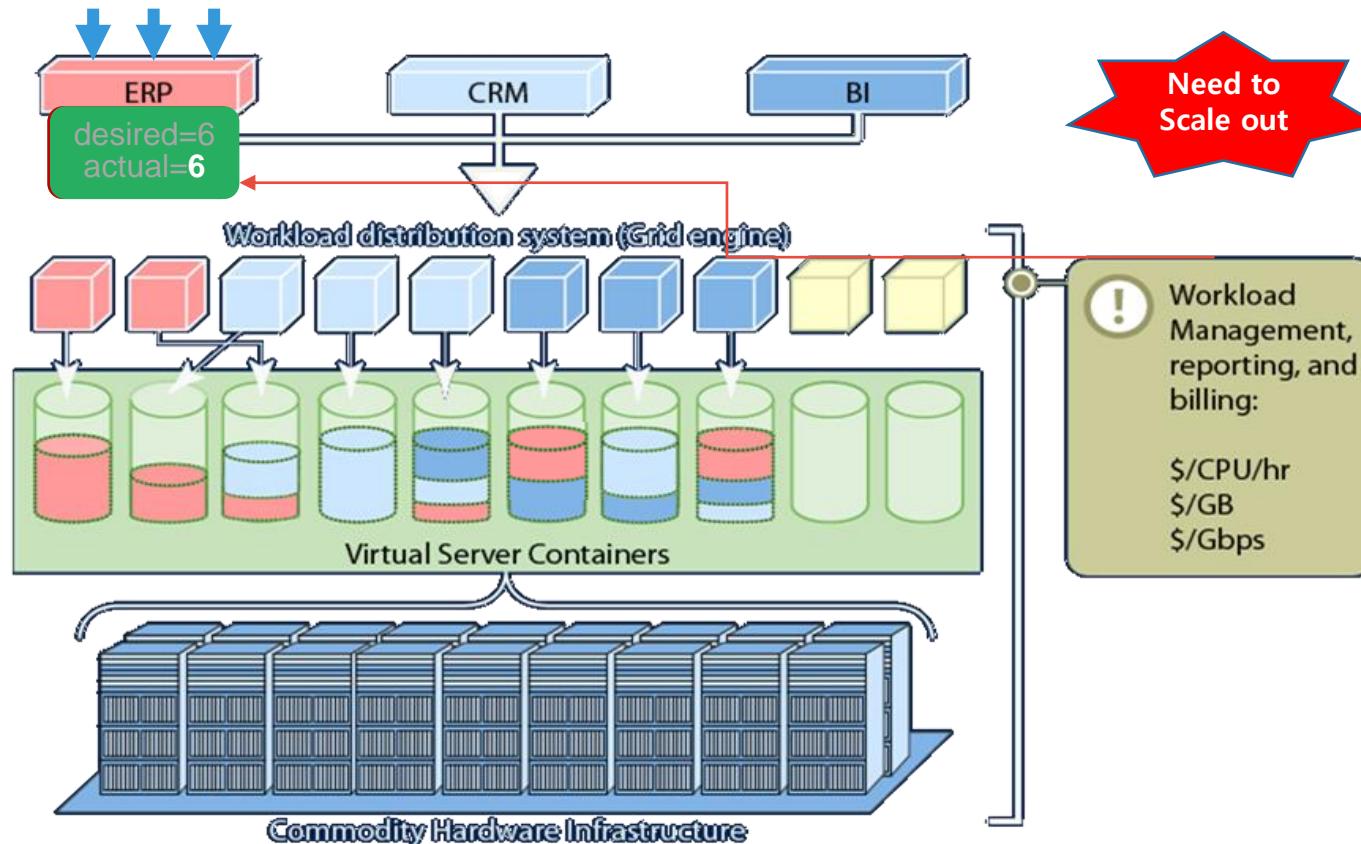
<https://www.cncf.io/>



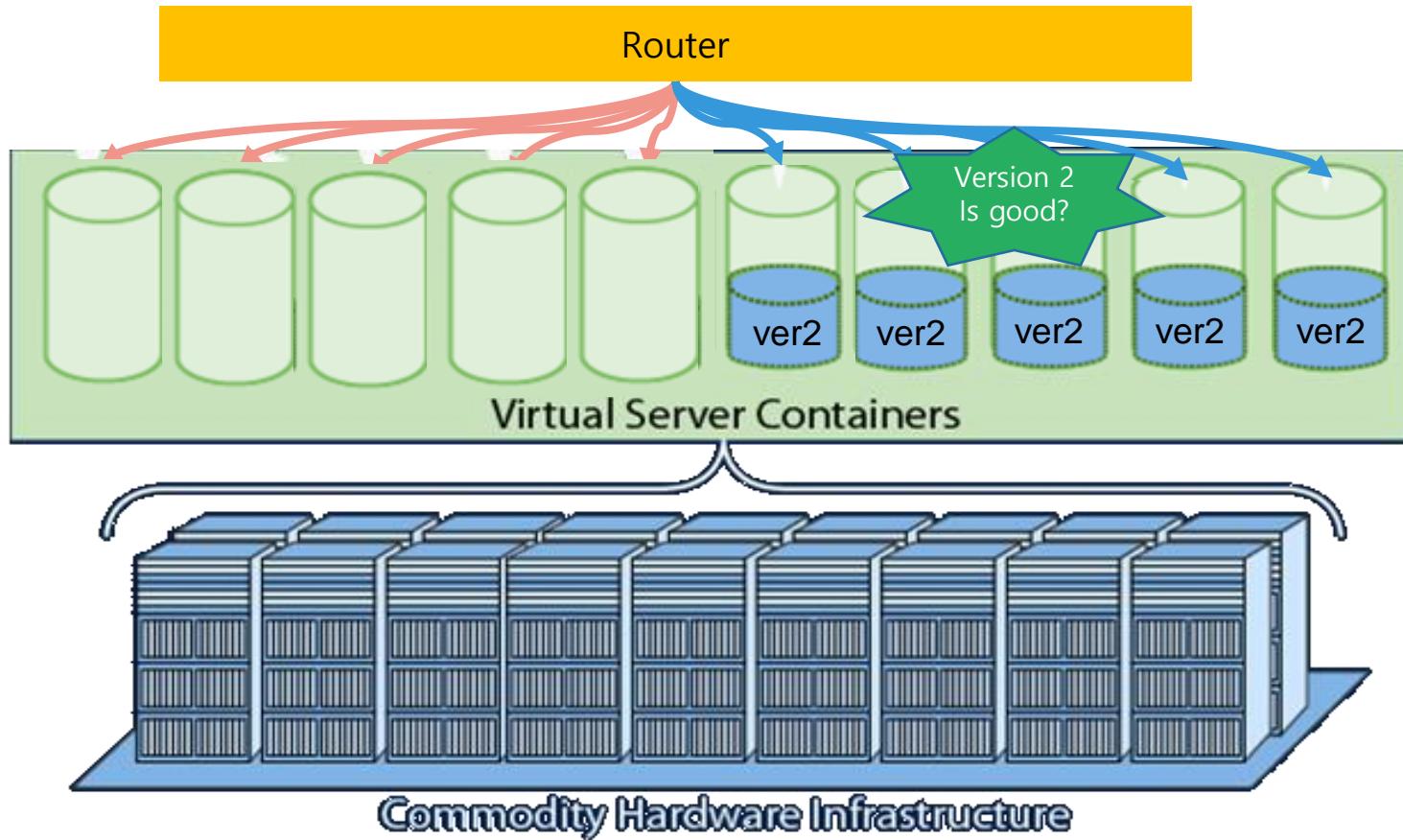
Container Orchestration – Self Healing Mechanism



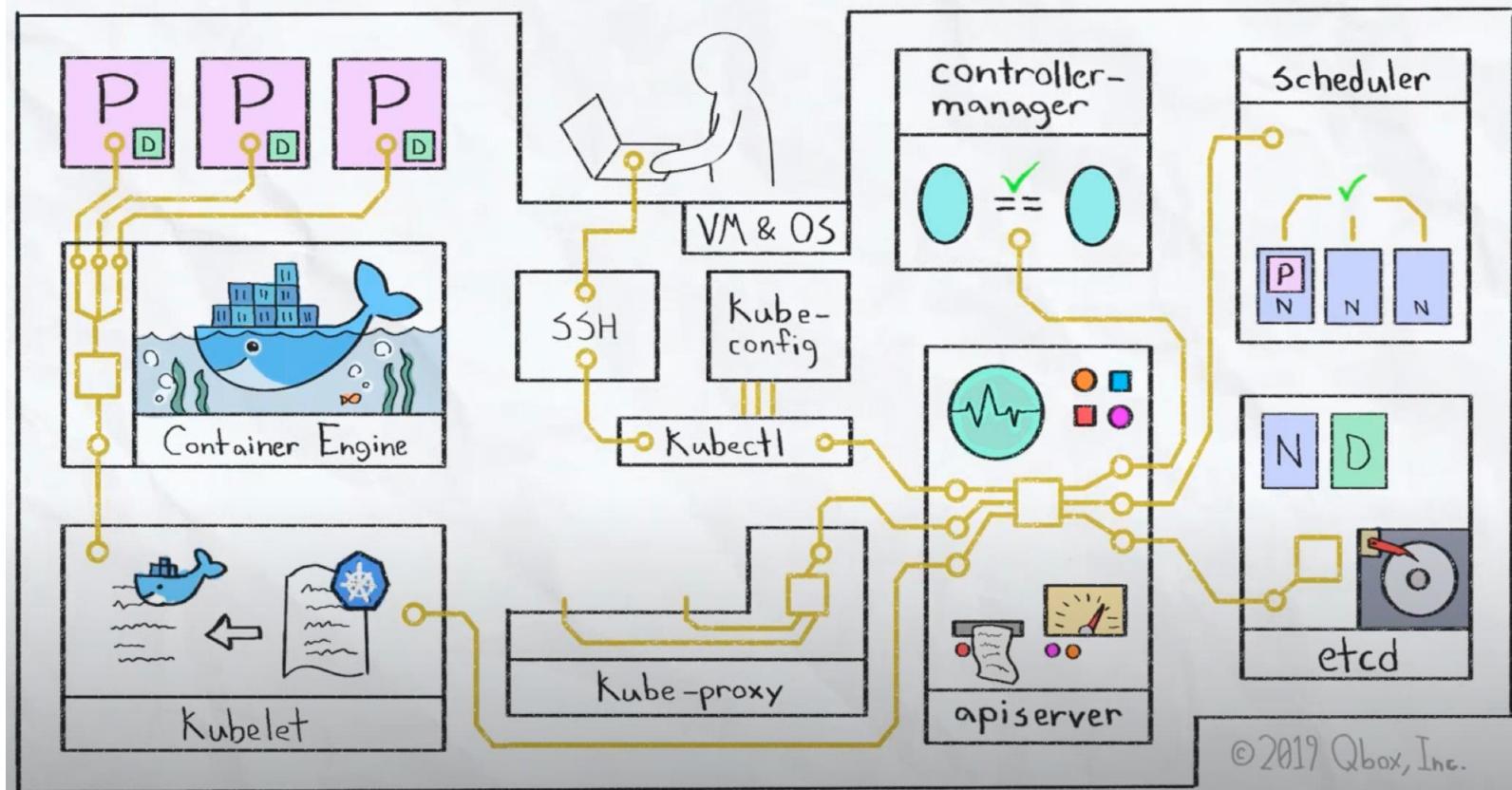
Container Orchestration – Scale out



Container Orchestrator – Zero Down Time Deploy

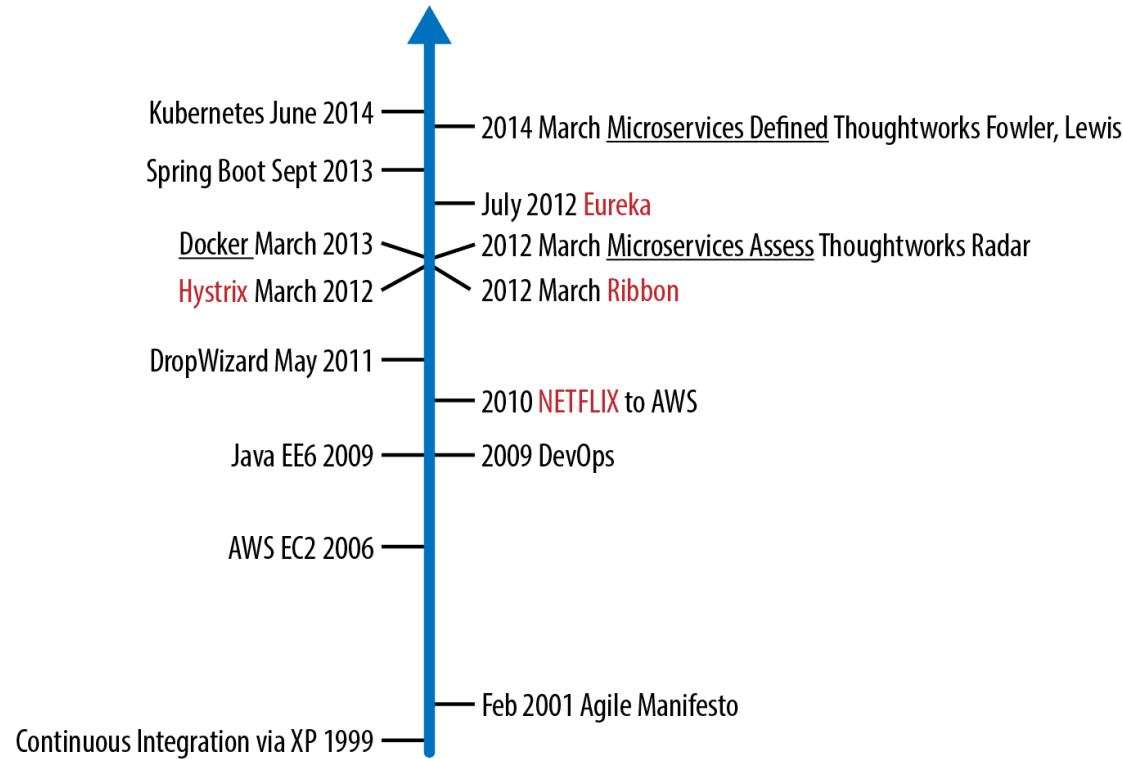


Anatomy of Kubernetes



Service Mesh

Microservices timeline



K8S vs Spring Cloud vs Istio

Capabilities	Spring Cloud	Istio	Kubernetes	Spring Cloud with Kubernetes & Istio on IaaS+
DevOps Experience				Self service, multi-environment capabilities
Auto Scaling & Self Healing				Pod/Cluster Autoscaler, HealthIndicator, Scheduler, Pool Ejection
Deployment & Scheduling				Deployment strategy, DarkLaunch, AB, Canary, Scheduler Strategy
Resilience & Fault Tolerance				HealthIndicator, Hystrix, HealthCheck, Process Check, Circuit Breaker/Timeout/Retry
Distributed Tracing				Zipkin, Jaeger
Centralized Metrics				Heapster, Prometheus, Grafana
Centralized Logging				EFK
API Gateway				Zuul, Traffic Control, Egress
Load Balancing				Ribbon, Service, Envoy
Chaos Engineering				Chaos Monkey for Spring Boot, Chaos Toolkit Kubernetes, Envoy
Service Discovery				Service
Configuration Management				Externalized Configuration, ConfigMaps, Secrets
Application Packaging				Spring Boot Maven/Gradle plugin
Job Management				Spring Batch, Scheduled Jobs
Process Isolation				Docker, Pods, Envoy
Environment Management				Namespaces, Authorization
Resource Management				CPU and Memory Limits, Namespace resource quotas
Operating System		IaaS+		Ubuntu, Atomic, ...
Virtualization				VMWare, Openstack, ...
Hardware, Storage, Networking				AWS, GCP, Azure, ...

Advanced Routing & Deployment Strategy

- Ingress / Egress Gateway
- Canary Deploy
 - 특정 유저의 신상, 지역, 권한, 접근 단말에 따른 다른 버전의 노출
- AB Testing / Shadow Deploy (Dark Launch)
 - 신규 버전의 오류 노출 없는 실질적 테스트

Advanced Resilience

- Retry (Kubernetes X, Spring Cloud O)
 - 서비스간 호출의 실패에 대한 재시도
- Circuit Breaking (Spring Cloud O) / Rate Limiting
 - 인스턴스의 보호
 - 전체 서비스 장애 차단
- Pool Ejection (Spring Cloud / Eureka)
 - 죽은 인스턴스의 제외
- Circuit Breaking + Pool Ejection + Retry = High Resilience

Advanced Security

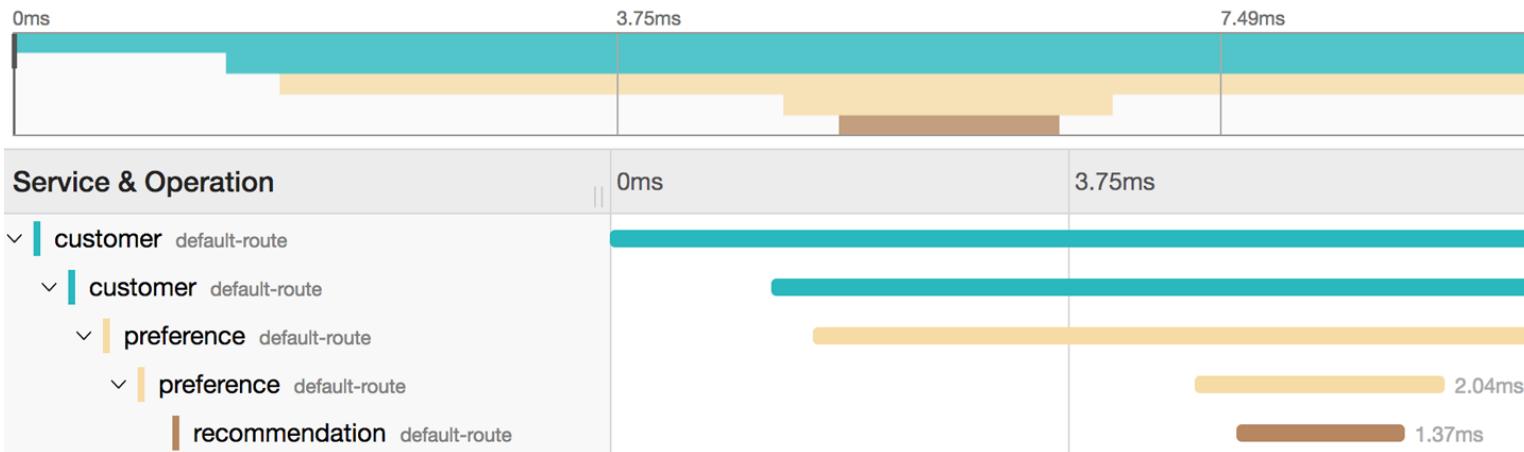
- TLS based Inter-Mi-services Communication
 - By Auth (신규모듈)
- Whitelist and Blacklist

Advanced Observability

- Distributed Tracing and Measure
 - 서비스간 호출의 내용 기록

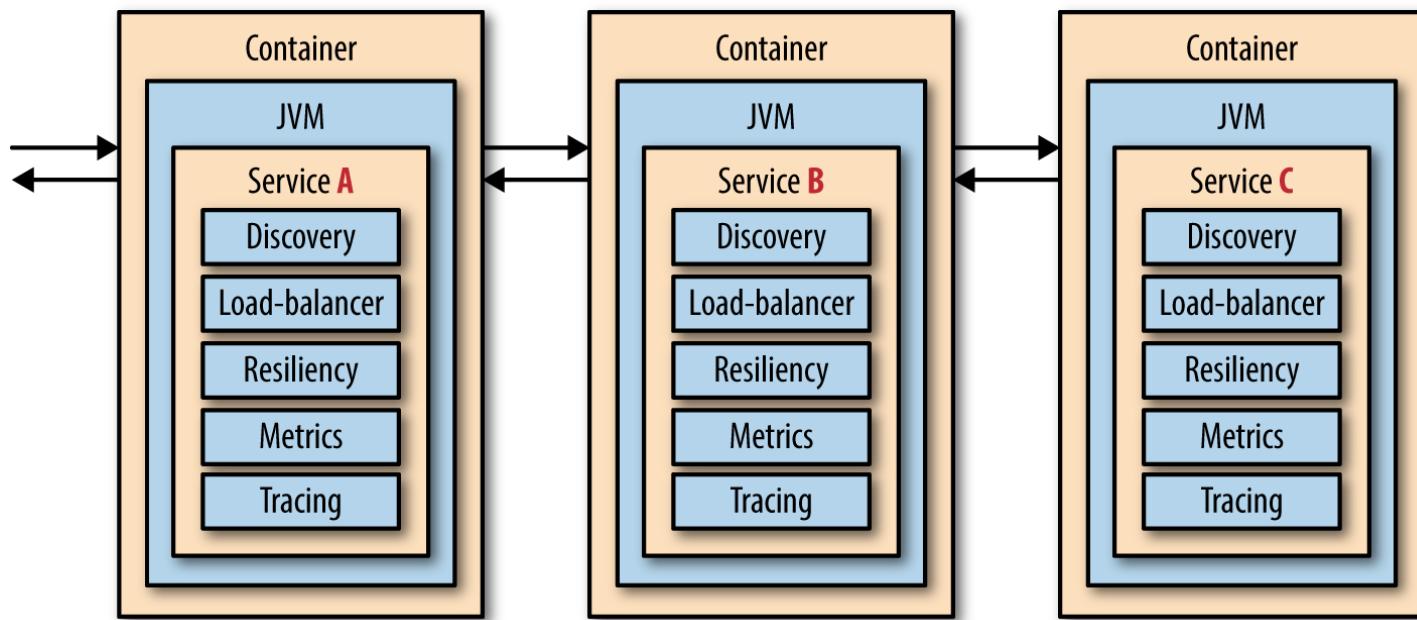
▼ customer: default-route

Trace Start: February 28, 2018 7:46 PM | Duration: 14.99ms | Services: 3 | Depth: 5 | Total Spans: 5

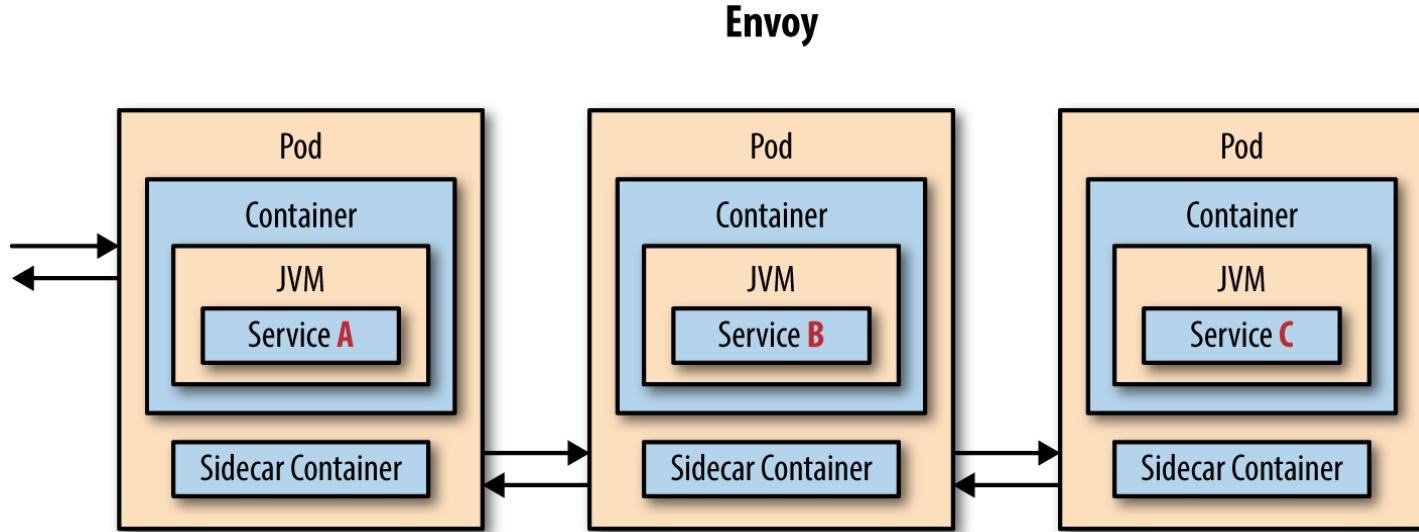


Spring Cloud + Netflix

Before Istio



After Istio (on K8S)



좋은점:

1. L7 레이어를 사용, 성능이 높음
2. Code 변경 없이 Cross-cutting 이슈를 다루어줌
3. Polyglot 다양한 언어에 무관하게 적용 가능
4. Main 서비스의 재배포 없이 Sidecar 를 관리 가능함

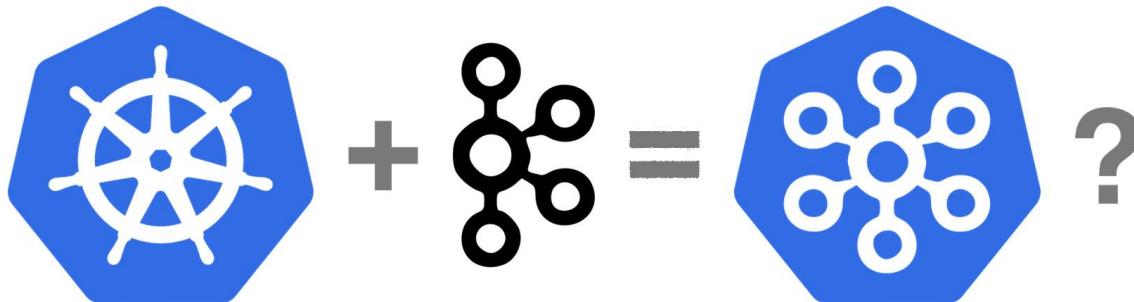
MSA 구현 기술들

- 1세대 MSA (Spring Cloud / Netflix OSS) 코드수정 ✓
- 2세대 Service Mesh (Kubernetes / Istio) 코드수정 X
- 3세대 Event Driven MSA (Serverless / Kafka) 간섭최소화

MSA 구현 기술들

비교	Spring Cloud / Netflix OSS	Kubernetes / Istio	Event Driven MSA (Kafka / K-Native)
제공기능	API GW, Service Registry, Circuit Breaker ...	Covers all of Netflix OSS + Canary Deploy, Dark Launch ...	PubSub, Materialized View, Realtime Streaming Processing
코드 변경	필요 (only Java)	불필요 (Polyglot)	N/A (with CDC)
서비스 간 커플링	Loosely-Coupled	Loosely-Coupled	가장 Loosely-coupled
데이터 프로젝션 성능	느림 (Blocked, Request-Response)	느림 (Blocked, Request-Response)	빠름 (Non-blocking, Event Sourcing)

Kubernetes + Istio + Kafka: → High Availability, High Business Continuity



- Self-Healing
- ZDT Deploy
- Auto Scale
- Non-blocking
- Time-Decoupled
- Pluggable subscriber

Reference Cloud Native Architecture

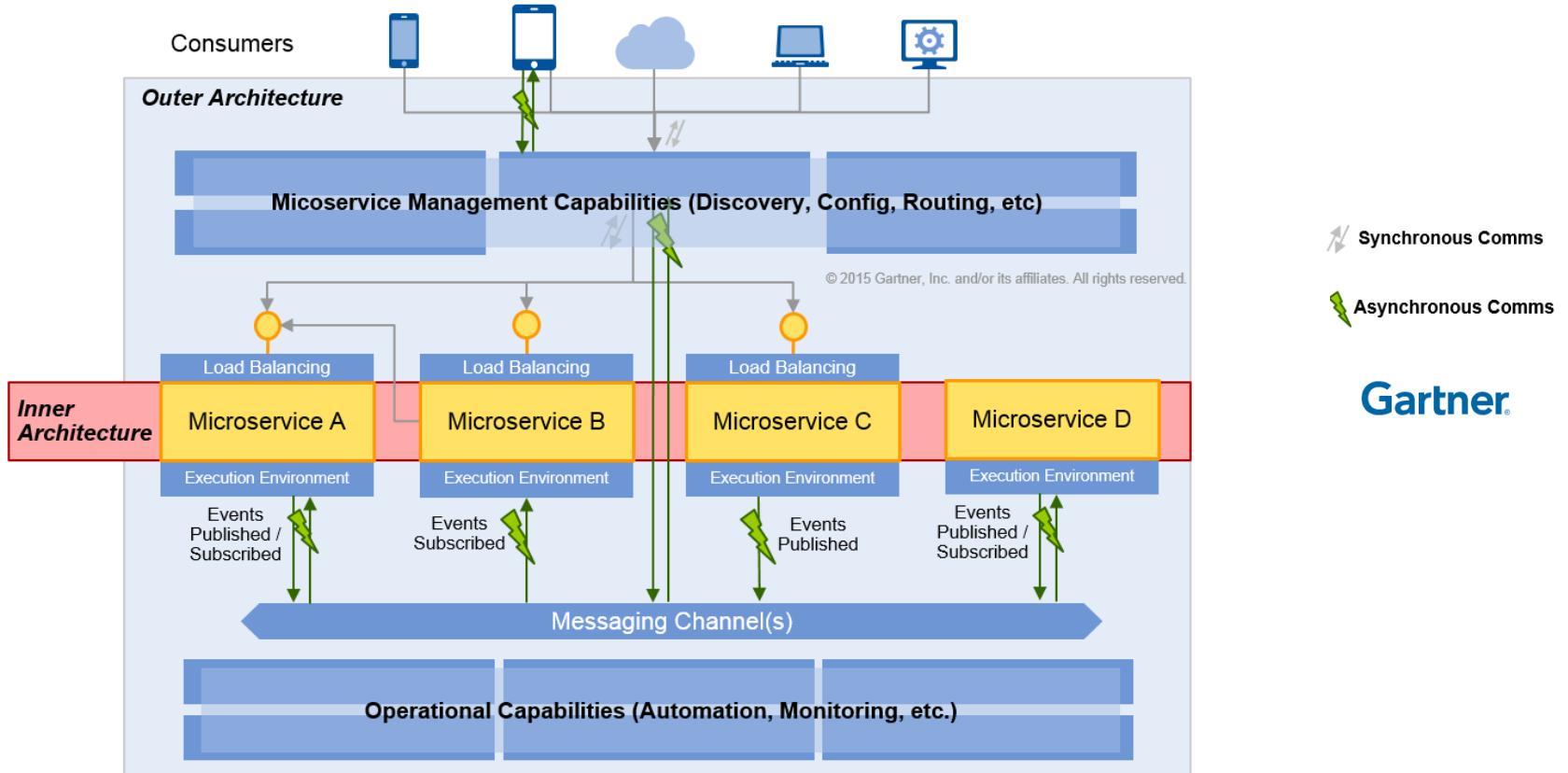
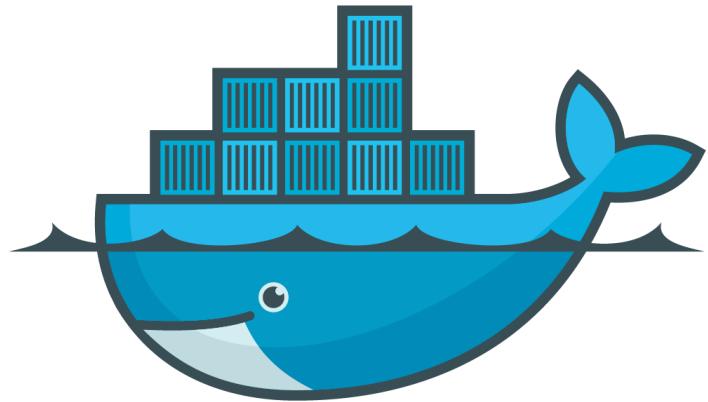


Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker) 
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

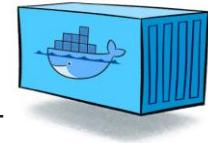


docker

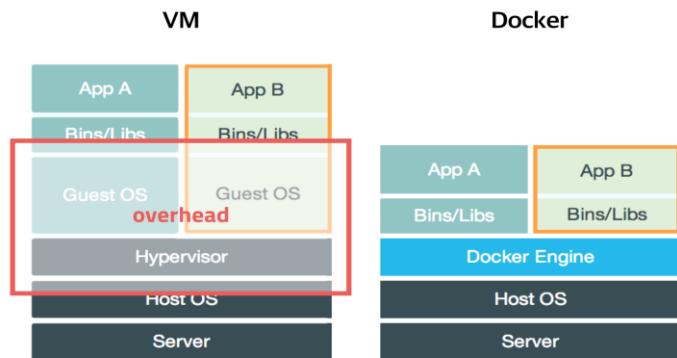
- 2013년 3월,
dotCloud 창업자 Solomon Hykes가 Pycon Conference에서 발표
- Go 언어로 작성된 “The future of linux Containers”

- Container based

- 프로세스 격리 기술
- 오픈소스 가상화 플랫폼



- 가상머신 .vs. Docker



Images & Containers

- **Image → Template**

- 가상머신 생성시 사용하는 ISO 와 유사한 개념의 이미지
- 여러 개의 층으로 된 바이너리 파일로 존재
- 컨테이너 생성시 읽기 전용으로 사용
- 도커 명령어로 레지스트리로부터 다운로드 가능

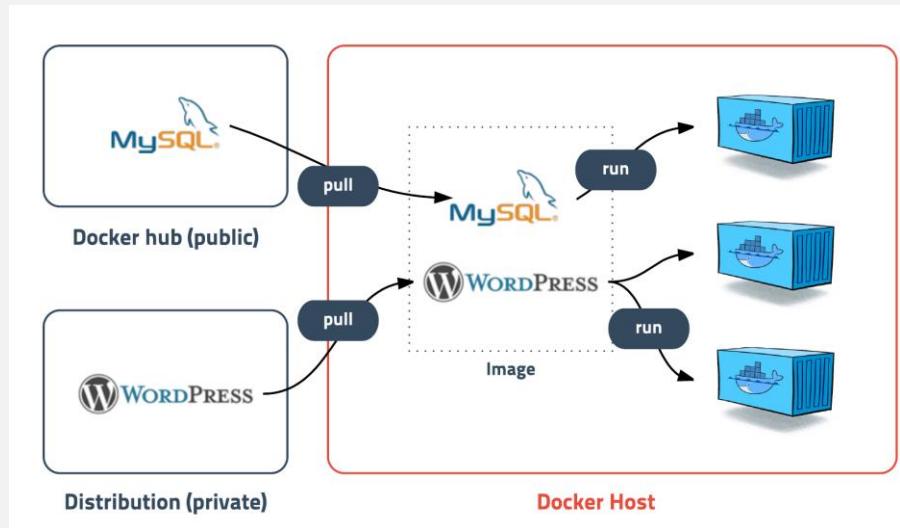


- 저장소 이름 : 이미지가 저장된 장소, 이름이 없으면 도커 허브(Docker Hub)로 인식
- 이미지 이름 : 이미지 이름, 생략 불가
- 이미지 버전 : 이미지 버전정보, 생략 시 latest로 인식

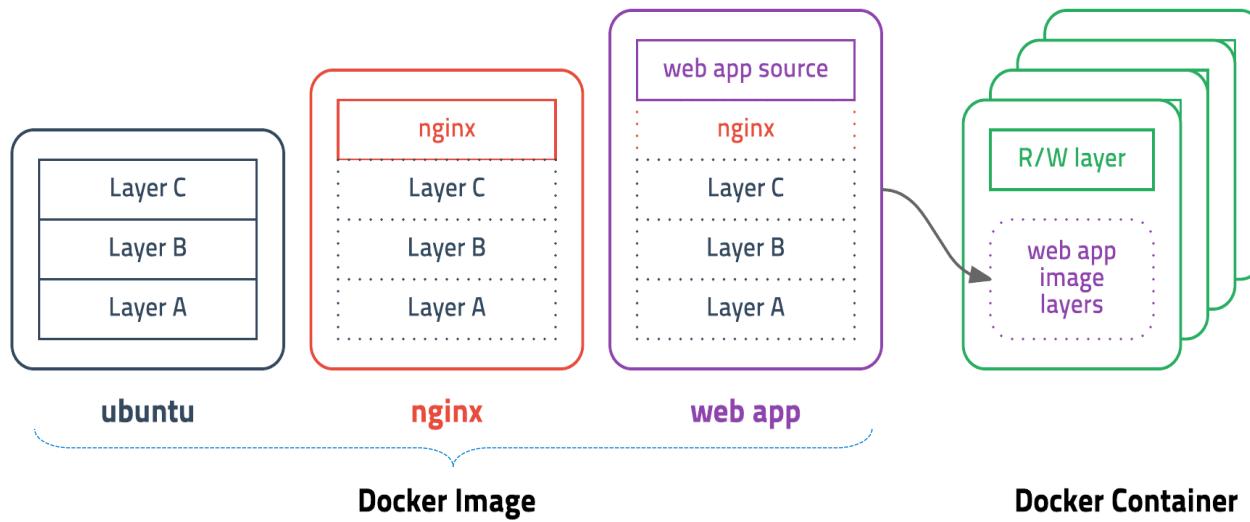
Images & Containers

- Containers → Instances

- 도커 이미지로 부터 생성
- 격리된 파일시스템, 시스템 자원, 네트워크를 사용할 수 있는 독립공간 생성
- 이미지를 읽기 전용으로 사용, 이미지 변경 데이터는 컨테이너 계층에 저장

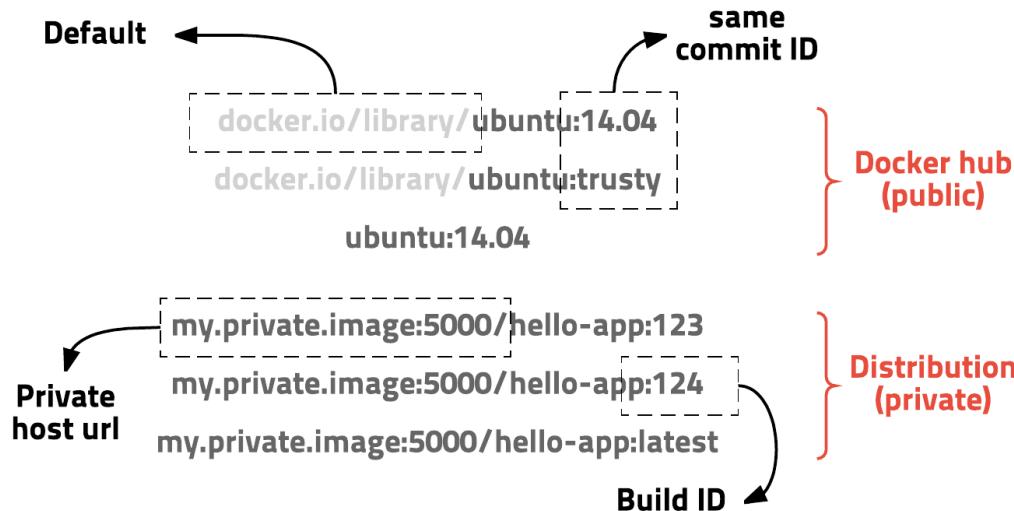


Layered Architecture



- **Image** : 여러 개의 읽기 전용(Read Only) 레이어로 구성
- **Container** : Image 위에 R/W 레이어를 두고, 실행 중 생성 또는 변경 내용 저장

Image Registry and Image Naming



- 이미지 Path는 <URL>/<namespace>/<Image_name>:<tag> 형식
- library는 도커허브 공식 이미지 Namespace로, 여기에 사용자 이름이 위치

Dockerfile

```
FROM openjdk:8-jdk-alpine
```

```
RUN apk --no-cache add tzdata && cp  
/usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

```
WORKDIR /app  
COPY hello.jar hello.jar  
COPY entrypoint.sh run.sh  
RUN chmod 774 run.sh
```

```
ENV PROFILE=local
```

```
ENTRYPOINT ["./run.sh"]
```

- **FROM** : 이미지를 생성할 때 사용할 기반 이미지를 지정한다.
- **RUN** : 이미지를 생성할 때 실행할 코드 지정한다. 예제에서는 패키지를 설치하고 파일 권한을 변경하기 위해 RUN을 사용
- **WORKDIR** : 작업 디렉토리를 지정한다. 해당 디렉토리가 없으면 새로 생성한다. 작업 디렉토리를 지정하면 그 이후 명령어는 해당 디렉토리를 기준으로 동작
- **COPY** : 파일이나 폴더를 이미지에 복사한다. WORKDIR로 지정한 디렉토리를 기준으로 복사
- **ENV** : 이미지에서 사용할 환경 변수 값을 지정한다. 컨테이너를 생성할 때 PROFILE 환경 변수를 따로 지정하지 않으면 local을 기본 값으로 사용
- **ENTRYPOINT** : 컨테이너를 구동할 때 실행할 명령어를 지정한다.

Docker Image Commands

- 도커 이미지 목록 확인
 - \$ docker images
- 도커 이미지 불러오기
 - 컨테이너 run 시에 이미지가 없으면 Docker Hub로부터 자동으로 Pull
 - \$ docker pull [ImageName:태그]
- 도커 이미지 삭제
 - docker image rm [이미지 ID]
 - docker image rm -f [이미지 ID] # 컨테이너를 삭제하기 전에 이미지 삭제
- 도커 모든 이미지 한 번에 삭제
 - \$ docker image rm \$(docker images -q)

Docker Container Commands

- 컨테이너 실행
 - \$ docker run [Options] [Image] [Command]
- 실행 중인 컨테이너 확인
 - \$ docker ps
 - \$ docker ps -a # 정지된 컨테이너 포함
- 컨테이너 시작, 재시작, 종료
 - \$ docker start / restart / stop [컨테이너 이름]
- 컨테이너 삭제
 - \$ docker container rm [컨테이너 ID]
- 모든 컨테이너 한번에 삭제 (중지 후 삭제)
 - \$ docker container rm \$(docker ps -a -q)

Docker Registry Commands

- Dockerfile로 이미지 생성
 - docker build --tag [생성할 이미지 이름]:[태그 이름].
 - # 맨 마지막의 .(마침표)은 Dockerfile의 위치
 - 이미지 이름은 URL(Docker hub, Cloud Container registry, Private registry)로 시작
- 이미지 Push
 - \$ docker login # 도커 로그인
 - \$ docker push [이미지 REPOSITORY]:[태그]
- Docker Hub에서 확인
 - <http://hub.docker.com> (login)

Lab. 12번가 서비스 배포



Lab Time

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes) 
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

Kubernetes key Features (1/2)

- **Automatic Scheduling**
 - 각 컨테이너가 필요로 하는 CPU와 메모리(RAM)를 쿠버네티스에게 요청하면, 쿠버네티스는 컨테이너를 노드에 맞추어서 자동으로 스케줄링
- **Self-healing**
 - Kubernetes는 실패한 노드에 대해, 컨테이너를 자동으로 교체하고, 재 스케줄링하며, 또한 Health check에 반응하지 않는 컨테이너를 정해진 규칙에 따라 다시 시작
- **Horizontal Scaling**
 - Kubernetes는 CPU 및 메모리와 같은 리소스 사용량을 기반으로 애플리케이션을 자동으로 확장 할 수 있으며, 메트릭을 기반으로 하는 동적 스케일링도 지원
- **Service discovery and load balancing**
 - Service Discovery 매커니즘을 위해 Application을 수정할 필요가 없으며, K8s는 내부적으로 Pod에 고유 IP, 단일 DNS를 제공하고 이를 이용해 load balancing

Kubernetes key Features (2/2)

- **Automated rollouts and rollbacks**

- Application, Configuration의 변경이 있을 경우 전체 인스턴스의 중단이 아닌 점진적으로 Container에 적용(rolling update) 가능
- Release revision이 관리되고 새로운 버전의 배포시점에 문제가 발생할 경우, 자동으로 이전의 상태로 Rollback 진행

- **Secret and configuration management**

- Kubernetes는 secrets와 Config 정보에 대해 이미지 재빌드없이 변경관리가 가능하고, Github 같은 저장소에 노출시키지 않고도 어플리케이션 내에서 보안정보 공유 가능

- **Storage Orchestration**

- Local storage를 비롯해서 Public Cloud(Azure, GCP, AWS), Network storage 등을 구미에 맞게 자동 mount 가능

Kubernetes: 발음하기

미국식: 큐브네리스

영국식: 쿠버네티스

인도식: 꾸-버네딕스

Kubernetes: 쓰기

Kubernetes



K8S ≠

KBS 

Kubernetes Object Model

`http://serviceld:8080`

Micro-Service

Service
(name: foo)

`http://external_ip:8080`

Load Balancer from
cloud provider

Redirects

Deployment

- Dynamic Service Binding

- e.g. <http://foo:8080>

- Type :

- LoadBalancer e.g. Ingress (API GW) or front-end
- ClusterIP e.g. 내부 마이크로 서비스들

- Zero-down time deployment

- (Kubernetes default is rolling-update)
e.g. Blue / Green

Creates / Manages

ReplicaSet
(Blue)

ReplicaSet
(Green)

- Keep replica count as desired
(replicas=2)

Pod

Container
(Main)

Container
(Side-car)

Pod

Container

Container

Pod

Container

Container

Pod

Container

Container

- Service Hosting

Declaration based configuration

> 12street.yml

> Desired state



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product
  labels:
    app: product
spec:
  replicas: 2
...
template:
  metadata:
    labels:
      app: product
  spec:
    containers:
      - name: product
        image: product:1.7.9
        ports:
          - containerPort: 80
...
apiVersion: apps/v1
kind: Service
metadata:
  name: product-service
  labels:
    app: product
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order
  labels:
    app: order
spec:
  replicas: 3
...
template:
  metadata:
    labels:
      app: order
  spec:
    containers:
      - name: order
        image: order:2.0
        ports:
          - containerPort: 8080
...
apiVersion: apps/v1
kind: Service
metadata:
  name: order-service
  labels:
    app: order
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db-deployment
  labels:
    app: db
spec:
  replicas: 2
...
template:
  metadata:
    labels:
      app: db
  spec:
    containers:
      - name: db
        image: mongo
        ports:
          - containerPort: 27017
```

```
apiVersion: apps/v1
kind: Service
metadata:
  name: mongo-service
  labels:
    app: mongo
```

Public Kubernetes Services



- 직관적이고 편한 UI
- 높은 윈도우 서비스 호환
- 국내 2개 리전 보유
- VM 생성 및 실행 속도



- 우세한 시장 점유율
- 높은 서비스 성숙도
- 광범위한 서비스 교육
- UI 및 사용성, 이용 요금



- Cloud-friendly Biz 디자인
- 전문적인 DevOps 보유
- 높은 컴퓨터 오퍼링
- 높은 Windows VM 비용

“

Kubernetes Basic Objects

- “Desired State” declaration
- Pod, Label, ReplicaSet, Deployment, Service, Volume, Configmap, Secret
- Liveness & Readiness

Kubernetes Core Concept : “Desired State”

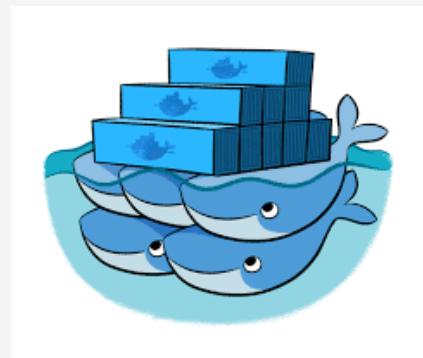


Declarative Model & Desired States

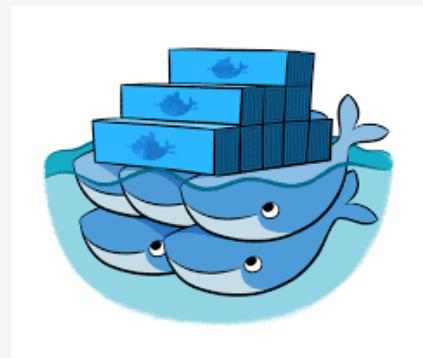
- **Kubernetes는 Current State을 모니터링하면서, Desired State를 유지하려는 습성**
- 직접적인 동작을 명령하지 않고, 원하는 상태를 선언(Not Imperative, But Declarative)
 - Imperative – “nginx 컨테이너를 3개 실행해줘, 그리고 80포트로 오픈해줘.”
 - Declarative – “80포트를 오픈한 채로, nginx 컨테이너를 3개 유지해줘.”

Pod ; Kubernetes 최소 배포 단위

- Pod : 미국식 [pa:d], 영국식 [pɒd]
 - “물고기, 고래” 작은 떼 (Docker의 심볼이 고래 모양에서 유래)
 - 발음하기 : “팟”, “파드”, “포드”



Pod



Pod

Kubernetes Object Model

- **apiVersion** : 해당 Object description 을 해석할 수 있는 API server 의 버전
- **kind** : 오브젝트의 타입 – 예제는 **Deployment**
- **metadata** : 객체의 기본 정보. 예) 이름
- **Spec (spec and spec.template.spec)** : 원하는 "Desired State" 의 세부 내역. 예제에서는 3개의 replica를 template 내의 pod 정의대로 찍어내어 유지하라는 desired state 설정임
- **spec.template.spec** : defines the desired state of the Pod. The example Pod would be created using **nginx:1.7.9**.

Once the object is created, the Kubernetes system attaches the **status** field to the object

```
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:1.7.9
          ports:
            - containerPort: 80
```

Deployment object Example

Lab. 기본적인 kubectl 명령어

- 객체 목록 불러오기
 - “[kubectl get \[객체 타입\]](#)” Ex) kubectl get pods : pod 목록을 불러온다.
- 객체 삭제하기
 - “[kubectl delete \[객체 타입\] \[객체 이름\]](#)”
Ex) “kubectl delete pods wordpress-5bb5dddcff-kwgf8”
 - “[kubectl delete \[객체타입,객체타입,...\] --all](#)”
Ex) “kubectl delete services,deployments,pods --all”
 - 실습 중에 잘못 생성되었거나 초기화가 필요할 경우 사용
 - 콤마(,)로 구분하여 붙여 적기
- 객체 상세 설명 확인하기
 - “[kubectl describe \[객체 타입\] \[객체 이름\]](#)”
Ex) “kubectl describe service wordpress”

Lab. 이미지를 통한 어플리케이션 배포

- 현재 작동 중인 pod들이 없는지 확인
 - [kubectl get pods](#)
- Nginx 이미지 예제로 배포하기
 - [kubectl create deploy first-deployment --image=nginx](#)
- 실행된 Pods 확인
 - [kubectl get pods](#)
 - 각 Pod들은 ‘Pod명-{hash}’ 형식의 고유한 이름을 가짐

Lab. Pod에 접속하기

- 로그 보기
 - `kubectl logs [복사된 pod 이름] -f`
- 복사된 pod이름으로 접속하기
 - `kubectl exec -it [복사된 pod 이름] -- /bin/bash`
- Pod 내에서 접속하기 (위 명령어로 진입후에는 리눅스 Shell 명령어 사용)
 - Curl 명령어를 사용하기 위한 업데이트를 한다. (Curl 명령이 없으면 설치)
`apt-get update`
`apt-get install curl`
 - Curl 명령어로 호출하기
`curl localhost`

설정 파일을 통한 pod 배포 (1/2)

- 아래 내용으로 nano editor 를 이용하여 declarative-pod.yaml 파일을 생성
 - Nginx 이미지를 기반으로 pod를 배포하는 설정파일

```
apiVersion: v1
kind: Pod
metadata:
  name: declarative-pod
spec:
  containers:
    - name: memory-demo-ctr
      image: nginx
```

설정 파일을 통한 pod 배포 (2/2)

- nano declarative-pod.yaml 파일을 직접 작성 후 배포
 - `kubectl create -f declarative-pod.yaml`
(-f 는 파일 경로를 설정해서 배포할 수 있는 옵션이다.)
- 배포된 Pod를 검색 후 접속
 - 이름이 설정대로 declarative-pod 로 생성됨을 확인
`kubectl get pods`
 - 생성된 Pod에 접속
`kubectl exec -it declarative-pod -- /bin/bash`
`apt-get update`
`apt-get install curl`
`curl localhost`

Pod Initialization

- `Initl.yaml` 파일 생성
 - `nano pod-initialize.yaml`
- Pod 생성
 - `kubectl create -f init.yaml`
- Pod 접속하기
 - `kubectl exec -it init-demo -- /bin/bash`
 - `cd /usr/share/nginx/html`
 - `cat index.html`

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html
  initContainers:
    - name: install
      image: busybox
      command:
        - wget
        - "-O"
        - "/work-dir/index.html"
        - "http://kubernetes.io"
      volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
        - name: workdir
          emptyDir: {}
```

Pod 초기화 시점에
실행

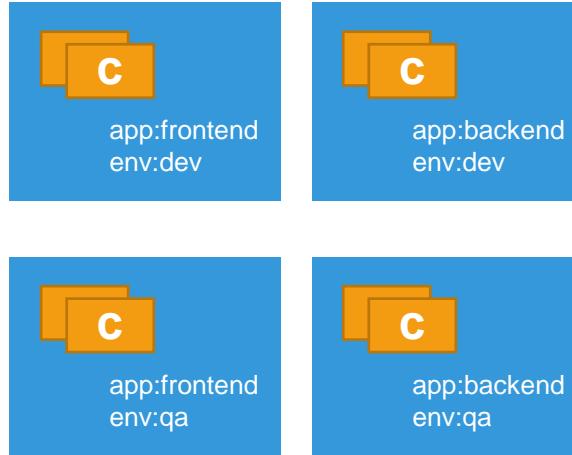
Lab. Pod



Lab Time

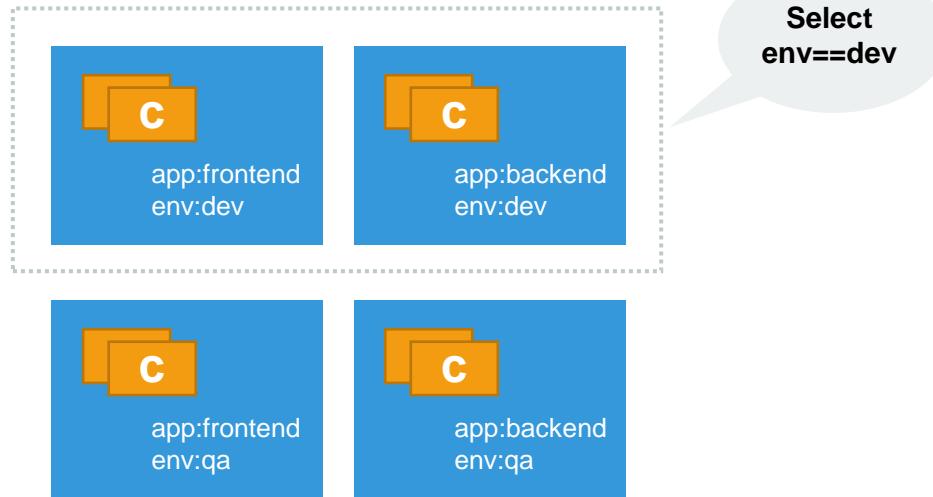
- Lab Script Location
 - Workflowy :

Labels



- Labels 은 객체 식별 정보로서 Kubernetes 객체라면 모두 붙일 수 있다.
- Label들은 요구사항에 맞춰 개체의 하위 집합을 구성하고 선택하는데 사용된다.
- Label들은 객체에 고유성을 제공하지 않아, 여러 객체들은 같은 label을 가질 수 있다.

Label Selectors



- Label Selectors들은 객체들의 집합을 선택하며, kubernetes는 2가지 종류를 지원한다.
 - **Equality-Based Selectors**
Uses the `=`, `==`, `!=` 연산자를 이용하여 Label key와 value 값을 기반으로 객체들을 필터링 할 수 있다.
 - **Set-Based Selectors**
`in`, `notin`, `exist` 연산자를 사용하며, value 값을 기반으로 객체들을 선택할 수 있다.

Lab. Label

- kubectl get po # 실행 중, Pod list 확인
- kubectl edit po <pod 명> # Pod 인스턴스에 Label 추가

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-02-16T11:22:56Z"
  labels:
    env: test
  name: init-demo
  namespace: default
  resourceVersion: "588586"
```

] ← vi 에디터로 편집

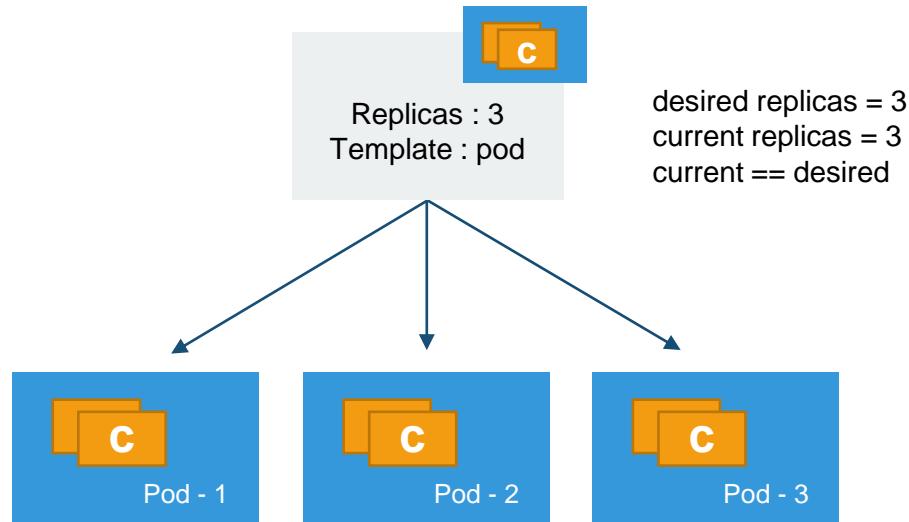
- kubectl get pods -l env=test
- kubectl get pods --selector env=test
- kubectl get pods --selector 'env in (test, test1)' # or 연산
- kubectl get po --selector 'env in(test, test1), app in (nginx, nginx1)' # and 연산
- kubectl get po --selector 'env,app notin(nginx)' # env가 있으면서, app이 nginx가 아닌 Pod

Replication Controllers

- Master node의 Controller Manager 중 하나
- Pod의 복제품이 주어진 개수(Desired State)만큼 작동하고 있는지 확인하고 개수를 조절 한다.
- Replication Controller는 Pod를 생성하고 관리한다.
 - 일반적으로 Pod는 자기 복구가 불가능 하기에 단독으로 배포를 하지 않는다.

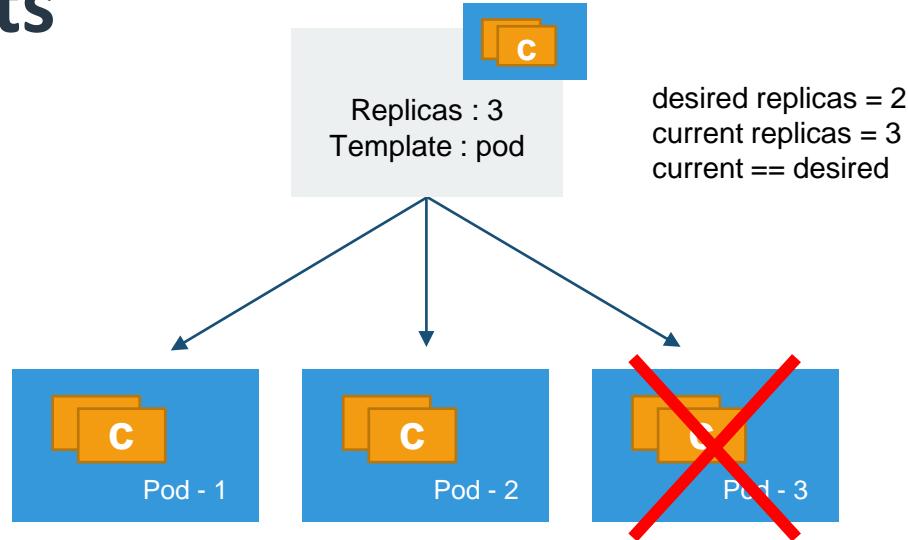


ReplicaSets



- ReplicaSet(rs)은 Replication Controller의 업그레이드 버전
- ReplicaSet은 equal 및 set 기반 Selector를 모두 지원하는 반면, Replication Controller는 equal기반 Selector만 지원

ReplicaSets



- 지정된 수의 Pod (Desired State)가 항상 실행되도록 보장
- ReplicaSets은 단독으로도 사용 가능하지만, 주로 Pod Orchestration에 사용(Pod creation, deletion, updates)
- Deployment가 ReplicaSets을 자동 생성하기 때문에 사용자는 관리에 신경 쓰지 않아도 됨

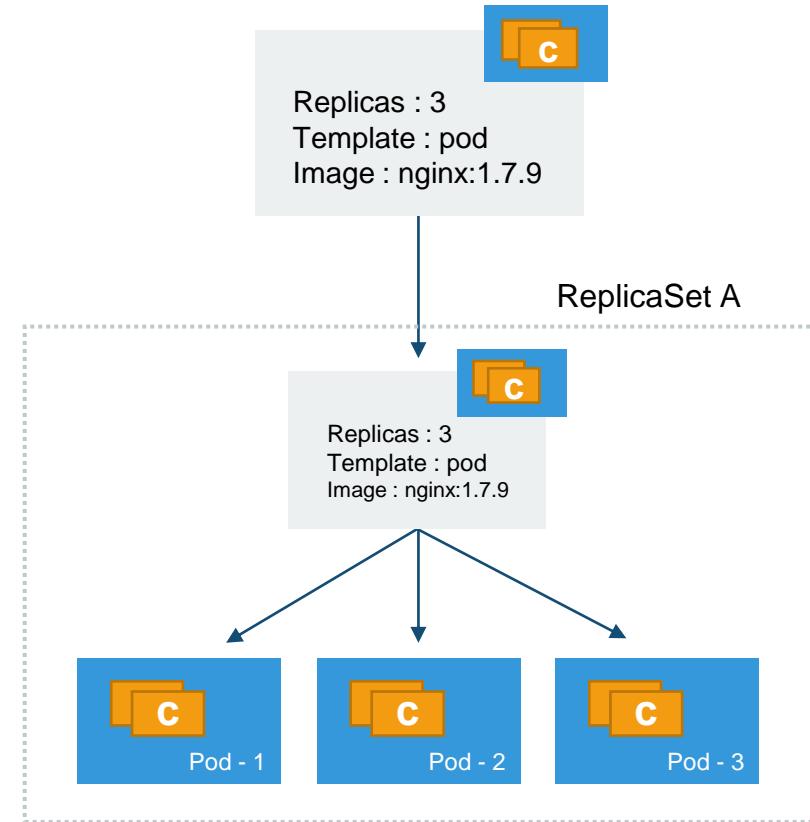
Lab. ReplicaSet

- ReplicaSet 파일 생성
 - `nano frontend.yaml`
- 파일을 기반으로 ReplicaSet 배포
 - `kubectl create -f frontend.yaml`
- ReplicaSet을 확인
 - `kubectl get pods`
 - `kubectl describe rs/frontend`
- ReplicaSet을 삭제
 - `kubectl delete rs/frontend`
(관련된 pod 전부 제거한다.)
 - `kubectl delete rs/frontend --cascade=false`
(ReplicaSet은 제거하지만 Pod는 유지)

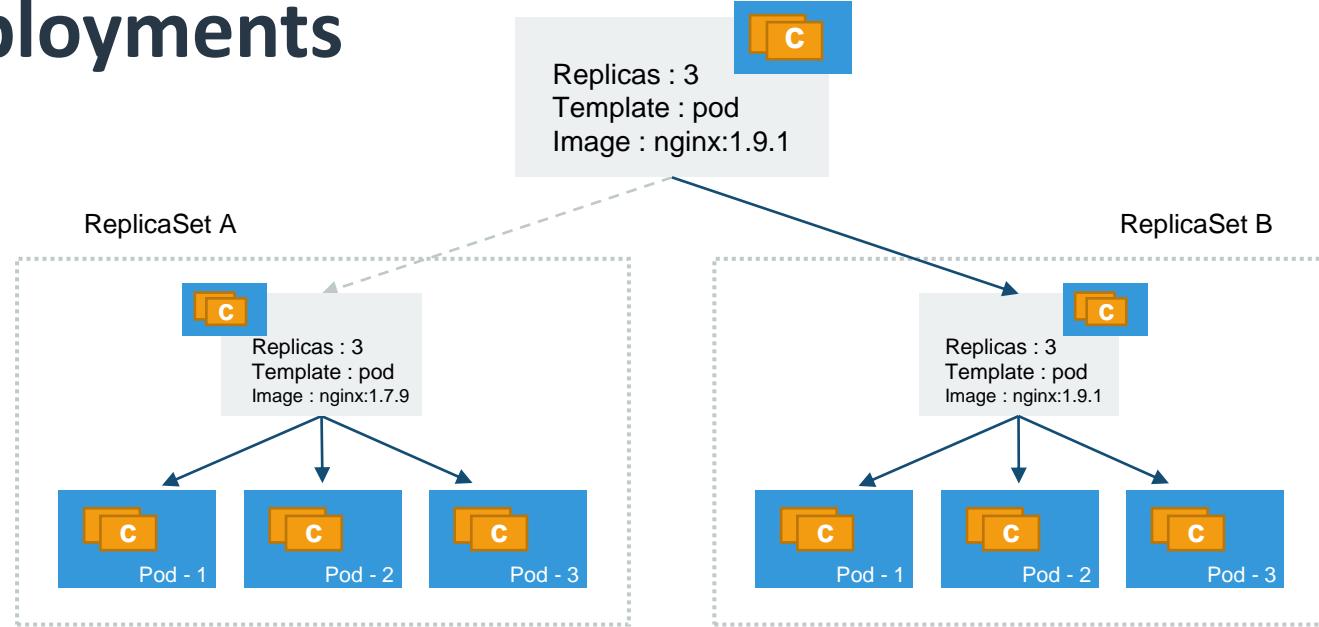
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
          ports:
            - containerPort: 80
```

Deployments

- Deployment 객체는 Pods와 ReplicaSets에 대한 선언적 업데이트를 제공한다.
- Deployment Controller는 Master node 컨트롤 관리자의 일부로 Desired state가 항상 만족이 되는지 확인한다.
- Deployment 가 ReplicaSet을 만들고 ReplicaSet은 그 뒤에 주어진 조건만큼의 Pod들을 생성한다.



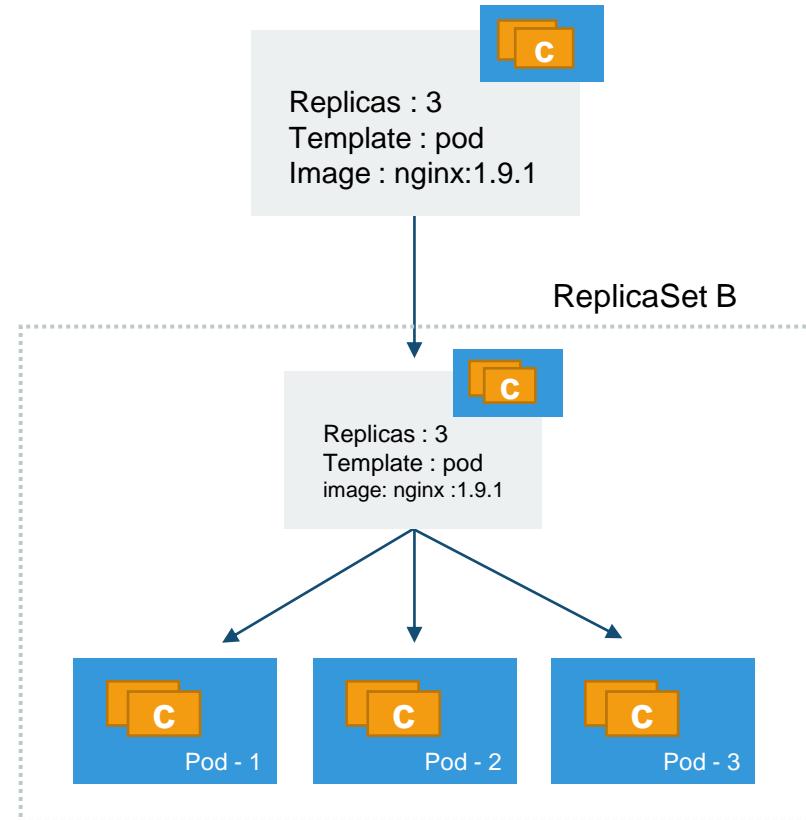
Deployments



- Deployment의 Pod template이 바뀌게 되면, 새로운 ReplicaSet이 생성되는데, 이를 **Deployment rollout**이라고 한다.
- Rollout은 Pod template에 변동이 생겼을 경우에만 동작하며, Scaling등의 작업은 ReplicaSet을 새로 생성하지 않는다.

Deployments

- 새로운 ReplicaSet이 준비되면 Deployment는 새로운 ReplicaSet을 바라본다.
- Deployment들은 Deployment recording등의 rollback 기능을 제공하며, 문제가 발생했을 경우, 이전 단계로 돌릴 수 있다.



Deployment 생성

- 설정 파일을 생성
 - nano nginx-deployment.yaml
- 파일을 기반을 배포
 - kubectl create -f nginx-deployment.yaml
- 생성된 deployment 확인
 - kubectl describe deployment nginx-deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

Pod 생성 시, 이 템플릿 참조
도커허브에서 nginx 1.7.9
이미지를 가져와 'nginx' 이름의
컨테이너 생성

Scaling Deployments

- Deployment의 replica의 개수를 확인한다.
 - `kubectl get pods`
- Deployment의 이름을 복사한다.
 - `kubectl get deployments`
- 해당 Deployment의 scale 조정
 - `kubectl scale deployments [deployment 이름] --replicas=3`
- 변경을 확인
 - `kubectl get pods`

Deployments 의 변경

- Deployment 파일을 변경
 - kubectl get deployments
 - nano nginx-deployment.yaml
(spec 아래 항목에 replicas: 5 속성 추가; 있으면 수정)
- 변경한 파일을 적용
 - kubectl apply -f nginx-deployment.yaml
- 변경 내용을 확인
 - kubectl get pods
- 설정파일에 추가된 replicas 속성을 삭제 후 다시 적용
 - Nano nginx-deployment.yaml (replicas)
 - kubectl apply -f nginx-deployment.yaml
 - kubectl delete pods --all
 - kubectl get pods

Rolling update

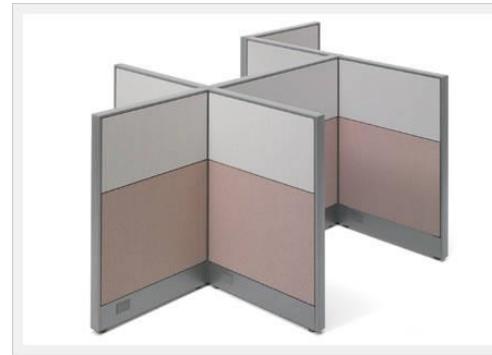
- 작동중인 pod와 deployments 확인
 - `kubectl get pods`
 - `kubectl get deployments`
- 새로운 버전의 deployments 배포 및 배포 상태 확인
 - `kubectl apply -f nginx-deployment.yaml`
 - `kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1`
 - `kubectl rollout status deployment/nginx-deployment`
- 변경 확인
 - `kubectl get deployments`
 - `kubectl get pods`
 - `kubectl describe pods [pod 이름]`
- Pod에 접속하여 확인
 - `kubectl exec -it podname -- /bin/bash`
 - `apt-get update`
 - `apt-get install curl`
 - `curl localhost`

Rollback

- 현재 실행중인 객체들을 확인
 - `kubectl get pods`
 - `kubectl get deployments -o wide` # deployment에 적용된 Image:버전 추가 표시
- 객체를 롤백 처리
 - `kubectl rollout undo deployment/nginx-deployment`
- 진행을 확인
 - `kubectl get deployments -o wide`

Namespaces

- Kubernetes는 동일 물리 클러스터를 기반으로 하는 복수의 가상 클러스터를 지원하는데 이들 가상 클러스터를 Namespace라고 한다.
- Namespace를 활용하면, 팀이나 프로젝트 단위로 클러스터 파티션을 나눌 수 있다.
- Namespace 내에 생성된 자원/객체는 고유하나, Namespace 사이에는 중복이 발생할 수 있다.



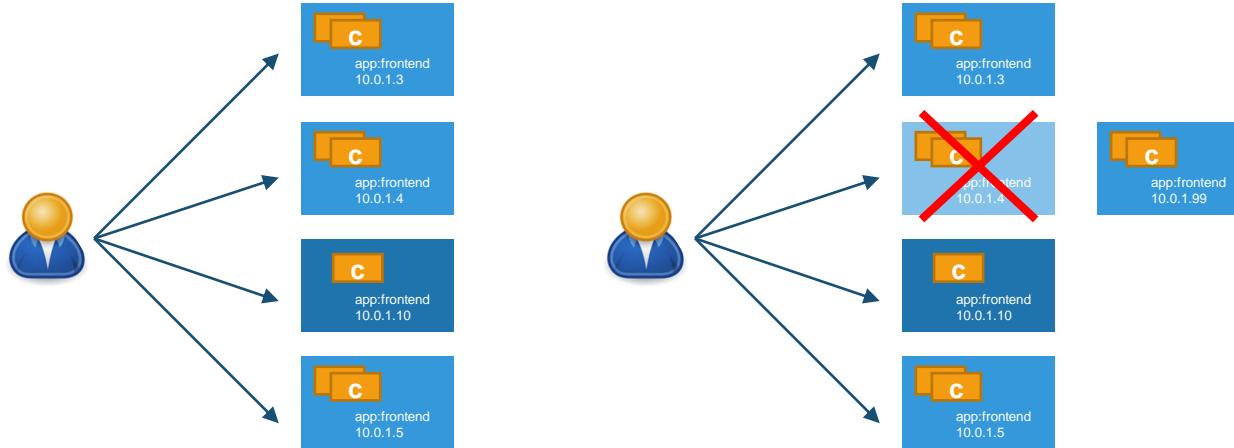
Namespaces

- Namespaces Object 조회

```
$ kubectl get namespaces
  NAME      STATUS   AGE
  default   Active   11h
  kube-public   Active   11h
  kube-system   Active   11h
```

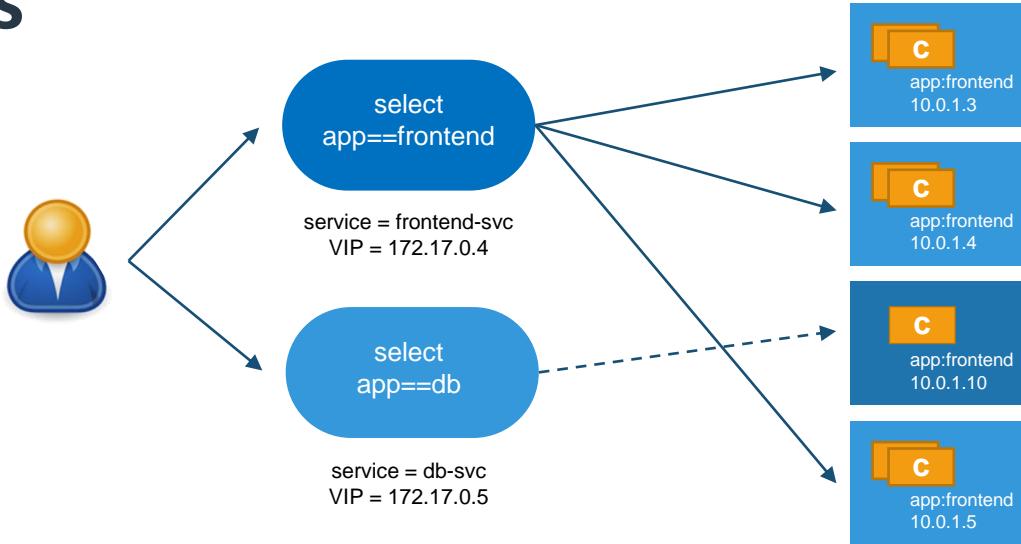
- Kubernetes는 처음에 3개의 초기 네임스페이스를 가진다.
 - default** : 다른 namespace를 갖는 다른 객체들을 가지고 있다.
 - kube-public** 은 클러스터 bootstrapping 같은 모든 유저가 사용 가능한 특별한 namespace 이다.
 - kube-system**: Kubernetes system에 의해서 생성된 객체를 가지고 있다.
- Resource Quotas를 사용하여 namespace 내에 존재하는 자원들을 나눌 수 있다.

Pod Access Issues



- 어플리케이션에 접근하기 위해서는, 사용자가 Pod에 접근해야 한다.
- Pod들은 언제든지 소멸 가능하기에 IP주소는 고정되어 있지 않다.
- 사용자가 직접 IP주소로 Pod에 연결되어 있을 때, Pod가 죽어서 새로 만들어지면 접속할 방법이 없다.
- 이 상황을 극복하기 위해서 추상화를 통해 Service라는 Pod들의 논리적 집단을 만들어 규칙을 설정하고 사용자들은 여기에 접속을 한다.

Services



- Selector를 사용하여 Pod를 논리적 그룹으로 나눌 수 있다.
- 각 논리적 그룹에 대해서 Service name이라는 이름을 부여할 수 있다.
- 사용자는 Service IP주소를 통해 Pod에 접속하게 된다.
 - 각 Service에 부여된 IP는 클러스터 IP 라고도 부른다.
- Service는 각 Pod에 대해 Load balancing을 자동으로 수행

Service Discovery

- Service는 클라이언트가 애플리케이션에 접근하는 Kubernetes의 채널로 런타임시, 이를 검색할 수 있는 방법이 필요
- DNS를 이용하는 방법
 - 서비스는 생성되면, [서비스 명].[네임스페이스명].svc.cluster.local이라는 DNS명으로 쿠버네티스 내부 DNS에 등록되고, 쿠버네티스 내부 클러스터에서는 이 DNS명으로 접근 가능한데, 이 때 DNS에서 리턴해 주는 IP는 외부 IP(External IP)가 아니라 Cluster IP(내부 IP) 임
- External IP를 명시적으로 지정하는 방법
 - 외부 IP는 Service의 Spec 부분에서 externalIPs 항목의 Value로 기술

ServiceType

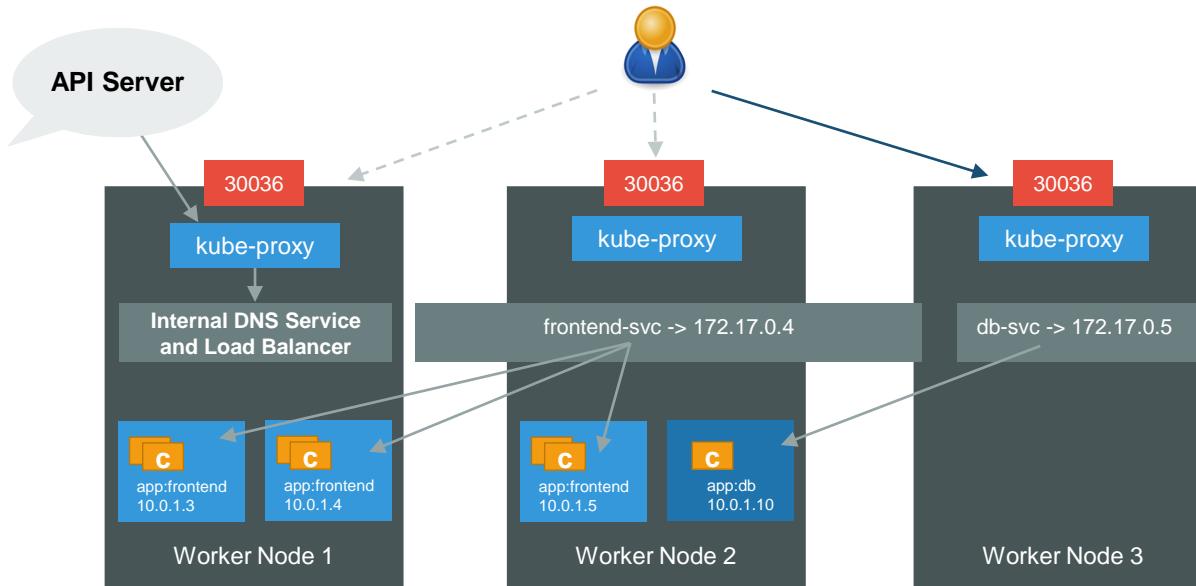
- Service를 정의할 때 Access Scope를 따로 정할 수 있다.
 - 클러스터 내에서만 접근이 가능한가?
 - 클러스터 내와 외부에서 접근이 가능한가?
 - 클러스터 밖의 리소스에 대한 Map을 가지는가?
- Service 생성 시, IP주소 할당 방식과 서비스 연계 등에 따라 4가지로 구분
 - ClusterIP
 - NodePort
 - LoadBalancer
 - ExternalName

ServiceType : ClusterIP

- 디폴트 설정으로 서비스에 클러스터 ip를 할당
- 쿠버네티스 클러스터 내에서만 이 서비스에 접근 가능
- 외부에서는 외부 IP를 할당 받지 못했기 때문에 접근이 불가능

ServiceType : NodePort

- 고정 포트(NodePort)로 각 노드의 IP에 서비스를 노출
- Cluster IP 뿐만 아니라, 노드의 IP와 포트를 통해서도(<NodeIP>:<NodePort>) 접근 가능

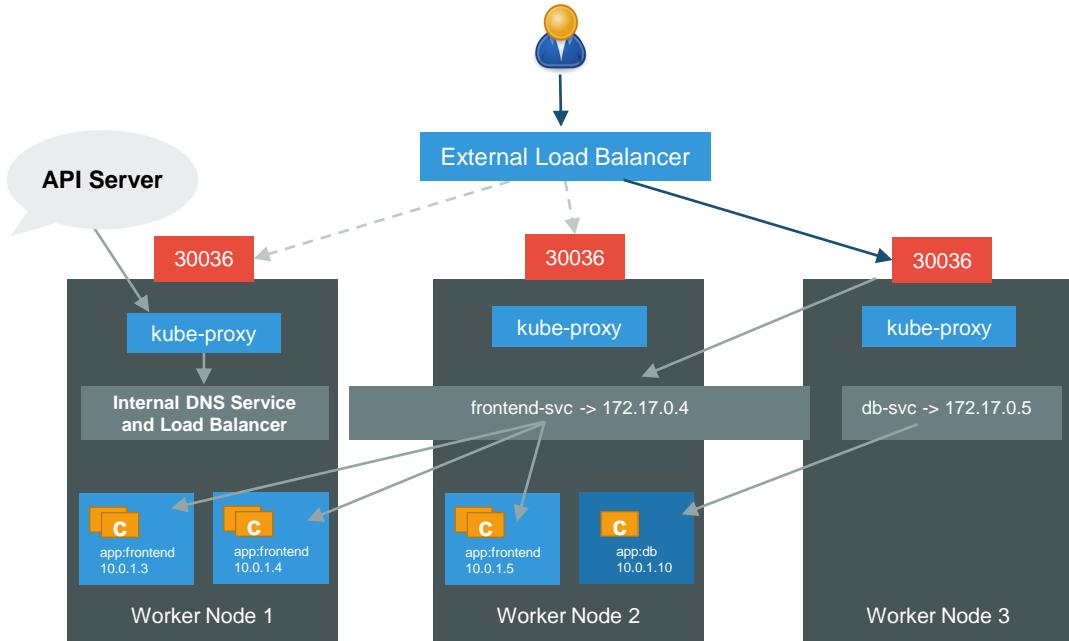


ServiceType : LoadBalancer

- 클라우드 밴더의 로드밸런싱 기능을 사용
 - NodePort와 ClusterIP Service들은 자동으로 생성되어 External Load Balancer가 해당 포트로 라우팅
 - Service들은 각 Worker node에서 Static port로 노출

LoadBalancer ServiceType은 기본 인프라가 Load balancer의 자동 생성을 제공하고, Kubernetes를 지원 할 경우에만 작동

Ex) Azure, Google Cloud Platform, AWS



Ingress

- Service는 L4 레이어로 TCP레벨에서 Pod를 로드밸런싱 함
- MSA에서는 Service 하나가 MSA의 서비스로 표현되는 경우가 많고 서비스는 하나의 URL(/orders, /products, ...)로 대표되는 경우가 많다.
- MSA 서비스간 라우팅을 위해 API Gateway를 두는 경우가 많은데 관리포인트가 생김
- URL기반의 라우팅 정도라면 L7 로드밸런서 정도로 위의 기능을 충족
- Kubernetes에서 제공하는 L7 로드밸런싱 컴포넌트를 ‘Ingress’ 라고 함

kubernetes.io의 의하면,

"An Ingress is a collection of rules that allow inbound connections to reach the cluster Services."

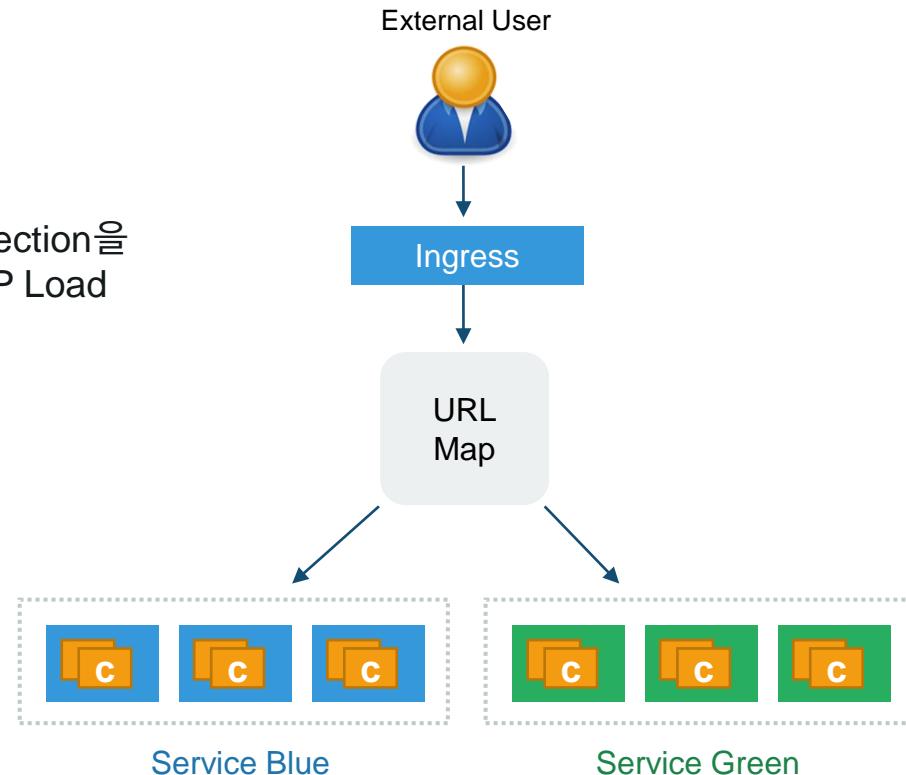
"Ingress는 인바운드 연결이 클러스터의 Service에게
라우팅되도록 하는 규칙의 집합체이다."



Ingress

- 아래와 같은 Service들의 Inbound Connection을 지원하기 위해 Ingress는 Layer7의 HTTP Load balancer 기능 제공

- TLS (SSL)
- Name-based virtual hosting
- Path-based routing
- Custom rules

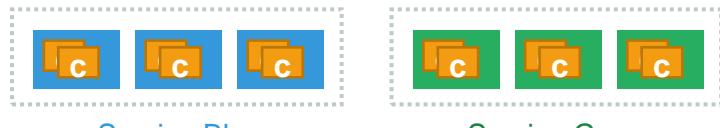


Ingress

URL Path based

example.com/blue
example.com/green

Ingress Controller



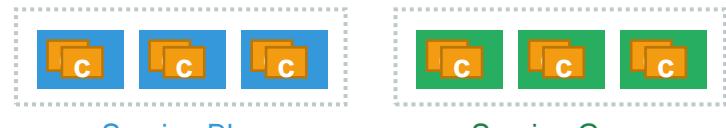
Service Blue

Service Green

Virtual Hosting

blue.example.com
green.example.com

Ingress Controller



Service Blue

Service Green

- 사용자들은 직접 Service에 접속하지 않는다.
- 유저는 Ingress에 먼저 접근하고, 요청은 해당 Service로 포워드 된다.
- Ingress 요청은 Ingress Controller에 의해 처리된다.

Ingress Controller

- "Ingress Controller"는 Ingress 리소스의 변경 사항을 마스터 노드의 API 서버에서 감시하고, 그에 따라 Layer 7로드 밸런서를 업데이트하는 응용 프로그램이다.
- Ingress Controller는 오픈소스 기반 구현체 및 클라우드 벤더사가 직접 구현체를 개발해 사용하기도 함
 - Nginx Ingress Controller, KONG, GCE L7 Load Balancer

Ingress Routing

- Host-based Routing
 - 사용자가 blue.example.com 와 green.example.com에 접근을 하게 되면 같은 Ingress endpoint에서,
 - 각각 nginx-blue-svc와 nginx-green-svc로 요청이 포워드

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
  namespace: ingress-basic
spec:
  rules:
    - host: blue.example.com
      http:
        paths:
          - backend:
              serviceName: nginx-blue-svc
              servicePort: 80
    - host: green.example.com
      http:
        paths:
          - backend:
              serviceName: nginx-green-svc
              servicePort: 80
```

<Host-based routing>

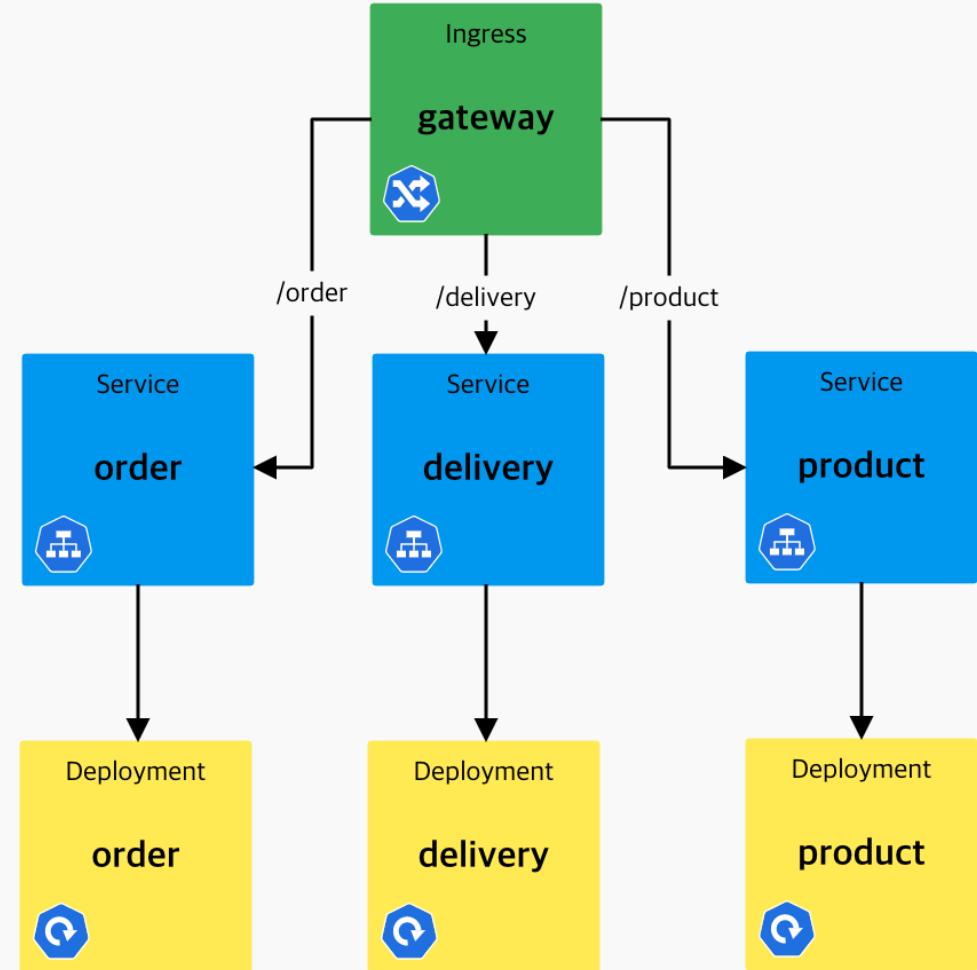
Ingress Routing

- Path-based Routing
 - Ingress는 또한 example.com/blue 와 example.com/green 형태의 요청에 대해,
 - 각각 nginx-blue-svc와 nginx-green-svc로 요청이 라우팅

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: web-ingress
  namespace: ingress-basic
spec:
  rules:
    - http:
        paths:
          - path: /blue/*
            backend:
              serviceName: nginx-blue-svc
              servicePort: 80
          - path: /green/*
            backend:
              serviceName: nginx-green-svc
              servicePort: 80
```

<Path-based routing>

Lab. Deploying 12-Street





여우야 여우야 뭐하니?

밥 먹는다

무슨 반찬?

개구리 반찬

살았니 죽었니?
살았다 야!

Live ness Probes & Readiness Probes

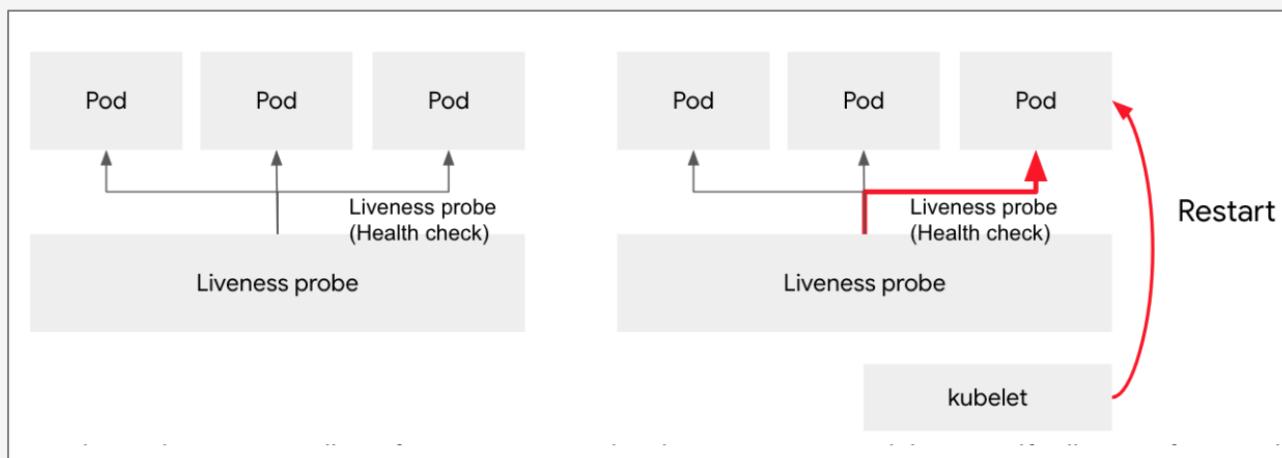
- 쿠버네티스는 각 컨테이너의 상태를 주기적으로 체크(Health Check)해서,
 - 문제가 있는 컨테이너를 자동으로 재시작하거나 또는 문제가 있는 컨테이너를 서비스에서 제외 한다.
- Liveness와 Readiness Probes은 kubelet이 pod내에서 실행되는 어플리케이션의 health를 조정하기 때문에 매우 중요하다.

Probe Types

- Liveness probe와 readiness probe는 컨테이너가 정상적인지 아닌지를 체크하는 방법으로 다음과 같이 3가지 방식을 제공한다.
 - Command probe
 - HTTP probe
 - TCP probe

Liveness Probes

- Pod는 정상적으로 작동하지만 내부의 어플리케이션이 반응이 없다면, 컨테이너는 의미가 없다.
 - 위와 같은 경우는 어플리케이션의 Deadlock 또는 메모리 과부화로 인해 발생할 수 있으며, 발생했을 경우 컨테이너를 다시 시작해야 한다.
- Liveness probe는 Pod의 상태를 체크하다가, Pod의 상태가 비정상인 경우 kubelet을 통해서 재시작한다.



Liveness Command probe

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/busybox
      args:
        - /bin/sh
        - -c
        - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
          - cat
          - /tmp/healthy
      initialDelaySeconds: 3
      periodSeconds: 5
```

- 왼쪽은 /tmp/healthy 파일이 존재하는지 확인하는 설정파일이다.
- periodSeconds 파라미터 값으로 5초마다 해당 파일이 있는지 조회한다.
- initialDelaySeconds 파라미터는 kubelet이 첫 체크하기 전에 기다리는 시간을 설정한다.
- 파일이 존재하지 않을 경우, 정상 작동에 문제가 있다고 판단되어 kubelet에 의해 자동으로 컨테이너가 재시작 된다.

Liveness HTTP probe

- Kubelet이 HTTP GET 요청을 /healthz로 보낸다.
- 실패 했을 경우, kubelet이 자동으로 컨테이너를 재시작 한다.

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 8080  
    httpHeaders:  
      - name: X-Custom-Header  
        value: Awesome  
    initialDelaySeconds: 3  
    periodSeconds: 3
```

Liveness TCP Probe

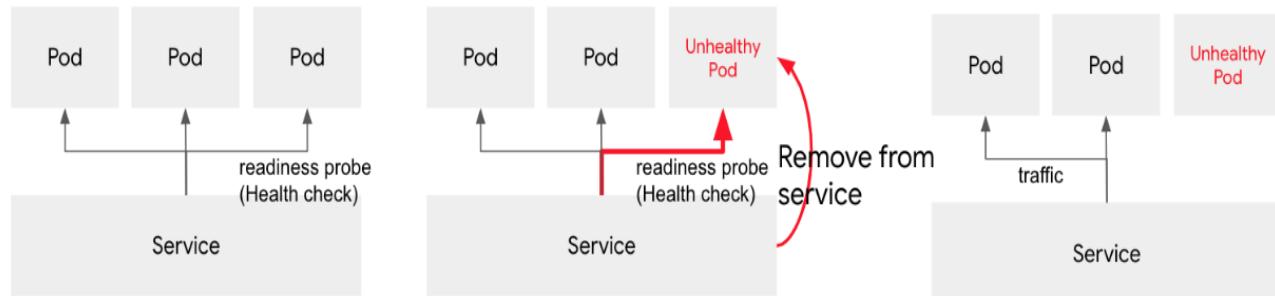
- kubelet은 TCP Liveness Probe를 통해, 지속적으로 어플리케이션이 실행중인 컨테이너의 TCP Socket을 열려고 한다.
- 정상이 아닌 경우 컨테이너를 재시작 한다.

```
livenessProbe:  
  tcpSocket:  
    port: 8080  
  initialDelaySeconds: 15  
  periodSeconds: 20
```

Readiness Probes

- Configuration을 로딩하거나, 많은 데이터를 로딩하거나, 외부 서비스를 호출하는 경우에는 일시적으로 서비스가 불가능한 상태가 될 수 있다.
- Readiness Probe를 사용하게 되면 주어진 조건이 만족할 경우, 서비스 라우팅하고, 응답이 없거나 실패한 경우, 서비스 목록에서 제외

```
readinessProbe:  
  exec:  
    command:  
      - cat  
      - /tmp/healthy  
  initialDelaySeconds: 5  
  periodSeconds: 5
```



Lab. Liveness, Readiness

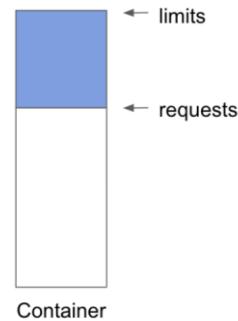


Lab Time

- Lab Script Location
 - Workflowy :

Resource Assign & Management

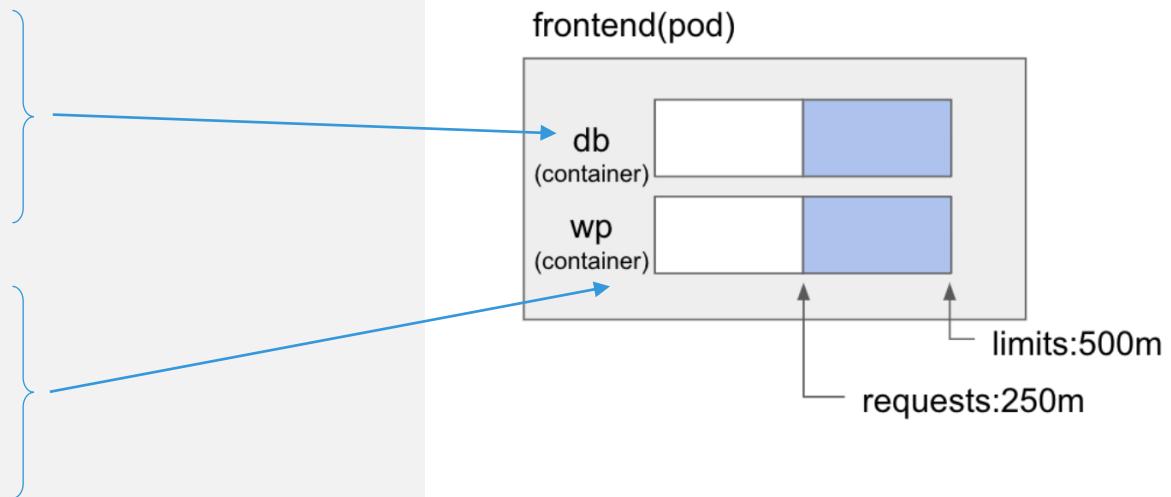
- 쿠버네티스에서 Pod를 어느 노드에 배포할지 결정하는 것을 스케줄링이라 함
- Pod에 대한 스케줄링시에, Pod내의 애플리케이션이 동작할 수 있는 자원(CPU,메모리 등) 정보를 알아야 그만한 자원이 가용한 노드에 Pod 배포 가능
- 리소스 단위
 - CPU의 경우 ms(밀리 세컨드)를 사용하는데 대략 1000ms가 1 vCore (가상 CPU 코어)
 - 메모리의 경우 Mb를 사용하며 64M(64×1000), 또는 64Mi (64×1024)로 계산
 - Request & Limit
 - Request : 컨테이너가 생성될 때 요청하는 리소스 양
 - Limit : 리소스가 더 필요한 경우 추가로 사용 가능한 양



Resource Assign & Management

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: db
      image: mysql
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
    - name: wp
      image: wordpress
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

- 샘플 설정에 따른 Pod내 CPU 리소스 할당



Resource Monitoring (1/2)

- Node의 자원 상태 모니터링을 위한 Metric Server(메트릭 서버) 설치
 - 설치 Commands

```
git clone https://github.com/kubernetes-incubator/metrics-server.git  
cd metrics-server/  
kubectl create -f deploy/kubernetes/
```

- 설치 확인

```
apexacme@APEXACME:~$ kubectl get all -l 'k8s-app in (metrics-server)' -n kube-system  
NAME                 READY   STATUS    RESTARTS   AGE  
pod/metrics-server-7668599459-9zxb9  1/1     Running   0          2d3h  
  
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE  
deployment.apps/metrics-server  1/1     1           1          2d3h  
  
NAME                      DESIRED   CURRENT   READY   AGE  
replicaset.apps/metrics-server-7668599459  1         1         1        2d3h
```

Resource Monitoring (2/2)

- Node의 자원 상태 모니터링

- \$ kubectl get nodes
- \$ kubectl describe nodes

Namespace	Name	CPU Requests	CPU Limits	Memory Requests	Memory Limits	AGE
ingress-basic	nginx-ingress-controller-5b85669986-g4n8t	0 (0%)	0 (0%)	0 (0%)	0 (0%)	5h14m
ingress-basic	nginx-ingress-default-backend-6b8dc9d88f-1bd9g	0 (0%)	0 (0%)	0 (0%)	0 (0%)	5h14m
kube-system	kube-proxy-47g4j	100m (5%)	0 (0%)	0 (0%)	0 (0%)	2d2h
kube-system	omsagent-8cr5p	75m (3%)	150m (7%)	225Mi (4%)	600Mi (13%)	7d
kube-system	tiller-deploy-7b98f7c844-t7cpn	0 (0%)	0 (0%)	0 (0%)	0 (0%)	6h55m
kube-system	tunnelfront-c8df6fcf-c7xz	10m (0%)	0 (0%)	64Mi (1%)	0 (0%)	7d
Allocated resources:						
(Total limits may be over 100 percent, i.e., overcommitted.)						
Resource	Requests	Limits				
cpu	185m (9%)	150m (7%)				
memory	289Mi (6%)	600Mi (13%)				
ephemeral-storage	0 (0%)	0 (0%)				
attachable-volumes-azure-disk	0	0				
Events:						
<none>						

- 현재 사용 중인 리소스 현황 모니터링

- \$ kubectl top nodes
- \$ kubectl top pods

Lab.

- Readiness and Zero Down-time Deploy
- Liveness and Self-healing
- HPA Autoscaling with Metric-service and Resource setting

“

Setup k8s Client & Creating Cluster

- Install Google Cloud SDK and Kubernetes Client
- Create GCP Project & set GCloud Config
- Create Cluster & fetch Credentials
- Configure Google Container Registry

Install Google Cloud SDK (Linux)

- # Add the Cloud SDK distribution URI as a package source
 - echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] http://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list
- # Import the Google Cloud public key
 - curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --keyring /usr/share/keyrings/cloud.google.gpg add -
- # Update the package list and install the Cloud SDK
 - sudo apt-get update && sudo apt-get install google-cloud-sdk

Initialize GCloud SDK

- # Initialize SDK

- gcloud init

```
[1] Re-initialize this configuration [gcp] with new settings
[2] Create a new configuration
[3] Switch to and re-initialize existing configuration: [default]
```

- # Configuration 항목 : GCP계정 설정, Project 설정, Region/ Zone 지정

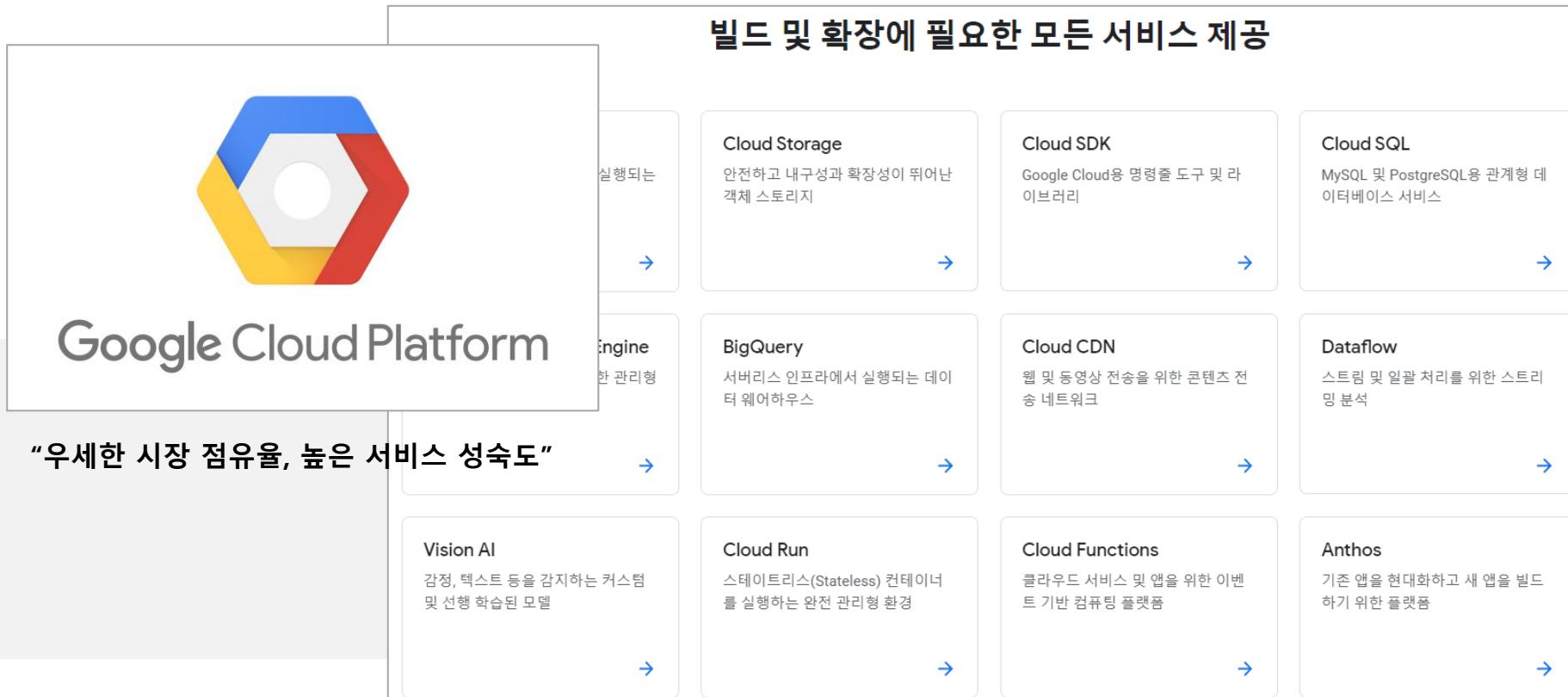
- # Verify Client Info

- gcloud auth list : 사용자 인증정보 보기
 - gcloud config list : SDK 구성속성 보기
 - gcloud info : GCloud SDK 설치정보 보기

Install Kubernetes Client

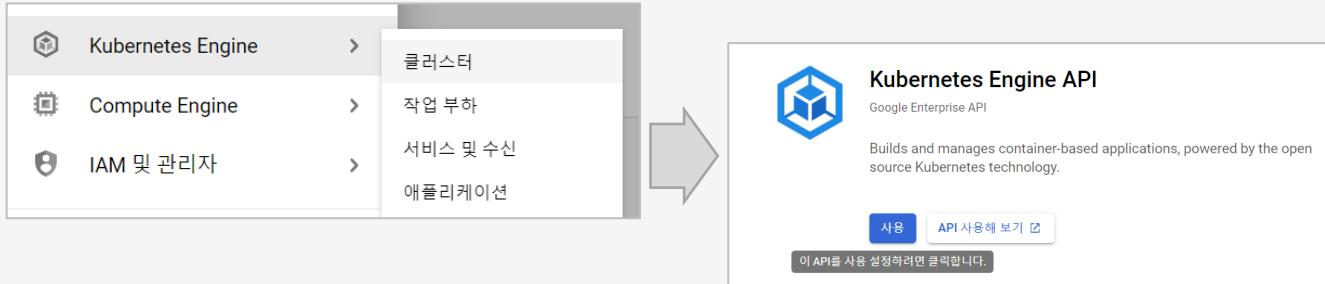
- 최신 릴리즈 다운로드
 - curl -LO "https://dl.k8s.io/release/\$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
- 설치
 - sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
- 설치 확인
 - kubectl version --client

Google GCP



Create GCP Project & Set Config

- Create Project & Set GCP Config
 - `gcloud projects create [PROJECT-NAME]`
 - `gcloud projects list` : 생성한 프로젝트 확인
 - `gcloud config set compute/zone asia-northeast3-a` : 한국/서울로 Zone 설정
 - `gcloud config set project [PROJECT-NAME]` : 생성한 project 설정
 - `gcloud config list` : 설정한 Config 확인
- 생성 후, GCP 관리콘솔에서 Kubernetes Engine API를 ‘사용’으로 변경



Create Cluster & fetch Credentials

- Create k8s Cluster on the Project

- `gcloud container clusters create [CLUSTER-NAME] --num-nodes 3 --machine-type n1-standard-2` : n1-standard-2 타입 워크노드 3개 생성
- `gcloud container clusters list` : 생성 클러스터 확인
- `gcloud container clusters get-credentials [CLUSTER-NAME]` : Cluster token 가져오기
- `kubectl config current-context` : Kubernetes Context 확인
- `kubectl get all` : cluster 객체 확인

Configure Google Container Registry

- Authorizing GCR
 - gcloud auth configure-docker
- Image Test
 - docker build -t gcr.io/PROJECT-NAME/IMAGE-NAME:v1 .
 - docker push gcr.io/PROJECT-NAME/IMAGE-NAME:v1
 - gcloud container images list --repository=gcr.io/PROJECT-NAME : 이미지 확인

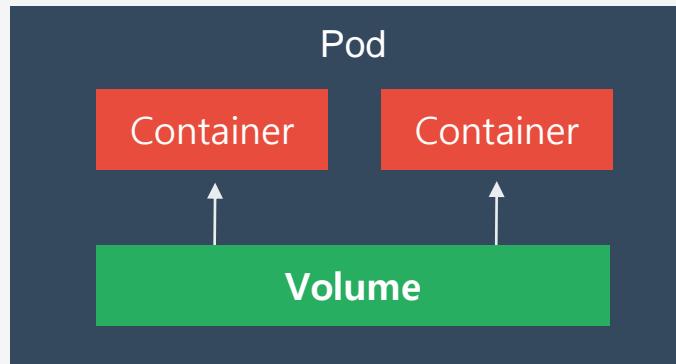
Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations 
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

Volumes

- 쿠버네티스는 여러 호스트에 걸쳐 Stateless한 컨테이너를 마이크로서비스로 배포하는 것이 목표이기에 영속성 있는 저장장치(Persistent Volume)를 고려해야 함



- Volume은 Pod에 장착되어, 그 Pod에 있는 Container 간에 공유

Types of Volumes

- Pod에 마운트된 디스크는 Volume type에 따라 사용 유형이 정의
- Volume Type 내 디스크의 크기, 내용 등의 속성 설정
- Types of Volumes

임시 볼륨	로컬 볼륨	네트워크 볼륨	네트워크 볼륨 (Cloud dependent)
emptyDir	hostPath	gitRepo, iSCSI, NFS cephFS, glusterFS	gcePersistentDisk, AzureDisk, Amazon EFS, Amazon EBS ...
✓ Pod내 컨테이너간 공유	✓ Host 디렉토리를 Pod와 공유 해 사용하는 방식	✓ 영구적으로 영속성 있는 데이터 관리 목적	
✓ Pod가 삭제되면, emptyDir도 지워지므로 휘발성 데이터 저장 용도	✓ 컨테이너에 nodeSelector를 지정안하면 매번 다른 호스 트에 할당 (e.g. 호스트의 Metric 수집해야 하는 경우)	✓ 쿠버네티스는 PV와 PVC의 개념을 통해 Persistent 볼 륨을 Pod에 제공 ✓ gitRepo, iSCSI, NFS와 같은 표준 네트워크 볼륨과 Cloud Vendor가 제공하는 볼륨으로 구분	

Volumes : emptyDir

```
apiVersion: v1
kind: Pod
metadata:
  name: shared-volumes
spec:
  containers:
    - image: redis
      name: redis
      volumeMounts:
        - name: shared-storage
          mountPath: /data/shared
    - image: nginx
      name: nginx
      volumeMounts:
        - name: shared-storage
          mountPath: /data/shared
  volumes:
    - name: shared-storage
      emptyDir: {}
```

- emptyDir의 생명주기는 컨테이너 단위가 아닌 Pod 단위로 Container 재기동에도 계속 사용 가능

- 생성된 Pod 확인

```
apexacme@APEXACME: ~/yaml$ kubectl get all
NAME           READY   STATUS    RESTARTS   AGE
pod/shared-volumes   2/2     Running   0          10m
```

- 지정 컨테이너 접속 후, 파일 생성
 - kubectl exec -it shared-volumes --container redis -- /bin/bash
 - cd /data/shared
 - echo test... > test.txt
- 다른 컨테이너로 접속 후, 파일 확인
 - kubectl exec -it shared-volumes --container nginx - /bin/bash
 - cd /data/shared
 - ls

Volumes : hostPath

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: somepath
          mountPath: /data/shared
  volumes:
    - name: somepath
      hostPath:
        path: /tmp
      type: Directory
```

- Node의 Local 디스크 경로를 Pod에 마운트
- 같은 hostPath에 있는 볼륨은 여러 Pod사이에서 공유
- Pod가 삭제되어도 hostPath에 있는 파일은 유지
- Pod가 재기동 되어 다른 Node에서 기동될 경우, 새로운 Node의 hostPath를 사용
- Node의 로그 파일을 읽는 로그 에이전트 컨테이너 등에 사용 가능
- Pod 생성 및 확인 (Pod 내, ls -al /data/shared)

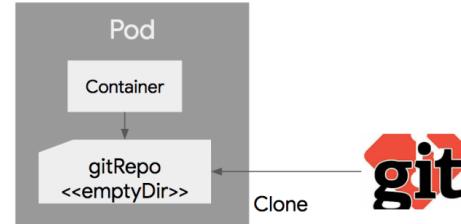
```
drwxrwxrwt 8 redis root 4096 Feb 17 03:08 .
drwxr-xr-x 3 redis redis 4096 Feb 17 03:07 ..
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .ICE-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .Test-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .X11-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .XIM-unix
drwxrwxrwt 2 redis root 4096 Feb 11 05:39 .font-unix
drwx----- 3 redis root 4096 Feb 11 05:44 systemd-private-fe55104f60e34b2ea4
```

Volumes example : gitRepo

```
apiVersion: v1
kind: Pod
metadata:
  name: gitrepo-volume-pod
spec:
  containers:
    - image: nginx:alpine
      name: web-server
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
          readOnly: true
    ports:
      - containerPort: 80
        protocol: TCP
  volumes:
    - name: html
      gitRepo:
        repository: https://github.com/luksa/kubia-website-example.git
        revision: master
        directory: .
```

- Pod 생성시 지정된 Git 리파지토리의 특정 리비전을 Cloning하여 디스크 볼륨 생성

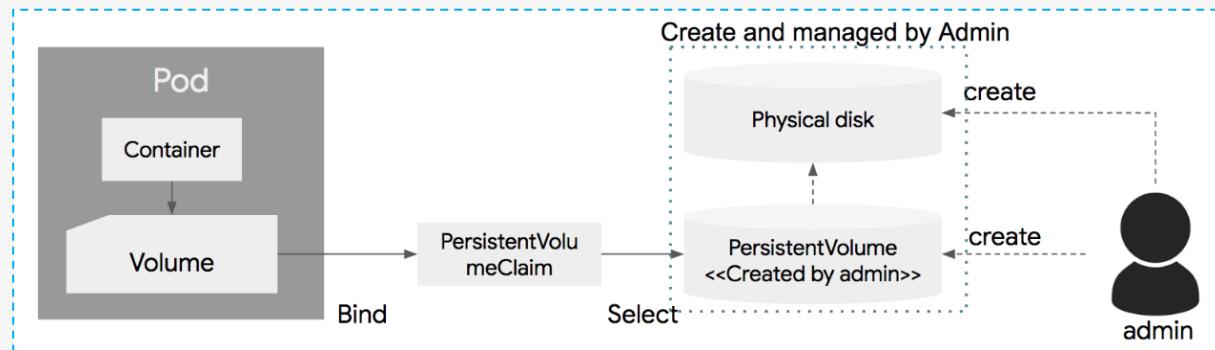
- 물리적으로는 emptyDir이 생성되고 Git Clone 수행



- HTML 같은 정적 파일 및 Nodejs 같은 스크립트 기반 코드 배포에 유용

PersistentVolume & PersistentVolumeClaim

- 특정 IT 환경에서는 영속성 있는 대용량 스토리지는 관리자에 의해 관리
 - 쿠버네티스 클러스터를 사용하는 개발자로부터 볼륨 프로비저닝 역할을 분리하는 사상
- 시스템 관리자가 실제 물리 디스크를 생성한 뒤, 이 디스크를 PersistentVolume 이라는 이름으로 Kubernetes에 등록
- 개발자는 Pod 생성 시, 볼륨을 정의하고, 해당 볼륨의 정의 부분에 PVC(PersistentVolumeClaim)를 지정하여 관리자가 생성한 PV와 연결



StorageClass - Dynamic PV Provisioning



- PV는 관리자에 의해 수동으로 생성될 수 있지만, 자동 생성도 가능(Dynamic Provisioning)
- StorageClass(SC) Object
- StorageClass 객체에 의해 PersistentVolumes 동적 제공 가능
 - StorageClass는 PersistentVolume를 만들기 위해 Cloud Provider별 CSI 인터페이스를 구현하여 제공
- PersistentVolumes 스토리지 관리를 제공하는 Volume Types :
 - GCEPersistentDisk, AWSElasticBlockStore, AzureDisk, NFS, iSCSI

GCP Storage Provisioner

- Default GCP StorageClass 확인 (총 3개의 프로비저너)

```
apexacme@APEXACME:~$ kubectl get sc
NAME          PROVISIONER           RECLAIMPOLICY  VOLUMEBINDINGMODE
premium-rwo   pd.csi.storage.gke.io  Delete         WaitForFirstConsumer
standard (default) kubernetes.io/gce-pd Delete         Immediate
standard-rwo   pd.csi.storage.gke.io  Delete         WaitForFirstConsumer
```

- Storage 접근모드에 따른 GCP 스토리지 클래스

접근 모드	스토리지 클래스	설명
ReadWriteOnce (RWO)	✓ premium-rwo (SSD) ✓ standard-rwo	✓ The volume can be mounted as read-write by a single node.
ReadWriteMany (RWX)	✓ premium-rwx (SSD) ✓ Standard-rwx	✓ The volume can be mounted as read-write by many nodes.
ReadOnlyMany (ROX)	✓ ""	✓ The volume can be mounted as read-only by many nodes. ✓ Specify "" as the storageClassName so it matches the PersistentVolume's StorageClass. # A nil storageClassName value uses the default StorageClass.

GCP Storage Provisioner

- CSI 드라이버 탑 및 설치(제거)

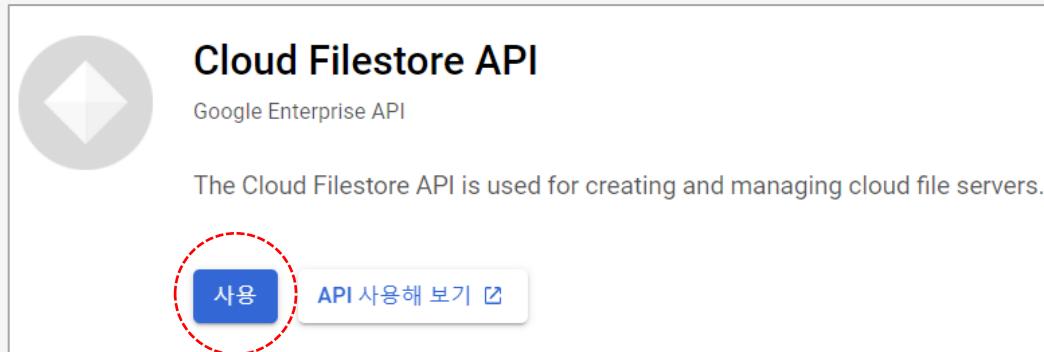
CSI 드라이버	StorageClass	클러스터에 드라이버 사용설정 <i>GKE version 1.21 or later required</i>
Compute Engine 영구 디스크 CSI	✓ premium-rwo ✓ standard-rwo	✓ gcloud container clusters update CLUSTER-NAME --update-addons=GcePersistentDiskCsiDriver=ENABLED(DISABLED) --zone ZONE-NAME
FileStore CSI	✓ premium-rwx ✓ Standard-rwx	✓ gcloud container clusters update CLUSTER_NAME --update-addons=GcpFilestoreCsiDriver=ENABLED(DISABLED) --zone ZONE-NAME

- FileStore CSI 설치 후, StorageClass 확인 (총 5개의 프로비저너)

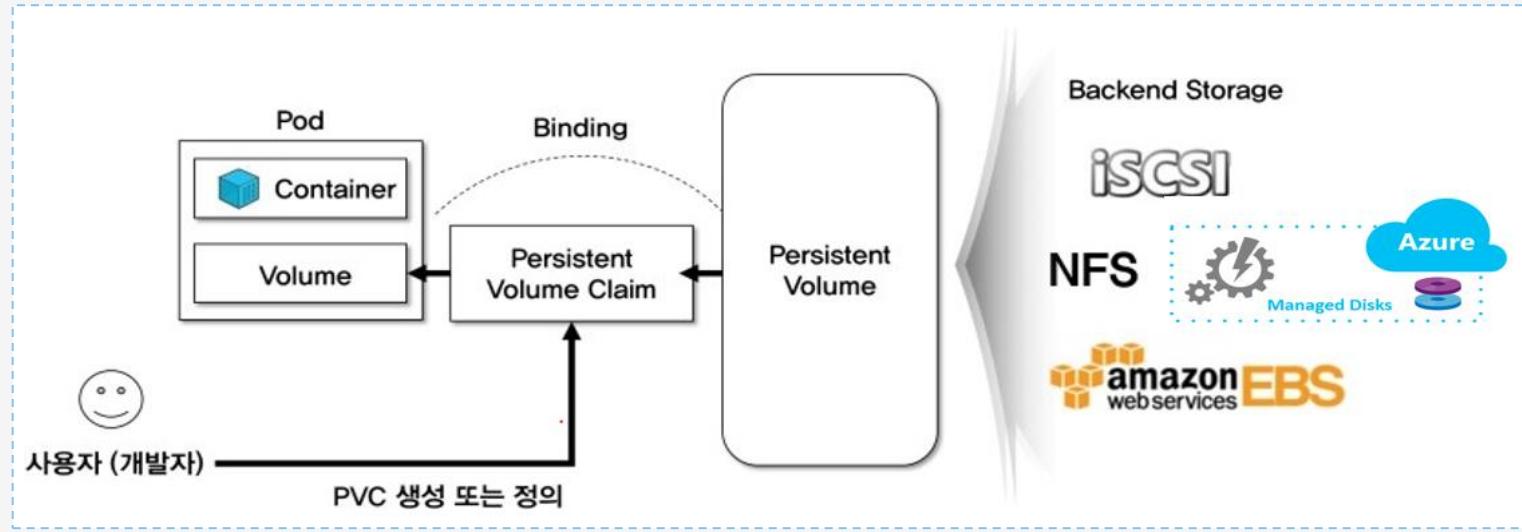
```
apexacme@APEXACME:~$ kubectl get sc
NAME          PROVISIONER           RECLAIMPOLICY   VOLUMEBINDINGMODE
premium-rwo   pd.csi.storage.gke.io Delete          WaitForFirstConsumer
premium-rwx   filestore.csi.storage.gke.io Delete          WaitForFirstConsumer
standard (default) kubernetes.io/gce-pd Delete          Immediate
standard-rwo   pd.csi.storage.gke.io Delete          WaitForFirstConsumer
standard-rwx   filestore.csi.storage.gke.io Delete          WaitForFirstConsumer
```

GCP Storage Provisioner

- FileStore CSI 설치 후, GCP 콘솔에서 Cloud Filestore API를 검색
- 사용중인 Project가 Cloud Filestore API를 사용하도록 설정



PersistentVolume Claims & Binding



- Pod가 크기, 접근 모드에 따라 PVC를 요청, 적합한 PersistentVolume 발견시 PersistentVolume Claim에 바인딩
- PVC 조건을 만족하는 PV가 없을 경우, PV를 StorageClass가 자동으로 Provisioning하여 바인딩

Create PersistentVolumeClaim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: podpvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: standard-rwx
  resources:
    requests:
      storage: 100Mi
```

Save as volume-pvc.yaml

- accessMode:
 - ReadWriteOnce : 하나의 Pod에만 마운트되고, 읽고 쓰기 가능
 - ReadOnlyMany : 여러 개의 Pod에서 마운트되고, 동시에 읽기만 가능 (쓰기는 불가능)
 - ReadWriteMany : 여러 개의 Pod에서 마운트되고, 동시에 읽고 쓰기 가능

- kubectl apply -f volume-pvc.yaml
- kubectl get pvc

```
apexacme@APEXACME:~$ kubectl get pvc -w
NAME      STATUS      VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
podpvc   Pending
```

- kubectl describe pvc

```
apexacme@APEXACME:~$ kubectl describe pvc podpvc
Name:         podpvc
Namespace:    default
StorageClass: standard-rwx
Status:       Bound
Volume:       pvc-d039c3ad-79d0-4fee-a310-6e747788ebb8
Labels:        <none>
Annotations:  pv.kubernetes.io/bind-completed: yes
              pv.kubernetes.io/bound-by-controller: yes
              volume.beta.kubernetes.io/storage-provisioner: filestore.csi.storage.gke.io
              volume.kubernetes.io/selected-node: gke-msaschool-education--default-pool-e2220b03-0x68
Finalizers:   [kubernetes.io/pvc-protection]
Capacity:     1Ti
Access Modes: RWX
VolumeMode:   Filesystem
Used By:      nginx-deployment-5fb7cb76d9-8c9z4
              nginx-deployment-5fb7cb76d9-jhbnd
              nginx-deployment-5fb7cb76d9-wjh6t
```

Create Pod with PersistentVolumeClaim

```
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
          volumeMounts:
            - mountPath: /usr/share/nginx/html
              name: mypvc
          volumes:
            - name: mypvc
          persistentVolumeClaim:
            claimName: podpvc
```

Save as pod-with-pvc.yaml

- kubectl apply -f pod-with-pvc.yaml
- kubectl get pod
- kubectl exec -it pod/nginx-deployment-*HashCode* -- /bin/bash
- df -k 로 PVC 마운트 확인

```
root@nginx-deployment-5fb7cb76d9-8c9z4:/usr/share/nginx/html# df -k
Filesystem      1K-blocks   Used   Available  Use% Mounted on
overlay          98868448 3094552  95757512   4% /
tmpfs             65536       0    65536   0% /dev
tmpfs             2015812      0   2015812   0% /sys/fs/cgroup
shm               65536       0    65536   0% /dev/shm
/dev/sdal         98868448 3094552  95757512   4% /etc/hosts
10.45.80.170:/vol1_1055763456 0 1002059776  0% /usr/share/nginx/html
tmpfs             2015812      12   2015800   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs             2015812      0   2015812   0% /proc/acpi
tmpfs             2015812      0   2015812   0% /proc/scsi
tmpfs             2015812      0   2015812   0% /sys/firmware
root@nginx-deployment-5fb7cb76d9-8c9z4:/usr/share/nginx/html#
```

Cloud Filestore의 최소 크기는 1TB로 설정

Lab. Volumes

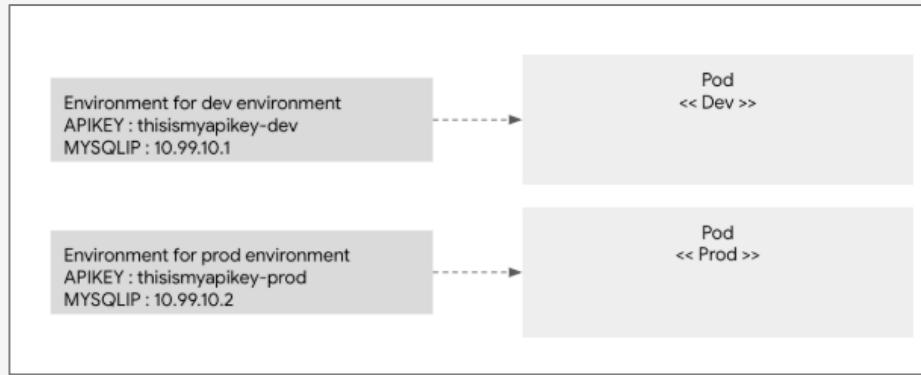


Lab Time

- Lab Script Location
 - Workflowy :

ConfigMaps

- ConfigMaps는 컨테이너 이미지로부터 설정 정보를 분리할 수 있게 해준다.
- 환경변수나 설정값들을 환경변수로 관리해 Pod가 생성될 때 이 값을 주입



- ConfigMaps은 2가지 방법으로 생성
 - 리터럴 값
 - 파일
- ConfigMaps는 etcd에 저장

Pod에서 ConfigMap 추가 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: cm-file-deployment
spec:
  replicas: 3
  minReadySeconds: 5
  selector:
    matchLabels:
      app: cm-file
  template:
    metadata:
      name: cm-file-pod
      labels:
        app: cm-file
    spec:
      containers:
        - name: cm-file
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/cm-file:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          env:
            - name: PROFILE
              valueFrom:
                configMapKeyRef:
                  name: cm-file
                  key: profile.properties
```

- 환경변수로 값 전달

- o cm-file configMap에서 키가 "profile.properties" (파일명)인 값을 읽어와서 환경 변수 PROFILE에 저장
- o 저장된 값은 파일의 내용인 아래 문자열이 됨
 - o myname=terry
email=myemail@mycompany.com
address=seoul

Pod에서 ConfigMap 추가 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: cm-file-deployment-vol
spec:
  replicas: 3
  minReadySeconds: 5
  selector:
    matchLabels:
      app: cm-file-vol
  template:
    metadata:
      name: cm-file-vol-pod
      labels:
        app: cm-file-vol
    spec:
      containers:
        - name: cm-file-vol
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/cm-file-volume:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: config-profile
              mountPath: /tmp/config
      volumes:
        - name: config-profile
          configMap:
            name: cm-file
```

- 디스크 볼륨으로 마운트 하기

- ConfigMap을 Volume으로 정의하고, 이 볼륨을 volumeMounts를 이용해 /tmp/config에 마운트 함
- 이때 중요한점은 마운트 포인트에 마운트 될 때, ConfigMap내의 키가 파일명이 됨

리터럴 값으로부터 ConfigMap 생성

- ConfigMap을 생성하는 명령어

```
$ kubectl create configmap my-config --from-literal=key1=value1 -  
-from-literal=key2=value2  
configmap "my-config" created
```

- 설정된 ConfigMap 정보 가져오기

```
$ kubectl get configmaps my-config -o yaml  
apiVersion: v1  
data:  
  key1: value1  
  key2: value2  
kind: ConfigMap  
metadata:  
  creationTimestamp: 2017-05-31T07:21:55Z  
  name: my-config  
  namespace: default  
  resourceVersion: "241345"  
  selfLink: /api/v1/namespaces/default/configmaps/my-config  
  uid: d35f0a3d-45d1-11e7-9e62-080027a46057
```

- o yaml 옵션은 해당 정보를 yaml 형태로 출력하도록 요청한다.
- 해당 객체는 종류가 ConfigMap이며 key-value 값을 가지고 있다.
- ConfigMap의 이름 등의 정보는 metadata field에 들어 있다.

파일로부터 ConfigMap 생성 (1/2)

- 아래와 같은 설정 파일을 만든다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: customer1
data:
  TEXT1: Customer1_Company
  TEXT2: Welcomes You
  COMPANY: Customer1 Company Technology Pct. Ltd.
```

- customer1-configmap.yaml**라는 이름으로 파일을 생성하였을 경우, 아래와 같이 ConfigMap를 생성한다.

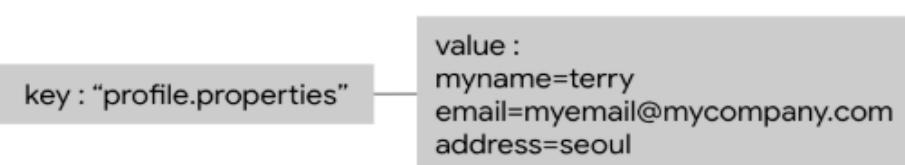
```
$ kubectl create -f customer1-configmap.yaml
configmap "customer1" created
```

파일로부터 ConfigMap 생성 (2/2)

- Userinfo.properties 파일을 생성하고,

```
myname=apexacme  
email=apexacme@uengine.org  
Address=seoul
```

- 파일을 이용해 ConfigMap을 만들 때는 --from-file을 이용해 파일명을 넘긴다.
- kubectl create configmap cm-file --from-file=./properties/profile.properties
 - 이때, 키는 파일명이 되고, 값은 파일 내용이 됨



Secrets

- ConfigMap이 일반적인 환경 설정 정보나 Config정보를 저장하도록 디자인 되었다면, 보안이 중요한 패스워드나 API 키, 인증서 파일들은 Secret에 저장
- Secret은 정보보안 차원에서 추가적인 보안 기능을 제공
 - 예를 들어, API서버나 Node의 파일에 저장되지 않고, 항상 메모리에 저장되므로 상대적 접근이 어려움
 - Secret의 최대 크기는 1MB (너무 커지면, apiserver나 Kubelet의 메모리에 부하 발생)
- ConfigMap과 기본적으로 유사하나, 값(value)에 해당하는 부분을 base64로 인코딩해야 함
 - SSL인증서와 같은 binary파일의 경우, 문자열 저장이 불가능하므로 인코딩 필요
 - 이를 환경변수로 넘길 때나 디스크볼륨으로 마운트해서 읽을 경우 디코딩 되어 적용

```
apiVersion: v1
kind: Secret
metadata:
  name: hello-secret
data:
  language: amF2YQo=
```

Pod에서 Secret 사용하기

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: hello-secret-deployment
spec:
  replicas: 1
  minReadySeconds: 5
  selector:
    matchLabels:
      app: hello-secret-literal
  template:
    metadata:
      name: hello-secret-literal-pod
      labels:
        app: hello-secret-literal
    spec:
      containers:
        - name: hello-secret
          image: (283210891307.dkr.ecr.ap-northeast-2.amazonaws.com)/hello-secret:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          env:
            - name: LANGUAGE
              valueFrom:
                secretKeyRef:
                  name: hello-secret
                  key: language
```

- Deployment.yaml 생성/ 실행
- kubectl create -f hello-secret-deployment.yaml
- \$ kubectl get deploy

Kubectl 명령어로 Secret 생성 및 확인

- 명령어로 Secret 만들기
 - \$ kubectl create secret generic my-password --from-literal=password=mysqlpassword
 - my-password라는 Secret을 생성하고, password라는 key와 mysqlpassword라는 value 값을 가지게 된다.
 - Value는 base64로 자동 encoding
 - **generic** : create a secret from a local file, directory or literal value
- Secret 확인 : kubectl get secret my-password -o yaml
 - echo [base64 value] | base64 --decode

Secret을 직접 만들기

- base64 형태로 인코딩하여 YAML파일내에 직접 생성 가능

```
$ echo mysqlpassword | base64  
bXIzcWxwYXNzd29yZAo=
```

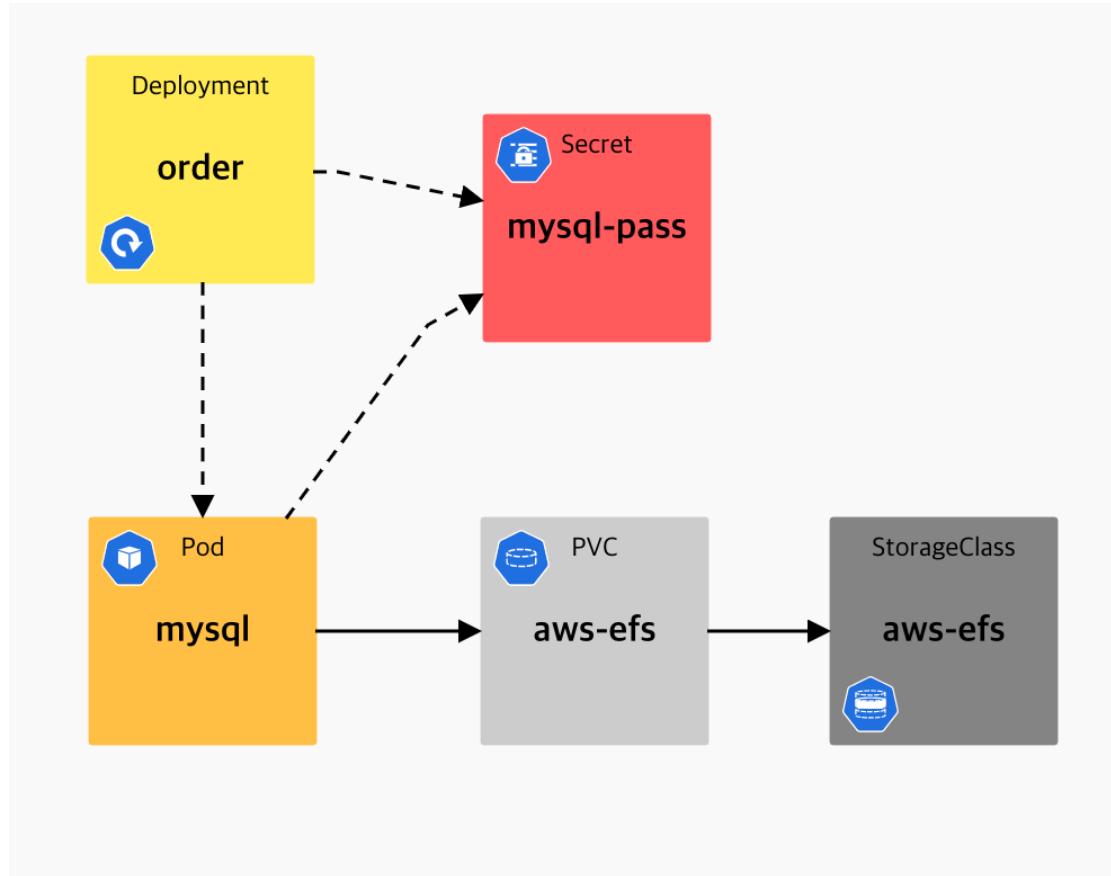
- 위 방식으로 인코딩 된 정보를 사용해 설정파일 생성

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: my-password  
type: Opaque  
data:  
  password: bXIzcWxwYXNzd29yZAo=
```

- base64** 인코딩은 바로 디코딩 됨으로 주의!

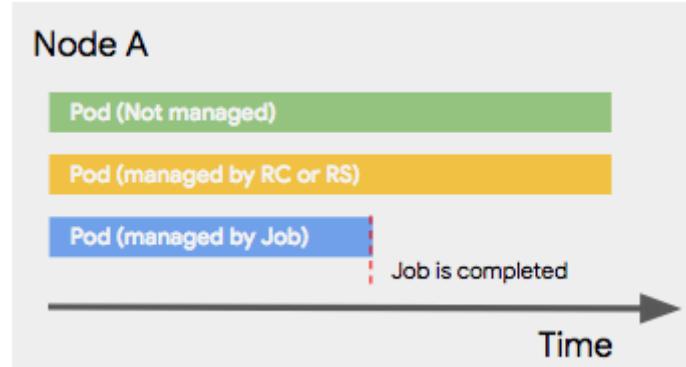
```
$ echo "bXIzcWxwYXNzd29yZAo=" | base64 --decode  
설정파일을 절대 소스코드에 넣지 않도록 주의한다!
```

Lab. Secret



Jobs

- 워크로드 모델 중, 배치나 한번 실행되고 끝나는 형태의 작업이 있을 수 있다.
- 예로, 원타임으로 파일 변환 작업을 하거나, 주기적으로 ETL 배치 작업을 하는 경우, Pod가 계속 떠 있을 필요 없이 작업을 할 때만 Pod를 실행한다.
- Job은 이러한 워크로드 모델을 지원하는 Controller



Jobs

- Job에 의해 관리되는 Pod는 Job이 종료되면 Pod도 같이 종료
- Job정의 시, Container Spec에 image뿐만 아니라, Job을 수행하기 위한 커맨드를 같이 입력

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle",
                     "print bpi(2000)"]
      restartPolicy: Never
      backoffLimit: 4
```

Job실패시, 처음부터 재시작 하지않음

Job실패시, 4번까지 재시도

Cron Jobs

- Job 컨트롤러에 의해 실행되는 작업을 주기적으로 스케줄링 해 주는 컨트롤러

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
  restartPolicy: OnFailure
```

Resource Quota

- ResourceQuota는 네임스페이스별로 사용 가능한 리소스 양을 정의

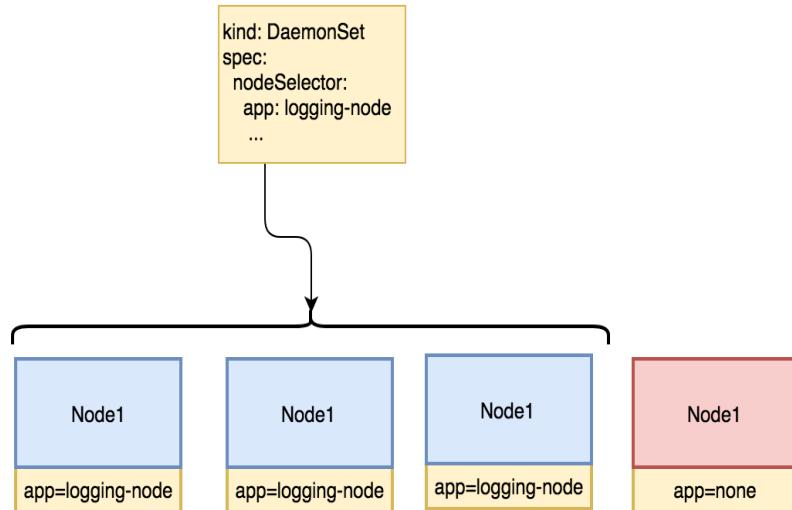
```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

<quota-mem-cpu.yaml>

- 네임스페이스에 ResourceQuota 적용
 - kubectl apply -f quota-mem-cpu.yaml --namespace=a-namespace

DaemonSets

- Pod를 각각의 노드에서 하나씩만 돌게 하는 형태로 Pod를 관리하는 컨트롤러
- 모니터링 데이터를 수집하거나, 스토리지 데몬을 실행하는 등, 모든 노드에서 항상 실행되는 특정 유형의 포드가 필요할 경우에 사용
 - Node 가 클러스터에 추가되면, 주어진 DaemonSet에 의해 Pod가 생성
 - DaemonSet이 삭제 되면, 관련된 모든 Node의 Pod들은 삭제
- 또한 특정 노드에만 Pod를 배포할 수 있도록, Pod의 “node selector”를 이용해서 라벨을 필터링하여 특정 노드만 선택 가능



StatefulSets

- 이전의 컨트롤러(Replica Set, Job) 같은 상태가 유지되지 않는 application을 관리하는 용도지만, StatefulSet은 단어의 의미 그대로 상태를 가지고 있는 포드들을 관리하는 컨트롤러
- 스테이트풀셋을 사용하면 볼륨을 사용해서 특정 데이터를 기록해두고 그걸 포드가 재시작했을 때도 유지할 수 있음
- StatefulSet controller는 이름, 네트워크 인증, 엄격한 순서 등의 독자성이 보장 되어야 할 때 사용
 - RS에 의해서 관리되는 Pod들은 기동이 될때 병렬로 동시에 기동 되나, DB의 경우에는 Master 노드가 기동 된 다음에, Slave 노드가 순차적으로 기동되어야 하는 순차성을 가지고 있는 경우가 있음
 - Ex) MySQL cluster, etcd cluster.
- 여러 개의 포드를 띄울 때 포드 사이에 순서를 지정해 지정된 순서대로 포드 실행 가능

StatefulSets

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: nginx
          image: k8s.gcr.io/nginx-slim:0.8
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
      volumeClaimTemplates:
        - metadata:
            name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        storageClassName: "standard"
        resources:
          requests:
            storage: 1Gi
```

- \$ kubectl get pod

NAME	READY	STATUS	RESTARTS	AGE
nginx-0	1/1	Running	0	13m
nginx-1	1/1	Running	0	12m
nginx-2	1/1	Running	0	12m

- \$ kubectl get pvc

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
www-nginx-0	Bound	pvc-e70627ea-2ecd-11e9-8d43-42010a920009	1Gi	RWO	standard	12m
www-nginx-1	Bound	pvc-f5f2d9d9-2ecd-11e9-8d43-42010a920009	1Gi	RWO	standard	12m
www-nginx-2	Bound	pvc-003c7376-2ece-11e9-8d43-42010a920009	1Gi	RWO	standard	12m

- StatefulSet은 Pod를 생성할 때 순차적으로 기동되고, 삭제할 때도 순차적으로(2 → 1 → 0) 삭제 됨

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture 

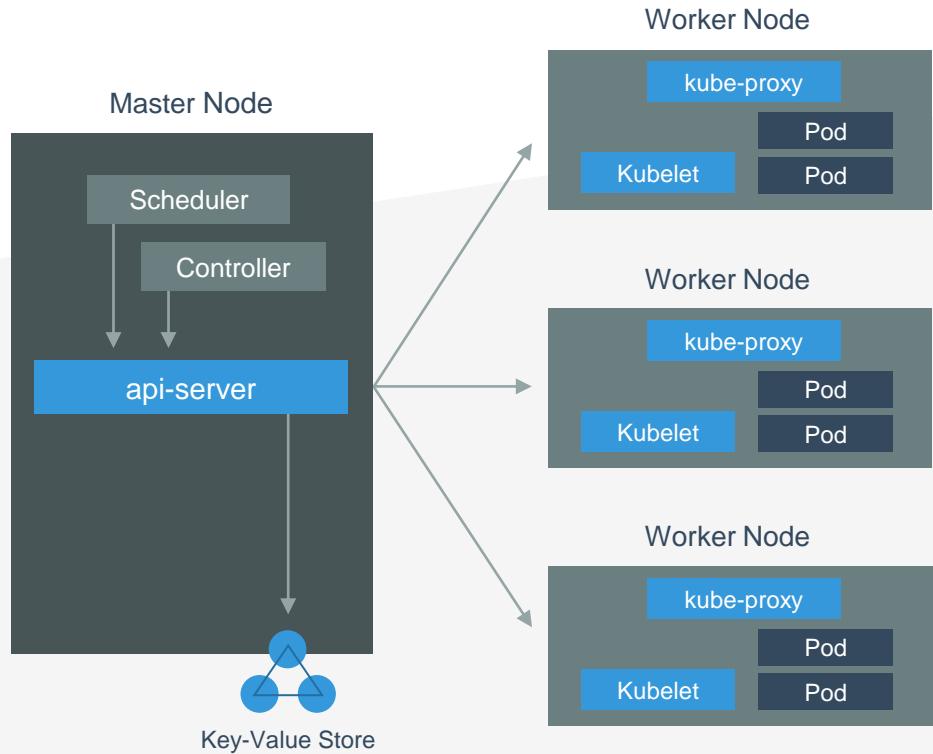
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

Kubernetes Architecture

- 하나 이상의 Master nodes
- 하나 이상의 Worker nodes
- etcd 와 같은 distributed key-value

User

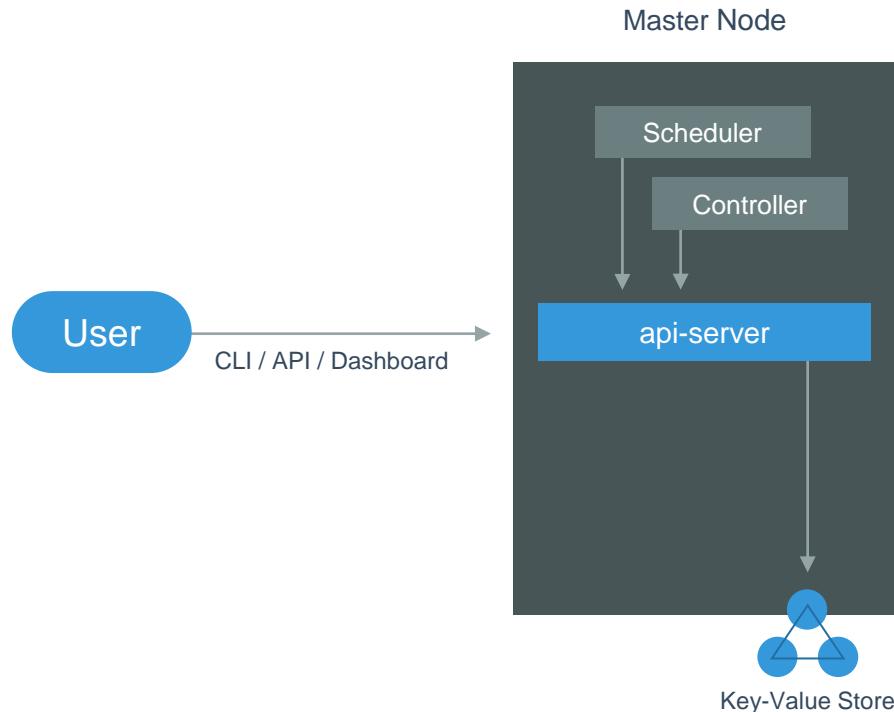
CLI / API / Dashboard



Master Node

- Master node는 Kubernetes 클러스터를 관리하며, 모든 관리자 업무의 시작점
- 다양한 모듈이 확장성을 고려하여 기능별로 분리
- CLI, GUI (Dashboard), APIs 등으로 Master node에 접근 가능

Master Node 구성요소



- API server
- Scheduler
- Controller manager
- Etcd. (distributed key-value store)

- ✓ 장애 대비하여, 하나 이상의 Master node를 한 클러스터에 구성하여 안정성을 높임
- ✓ 하나 이상의 Master node 가 존재할 경우, HA (High Availability)모드로 그 중 하나가 리더(leader)로써 작동하고 나머지는 팔로워(followers)로 운영

Master Node 구성요소 :

#1. API Server

- 원하는 상태를 Key-Value 저장소에 저장하고 저장된 상태를 조회하는 작업 수행
- 모든 administrative tasks는 master node에 있는 API server를 통해 수행
- 사용자/운영자가 보낸 요청은 API server를 통해 처리되고, 실행 결과는 분산 키-값 저장소 (Distributed key-value store)에 저장

Master Node 구성요소 :

#2. Scheduler

- Pod를 여러가지 조건(필요한 자원, 라벨)에 따라 적절한 노드에 할당해 주는 모듈
- Scheduler가 개별 worker node에게 업무를 분담
- Scheduler는 각 worker node의 리소스 사용 정보를 가짐
- “disk==ssd” 와 같은 label이 설정된 worker node에 업무 부여하는 등 사용자/운영자가 설정한 제약에 대해 인지하고 있음
- Scheduler는 Pods 와 Services 단위로 업무 수행

Master Node 구성요소 :

#3. Controller Manager

- **Kubernetes Cluster에 있는 모든 오브젝트의 상태를 관리하는 모듈**
- Controller manager는 Kubernetes Cluster의 상태를 조절하는 non-terminating control loops 관리
- 각각의 control loops는 자신이 관리하는 객체의 desired state를 알고 그 객체들의 현 상태를 API 서버를 통해서 모니터링
- 제어 루프에서 관리하는 객체의 현 상태가 원하는 상태와 다르면, 제어 루프는 수정 단계를 수행하여 현 상태와 원하는 상태를 일치시키는 작업 수행
- Kubernetes release 1.6, kube-controller-manage와 cloud-controller-manager로 세분

Master Node 구성요소 :

#4. Etcd

- RAFT* 알고리즘을 이용한 클러스터 상태를 저장하는 **key-value** 저장소
- 단순 R/W 기능외, Watch기능이 있어, 상태가 변경되면 트리거 로직 실행 가능
- 클러스터의 모든 설정, 상태가 저장되므로 Cluster Backup / Restore 유리
- Etcd는 오직 API 서버와 통신하고, 다른 모듈은 API서버를 통해 etcd 접근

* RAFT 알고리즘: 분산 환경에서 상태를 공유하는 알고리즘, consensus algorithm(Paxos, RAFT)

Master Node 구성요소 :

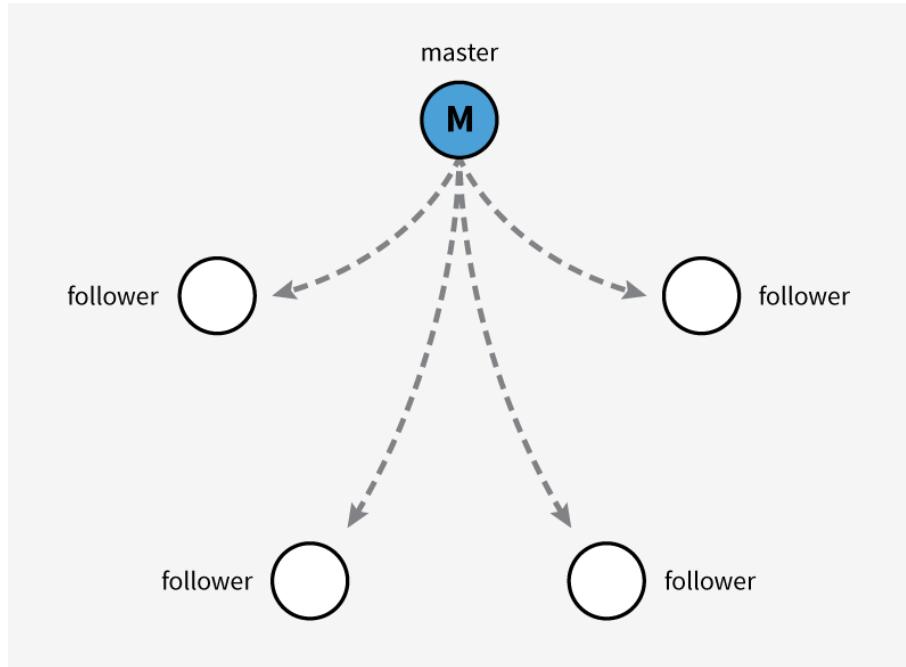
#4. Etcd - State 관리

- Kubernetes는 etcd를 사용해 클러스터의 상태를 저장
- etcd는 ‘Raft Consensus Algorithm’을 기반으로 하는 distributed key-value store
 - Raft는 장치의 집합으로써 몇몇 구성원에 장애가 발생하여도 살아남아 일관적인 집단으로 작동할 수 있도록 해 줌
- etcd는 Go 프로그래밍 언어로 작성
- Kubernetes에서는 클러스터의 상태를 저장하는 기능 말고도 Subnets, ConfigMaps, Secrets 등을 저장하기도 함

Master Node 구성요소 :

#4. Etcd - State 관리

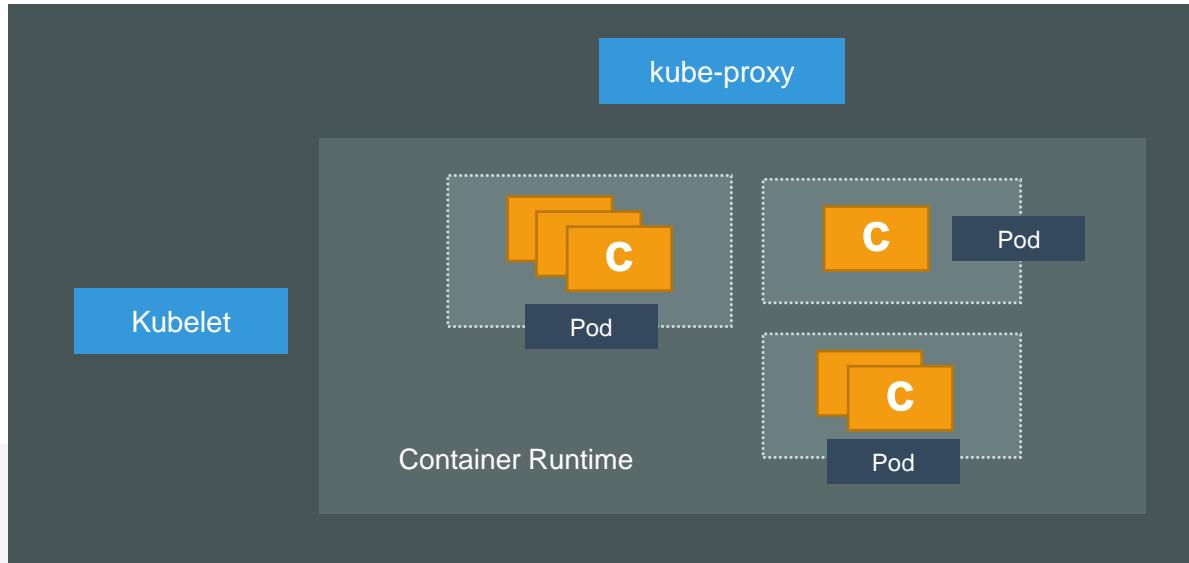
- 그룹의 노드 중 하나는 Master로 작동을 하고, 나머지는 Follower로 작동
- 모든 노드는 Master 후보



Worker Node

- 마스터 서버와 통신하면서 필요한 Pod를 생성하고 네트워크와 볼륨을 설정
- Worker node는 Master node에 의해 관리되며, Pod를 이용 어플리케이션을 실행
- Pod는 Kubernetes의 스케줄링 단위
- 하나 이상의 컨테이너의 논리적 집합이며, 항상 같이 스케줄링 됨

Worker Node 구성요소



- Container runtime
- kubelet
- kube-proxy

Worker Node 구성요소 :

#1. Container Runtime

- 컨테이너의 lifecycle을 관리/실행 하기위한 Worker node상의 Container runtime
 - (runtimes: containerd, cri-o, RKT, LXD)
 - ✓ 가끔, Docker가 Container runtime으로 언급되는 경우가 있다.
 - ✓ 정확하게는 Docker는 플랫폼이며, **containerd**를 Container runtime으로 사용한다.

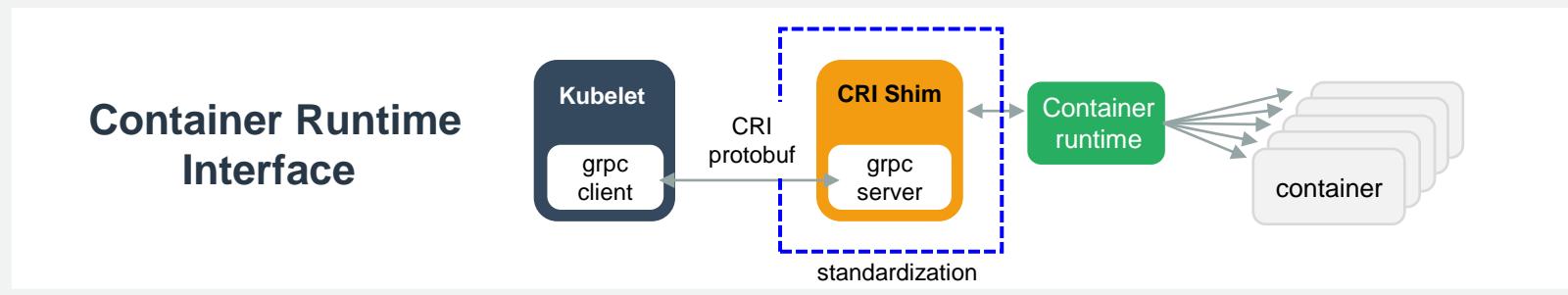
Worker Node 구성요소 :

#2. Kubelet

- **Node에 할당된 Pod의 생명 주기를 관리**
- kubelet은 master node와 통신하기 위해 worker node에서 작동하는 에이전트
- Pod 설정을 수신하고 해당 Pod와 관련된 컨테이너를 실행
- 항상 컨테이너가 정상 작동하는지 확인
- Kubelet은 Container Runtime Interface(CRI)를 이용하여 컨테이너 런타임에 연결
 - CRI는 protocol buffers, gRPC API, and libraries 등을 포함

Worker Node 구성요소 :

#2. Kubelet



- Kubelet이 명령을 받으면 Docker runtime을 통해 Container를 생성하거나 삭제
- Docker이외의 여러 컨테이너 기술이 나오면서 매번 Kubelet 코드를 수정하는 번거로움에 따라, Kubelet과 Container runtime 사이에 표준 인터페이스가 제정되었는데, 이를 CRI Shim이라 함
- 새로운 Container runtime은 CRI Shim스펙에 맞춰 CRI 컴포넌트를 구현

* gRPC : Google에서 처음 개발한 공개 원격 프로시저 호출(RPC) 시스템, 인터페이스 설명언어로 프로토콜 버퍼 사용

* Protocol buffer 줄임말로 크기를 줄인 직렬화 데이터 구조(이진 메시지 형식)

Worker Node 구성요소 :

#2. Kubelet : CRI Shims



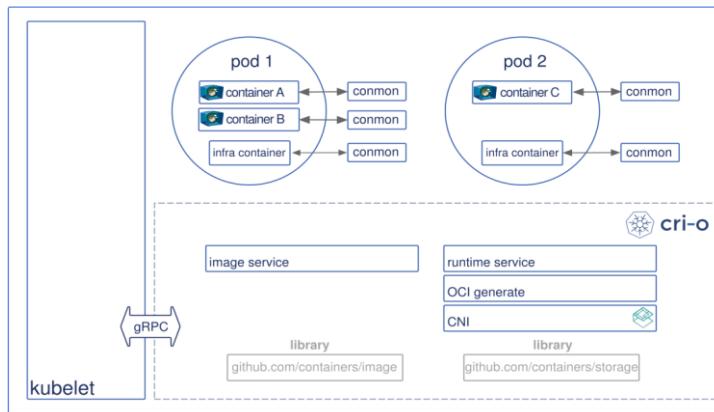
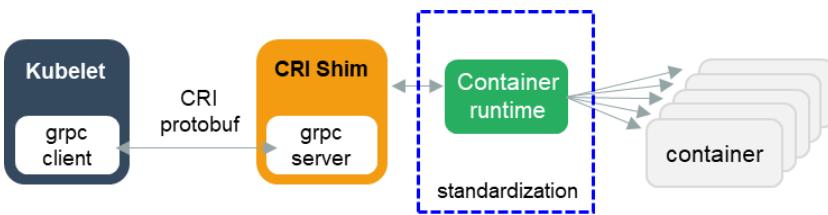
- **Docker shim**

Docker shim은 Worker node에 설치된 Docker를 이용하여 컨테이너를 생성

- 내부적으로는 Docker는 containerd를 이용해 컨테이너를 관리

Worker Node 구성요소 :

#2. Kubelet : CRI-O



- **Container Runtime 표준화**

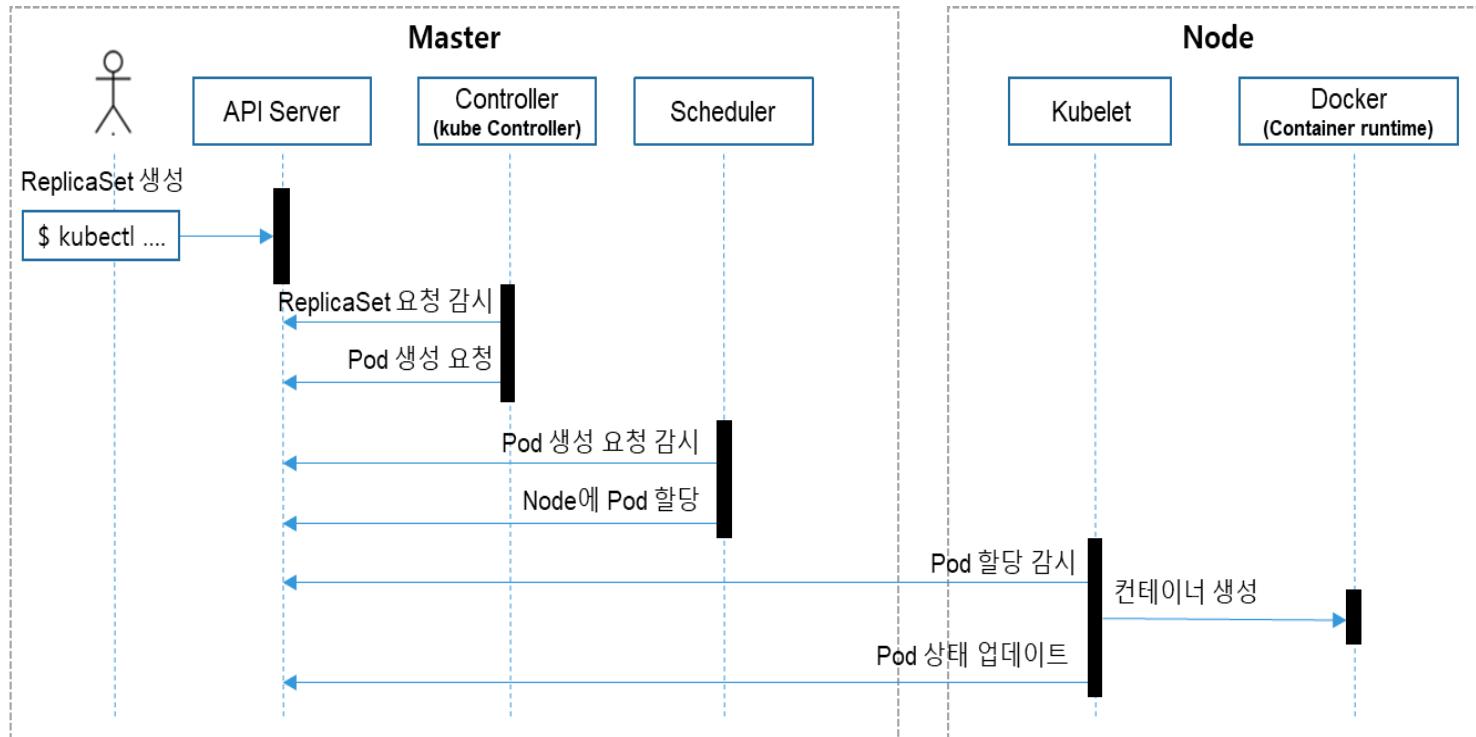
- 각 Container마다 CRI Shim을 개발해야 하는 비용적 이슈에서 CRI-O 등장
- Container runtime이 OCI(Open Container Initiative)에서 정의한 표준을 따르면 CRI-O를 CRI shim 으로 사용 가능

Worker Node 구성요소 :

#3. Kube-proxy

- Kubelet이 Pod를 관리한다면, Proxy는 Pod로 연결되는 네트워크를 관리
- 초기, kube-proxy가 프록시 서버로 동작하면서 실제 요청을 받아 각 Pod로 포워딩
 - 성능 이슈로 iptables를 설정하는 방식으로 변경, 최근엔 IPVS 지원 시작
- 애플리케이션에 액세스하기 위해 포드에 직접 연결하는 대신 "서비스"라는 논리적 구성을 연결 엔드포인트로 사용
 - Service 그룹과 관련된 Pod에 접속 시 자동으로 Load Balancing 됨
- kube-proxy는 Network proxy로서 각각의 Worker node에서 작동하며, 각각의 Service endpoint를 생성/삭제하기 위해 API server를 지속적으로 Listening
 - kube-proxy는 각 Service endpoint에 접근할 수 있는 경로 설정

Pod Creation Process (1/2)



Pod Creation Process (2/2)

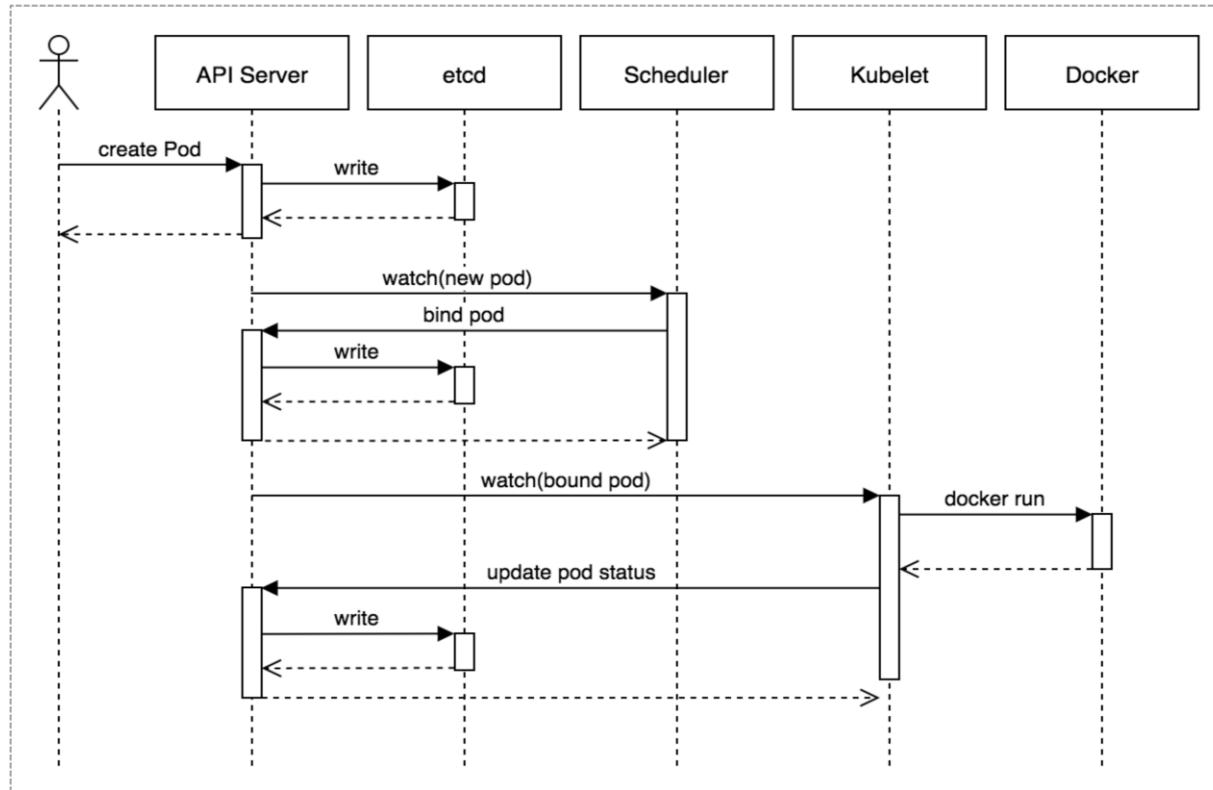
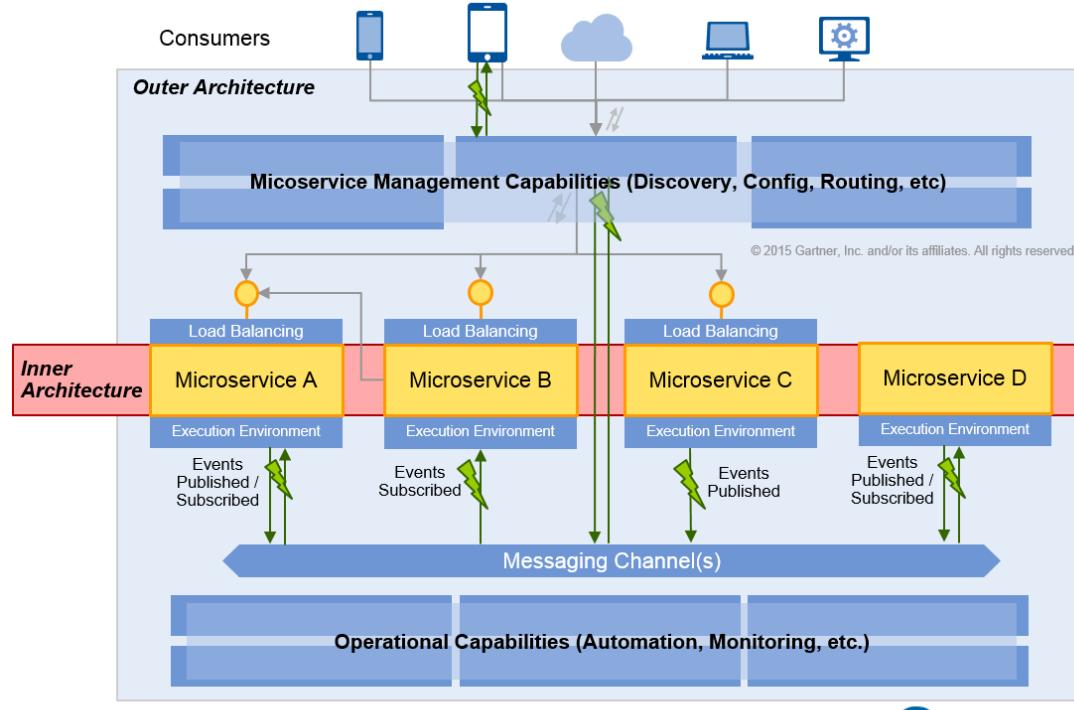


Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio) 
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract

Service Mesh in Microservice Architecture



Outer Architecture:

マイクロサービス外부를 아우르는 영역으로 트래픽 통제 및 서비스간 어떻게 커뮤니케이션 하는가 등, 클라우드 담당자가 공히 겪게 되는 복잡성들
(e.g. 서비스 발견, 탄력성, 장애 내성, 등)을 네트워크 레벨에서 처리

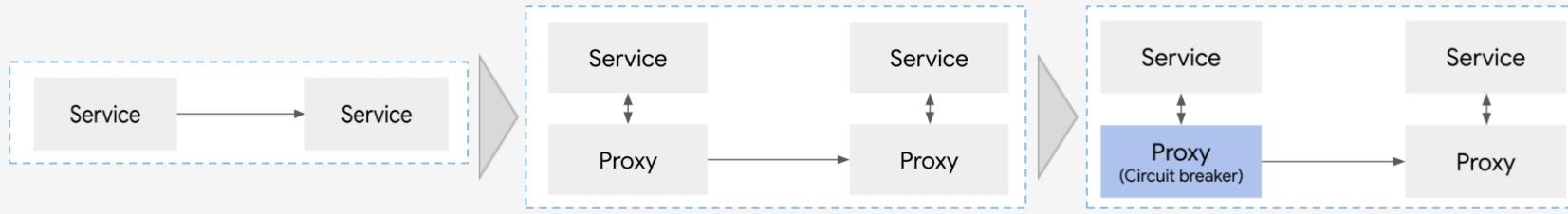
Inner Architecture:

개별 마이크로서비스 영역으로 가능한 간결, 명확하고 단순하게 구현하는데 중점

Service Mesh vs. MSA Framework

기 능	Istio	Spring Cloud & Netflix OSS
Application Language	<ul style="list-style-type: none">Polyglot	<ul style="list-style-type: none">Java
Architecture	<ul style="list-style-type: none">사이트카 패턴 (uses Envoy as proxy)	<ul style="list-style-type: none">애플리케이션에 라이브러리 추가
Implementation Level	<ul style="list-style-type: none">Platform (using YAML)	<ul style="list-style-type: none">Application (using Code)
Service Discovery	<ul style="list-style-type: none">Implicit (자동 등록)	<ul style="list-style-type: none">Explicit (명시적 선언)
Circuit Breaker	<ul style="list-style-type: none">YAML Configuration	<ul style="list-style-type: none">Coding/ Annotation
Tracing	<ul style="list-style-type: none">Envoy Proxy들을 통한 호출 추적	<ul style="list-style-type: none">추적 데이터를 명시적으로 로깅
Security	<ul style="list-style-type: none">Service to ServiceMutual TLS through Envoy	<ul style="list-style-type: none">Application Specific

Service Call through Proxy, Service Mesh



- 서비스를 직접 호출하는 것이 아니라, 서비스마다 프록시(Proxy, Sidecar)를 통한 호출
- 프록시를 통해, 들어오거나 나가는 트래픽을 통제함으로써 Circuit Breaker Pattern 지원
- 클라이언트의 OS나, 브라우저 환경에 따른 스마트 라우팅(L7 Layer)도 프락시에서 통제 가능
 - 기존 Ingress와 같은 API Gateway로는 한계

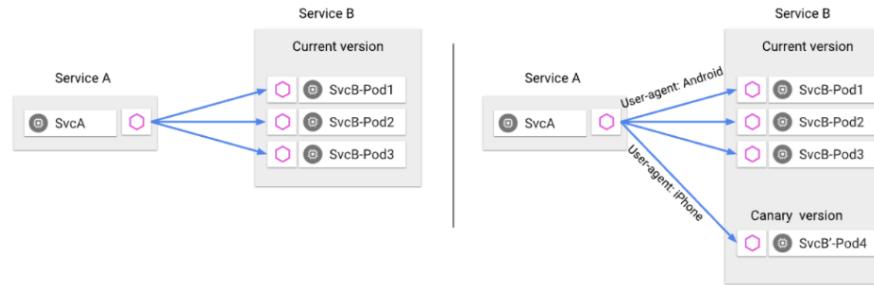
Istio Service Mesh key features

- Improved Routing & Deployment Strategy
- Improved Smart Resilience
- Improved Security
- Improved Observability



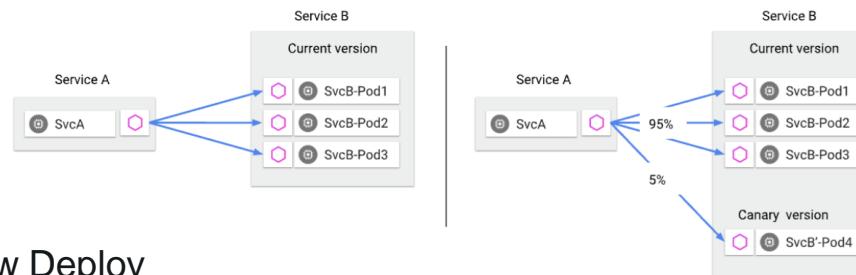
1. Improved Routing & Deployment Strategy

- Traffic Routing Controls



- Canary Deploy

- 특정 유저의 신상, 지역, 권한, 접근 단말에 따른 다른 버전의 노출



- A/B Testing, Shadow Deploy

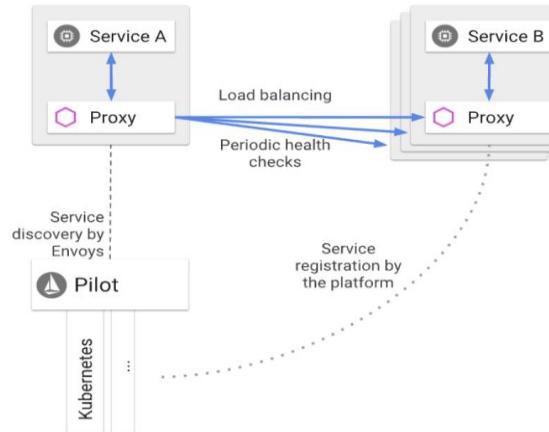
- 신규 버전의 오류 노출 없는 실질적 테스트

2. Improved Resilience

- Retry
 - 서비스간 호출의 실패에 대한 재시도
- Circuit Breaking / Rate Limiting
 - 인스턴스의 보호
 - 전체 서비스 장애 차단
- Pool Ejection
 - 죽은 인스턴스의 제외
- Health Check & Service Discovery
 - 여러 인스턴스에 대한 로드 밸런싱
 - 서비스 앤드 포인트 관리



Circuit Breaking + Pool Ejection + Retry
= High Resilience

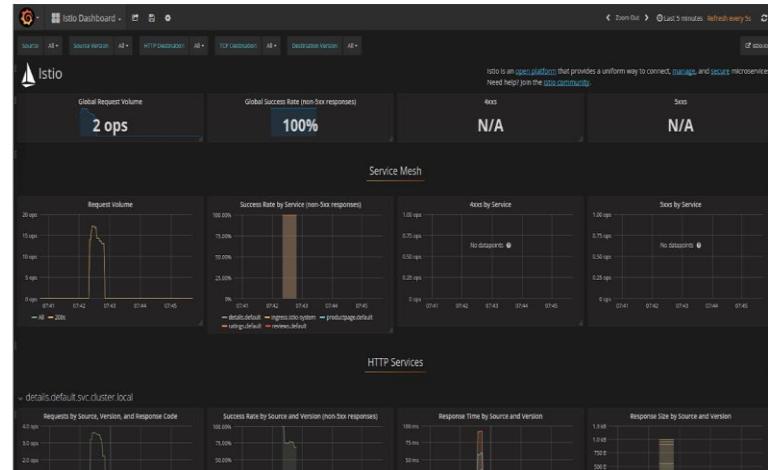
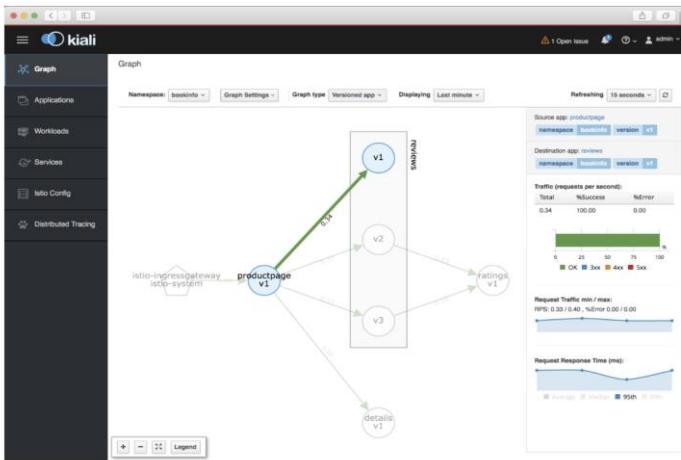


3. Improved Security

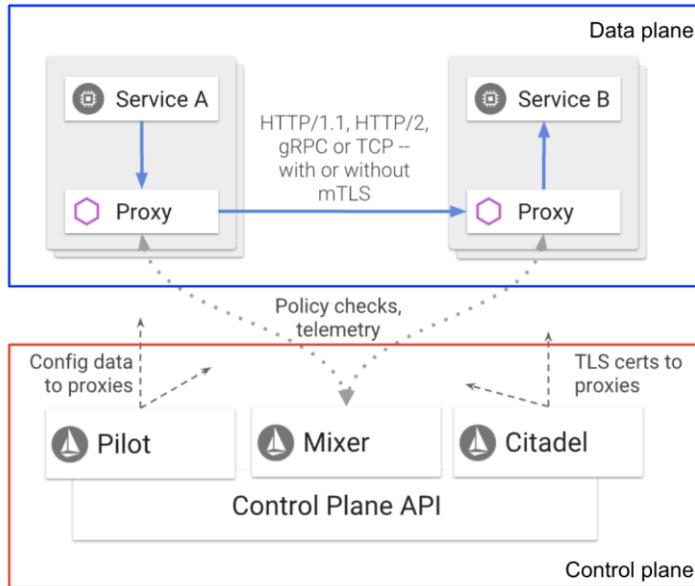
- TLS based Inter Microservices Communication
 - Envoy로 통신하는 모든 트래픽을 자동으로 TLS를 이용해 암호화
- Service Authentication & Authorization
 - JWT Token을 이용한 서비스 접근 Client 인증
 - 역할기반(RBAC) 권한 인증 지원
- Whitelist and Blacklist

4. Improved Observability

- Distributed Tracing and Measure
 - 서비스간 호출 내용 기록
 - Istio 설치 시, Kiali, Jaeger가 default로 설치되어 각각 Monitoring 및 Tracing 지원



Istio Service Mesh architecture



- **Data Plane(데이터 플래인)**

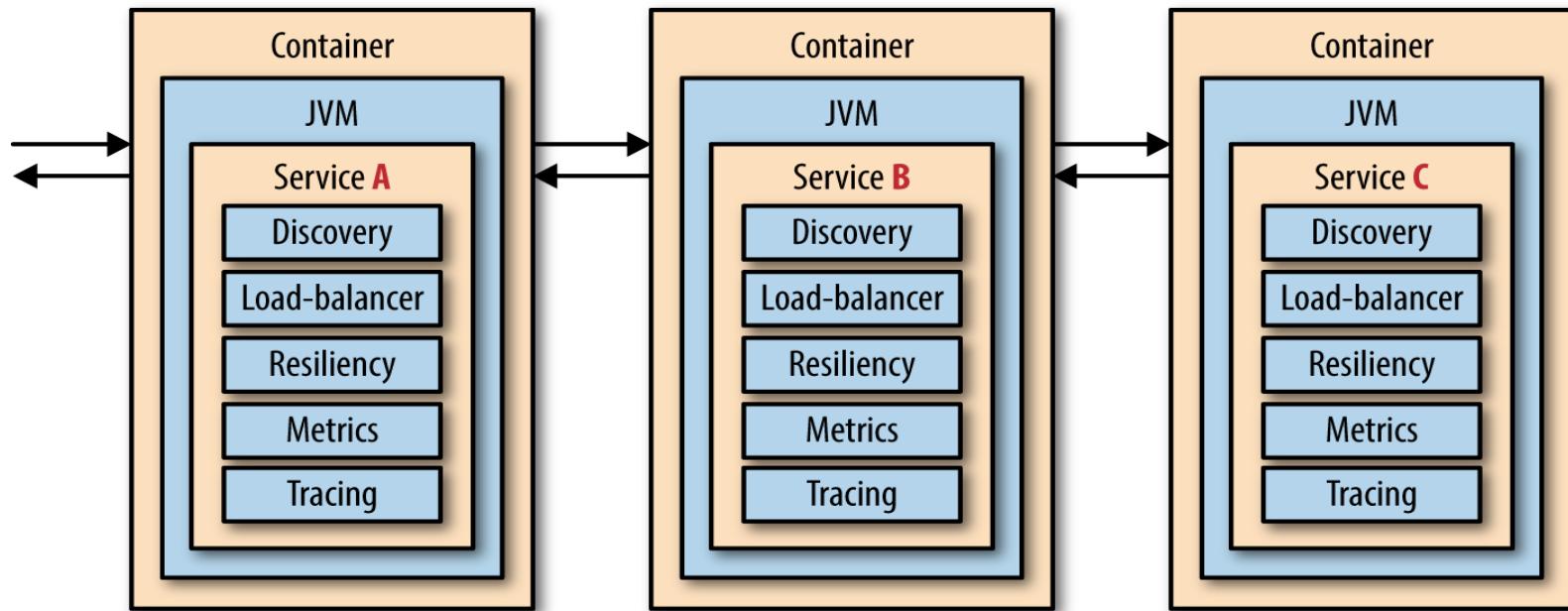
- 서비스 옆에 사이드카(Envoy)를 붙여, 서비스로 들어오고 나가는 트래픽을 Envoy를 통해 통제
- 유입/유출 트래픽 통제 : Ingress/Egress Controller

- **Control Plane(컨트롤 플래인)**

- **Pilot(파일럿)** : Envoy에 대한 설정 관리 및 서비스 디스커버리 기능 제공
- **Mixer(믹서)** : 액세스 컨트롤 및 다양한 모니터링 지표 수집
- **Citadel(시터덜)** : 보안관련 기능 담당 모듈로 사용자 인증/ 인가 및 TLS 통신을 위한 인증서 관리

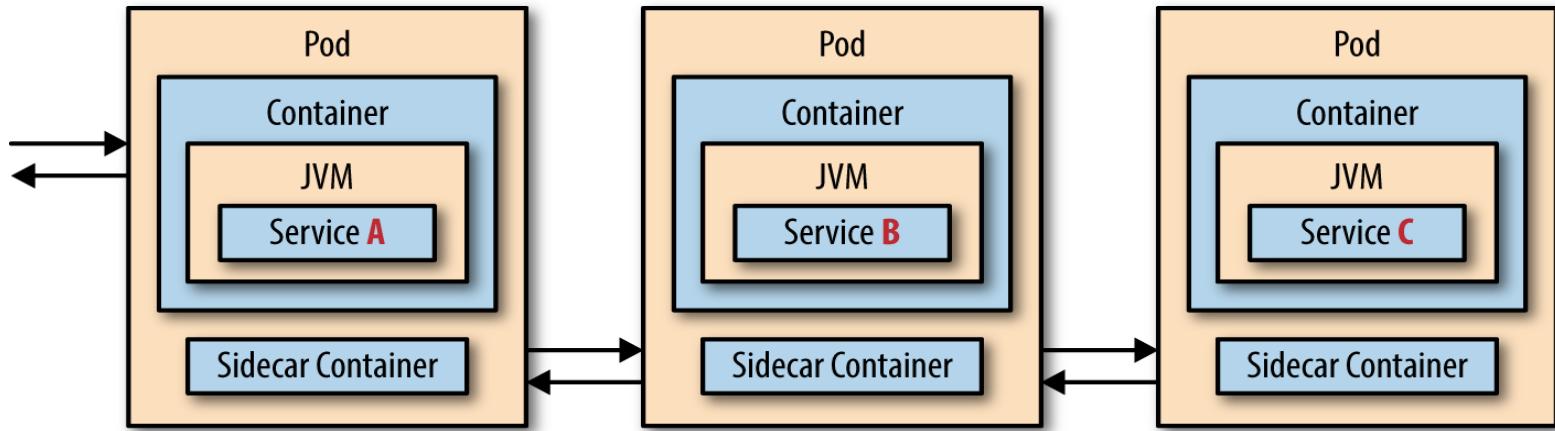
Before Istio : Spring Cloud + Netflix

Before Istio



After Istio

Envoy



좋은점 :

1. L7 레이어를 사용, 성능이 높음
2. Code 변경 없이 Cross-cutting 이슈를 다루어 줌
3. Main 서비스의 재배포 없이 Sidecar 를 관리 가능함

Lab. Istio Install

- Istio 설치

- curl -L https://git.io/getLatestIstio | ISTIO_VERSION=1.4.5 sh -
- cd istio-1.4.5
- export PATH=\$PWD/bin:\$PATH
- for i in install/kubernetes/helm/istio-init/files/crd*yaml; do kubectl apply -f \$i; done
- kubectl apply -f install/kubernetes/istio-demo.yaml

- 설치확인

- \$ kubectl get pod -n istio-system



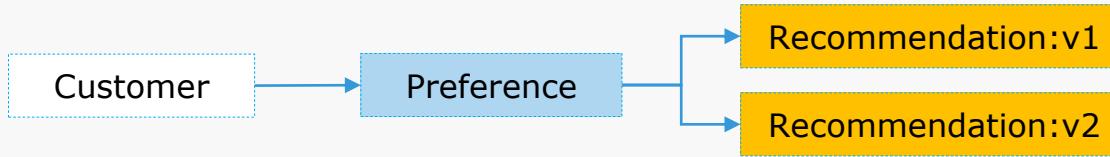
```
apexacme@APEXACME:~/istio-1.4.5$ kubectl get pod -n istio-system
NAME                               READY   STATUS    RESTARTS   AGE
grafana-6bb6bcf99f-78p9q          1/1    Running   0          76s
istio-citadel-597559bf4-lpqxc   1/1    Running   0          75s
istio-egressgateway-5c68d8857-qp14x 0/1    Running   0          76s
istio-galley-769849c5fb-cpvkp    0/1    Running   0          76s
istio-grafana-post-install-1.4.5-frqk4 0/1    Completed 0          78s
istio-ingressgateway-665c7c6b8-k9qhs 0/1    Running   0          76s
istio-pilot-6bd54f544b-65tnf     1/2    Running   1          75s
istio-policy-6b7d856769-xcip2    2/2    Running   2          75s
istio-security-post-install-1.4.5-p9wtc 0/1    Completed 0          78s
istio-sidecar-injector-6d9f967b5-vl57x 1/1    Running   0          75s
istio-telemetry-657bbb5677-pg2th   2/2    Running   2          75s
istio-tracing-56c7f85df7-71kld8   1/1    Running   0          75s
kiali-7b5c8f79d8-dm46s           1/1    Running   0          76s
prometheus-74d8b55f54-r9w7v      1/1    Running   0          75s
apexacme@APEXACME:~/istio-1.4.5$
```

Lab. Istio Tutorial Setup (1/2)

- Git repository에서 Tutorial 리소스 가져오기
 - cd ~
 - mkdir git
 - cd git
 - git clone https://github.com/redhat-developer-demos/istio-tutorial
 - cd istio-tutorial
 - 네임스페이스 생성
 - kubectl create namespace tutorial
 - Customer 애플리케이션 배포
 - kubectl apply -f <(istioctl kube-inject -f customer/kubernetes/Deployment.yml) -n tutorial
 - kubectl create -f customer/kubernetes/Service.yml -n tutorial
 - kubectl create -f customer/kubernetes/Gateway.yml -n tutorial
 - 배포 확인 및 외부접속 설정
 - kubectl -n tutorial edit svc customer
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl -n tutorial get svc
 - 브라우저로 EXTERNAL-IP:8080 접속
- | NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------------|--------------|-------------|----------------|----------------|------|
| service/customer | LoadBalancer | 10.0.254.95 | 52.231.113.110 | 8080:30926/TCP | 6m4s |

Lab. Istio Tutorial Setup (2/2)

- 서비스 흐름



- 테스트 애플리케이션 배포 (preference, recommendation)
 - kubectl apply -f <(istioctl kube-inject -f preference/kubernetes/Deployment.yml> -n tutorial
 - kubectl create -f preference/kubernetes/Service.yml -n tutorial
 - kubectl apply -f <(istioctl kube-inject -f recommendation/kubernetes/Deployment.yml> -n tutorial
 - kubectl create -f recommendation/kubernetes/Service.yml -n tutorial

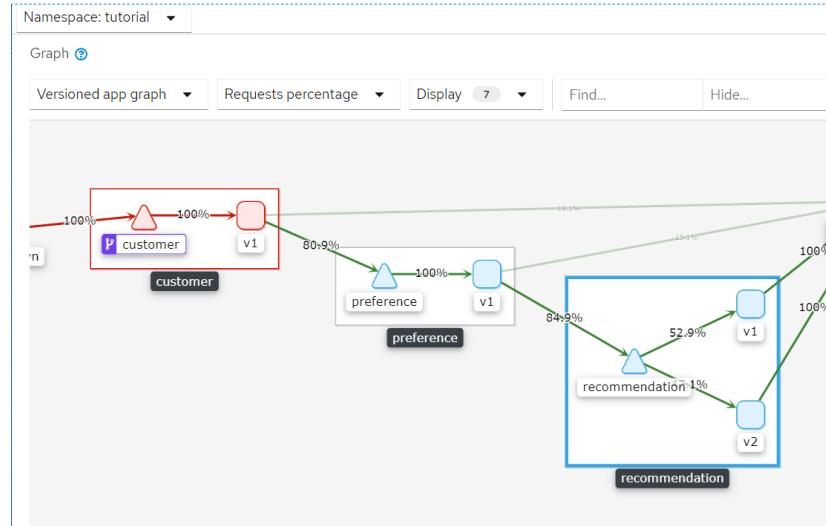
- 모니터링시스템 외부접속 설정

- kubectl edit svc jaeger-query -n istio-system
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl edit svc kiali -n istio-system
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl get svc -n istio-system
 - Jaeger 접속 : EXTERNAL-IP :16686
 - Kiali 접속 : EXTERNAL-IP:20001

jaeger-query	LoadBalancer	10.0.181.81	52.231.113.45	16686:32721/TCP
kiali	LoadBalancer	10.0.217.7	40.82.159.29	20001:30150/TCP

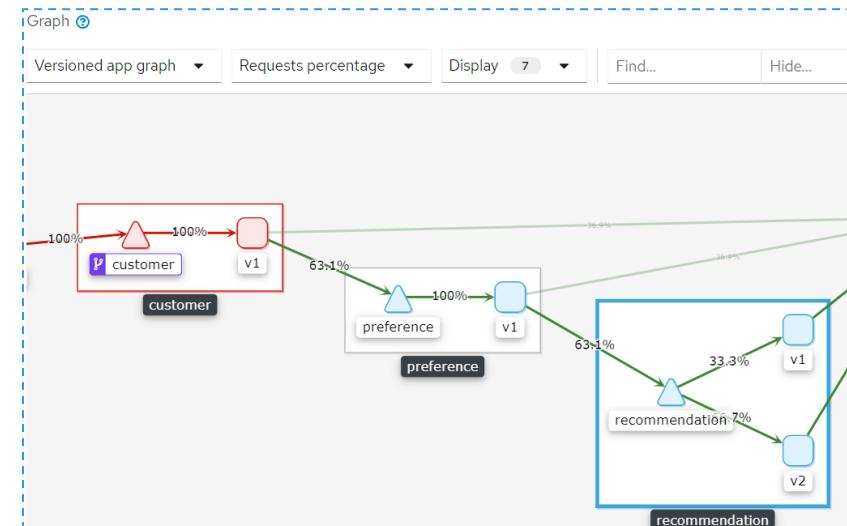
Lab. Istio – Simple Routing (1/2)

- Istio의 Simple Routing 확인
 - recommendation 서비스 추가 배포 (version. two)
 - kubectl apply -f <(istioctl kube-inject -f recommendation/kubernetes/Deployment-v2.yaml) -n tutorial
 - 서비스 호출
 - 브라우저에서 Customer 서비스(External-IP:8080 접속) 호출
 - F5(새로고침)를 10회 이상 클릭하여 다수의 요청 생성
- Routing 결과 확인
 - Kiali(External-IP:20001) 접속
 - Left 영역 : Graph 선택
 - Main 영역 : Versioned app > Requests percentage
 - Recommendation 서비스의 v1, v2에 균등한 라우팅 (Round Robin) 비율 확인 - 52.9% : 48.1%



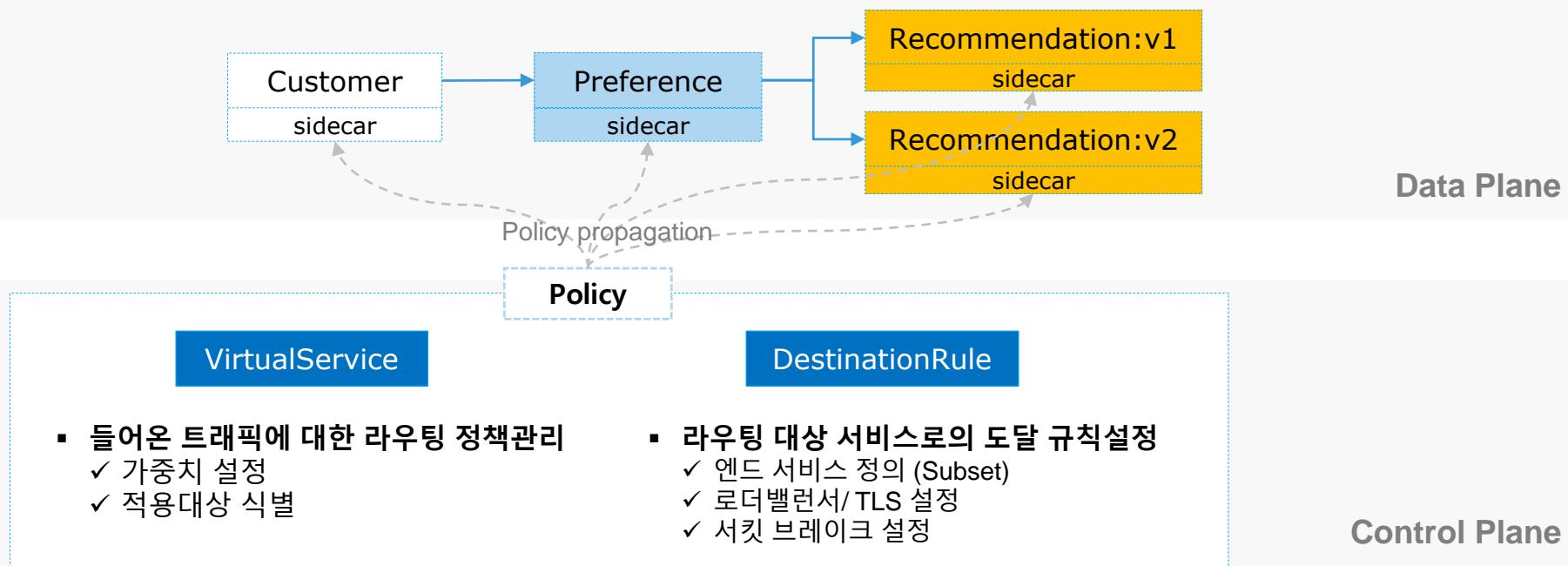
Lab. Istio – Simple Routing (2/2)

- recommendation 서비스 Scale Out
 - 서비스의 v2 의 replica 를 2로 설정
 - kubectl scale --replicas=2 deployment/recommendation-v2 -n tutorial
 - kubectl get po -n tutorial
 - 서비스 호출
 - 브라우저에서 Customer 서비스(External-IP:8080 접속) 호출
 - F5(새로고침)를 10회 이상 클릭하여 다수의 요청 생성
- Routing 결과 확인
 - Kiali(External-IP:20001) 접속
 - Left 영역 : Graph 선택
 - Main 영역 : Versioned app > Requests percentage
 - Recommendation 서비스의 v1, v2에 균등한 라우팅 (Round Robin) 비율 확인 - 33.3% : 66.6%



Istio Improved Routing & Rule

- 이스티오 정책 : VirtualService, DestinationRule



Lab. Istio – Improved Routing (1/3)

- 정책(VirtualService, DestinationRule) 설정 실습
 - 현재 정책 확인
 - kubectl get VirtualService -n tutorial -o yaml, kubectl get DestinationRule -n tutorial -o yaml

recommendation 서비스 라우팅 정책 설정

VirtualService/ DestinationRule 설정

- kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yaml -n tutorial
- kubectl create -f istiofiles/virtual-service-recommendation-v2.yaml -n tutorial

설정 확인

- kubectl get VirtualService -n tutorial -o yaml

```
spec:  
  hosts:  
    - recommendation  
  http:  
    - route:  
      - destination:  
          host: recommendation  
          subset: version-v2  
          weight: 100
```

- kubectl get DestinationRule -n tutorial -o yaml

```
spec:  
  host: recommendation  
  subsets:  
    - labels:  
        version: v1  
        name: version-v1  
    - labels:  
        version: v2  
        name: version-v2
```

서비스 확인

- 브라우저에서 Customer 서비스(External-IP:8080 접속)호출
- Kiali(External-IP:20001)에서 모니터링

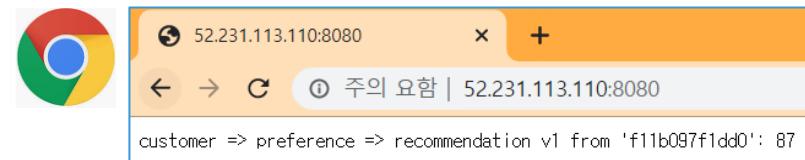
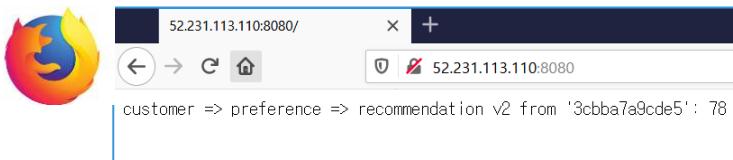


Lab. Istio – Improved Routing (2/3)

- 가중치 기반 Service Routing 실습
 - recommendation 서비스 v1의 가중치를 100으로 변경
 - kubectl replace -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial
 - 서비스 호출 및 Kiali(Externl-IP:20001)에서 모니터링
- VirtualService 삭제 시, Round-Robin 방식으로 동작
 - kubectl delete -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial
- 이와 같이 가중치를 사용해 카나리 배포(Canary Deploy)가 가능함
 - 90 : 10
 - kubectl create -f istiofiles/virtual-service-recommendation-v1_and_v2.yml -n tutorial
 - 75 : 25
 - kubectl replace -f istiofiles/virtual-service-recommendation-v1_and_v2_75_25.yml -n tutorial
- 삭제
 - kubectl delete dr recommendation -n tutorial
 - # kubectl delete vs recommendation -n tutorial
 - kubectl scale --replicas=1 deployment/recommendation-v2 -n tutorial

Lab. Istio – Improved Routing (3/3)

- Client 브라우저 유형별 Service Routing 실습
 - Firefox 브라우저로 접속 시, v2로 라우팅되도록 설정
 - kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yml -n tutorial
 - kubectl create -f istiofiles/virtual-service-firefox-recommendation-v2.yml -n tutorial
 - Firefox 브라우저와 다른 브라우저에서 접속 확인
 - Browser 환경이 지원되지 않을 경우,
 - curl -A Safari External-IP:8080
 - curl -A Firefox External-IP:8080



- 삭제
 - kubectl delete dr recommendation -n tutorial
 - kubectl delete vs recommendation -n tutorial

Lab. Istio



Lab Time

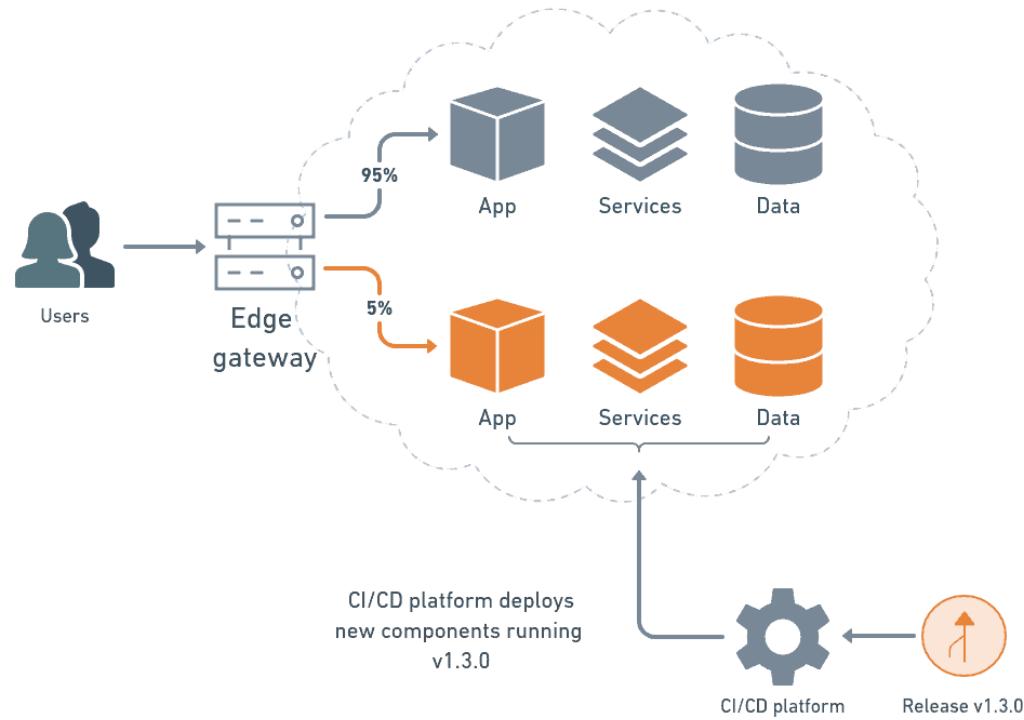
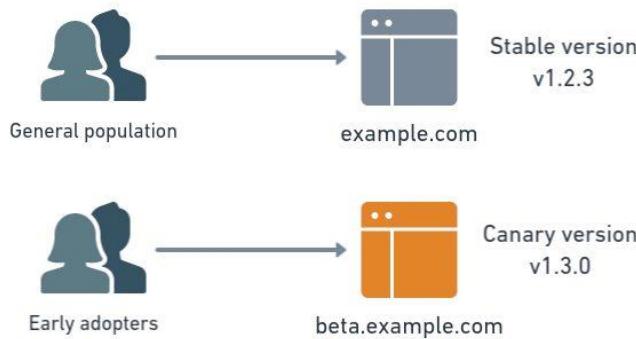
- Lab Script Location
 - Workflowy :

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio ✓
8. Contract Test with Spring Cloud Contract

Canary Deployment



Kubernetes Default Deployment Strategy – Rolling Updates

새 버전의 배포

```
kubectl set image deploy order order=order:v2
```

방금 디플로이의 롤백 (복구)

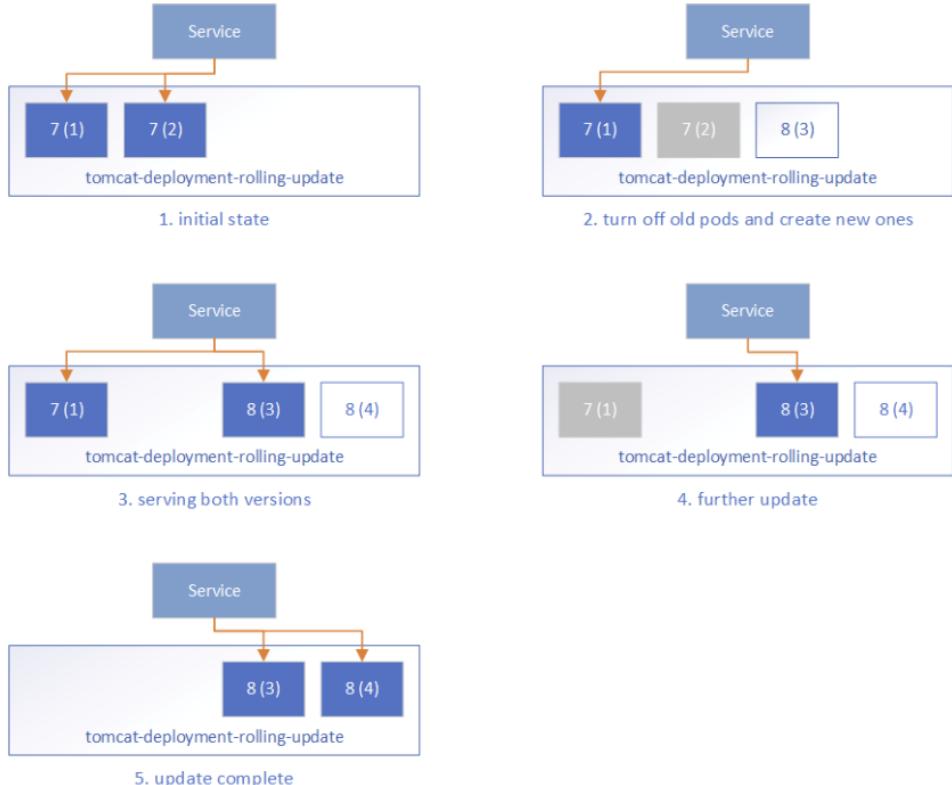
```
kubectl rollout undo deploy order
```

히스토리 보기

```
kubectl rollout history deploy order
```

특정 버전으로 복구

```
kubectl rollout undo deploy order --to-revision 2
```



Argo Rollouts – Basic Canary

Monitoring

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollouts-demo
spec:
  replicas: 5
  strategy:
    canary:
      steps:
        - setWeight: 20
        - pause: {}
        - setWeight: 40
        - pause: {duration: 10}
        - setWeight: 60
        - pause: {duration: 10}
        - setWeight: 80
        - pause: {duration: 10}
```

Step Setting

Deploy Command

```
kubectl argo rollouts set image rollouts-demo order=order:v2
```

Rollback Command

```
kubectl argo rollouts undo rollouts-demo
```

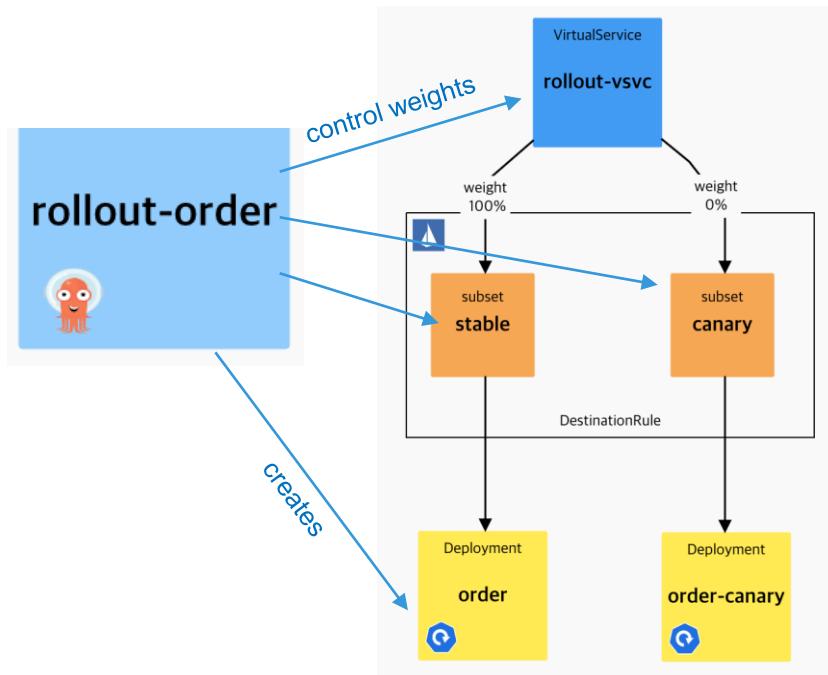
```
$ kubectl argo rollouts get rollout demo
Name:          demo
Namespace:     default
Status:        ⚡ Paused
Strategy:     Canary
  Step:        3/4
  SetWeight:   50
  ActualWeight: 33
Images:       argoproj/rollouts-demo:blue (stable)
                  argoproj/rollouts-demo:orange (canary)

Replicas:
  Desired:    4
  Current:    6
  Updated:    2
  Ready:      6
  Available:  6

NAME                      KIND   STATUS AGE INFO
C demo                     Rollout ⚡ Paused 21m
  # revision:2
    └─demo-f66bdb575
      ├─demo-f66bdb575-tzkhf
      │ ├─Pod
      │ └─Pod
      └─demo-f66bdb575-hgrw2
        ├─Pod
        └─Pod
  # revision:1
    └─demo-8478fd7f57
      ├─demo-8478fd7f57-7kmg6
      ├─demo-8478fd7f57-dmmf8
      ├─demo-8478fd7f57-dtmbf
      └─demo-8478fd7f57-fb66d
        ├─Pod
        └─Pod
```

Argo Rollouts

– Canary with Istio (Blue/Green with Splitting Traffic)



```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-order
spec:
  replicas: 5
  strategy:
    canary:
      trafficRouting:
        istio:
          virtualService:
            name: rollout-vsvc           # required
            routes:
              - primary                 # required
            destinationRule:
              name: rollout-destrule   # required
              canarySubsetName: canary # required
              stableSubsetName: stable # required
  steps:
    - setWeight: 5
    - pause:
        duration: 10s
    - setWeight: 20
    - pause:
        duration: 10s
    - setWeight: 40
    - pause:
        duration: 10s
    - setWeight: 60
```

Argo Rollouts

Canary Analysis with AnalysisTemplate

카나리 테스트를 통과하지
않으면 자동 롤백

```
---  
apiVersion: argoproj.io/v1alpha1  
kind: AnalysisTemplate  
metadata:  
  name: success-rate  
spec:  
  args:  
    - name: service-name  
  metrics:  
    - name: success-rate  
      successCondition: result[0] >= 0.95  
      interval: 30s  
      failureLimit: 1  
    provider:  
      prometheus:  
        address: http://prometheus.istio-system.svc.cluster.local:9090  
        query: |  
          sum(irate(  
            istio_requests_total{reporter="source",destination_service=""}  
          ) /  
          sum(irate(  
            istio_requests_total{reporter="source",destination_service=""}  
          ))
```

Labs

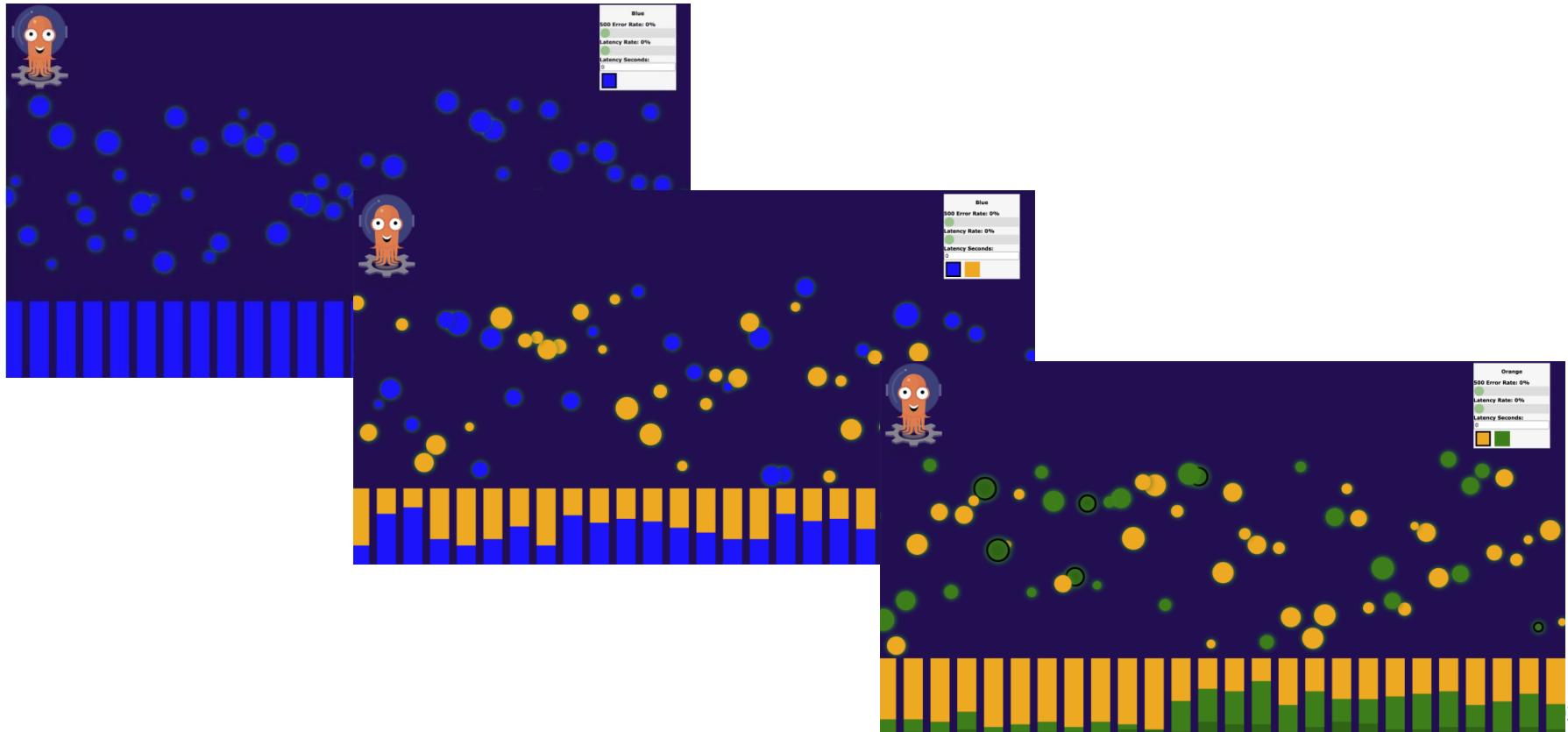


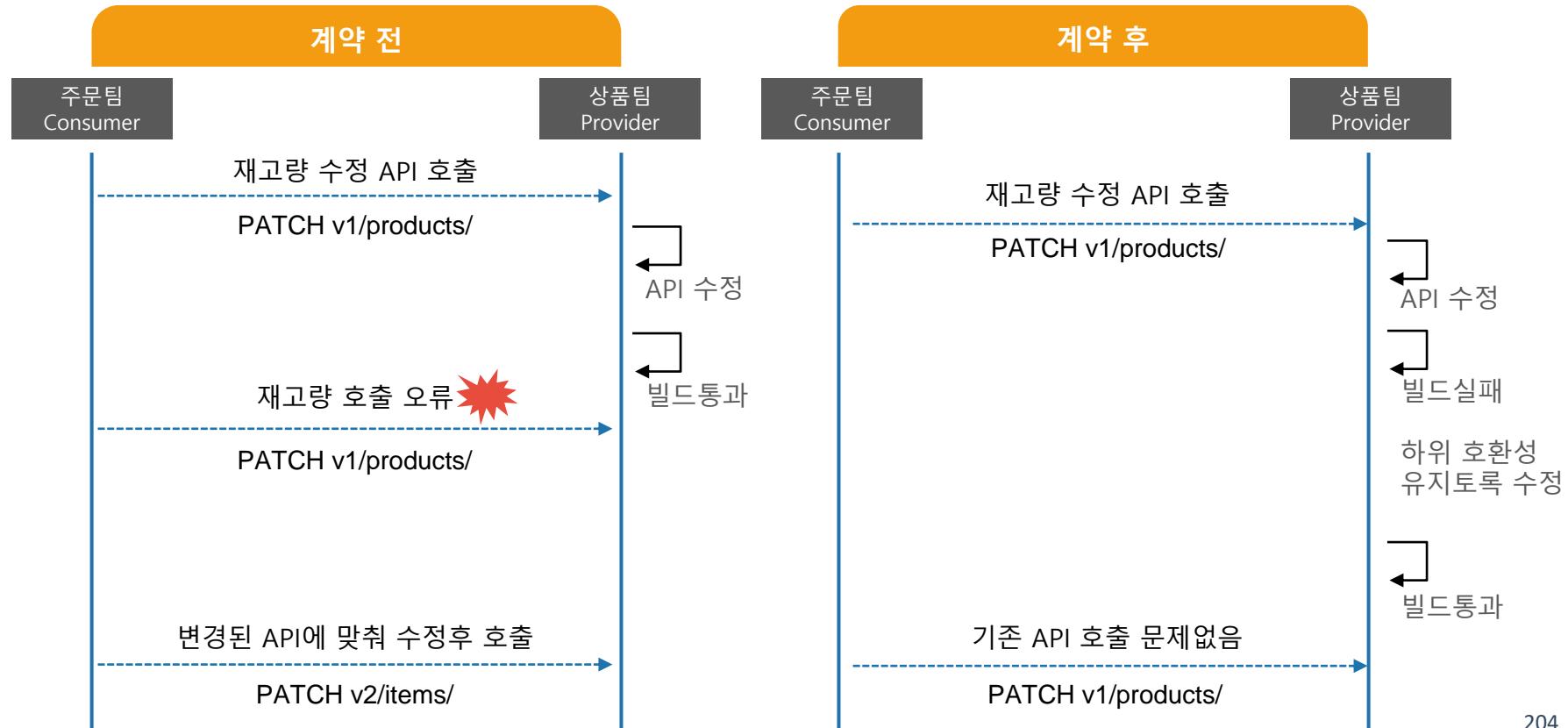
Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract



Contract Test



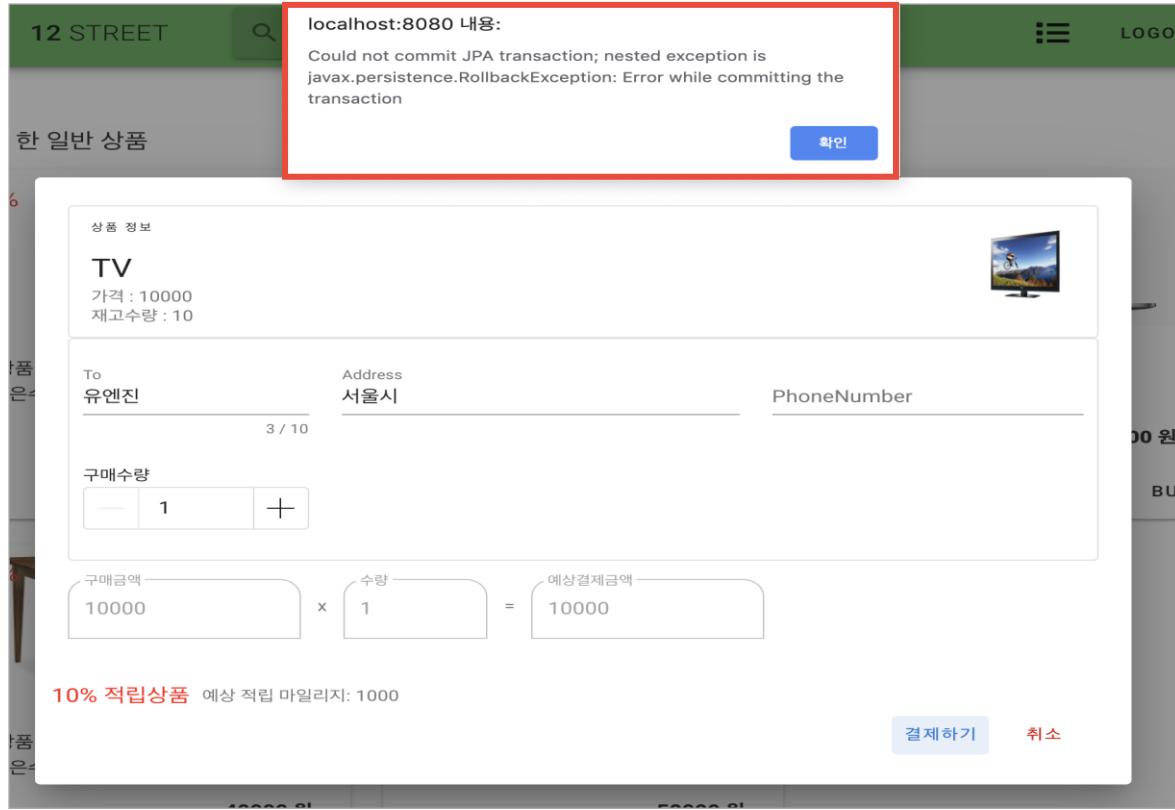
상품팀에서 API를 일방적으로 변경

소스 위치

products / src / main / java / com / example / template / ProductController.java

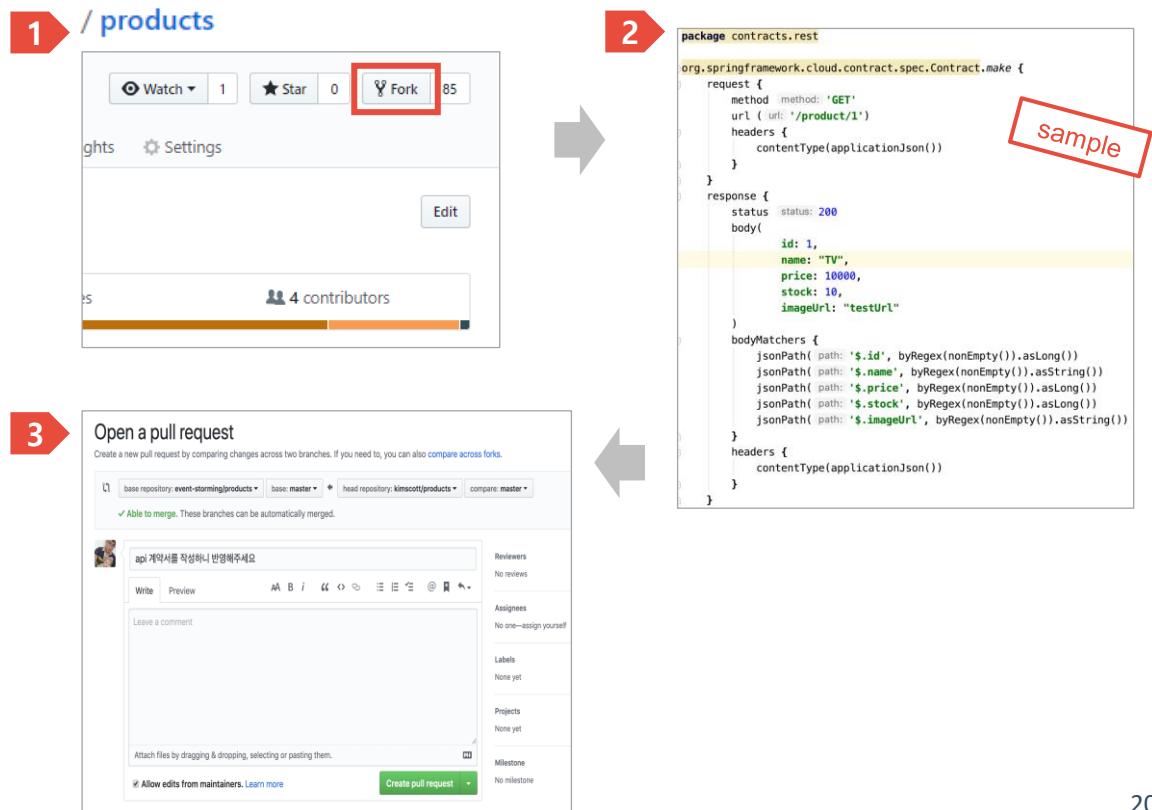
```
13  
14     @GetMapping("/product/{productId}") /item/{productId} 로 변경  
15     Product productStockCheck(@PathVariable(value = "productId") Long productId) {  
16         return this.productService.getProductById(productId);  
17     }  
18 }  
19
```

주문팀의 서비스 장애 발생



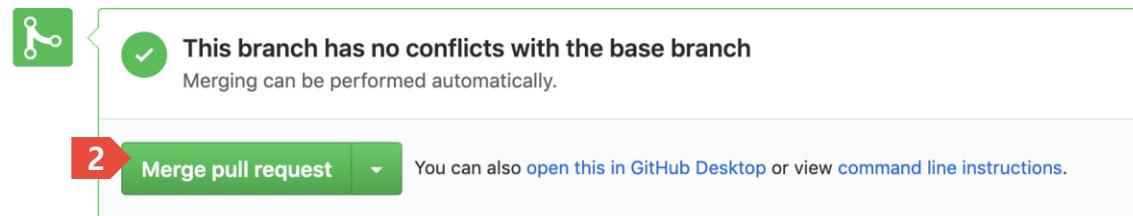
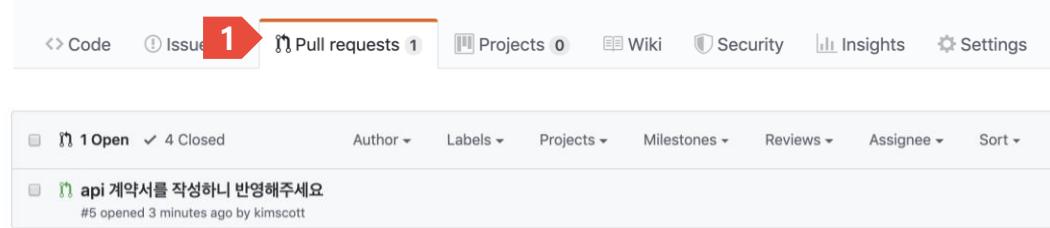
계약체결(1/2) - 주문팀에서 계약서 작성 후, 상품팀에 체결 요청

1. 상품팀 소스 복사 (포크 생성)
2. 주문팀이 계약서 생성
3. 주문팀에서 생성한 계약서 (productGet.groovy) 파일을 상품팀에 Pull request 요청함



계약체결(2/2) - 상품팀에서 계약서 수락

- 상품팀 : Pull Request 메뉴 선택
- 상품팀은 해당 계약서를 accept 하여 반영함
- 상품 서비스는 이제부터는 계약서를 안지켰을때 아예 배포가 안됨



계약체결 후, 상품팀은 계약 위반으로 배포 실패함

CloudBuild에서 mvn package 단계 실패

빌드 정보

상태	빌드 실패
빌드 ID	6db60b82-3b99-4aef-9870-ce85dce23cccd
이미지	-
트리거	master 브랜치에 푸시 (products)
소스	GitHub event-storming/products ↗ 9997fc8cebfff7e444d94b790378bb58dfc324362c ↗
Git 커밋	9997fc8cebfff7e444d94b790378bb58dfc324362c ↗
환경 변수	CLOUDSDK_COMPUTE_ZONE=asia-northeast1-a CLOUDSDK_CONTAINER_CLUSTER=standard-cluster-1
시작 시간	2019년 10월 29일 오후 3시 42분 59초 UTC+9
걸린 시간	1분 54초

로그가 너무 커서 이 페이지에 완전히 표시할 수 없습니다. 빌드 단계의 로그가 누락되거나 완료되지 않을 수 있습니다.

모두 펼치기

빌드 단계

build gcr.io/cloud-builders/mvn -- clean package	1분 46초
-----------------------------------------------------	--------

실패 LOG

```
Step #0 - "build": 2019-10-29 06:44:51.547 DEBUG 69 --- [           main] o.s.c.c.v.m.stream.StreamStubMessages      : Picked channel name is [event-out]
Step #0 - "build": [ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 4.976 s <<< FAILURE! - in com.example.template.MessagingTest
Step #0 - "build": [ERROR] validate_productChanged(com.example.template.MessagingTest)  Time elapsed: 0.221 s  <<< ERROR!
Step #0 - "build": com.jayway.jsonpath.PathNotFoundException: No results for path: ${'productName'}
Step #0 - "build":         at com.example.template.MessagingTest.validate_productChanged(MessagingTest.java:43)
Step #0 - "build": 
```

상품팀에서는 하위 호환성을 유지하며, 추가 API 제공

products / src / main / java / com / example / template / ProductController.java

```
@GetMapping("/item/{productId}")
Product productStockCheck(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}

@GetMapping("/product/{productId}")
Product productStockCheck1(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}
```

기존의 하위
호환성 API 유지

Table of content

Microservice and
Event-storming-Based
DevOps Project

1. DevOps Process Changes
2. Packaging Applications with Container (Docker)
3. Improving SLA with Container Orchestrator (Kubernetes)
4. Advanced Kubernetes Configurations
5. Anatomy of Kubernetes Architecture
6. Advanced Service Resiliency with Service Mesh (Istio)
7. Zero Down-Time (Canary) Deploy with Argo Rollouts and Istio
8. Contract Test with Spring Cloud Contract



THANKS!

Any Question?

You can find me at:

jyjang@uengine.org

<https://github.com/jinyoung>

<https://github.com/TheOpenCloudEngine>

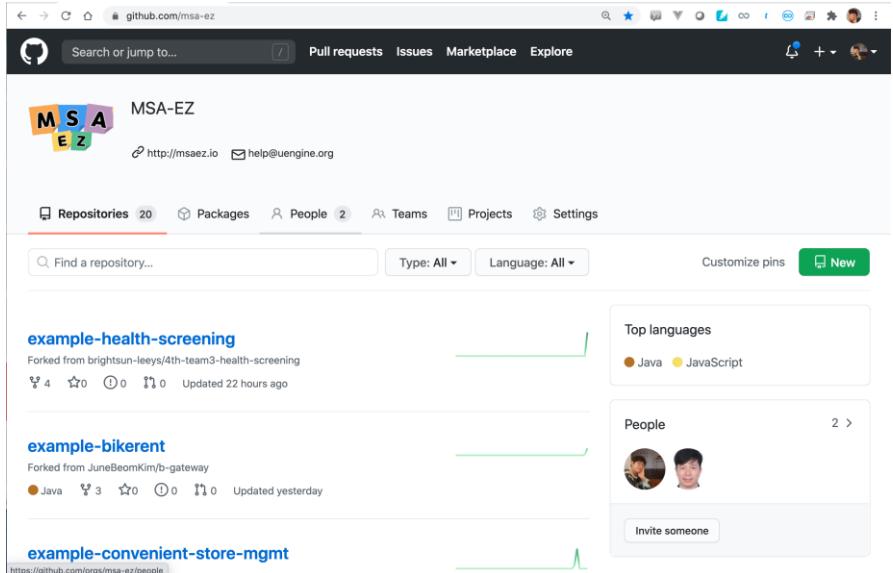
Recommended Books

- **Overall MSA Design patterns:**
<https://www.manning.com/books/microservices-patterns>
- **Microservice decomposition strategy:**
 - DDD distilled: <https://www.oreilly.com/library/view/domain-driven-design-distilled/9780134434964/>
 - Event Storming: https://leanpub.com/introducing_eventstorming
- **API design and REST:**
<http://pepa.holla.cz/wp-cont.../2016/01/REST-in-Practice.pdf>
- **Database Design in MSA:**
 - Lightly:
 - [https://www.confluent.io/wp-content/uploads/2016/08/Making Sense of Stream Processing Confluent 1.pdf](https://www.confluent.io/wp-content/uploads/2016/08/Making_Sense_of_Stream_Processing_Confluent_1.pdf)
 - Deep dive:
 - https://dataintensive.net/?fbclid=IwAR3OSWkhqRjLI9gBoMpbsk-QGxeLpTYVXIJVCSaw_A5eYrBDc0piKSm4pMM

github.com/msa-ez

Reference MSA Projects

github.com/msa-ez



The screenshot shows the GitHub organization page for `github.com/msa-ez`. The page features a header with the organization logo (MSA EZ), a search bar, and navigation links for Pull requests, Issues, Marketplace, and Explore. Below the header, there's a banner with the URL `http://msaez.io` and an email address `help@uengine.org`. The main content area displays three repository cards:

- example-health-screening**: Forked from `brightsun-leeyi/4th-team3-health-screening`. It has 4 stars, 0 forks, 0 issues, 0 pull requests, and was updated 22 hours ago.
- example-bikernet**: Forked from `JuneBeomKim/b-gateway`. It has 3 stars, 0 forks, 0 issues, 0 pull requests, and was updated yesterday.
- example-convenient-store-mgmt**: A link to the organization's people page.

On the right side, there are sections for Top languages (Java, JavaScript) and People, which lists two individuals with their profile pictures. There's also a button to "Invite someone".

MSA School

msaschool.io

The screenshot shows the 'MSA School 소개' (MSA School Introduction) page. The header includes the site's logo and navigation links for 소개, 계획단계, 설계/구현/운영단계, 참고자료, 커뮤니티 및 교육, and //Engine. The main content area features a large image of a person working on a laptop, overlaid with text: '험난한 MSA 구축 여정의 길라잡이'. To the right of the image is a circular diagram divided into six segments labeled 1단계 (분석), 2단계 (설계), 3단계 (구현), 4단계 (통합), 5단계 (배포), and 6단계 (운영). Below the diagram, the text 'Biz Dev Ops' is prominently displayed. A callout box highlights '계획에서 운영까지 End-to-End 학습을 위한 단계별 효율적인 실천법 제시'. On the left sidebar, there are links for MSA School 소개, 예제 도메인, 사용 플랫폼, 예제 애플리케이션 둘러보기, 관련 리소스, 유필리티 및 도구.