

# マイクロサービスを 위한 Istio & Kubernetes

Open Infra Days Korea 2018

June 29, 2018

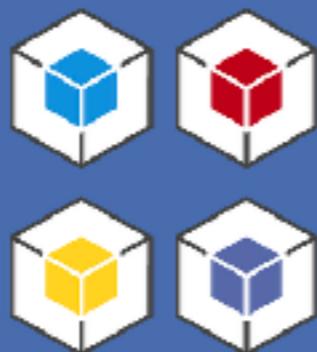
KIM CHUNGSUB

Creative Lab Director

 subicura

 subicura

 purpleworks



Istio 0.8



Kubernetes 1.10

## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

## Live Demo

Istio 데모

## 결론

마무리

---

개요\_소개

## 김충섭

퍼플웍스 개발자 (2010~)

Ruby on rails / Golang / Typescript / ReactJS

Docker / Kubernetes / Istio

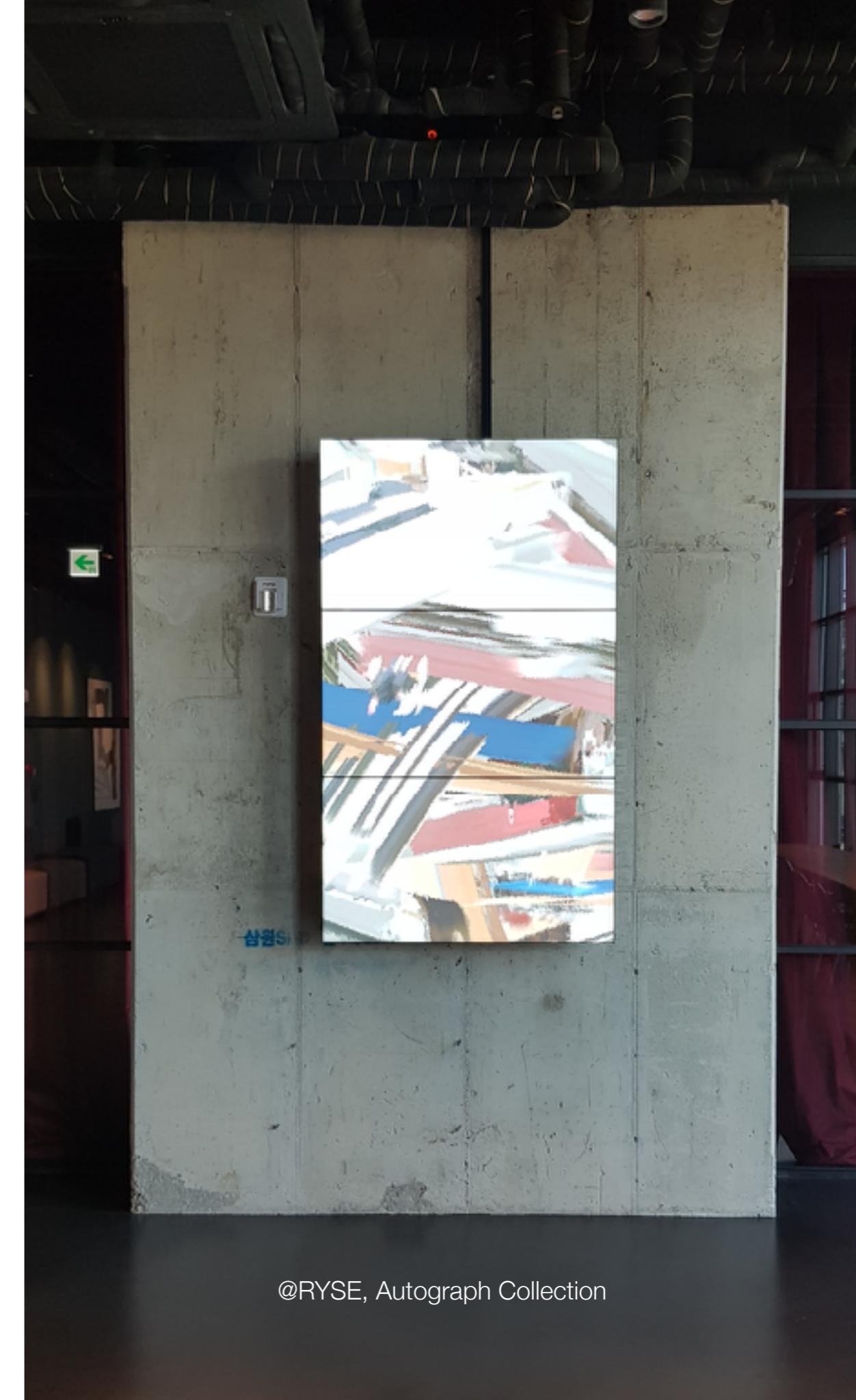
Openframeworks / OpenGL

Microservices / Devops에 관심

OpenContainer Korea (구 Docker Korea) 운영진

remotty 멤버

[subicura.com](http://subicura.com) 블로그 운영중



## 오늘 할 이야기

**마이크로서비스** microservices 도입에 관심이 있거나 이미 도입한 곳에서  
**컨테이너 스케줄러** container scheduler와 **서비스 메시** service mesh가 어떤 역할을 하는지  
그 중에 왜 **Kubernetes**와 **Istio**가 좋은 선택인지  
이야기합니다.

트랜드에 따라 도구를 선택하는 것이 아닌  
**현재 문제를 해결하는데 필요한 도구인지 선택할 수 있도록**  
정보를 전달합니다.

## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

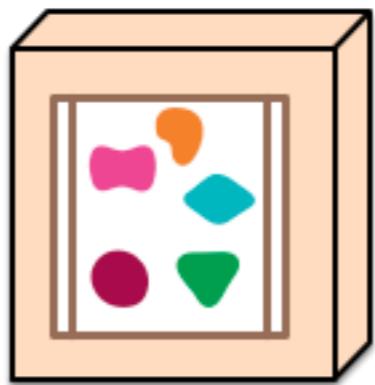
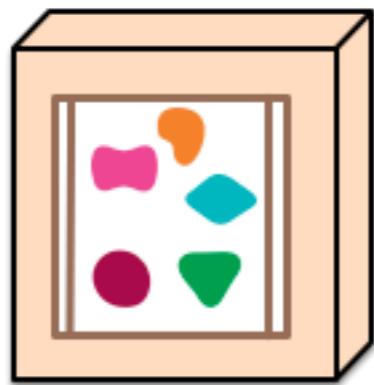
## Live Demo

Istio 데모

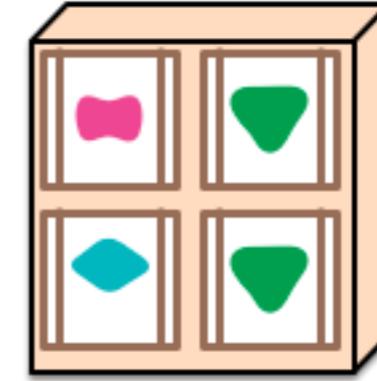
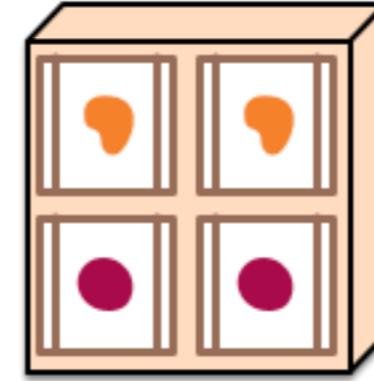
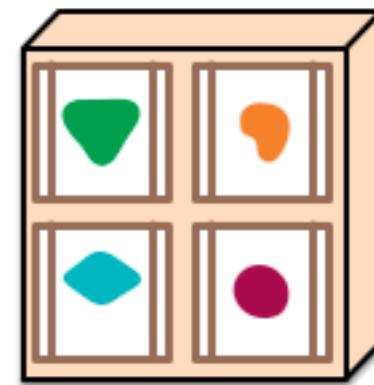
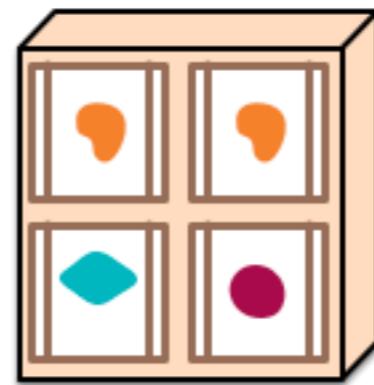
## 결론

마무리

## Monolithic vs Microservices



**monolithic**



**microservices**

## Ruby on Rails

- 인기 있었는 Ruby 기반 MVC 웹 프레임워크
- DRY (Don't Repeat Yourself) 철학
- 설정 보다는 관습 (Convention over configuration)
- 빠른 웹 개발 가능 (How to build a blog in 15 minutes with Rails / 2005)
- 다양한 플러그인 (gem package manager)
  - oAuth (facebook, twitter, google, naver, ...)
  - Background Job, CronJob
  - File upload
  - 메일 발송
- rubocop (정적코드분석)
- rspec (테스트 프레임워크)



## 마이크로서비스\_ 마이크로서비스 도입하기

# 배너 관리 시스템

오늘 < >

2018년 6월

주 월 2개월

구분	06.01	06.02	06.03	06.04	06.05	06.06	06.07	06.08	06.09	06.10	06.11	06.12	06.13	06.14	06.15	06.16	06.17	06.18	06.19	06.20	06.21		
	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00		
<b>▼ MOBILE</b>																							
1 slot	오후 12:00	배너_1																					
2 slot	오전 12:00	배너_2																					
3 slot	오후 12:00	배너_3																					
4 slot	오후 6:30	배너_4																					
5 slot	오전 9:00	배너_5																모전 9:00	배너_6				
6 slot	오전 9:00	배너_7																					
<b>▼ DESKTOP</b>																							
1 slot	오후 12:00	배너_1																					
2 slot	오전 12:00	배너_2																					
3 slot	오후 12:00	배너_3																					
4 slot	오후 6:30	배너_4																					
5 slot	오전 9:00	배너_5																	오전 9:00	배너_6			
6 slot	오전 9:00	배너_7																					

배너 관리

저장

---

마이크로서비스 \_ 마이크로서비스 도입하기

## 배너 관리 시스템

열심히 만들었는데 다른 서비스에서 쓸 순 없을까...?



---

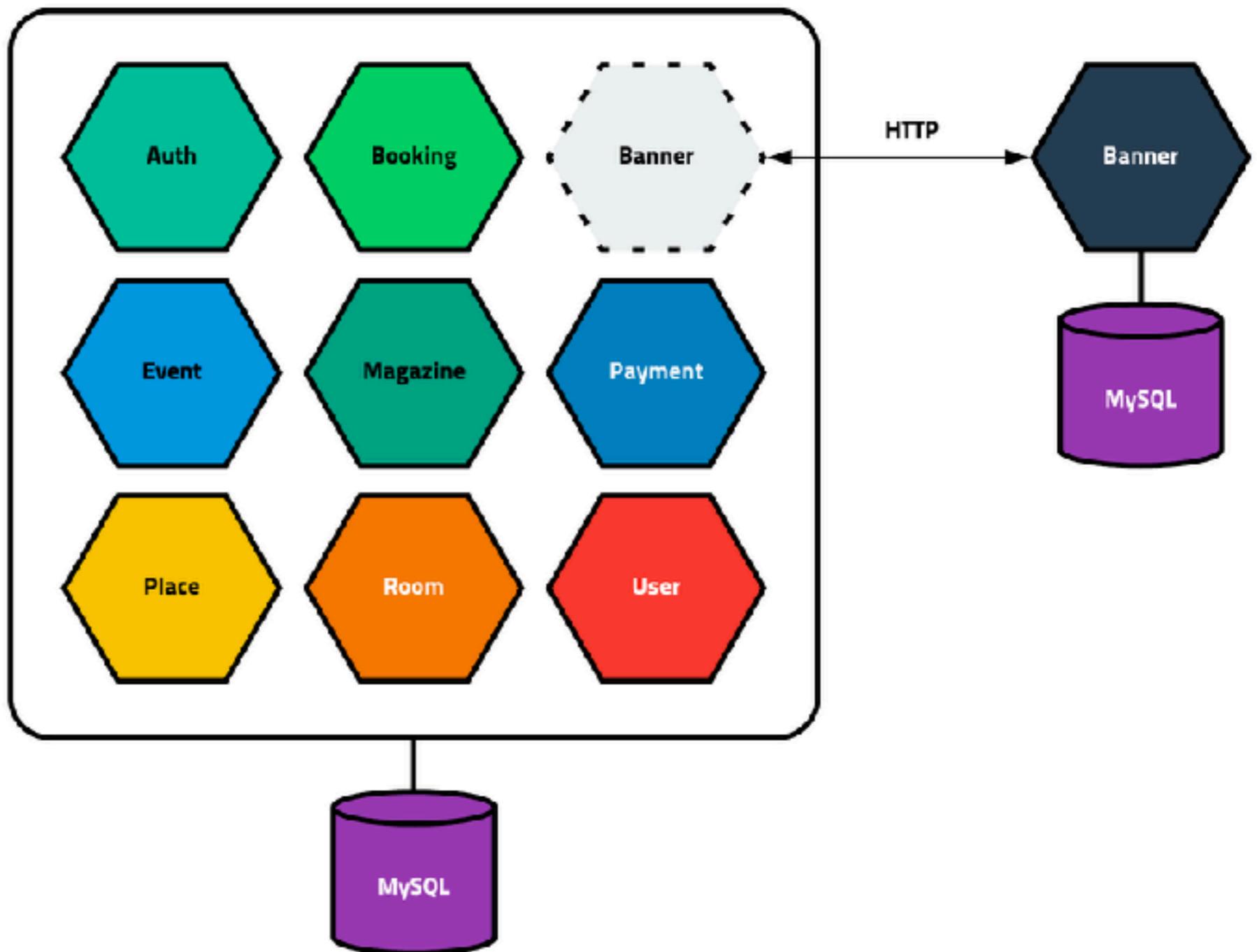
마이크로서비스\_ 마이크로서비스 도입하기

~~배너 관리 시스템~~ → 배너 관리 서비스!



## 마이크로서비스\_ 마이크로서비스 도입하기

# 서비스 분리

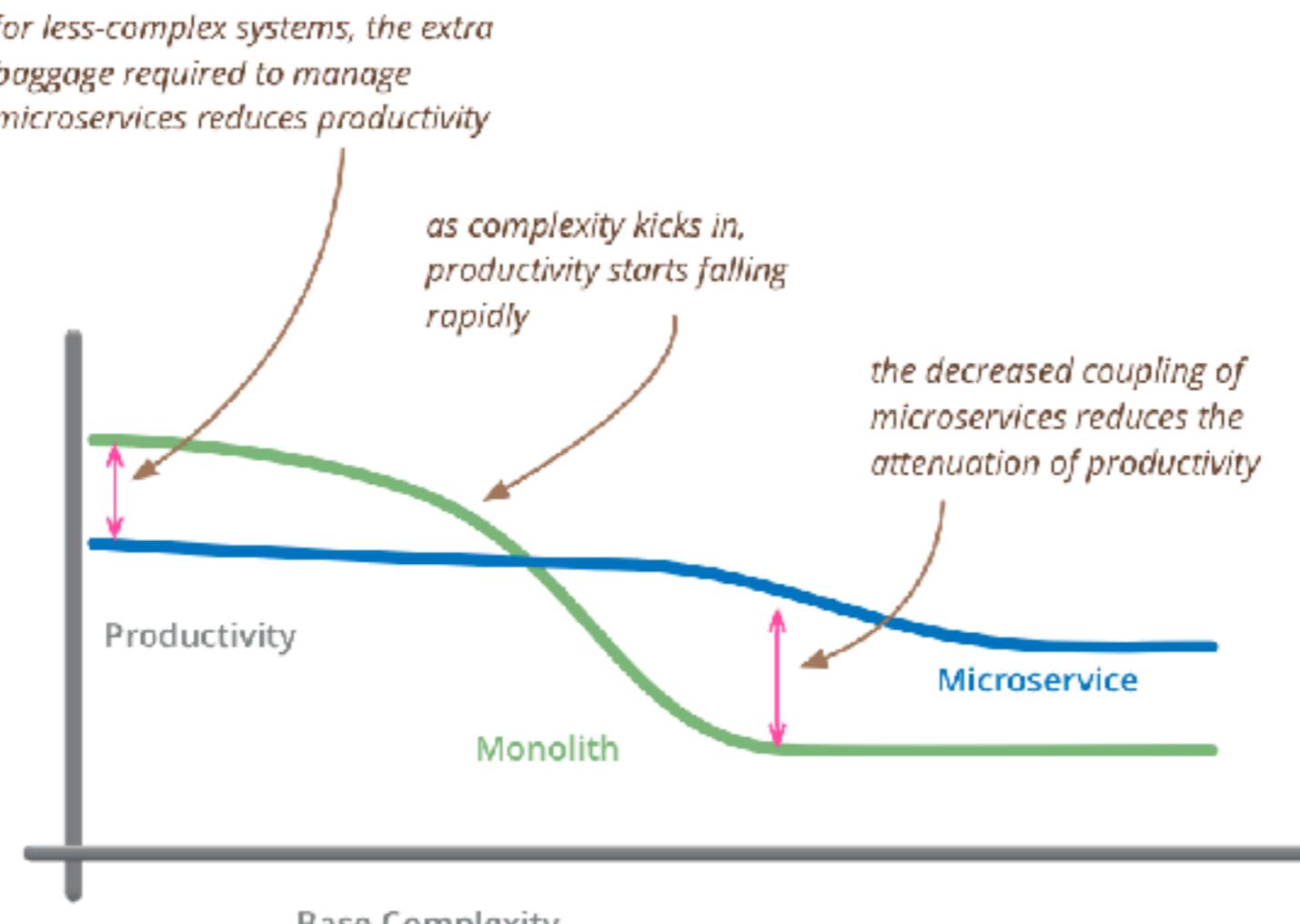


## 서비스가 커지면서 생기는 문제

- 작은 수정이 전체 서비스에 영향을 미침
- 전체 소스를 파악하기 힘듬
- 프레임워크 / 라이브러리 버전이 업데이트 되면 전체 검수 필요
- 빌드 속도 느려짐
- rubocop(정적분석도구) + rspec(테스트도구) 실행 시간이 길어짐
- QA 시간이 길어짐
- 새로운 언어 / 프레임워크 도입이 힘듬 #Polyglot

→ 빠르게 변하는 환경에 대응이 어렵고 빠른 개발 / 빠른 배포가 어려움

## 문제를 해결해야 할 시점



*but remember the skill of the team will outweigh any monolith/microservice choice*

## 서비스 분리

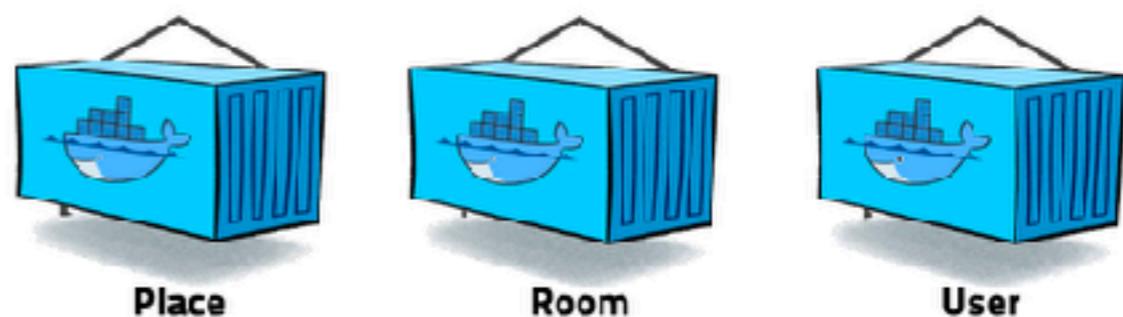
- 작게 쪼개고 분산하자
- 시스템 전체가 죽진 않는다
- 더 자주, 더 빠르게 배포할 수 있다
- 다양한 기술을 도입할 수 있다  
(golang, nodejs, python, ...)



## 마이크로서비스\_ 마이크로서비스 도입하기

### 도커

도커 컨테이너로 관리

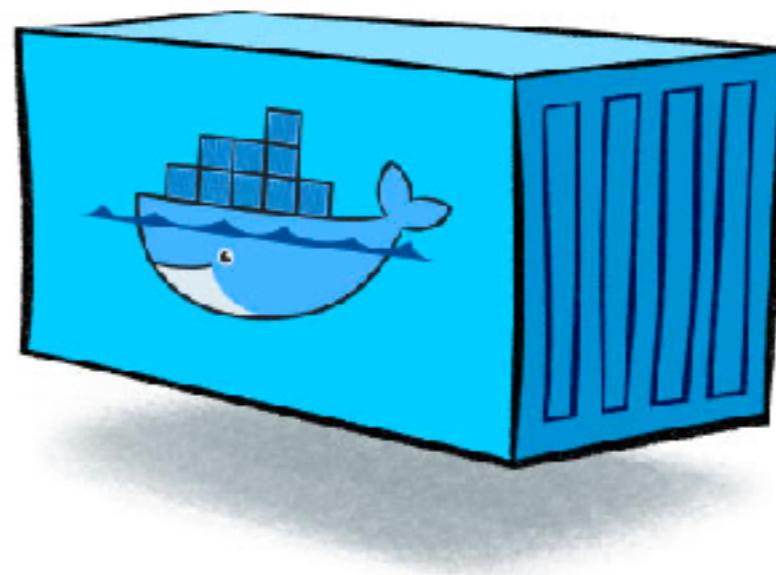


---

## マイクロサービス\_マイクロ서비스導入하기

# 컨테이너

Build, Ship, and Run Any App, Anywhere



```
docker run -d p 3000:3000 xxxx/awesome-banner
```

## 마이크로서비스

In short, the microservice architectural style is an approach to developing a single application as a suite of **small services**, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and **independently deployable** by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in **different programming languages** and use **different data storage** technologies.

- James Lewis and Martin Fowler

---

마이크로서비스 \_ 마이크로서비스 도입하기

## 마이크로서비스

복잡해지는 프로젝트의 단점을 해결하기 위한 방법의 하나

Netflix / Amazon / Ebay / Uber / Twitter / Coupang / 11st / ...에서 사용중  
자세한 이론은 생략!

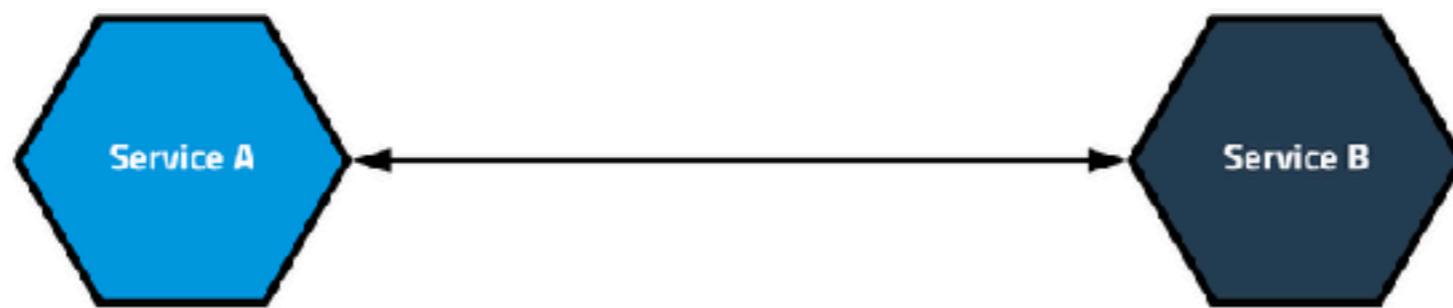
---

마이크로서비스\_ 마이크로서비스 도입하기

## 마이크로서비스 도입 성공?

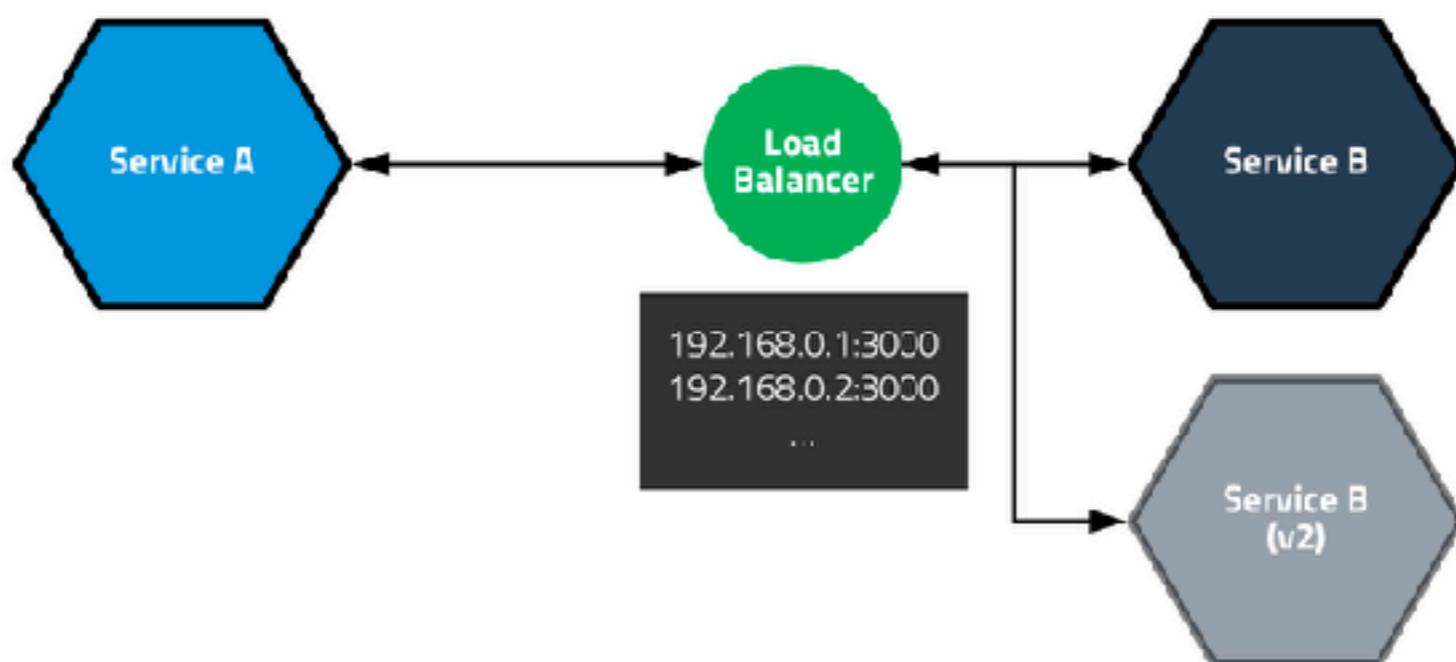


## 마이크로서비스와 운영



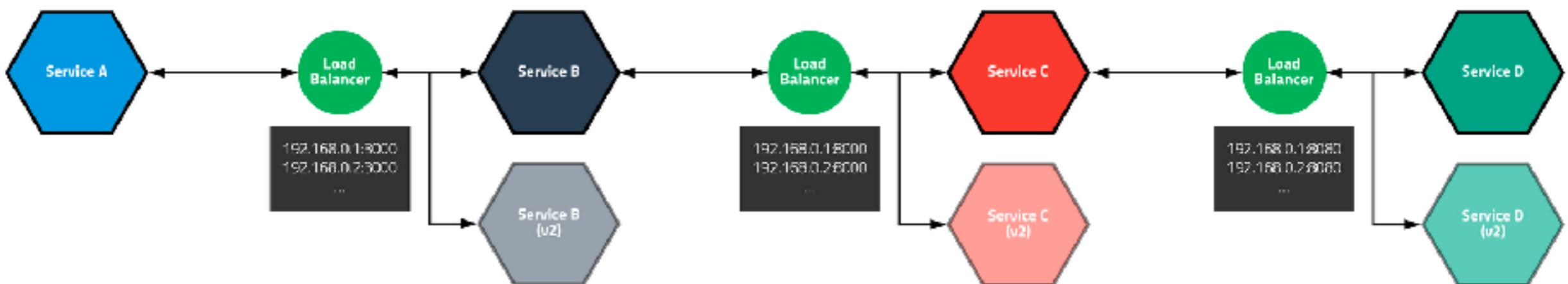
## 마이크로서비스와 운영

운영 복잡도/비용 증가



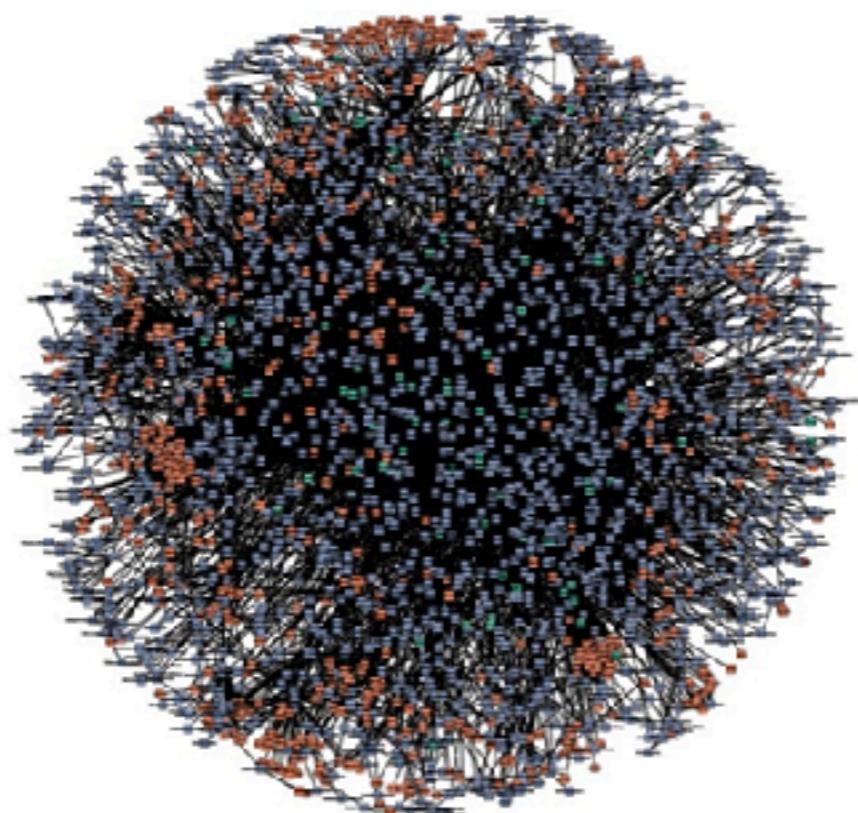
## 마이크로서비스와 운영

운영 복잡도/비용 증가



## 마이크로서비스 \_ 마이크로서비스 도입하기

# マイクロサービスと運用



amazon.com



NETFLIX

## 마이크로서비스 도입 후 발생한 문제점

- 자동 배포 / 확장 (Deployment)
- 서비스 검색 (Service Discovery)
- 부하 분산 (Load Balance)
- 장애 복원 (Resilient)
- 설정 공유 (Configuration Management)
- 로그 (Logging)
- 보안 (Security)
- 라우팅 설정 (Routing Management)
- 리소스 모니터링 (Metric Monitoring)
- 요청 / 응답 추적 (Tracing)
- 부하 방지 (Circuit breaker)
- 장애 허용 (Fault tolerance)

## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

## Live Demo

Istio 데모

## 결론

마무리

---

컨테이너 오케스트레이션 \_ 컨테이너 오케스트레이션이란?

## 컨테이너 오케스트레이션

여러 대의 서버에서 여러 개의 컨테이너를 편리하게 관리해주는 도구

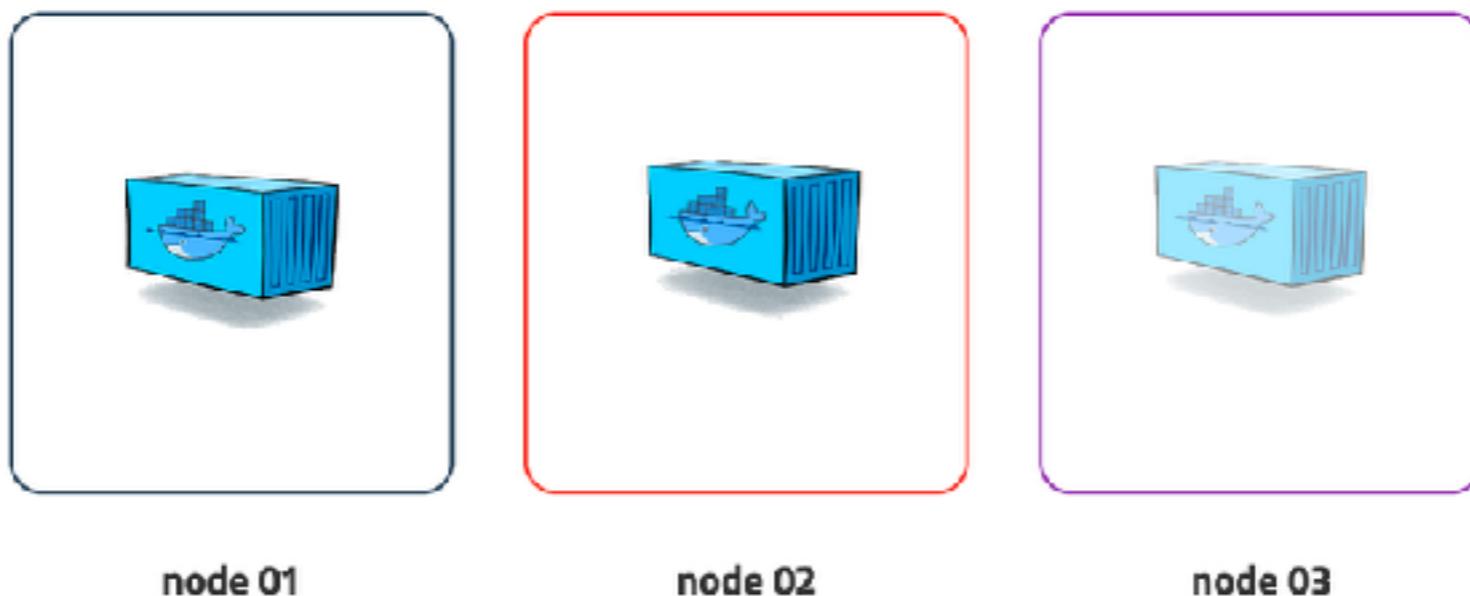
- 스케줄링 (Scheduling)
- 클러스터링 (Clustering)
- 서비스 검색 (Service Discovery)
- 부하 분산 (Load Balance)
- 확장 / 축소 (Scale up & down)
- 복구 (Rollback)
- 상태 확인 (Healthcheck)
- 설정 관리 (Config & Secret)

---

컨테이너 오케스트레이션 \_ 컨테이너 오케스트레이션이란?

## 스케줄링 Scheduling

컨테이너를 적절한 서버에 배포하고 상태를 유지하게 도와줌

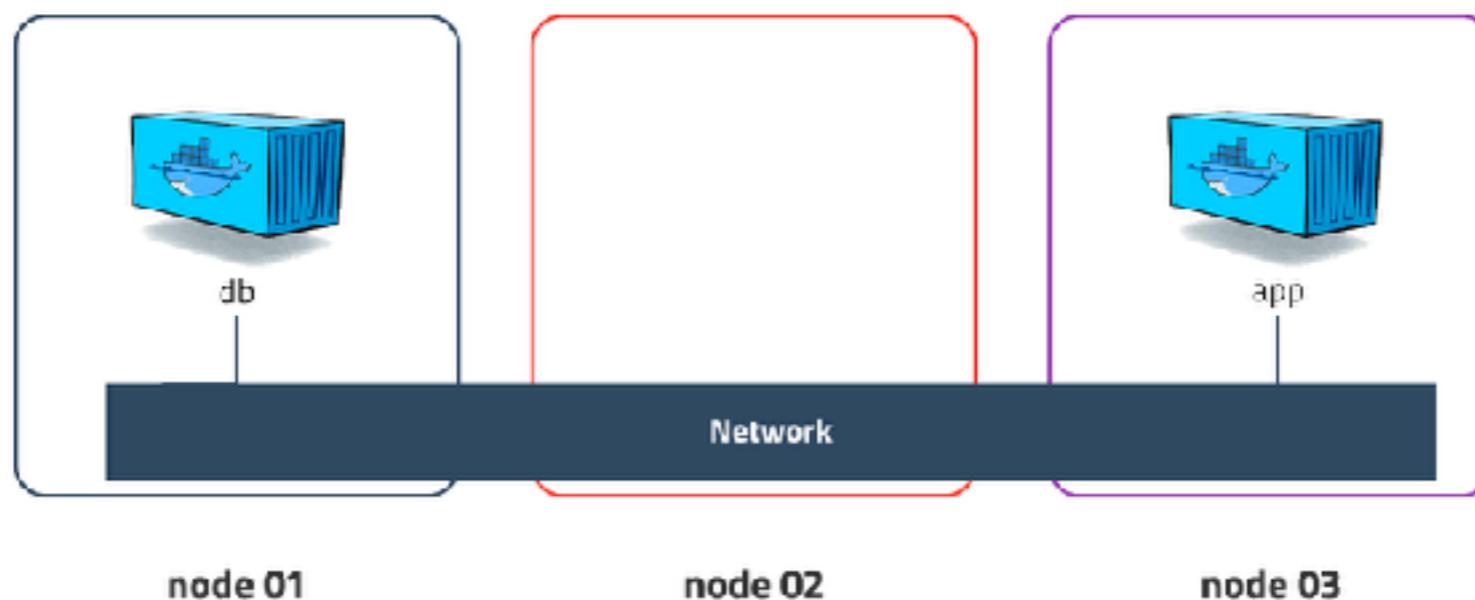


---

## 컨테이너 오케스트레이션 \_ 컨테이너 오케스트레이션이란?

### 클러스터링 Clustering

여러개의 서버를 하나의 서버처럼 관리하고 가상 네트워크를 이용해 바로 접근할 수 있게 도와줌

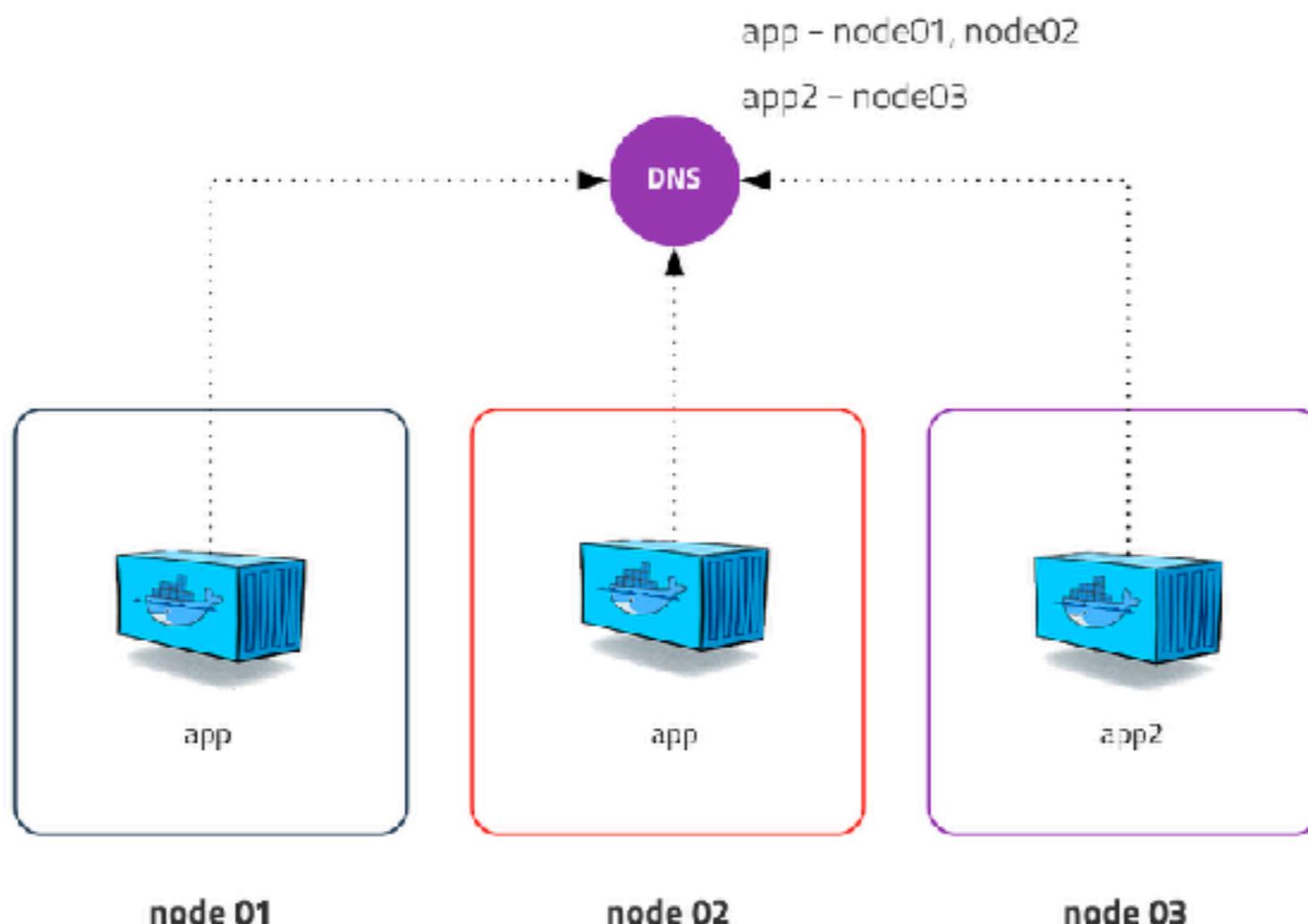


---

## 컨테이너 오케스트레이션 \_ 컨테이너 오케스트레이션이란?

## 서비스 검색 Service Discovery

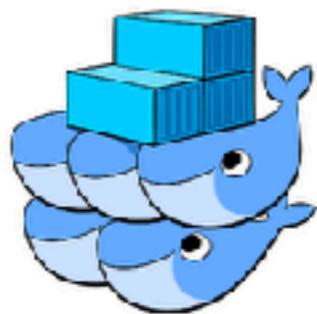
컨테이너(서비스)의 IP / Port 정보를 저장하고 서비스 레지스트리(DNS, ...)에 저장



---

컨테이너 오케스트레이션 \_ 컨테이너 오케스트레이션이란?

## 컨테이너 오케스트레이션 도구



## Docker swarm vs Kubernetes

Docker Swarm	Kubernetes
도커 컨테이너 스케줄링 O	도커(+rkt, ...) 컨테이너 스케줄링 O
클러스터링 (overlay network)	클러스터링 (다양한 네트워크 플러그인)
서비스 검색 O (DNS)	서비스 검색 O (DNS, Endpoint)
컨테이너를 한개씩 관리	여러개의 컨테이너를 하나의 팟으로 관리
Ingress controller 지원 X (3rd party)	Ingress controller 지원 O
Access Control 지원 (enterprise only)	RBAC 지원
작은 플러그인	다양한 플러그인
설치 쉬움	설치 어려움

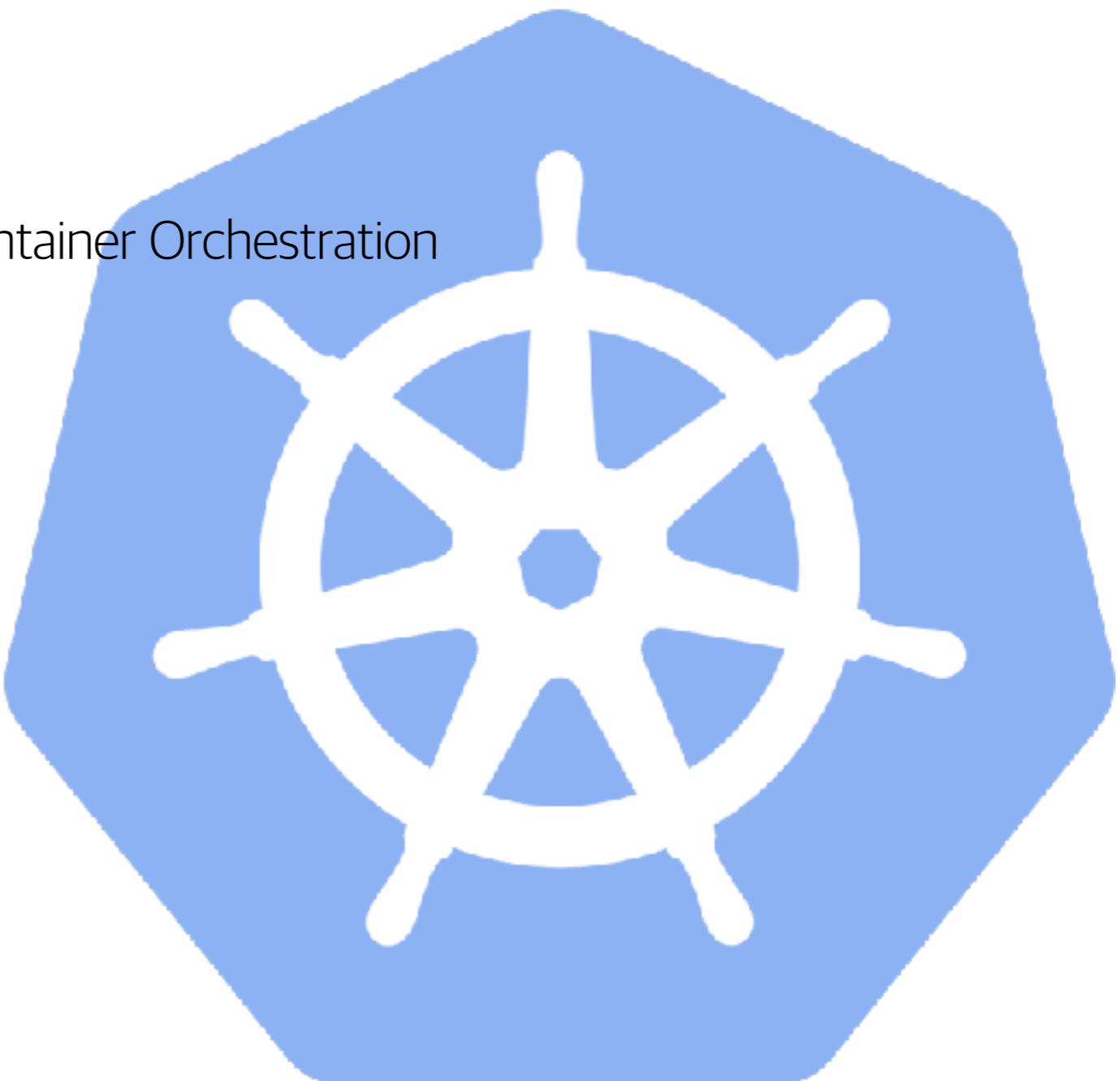
## 사실상 표준 De facto

**Kubernetes** Production-Grade Container Orchestration

선언적인(Declaratively) 서비스 관리

그리스어로 조타수 또는 조종사

K8s로 줄여서 사용

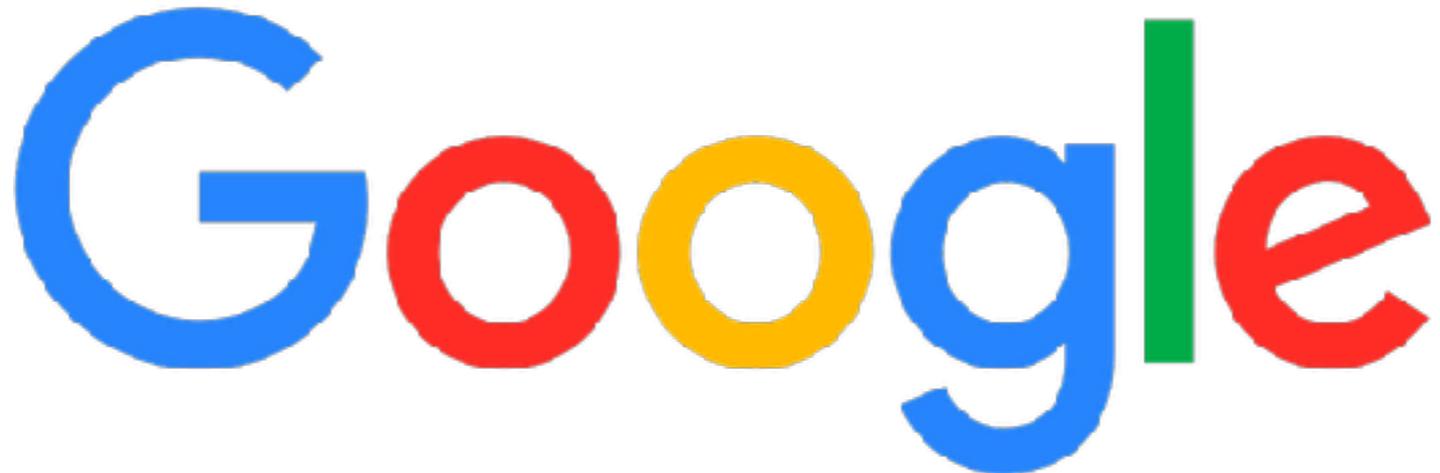


---

컨테이너 오케스트레이션 \_ Kubernetes

by Google

1주일에 20억개의 컨테이너를 생성하는 google이  
컨테이너 배포 시스템으로 사용하던 borg를 참고하여 go언어로 만든 오픈소스



## 클라우드 지원

**AWS** EKS or kops or ...

**Google Cloud** GKE

**Azure** Container Service

## 업데이트

**v1.11** 2018/6/27

**v1.10** 2018/3/26

**v1.9** 2017/11/9

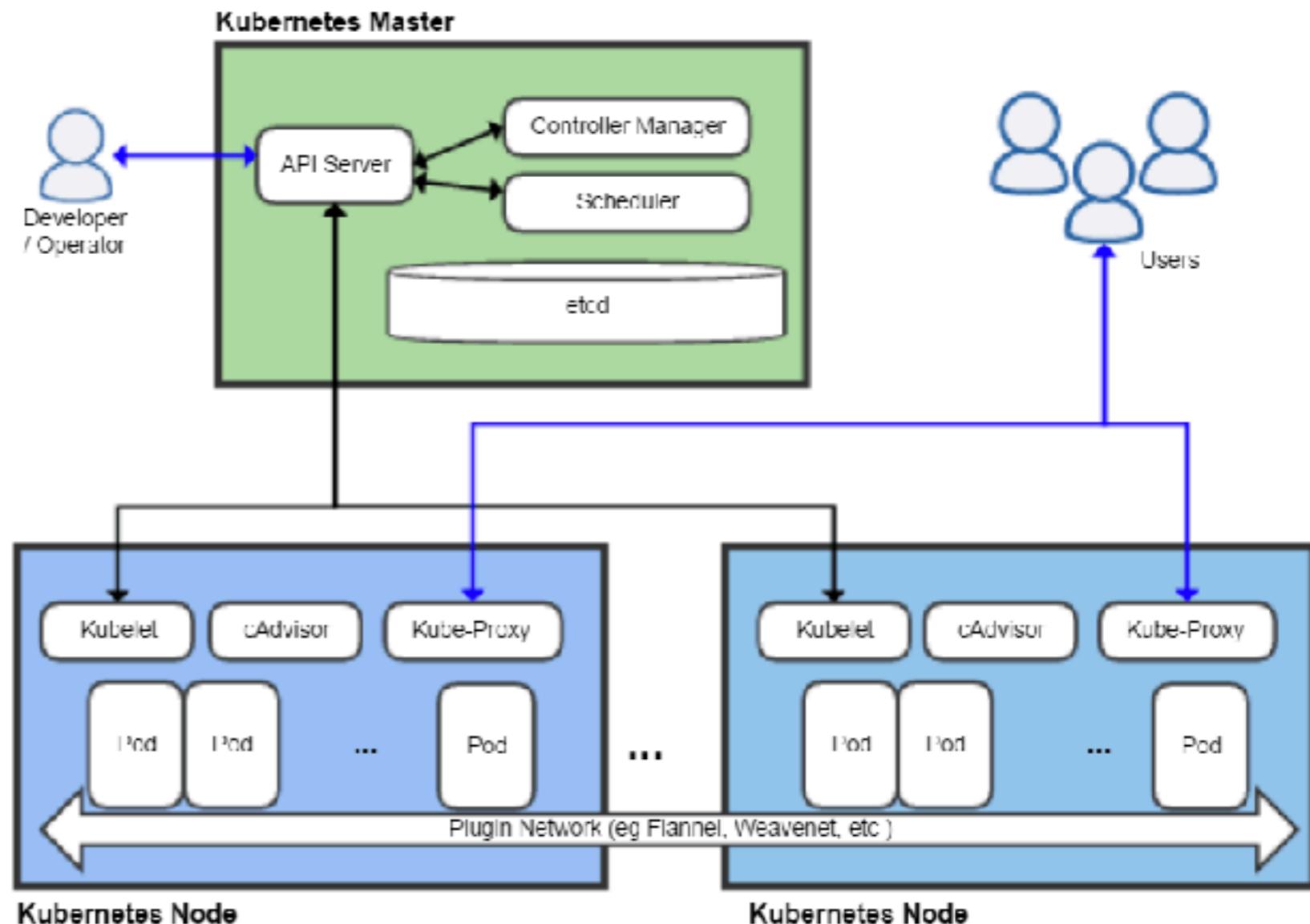
**v1.8** 2017/8/25

**v1.7** 2017/6/14

**v1.6** 2017/3/23

## 아키텍처

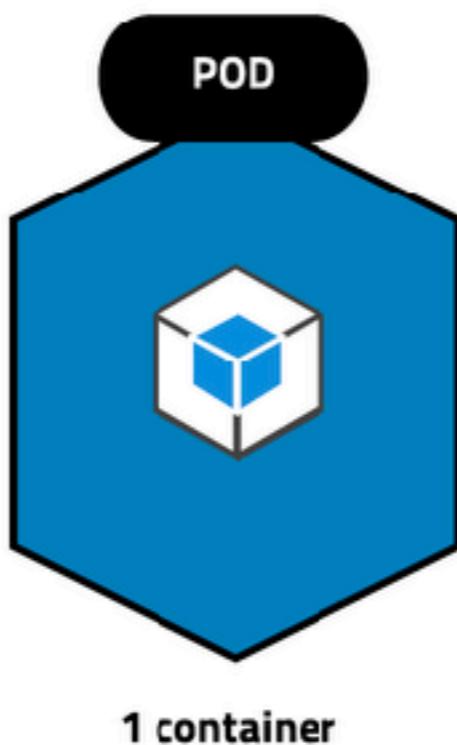
Master - Node



## Pod

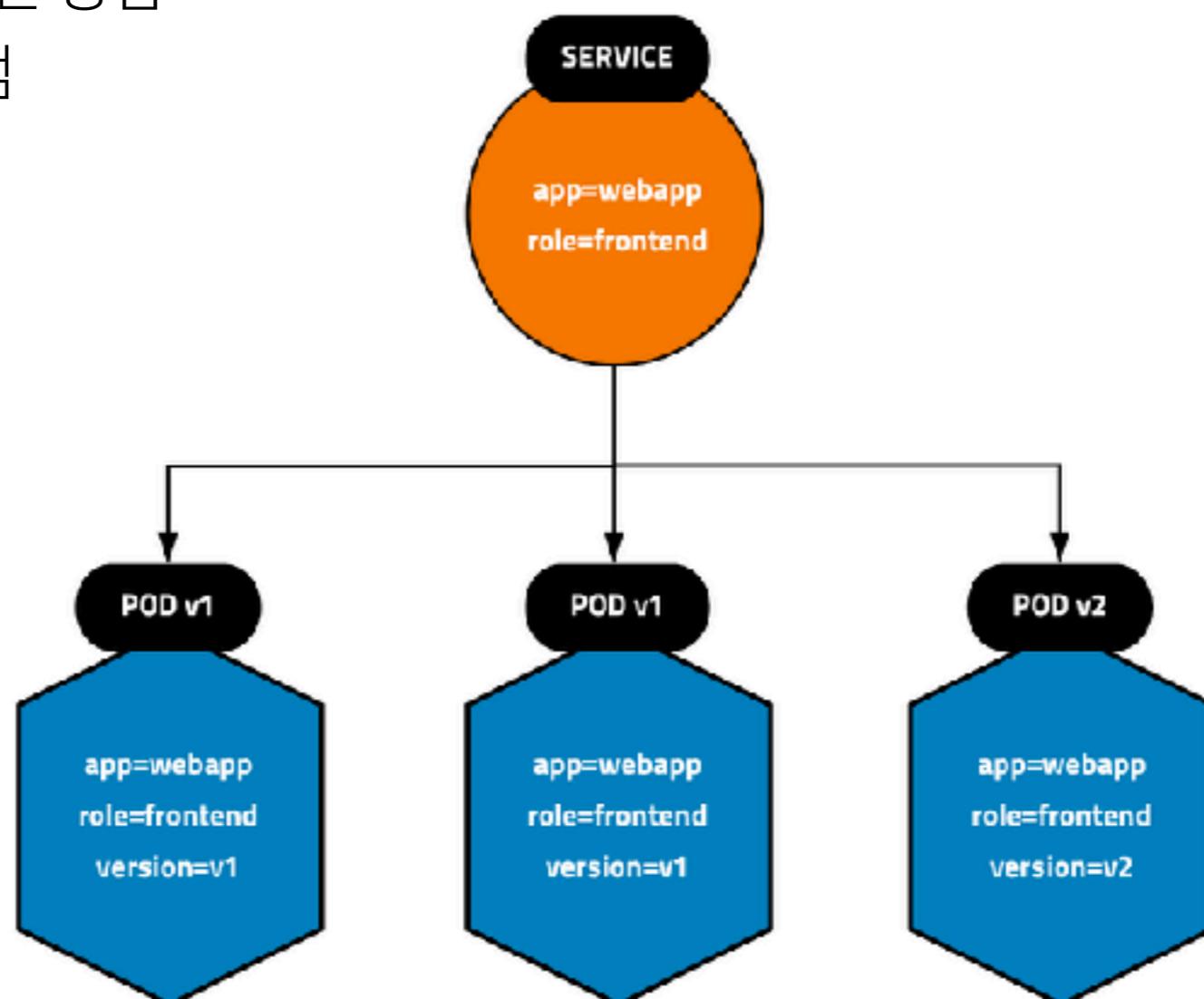
최소 배포 단위 - 1개 이상의 컨테이너 포함

Pod 내에서 컨테이너는 localhost로 접근하고 볼륨을 공유



# Service

Pod과 Pod이 통신하는 방법  
로드밸런서 기능은 덤



## 해결한 문제 + 남은 문제

- 자동 배포 / 확장 (Deployment)
  - 서비스 검색 (Service Discovery)
  - 부하 분산 (Load Balance)
  - 장애 복원 (Resilient)
  - 설정 공유 (Configuration Management)
  - 로그 (Logging)
  - 보안 (Security)
  - 라우팅 설정 (Routing Management)
  - 리소스 모니터링 (Metric Monitoring)
  - 요청 / 응답 추적 (Tracing)
  - 부하 방지 (Circuit breaker)
  - 장애 허용 (Fault tolerance)
- 
- deployment ✓
- service communication

## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

## Live Demo

Istio 데모

## 결론

마무리

## 서비스 간 통신 트랜드

### 2000년대 - SOA

- SOAP, XML, binary format
- 무거운 메시지 큐 (Heavyweight MQ)
- Enterprise Service Bus (ESB)

### 2010년대 - Microservices

- HTTP, JSON, REST, gRPC
- 가벼운 메시지 큐 (Lightweight MQ)
- Event sourcing / CQRS

## 앞서가는 회사들 전략

### Netflix

- Karyon + HTTP/JSON or RxNetty RPC + Eureka + Hystrix + Zuul + Ribbon + ...

### Twitter

- Finagle + Thrift + ZooKeeper + Zipkin

### AirBnB

- HTTP/JSON + SmartStack + ZooKeeper + Charon/Dyno

### Google

- gRPC, Stubby + GSLB + GFE + Dapper

---

서비스 메시 \_ service to service communication

## 서비스 메시로

기존 라이브러리/서비스의 장점을 모아 조금 다른 방식으로 구현



Eureka Service registry	Ribbon Client Side LB	Hystrix Circuit Breaker	Zipkin Distributed Tracing	Prometheus Monitoring	Grafana Data Visualization
----------------------------	--------------------------	----------------------------	----------------------------------	--------------------------	-------------------------------

---

## 서비스 메시\_서비스 메시란?

# 서비스 메시|Service mesh

A service mesh is a configurable **infrastructure layer for a microservices application**. It makes **communication between service** instances flexible, reliable, and fast. The mesh provides service discovery, load balancing, encryption, authentication and authorization, support for the circuit breaker pattern, and other capabilities.

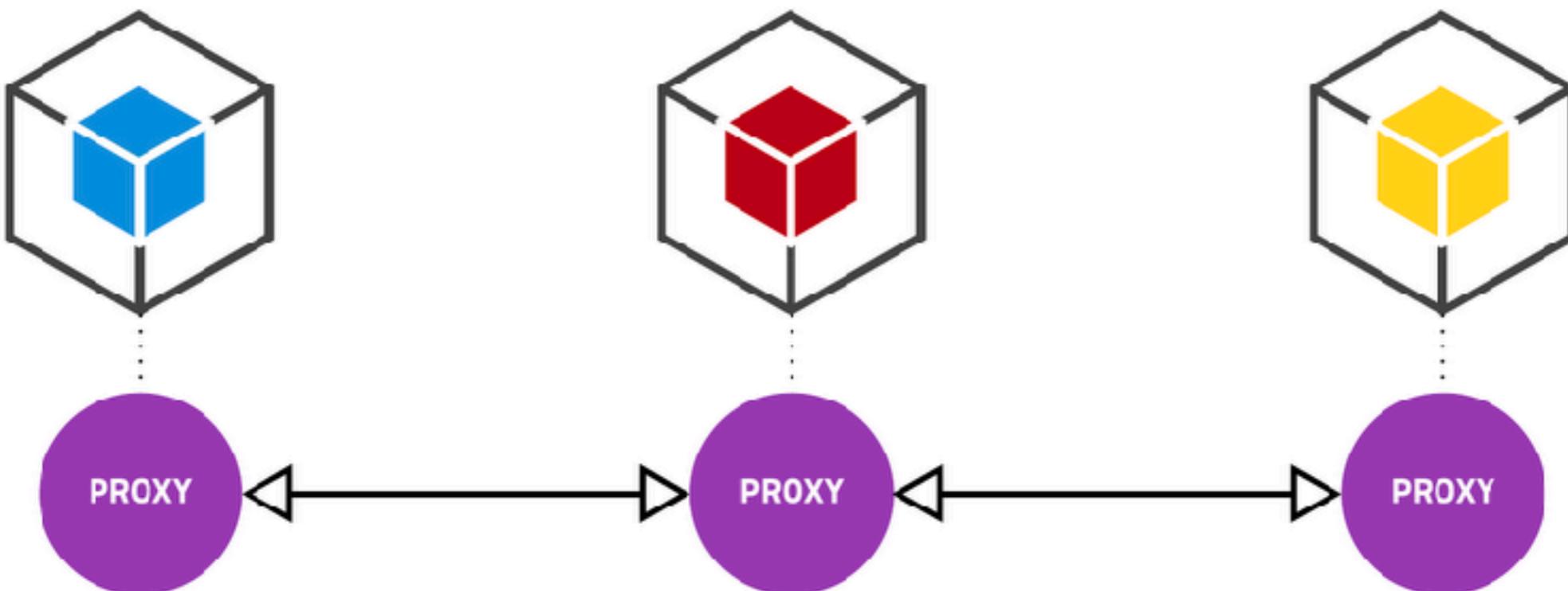
- nginx

---

서비스 메시\_서비스 메시란?

## 서비스 메시|Service mesh

서비스 간 통신을 처리하기 위한 전용 인프라 계층

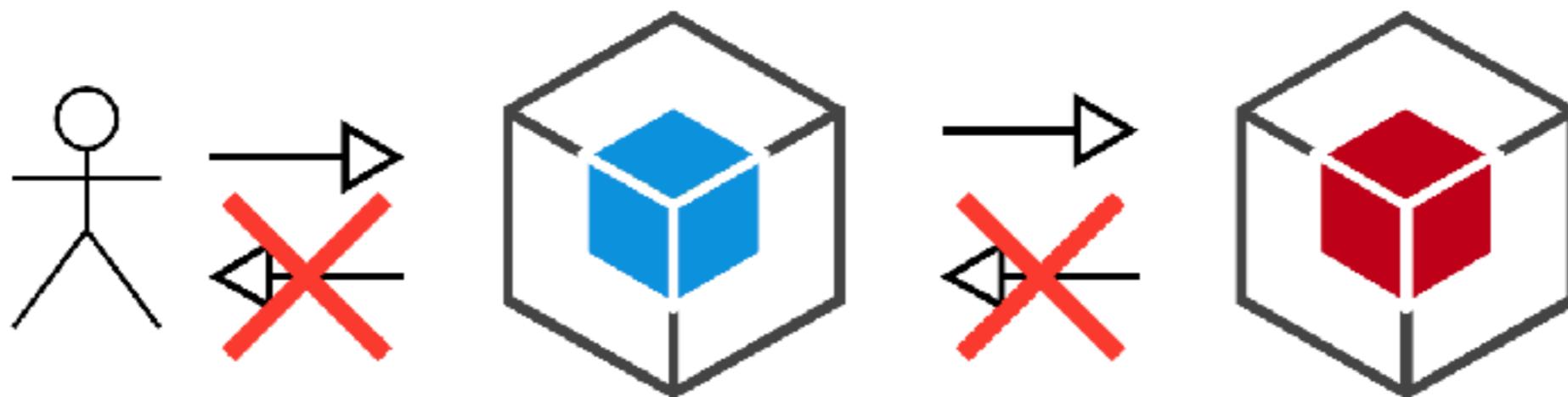


---

서비스 메시\_ 서비스 메시란?

## 흔한 서비스 오류

내부 서비스 중 하나가 오류가 발생했을 때

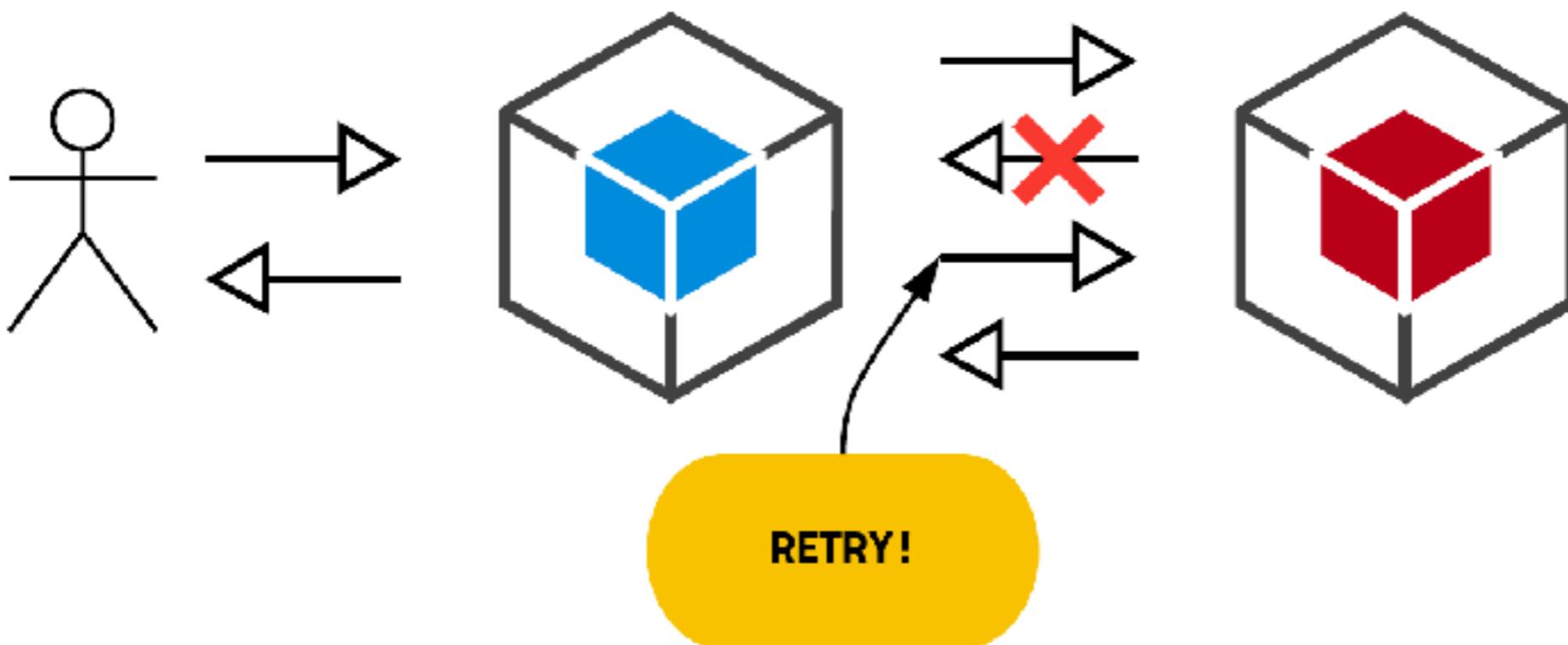


---

서비스 메시\_ 서비스 메시란?

## 재시도 정책Retry policy

서비스 호출 시 오류가 발생하면 몇번 더 재시도



---

서비스 메시\_ 서비스 메시란?

## 재시도 정책Retry policy

코드로 작성한다면

```
begin
  retries ||= 0
  @banners = BannerService.get_all
rescue
  retry if (retries += 1) < 3
end
```

ruby

서비스 메시\_ 서비스 메시란?

## 재시도 정책Retry policy

라이브러리를 사용한다면

```
// ...
@EnableRetry
public class MyApp {
// ...
    @Retryable
    @RequestMapping("/hit-api")
    public Object hitApi() {
// ...
```

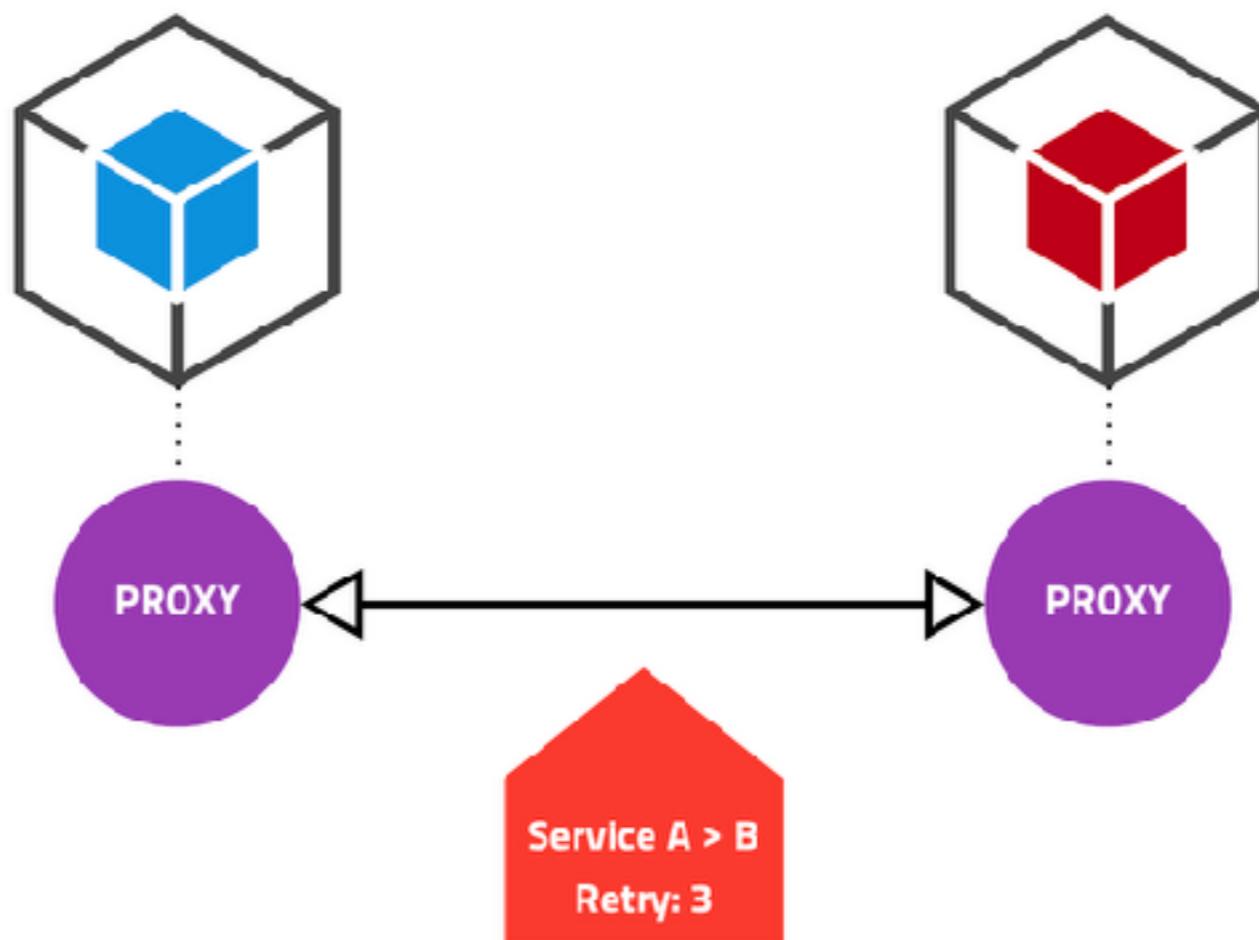
java

---

서비스 메시\_ 서비스 메시란?

## 재시도 정책Retry policy

서비스메시를 사용한다면?!



## Envoy

**Lyft**에서 C++로 만든 서비스간 통신을 처리하는 고성능 프록시

- 어플리케이션 수정 없이 각 어플리케이션의 요청과 응답을 투명하게 처리
  - 다양한 언어로 작성된 어플리케이션들과 함께 사용할 수 있음 C++, Go, PHP, Ruby..
- L3/L4 filter
- HTTP L7 filter
- HTTP/2, gRPC, MongoDB, DynamoDB
- 서비스 검색 (Service Discovery)
- 헬스 체크 (Health Checking)
- 향상된 부하 분산 (Advances load balancing)
- 관찰가능성 (Observability) - Metrics, Tracing, ...
- 동적 설정 (Dynamic configuration)

## Battle Test @lyft

100+ services

10,000+ VMs

2M req/s



## Used by

**lyft**

**Google**

**IBM**

**verizon<sup>✓</sup>**

**ebay**



**Microsoft**

**stripe**



**Tencent 腾讯**

**twilio**



**NETFLIX**

**Pinterest**

**Medium**

**Booking.com**



**Square**

**airbnb**

**YAHOO! JAPAN**

**vmware<sup>®</sup>**

**cookpad**

---

## 서비스 메시\_Istio

### Istio

사이드카(Sidecar) 패턴을 이용한 서비스메시 구현체

- Google, IBM, Lyft에서 만든 오픈소스 프로젝트
- Envoy의 확장된 버전을 이용하여 서비스간 통신을 처리
- 쿠버네티스에 최적화되어 있고 다른 스케줄러도 지원 예정
- 그리스어로 뜻/항해(sail)에서 따옴
- 2017년 5월에 개발
- 2018년 6월 v0.8 릴리즈 / 2018년 1.0 목표?
- Route Rules v1Alpha3 API

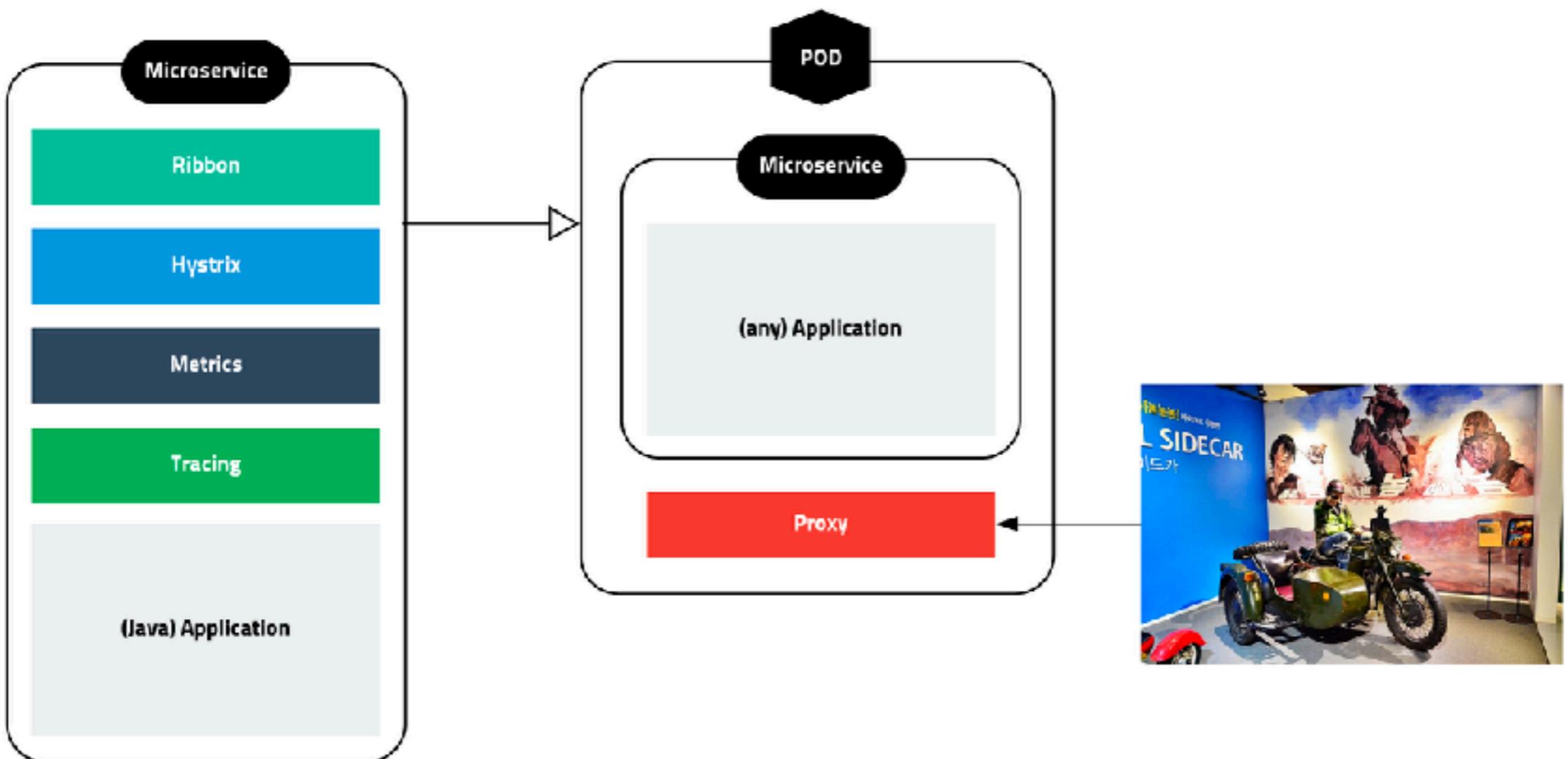


## Istio가 제공하는 기능

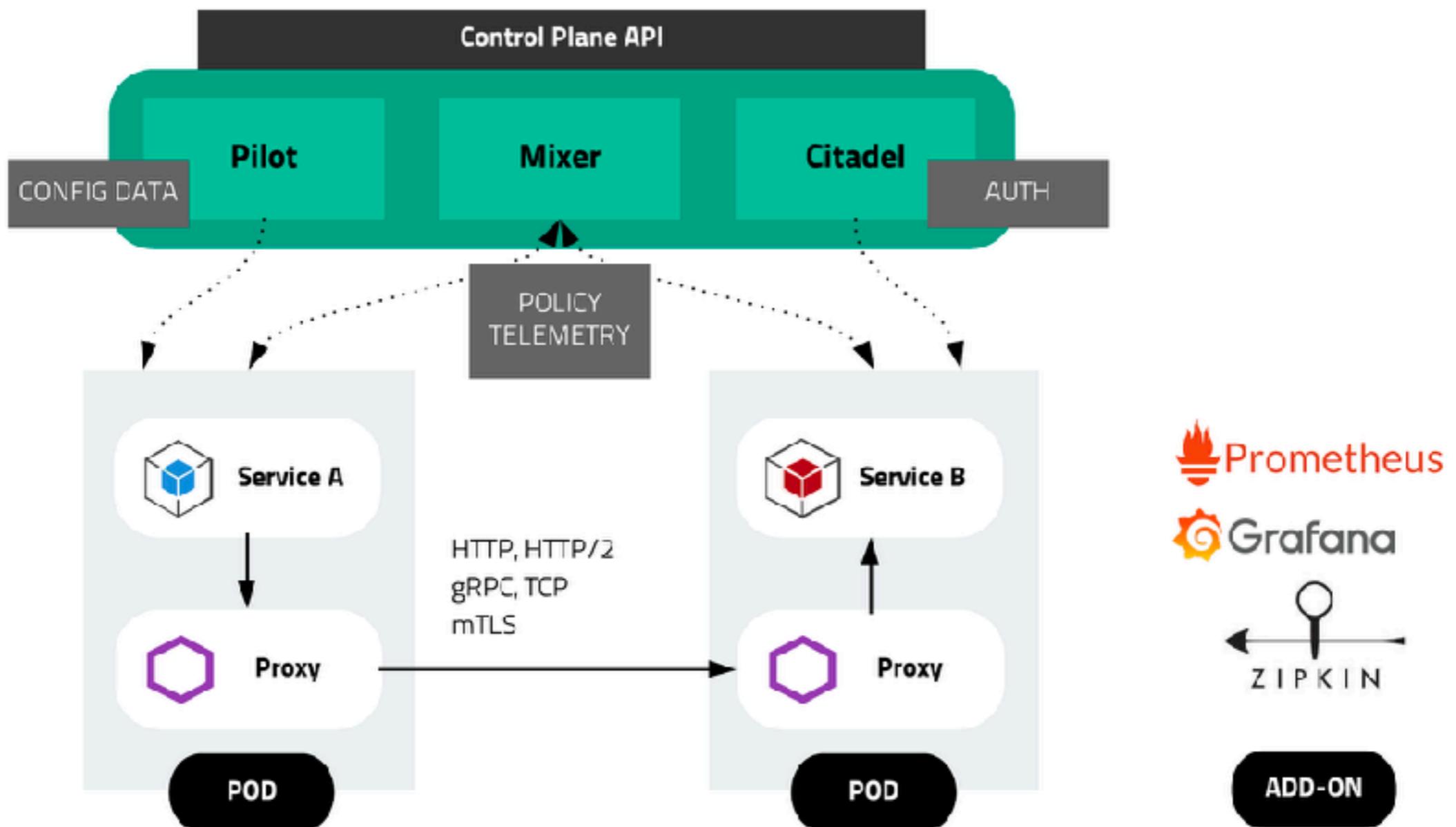
- Dynamic service discovery
- Load Balancing
- TLS termination
- HTTP/2
- gRPC proxying
- Traffic Control
- Circuit Breaker
- Monitoring, Distributed tracing
- Fault Injection
- ...

## Focus

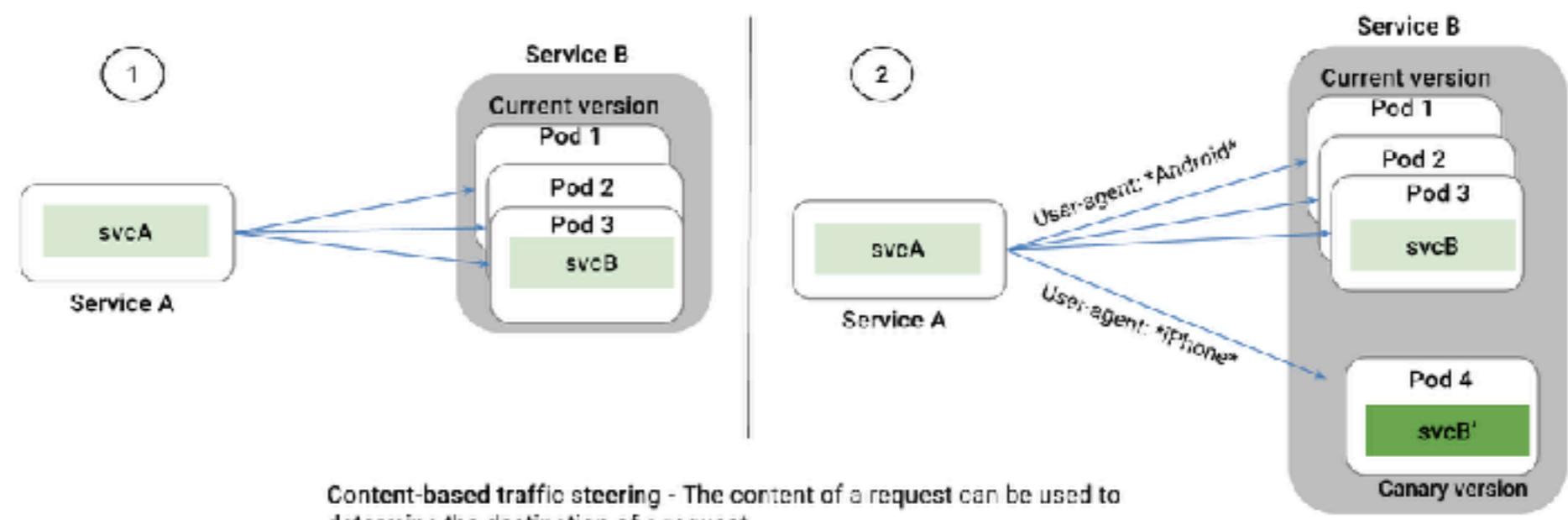
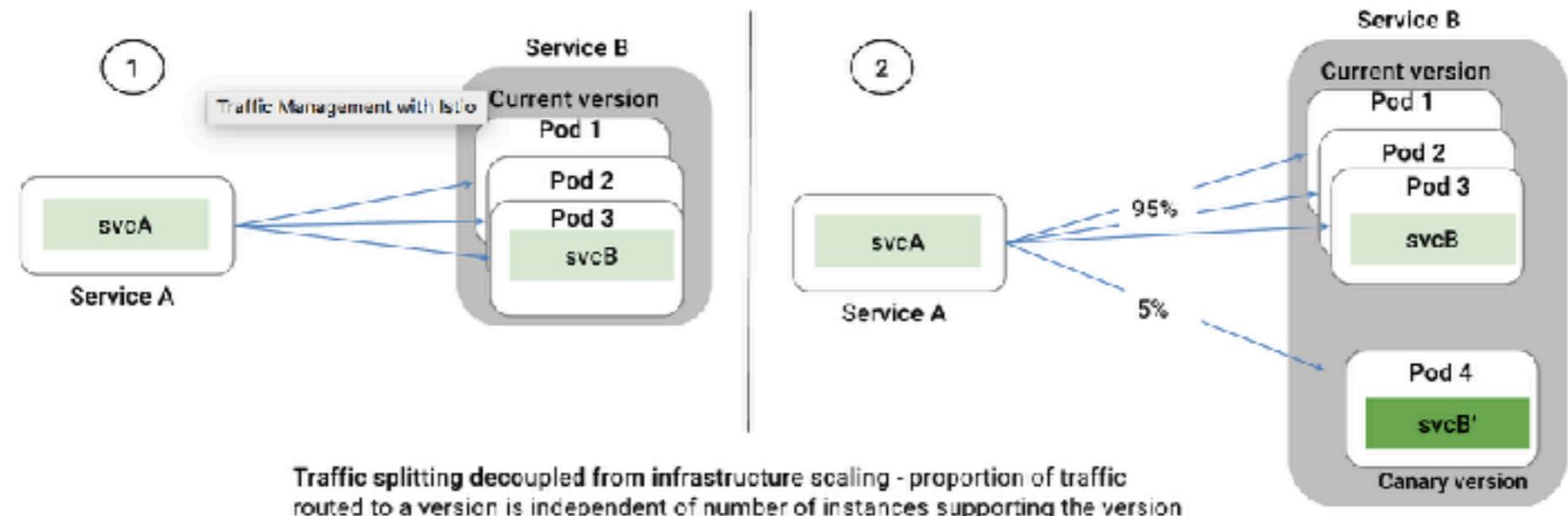
기능 구현에 집중. 나머지는 설정으로



# Istio architecture



## Traffic control



## Circuit Break

```
● ● ●

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews-cb-policy
spec:
  host: reviews.prod.svc.cluster.local
  trafficPolicy:
    ...
  outlierDetection:
    http:
      consecutiveErrors: 7
      interval: 5m
      baseEjectionTime: 15m
```

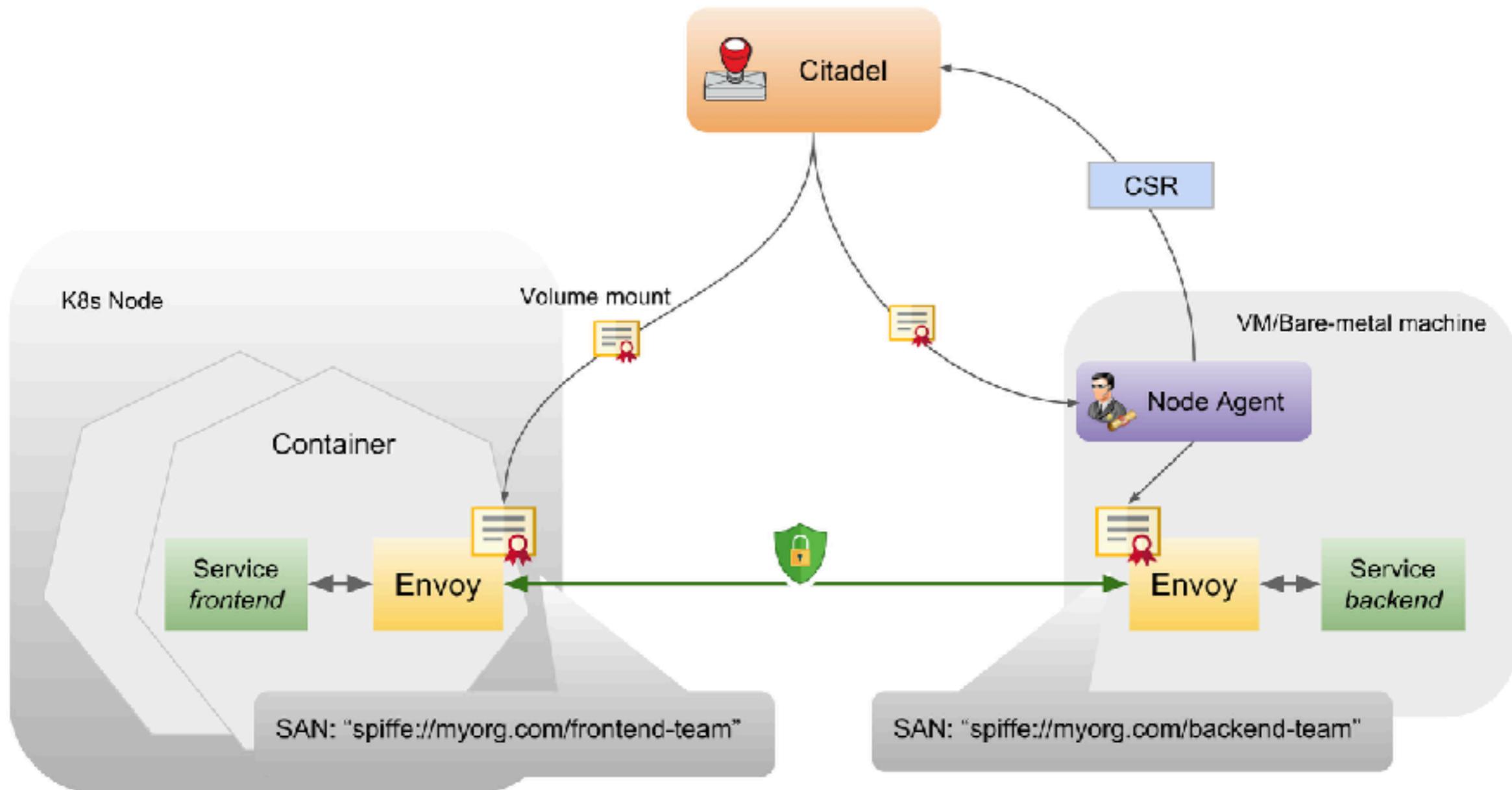
## Fault Injection

```
● ● ●

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        percent: 10
        fixedDelay: 5s
  route:
  - destination:
      host: ratings
      subset: v1
```

## 서비스 메시\_Istio

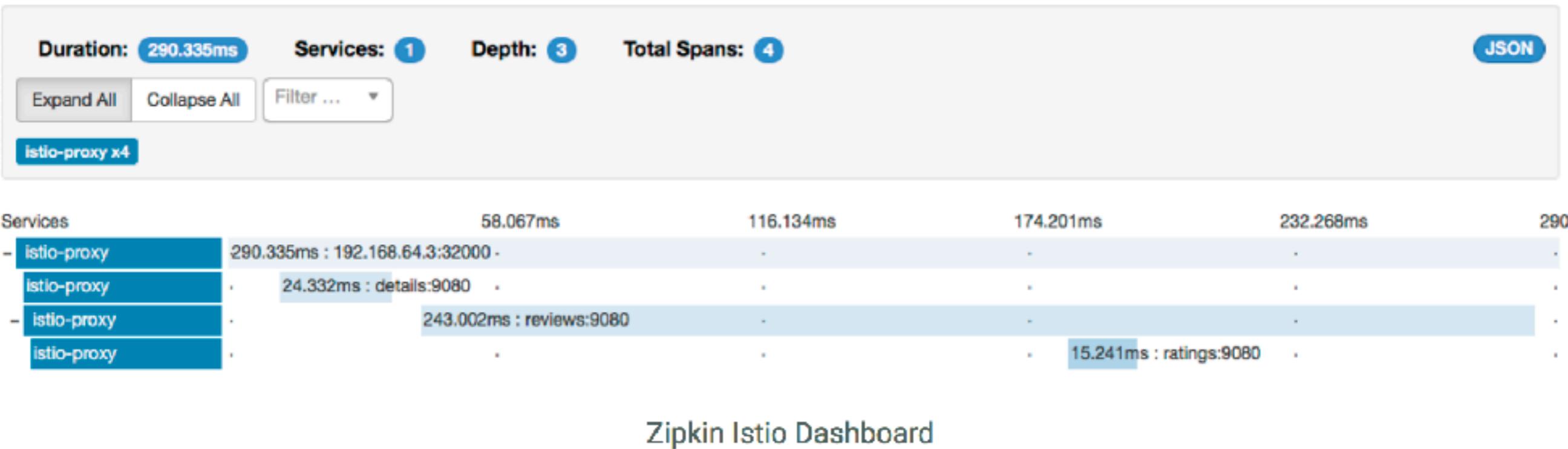
# mutual TLS



## 서비스 메시\_Istio

# Tracing

zipkin



## Tracing without code

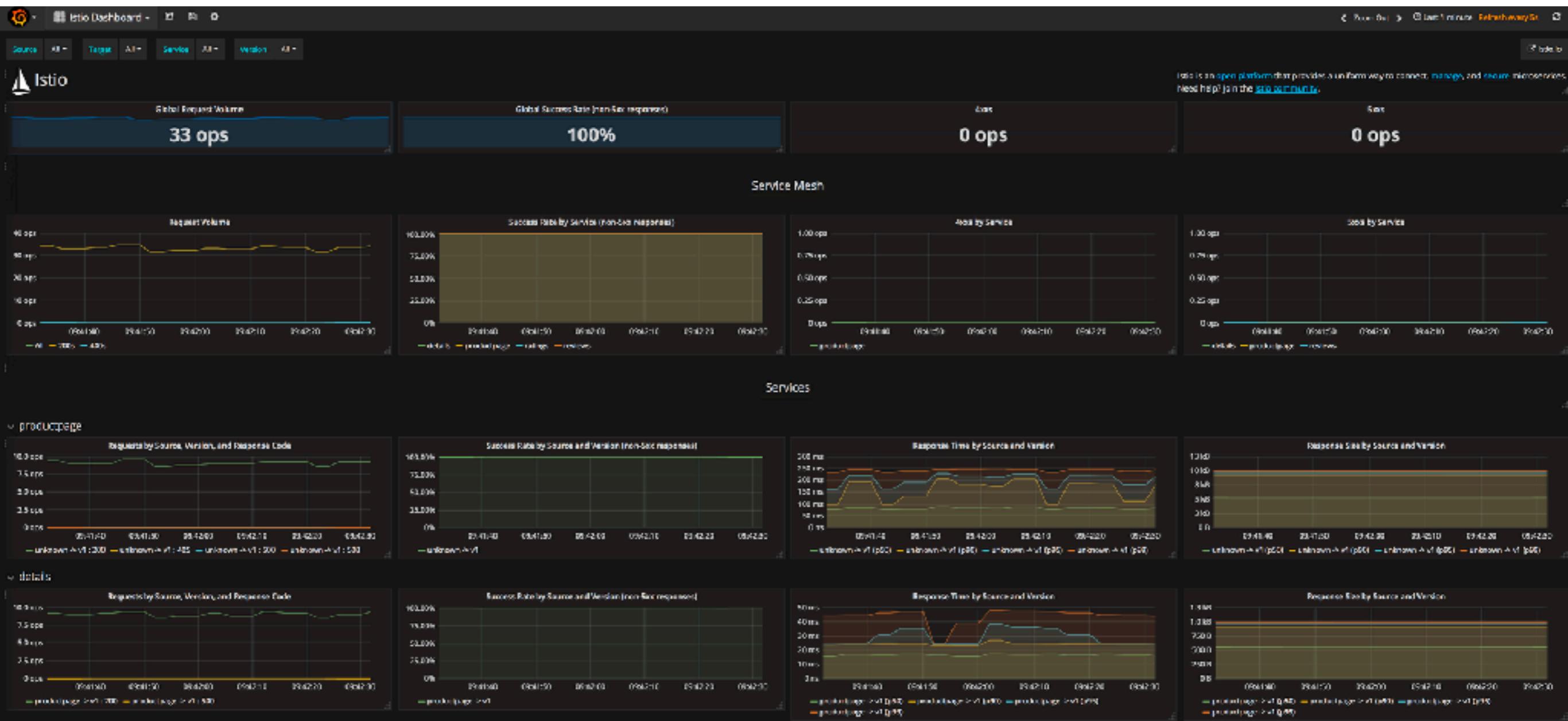
```
● ● ●  
  
// create new span using span found in context as parent (if none is found,  
// our span becomes the trace root).  
span, ctx := opentracing.StartSpanFromContext(ctx, "Concat")  
defer span.Finish()  
  
// assemble URL query  
url := fmt.Sprintf(  
    "%s/concat/?a=%s&b=%s", c.baseURL, url.QueryEscape(a), url.QueryEscape(b),  
)  
  
// create the HTTP request  
req, err := http.NewRequest("GET", url, nil)  
if err != nil {  
    return "", err  
}  
  
// use our middleware to propagate our trace  
req = c.traceRequest(req.WithContext(ctx))  
  
// execute the HTTP request  
resp, err := c.httpClient.Do(req)  
if err != nil {  
    // annotate our span with the error condition  
    span.SetTag("error", err.Error())  
    return "", err  
}  
defer resp.Body.Close()
```



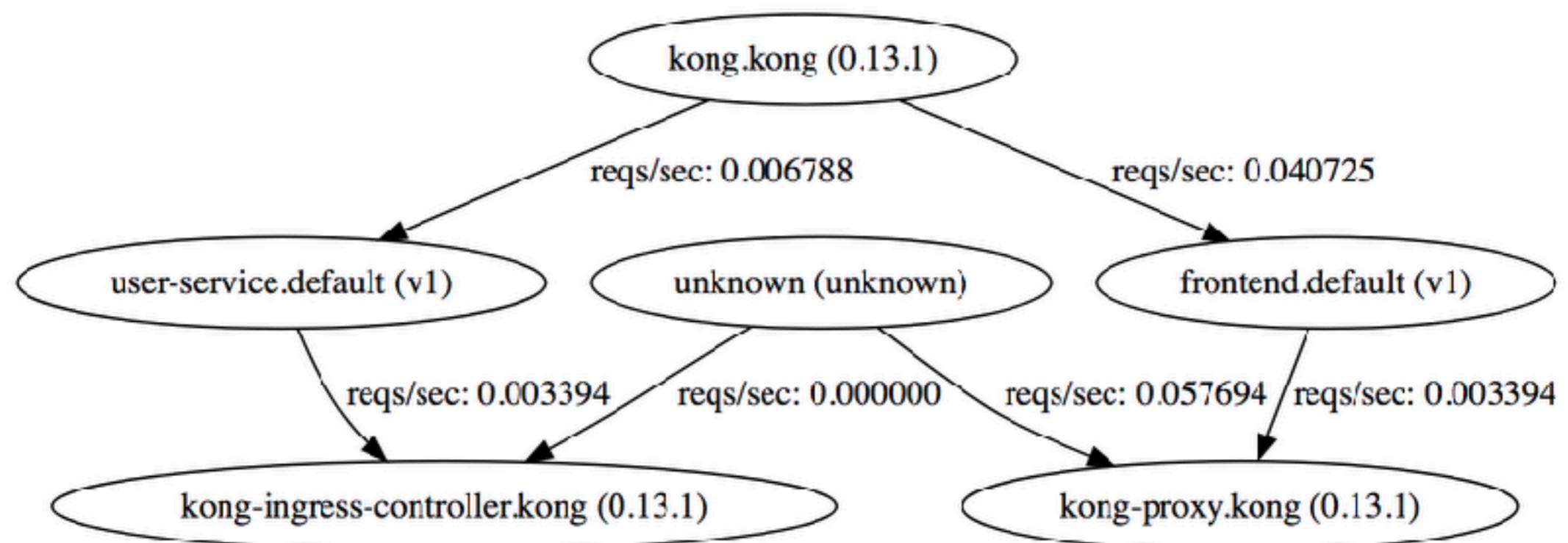
## 서비스 메시\_Istio

# Monitoring

prometheus + grafana

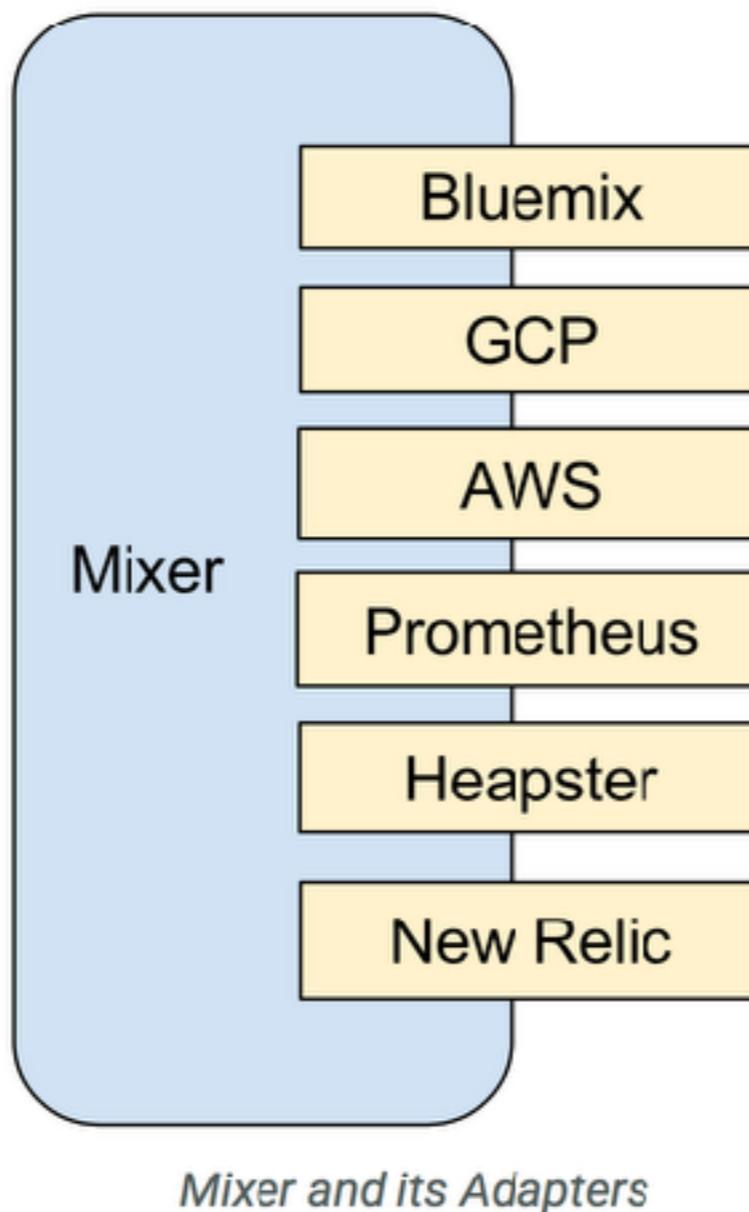


## Visualization



# Configurable

adapters



## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

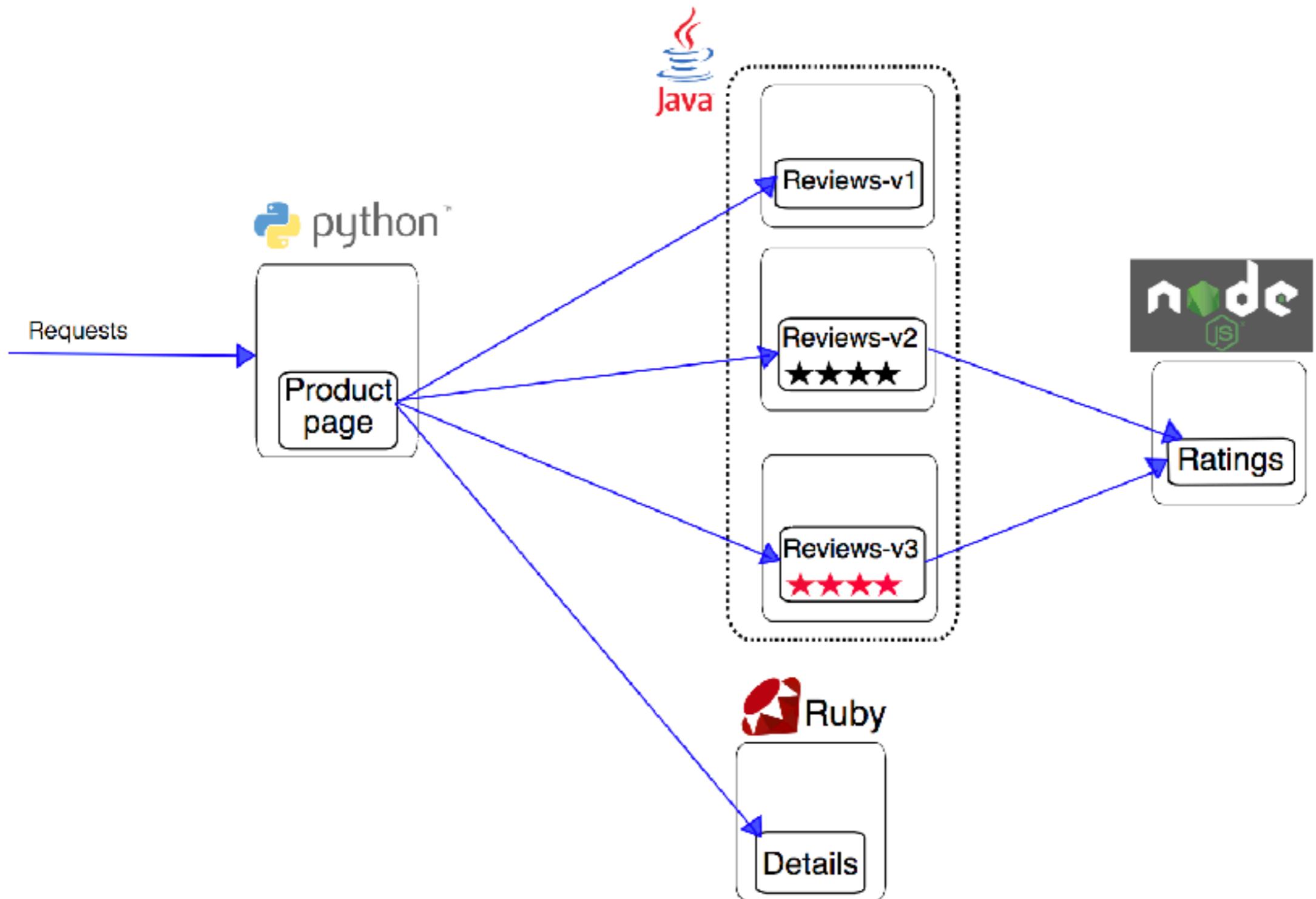
Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

## Live Demo

Istio 데모

## 결론

마무리



*Bookinfo Application without Istio*

# DEMO

## 개요

소개

## 마이크로서비스

마이크로서비스 도입하기  
문제점

## 컨테이너 오케스트레이션

컨테이너 오케스트레이션이란?  
Docker Swarm  
Kubernetes

## 서비스 메시

Service to Service Comm  
서비스 메시란?  
Envoy  
Istio

## Live Demo

Istio 데모

## 결론

마무리

---

컨테이너 오케스트레이션 \_ Kubernetes

**Microservices + Istio + Kubernetes = ❤️**



함께 해요 :)

<http://www.purpleworks.co.kr/recruit>

THANKS