

# Istio를 활용한 Service Mesh 구성하기

V.1.0.0

2018.08.20

Copyright©2018 by SK DIGITAL LABS All rights reserved.



# Table of Contents

---

01 | Service Mesh는 왜 필요한가?

02 | Service Mesh 구현체: Istio

# PART 01. Service Mesh는 왜 필요한가?

---

01

마이크로서비스의 복잡성

02

Service Mesh 란?

03

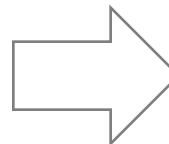
Service Mesh 구현체

### CLOUDZ LABS

마이크로서비스 아키텍처는 기존 모놀리식 아키텍처의 단점을 보완하고 클라우드 환경에서의 이점을 극대화하기 위해 많이 사용되고 있습니다.



Monolithic



Microservice

*Agility*  
*Scale*  
*Speed*  
*Safety*

### CLOUDZ LABS

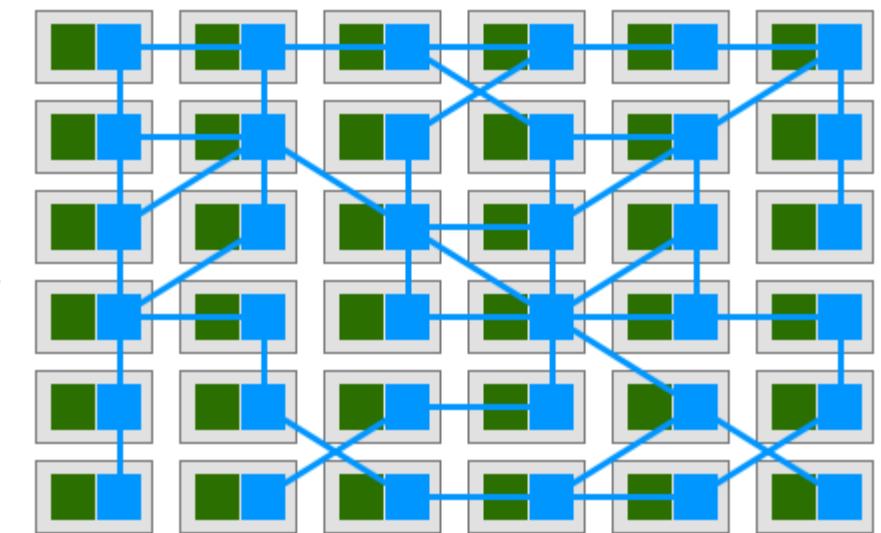
하지만 기존의 모놀리식 애플리케이션에서 마이크로서비스 아키텍처로 전환함에 따라 또 다른 문제점들이 발생합니다.

이를 Service Mesh를 통해 해결할 수 있습니다.



#### MSA Challenges

- Discovery**
- Load Balancing**
- Failure Recovery**
- Metrics**
- Monitoring, Logging**
- A/B Testing, Canary Releases,**
- Access Control**
- Authentication**
- Distribute tracing**



# PART 01. Service Mesh는 왜 필요한가?

01

마이크로서비스의 복잡성

02

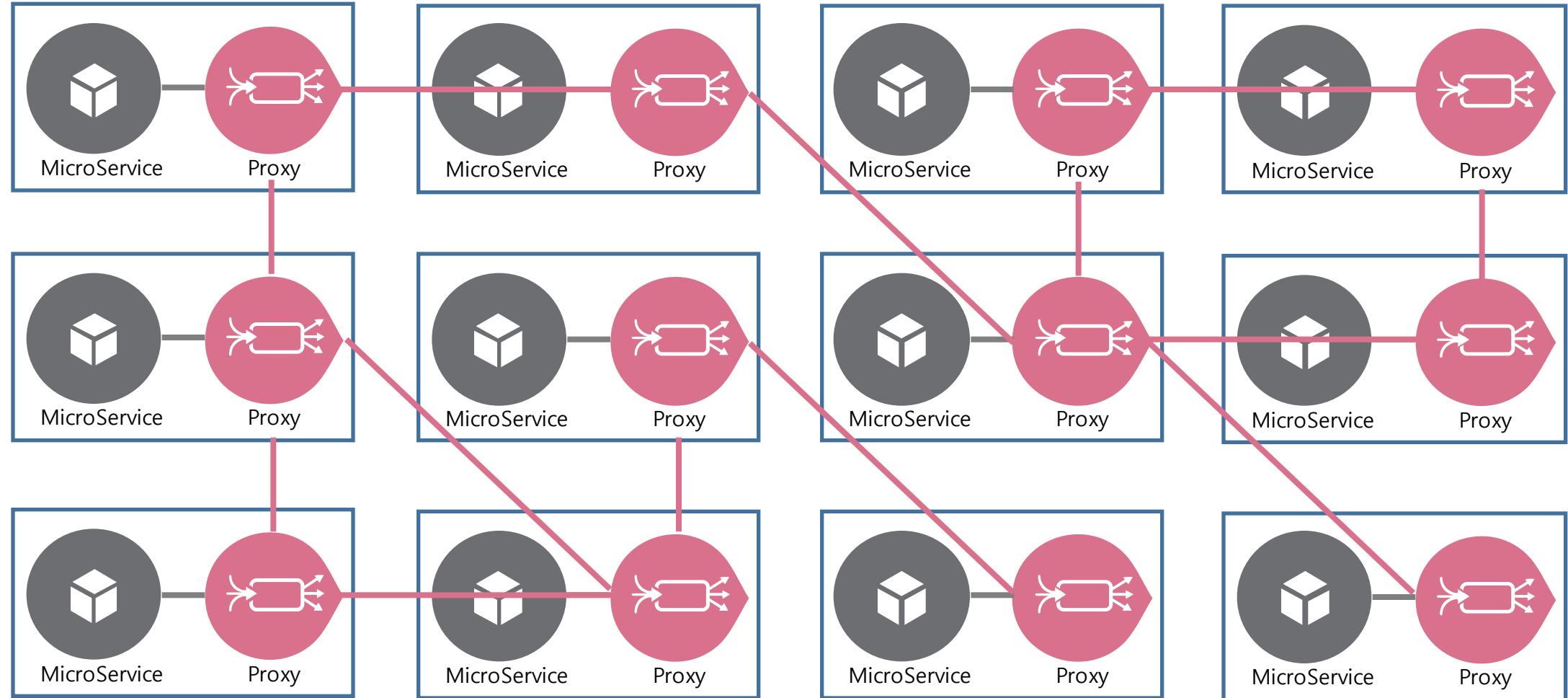
**Service Mesh** 란?

03

**Service Mesh** 구현체

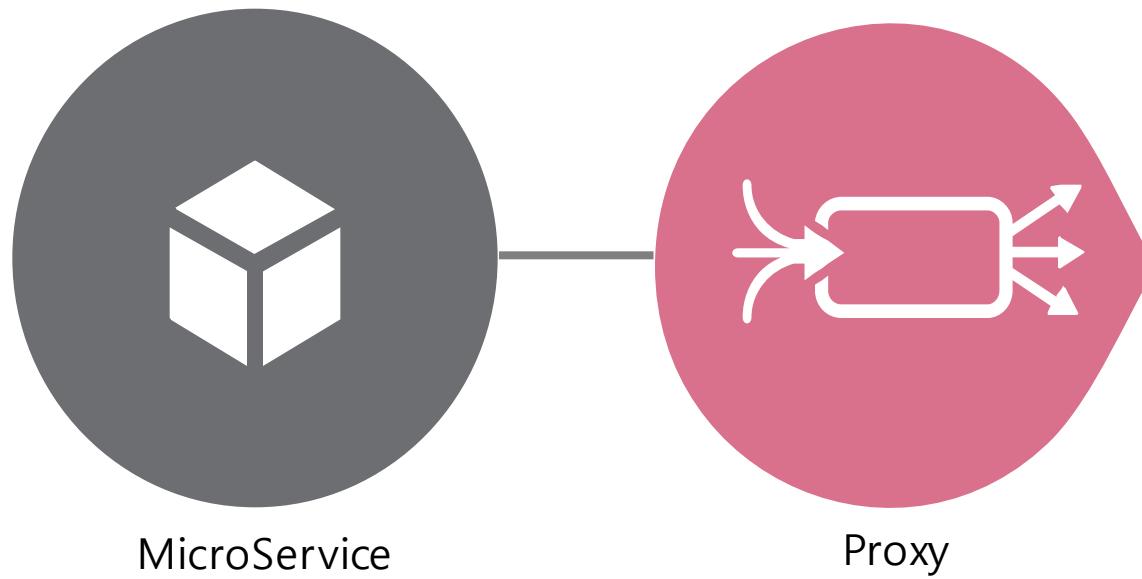
## CLOUDZ LABS

**Service Mesh**는 서비스 간 통신을 추상화하여 안전하고, 빠르고, 신뢰할 수 있게 만드는 전용 **InfraStructure Layer**입니다.



## CLOUDZ LABS

**Service Mesh**의 구현은 보통 서비스의 앞단에 경량화 프록시를 **Sidecar** 패턴으로 배치하여 서비스 간 통신을 제어하는 방법으로 구현합니다.



### ▪ **Sidecar** 패턴이란?

- **Container** 디자인 패턴의 일종
  - 기본 애플리케이션 외 필요한 추가 기능을 별도의 애플리케이션으로 구현하고 이를 동일한 프로세스 또는 컨테이너 내부에 배치하는 것
  - 애플리케이션과 함께 컨테이너에 배치된 **Sidecar**는 저장 공간, 네트워크 등의 리소스를 공유하며 이를 통해 모니터링에 필요한 **metrics** 수집, 로깅, 프록시 등을 수행함
- 
- 장점
  - 기본 애플리케이션의 로직을 수정하지 않고 별도의 추가 기능을 수행 가능함
  - 애플리케이션의 **polyglot** 프로그래밍이 가능하기 때문에 최적화된 언어로 개발을 진행함

# PART 01. Service Mesh는 왜 필요한가?

01

마이크로서비스의 복잡성

02

Service Mesh 란?

03

Service Mesh 구현체

## CLOUDZ LABS

**Service Mesh** 구현체 (Envoy, Linkerd -> Istio, Conduit)



envoy



Istio

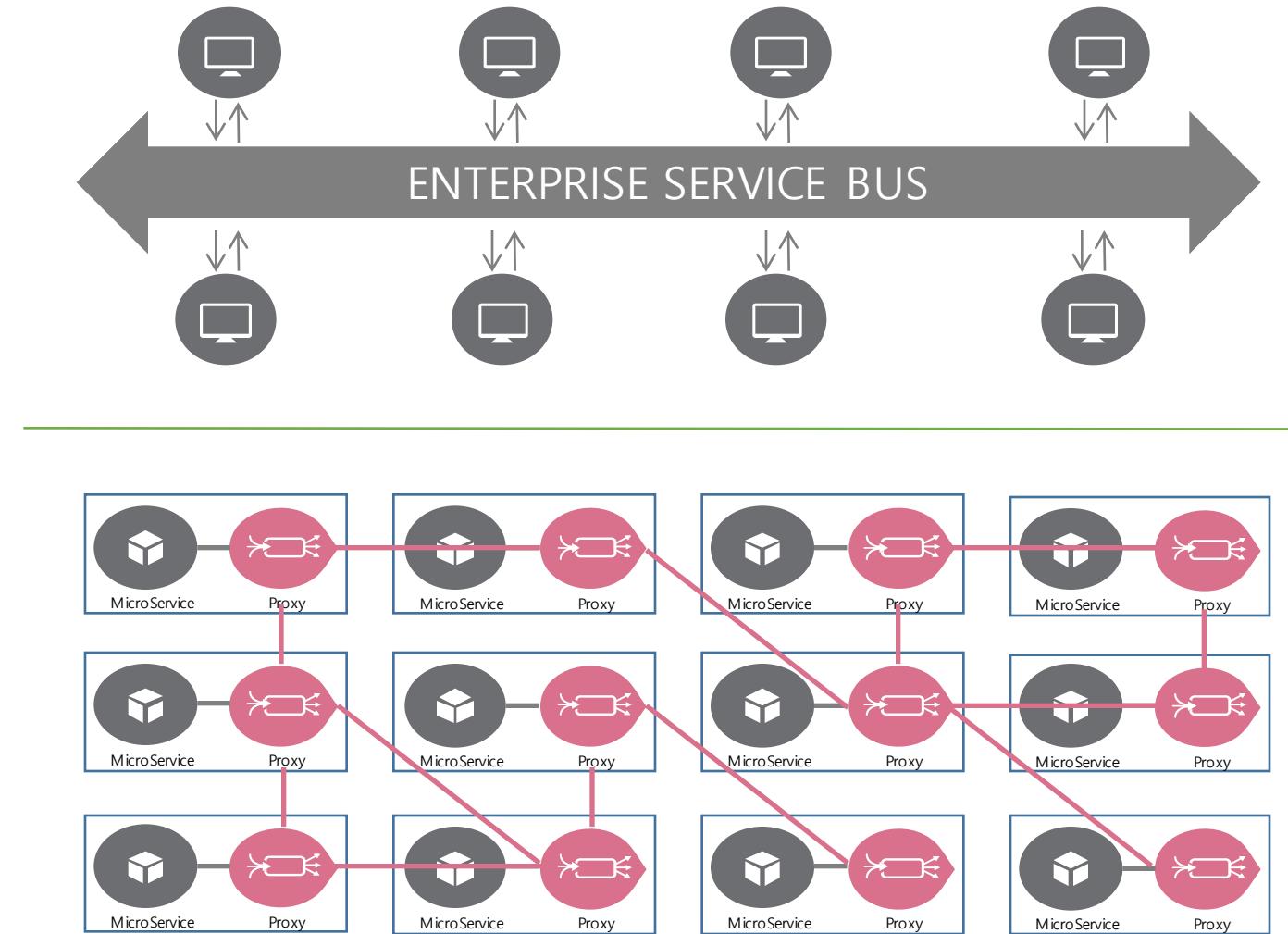


LINKERD



CONDUIT

## Service Mesh 와 SOA - ESB 의 비교





VS.



		Spring Cloud	Service Mesh
장점	<ul style="list-style-type: none"> <li>- 레퍼런스가 많음</li> <li>- 보다 안정적</li> <li>- 기본적인 설정만으로 동작</li> <li>- 세부적인 설정 가능</li> </ul>	<ul style="list-style-type: none"> <li>- Polyglot 프로그래밍이 가능 (언어 종속성 없음)</li> <li>- 높은 이식성(코드, 플랫폼 종속성 없음)</li> <li>- Mesh 설정 변경이 자유로움</li> <li>- 런타임 복잡성에 대한 이슈를 해결</li> <li>- Spring Cloud에 비해 보다 많은 기능을 지원 (가중치 기반 라우팅, TLS, Mirroring, Fault Injection 등)</li> </ul>	
단점	<ul style="list-style-type: none"> <li>- 언어, 코드, 런타임에 종속성</li> <li>- 라이브러리도 업데이트 및 관리가 필요함</li> </ul>	<ul style="list-style-type: none"> <li>- 시스템의 런타임 인스턴스 수가 증가</li> <li>- 새로운 기술로 레퍼런스가 많지 않음</li> </ul>	

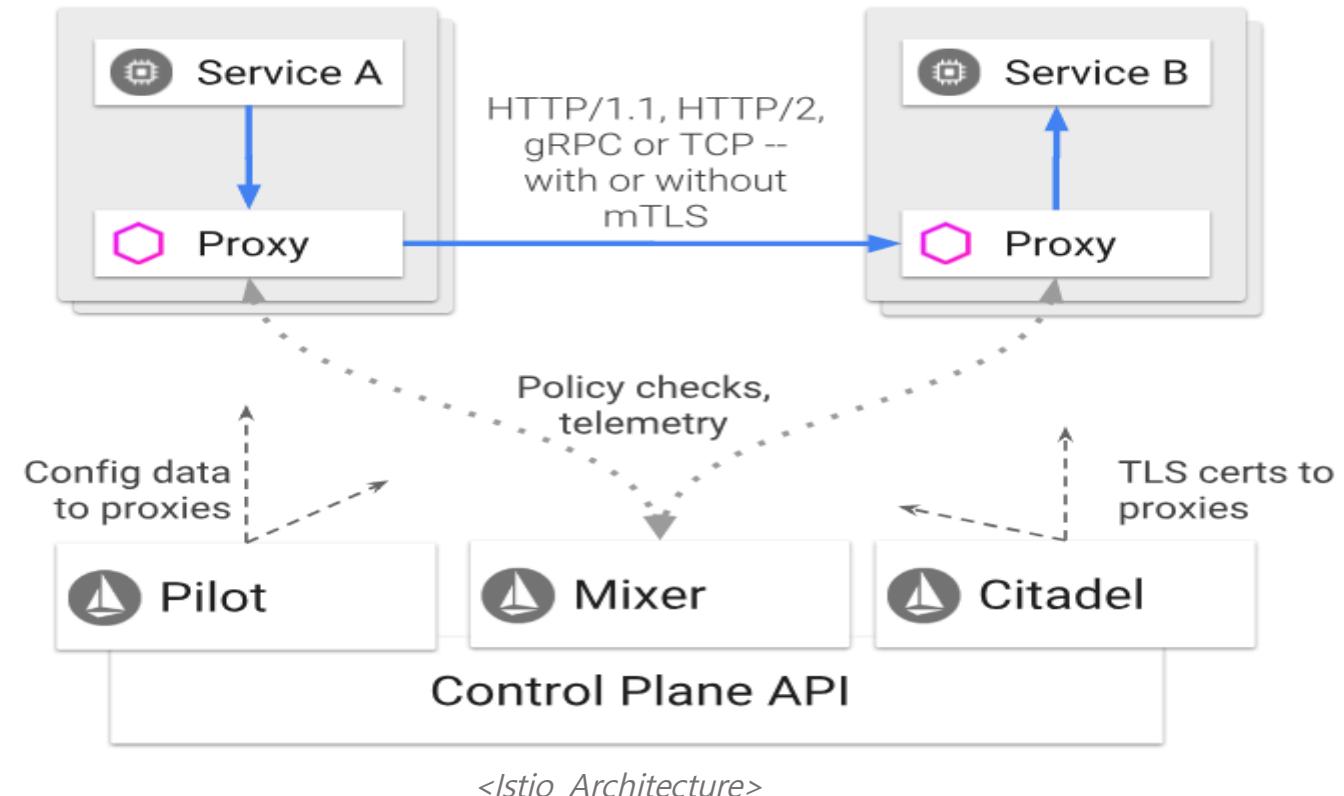
## PART 02. Service Mesh 구현체: Istio

---



## CLOUDZ LABS

Istio는 사이드카 프록시 영역인 **Data Plane**과 그 프록시를 구성하고 관리하는 **Control Plane**으로 구성됩니다.

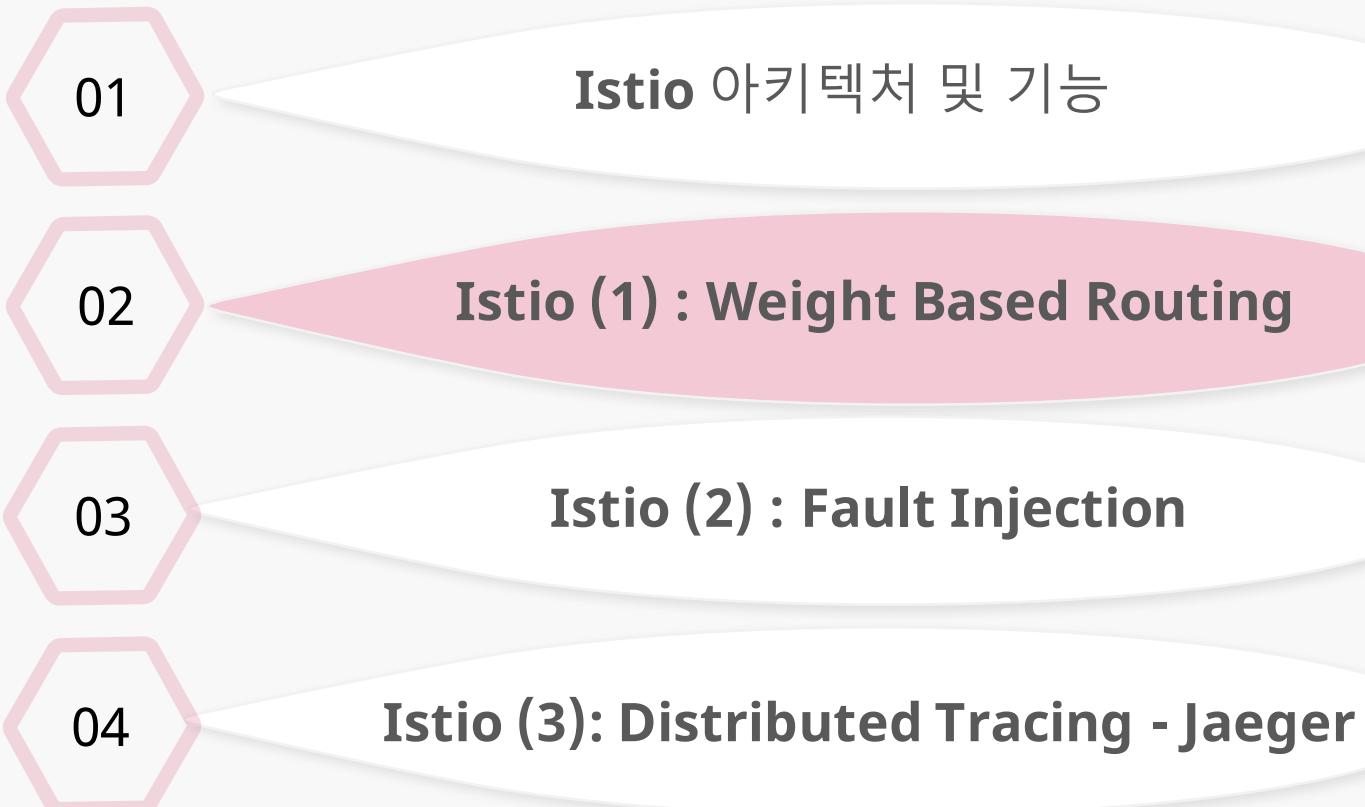


## ✓ Istio 아키텍처

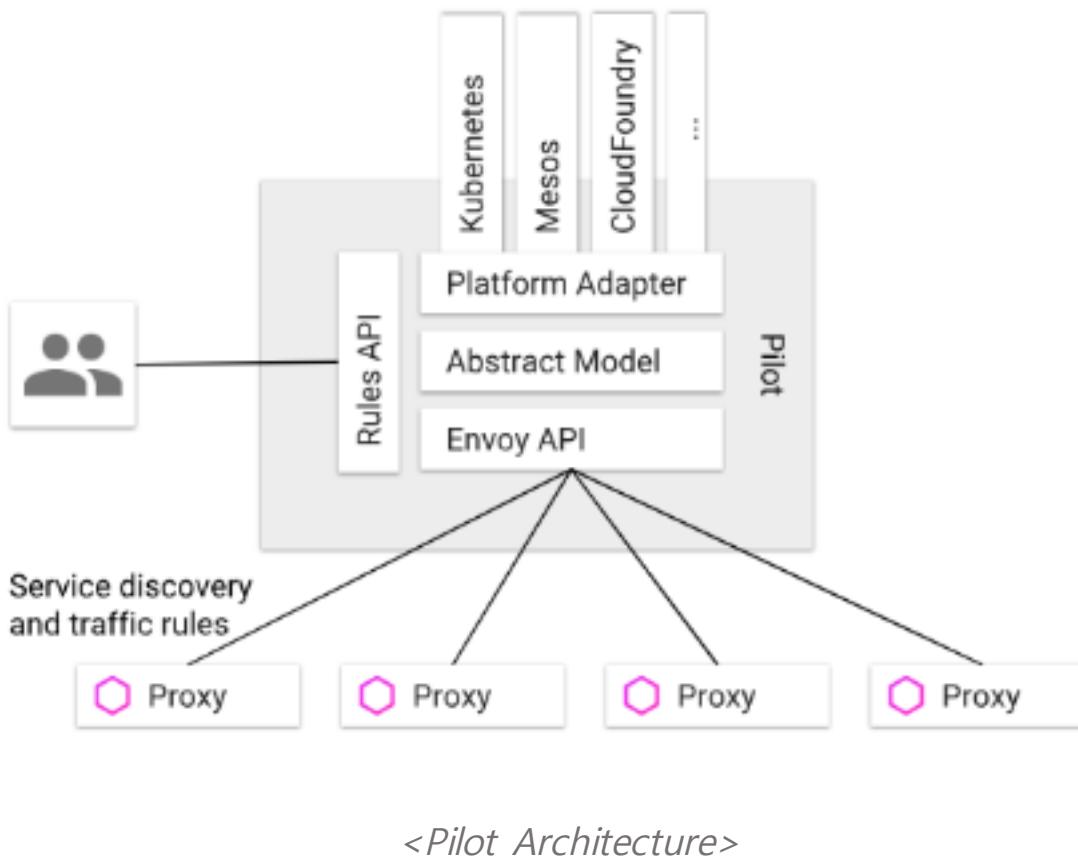
- Data Plane + Control Plane
- Data Plane : 사이드카로 배치되는 일련의 프록시(Envoy)로 구성
- Control Plane : 프록시를 구성하고 관리하며, 원격 측정을 수집하도록 믹서를 구성 (Pilot, Mixer, Citadel)

- ✓ Traffic Management
  - **Dynamic request routing** (**weight-based routing**, content-based routing )
  - Discovery and load balancing
  - Handling failures (circuit breaking, retry, timeout, health check ...)
  - **Fault injection** (**http/tcp error code**, delay)
- ✓ Policies Enforcement (traffic limit, access control)
- ✓ Telemetry (**jaeger**, grafana, prometheus, service graph ... )
- ✓ Security

## PART 02. Service Mesh 구현체: Istio



## CLOUDZ LABS

트래픽 관리의 핵심 구성요소인 **Pilot**✓ **Pilot**

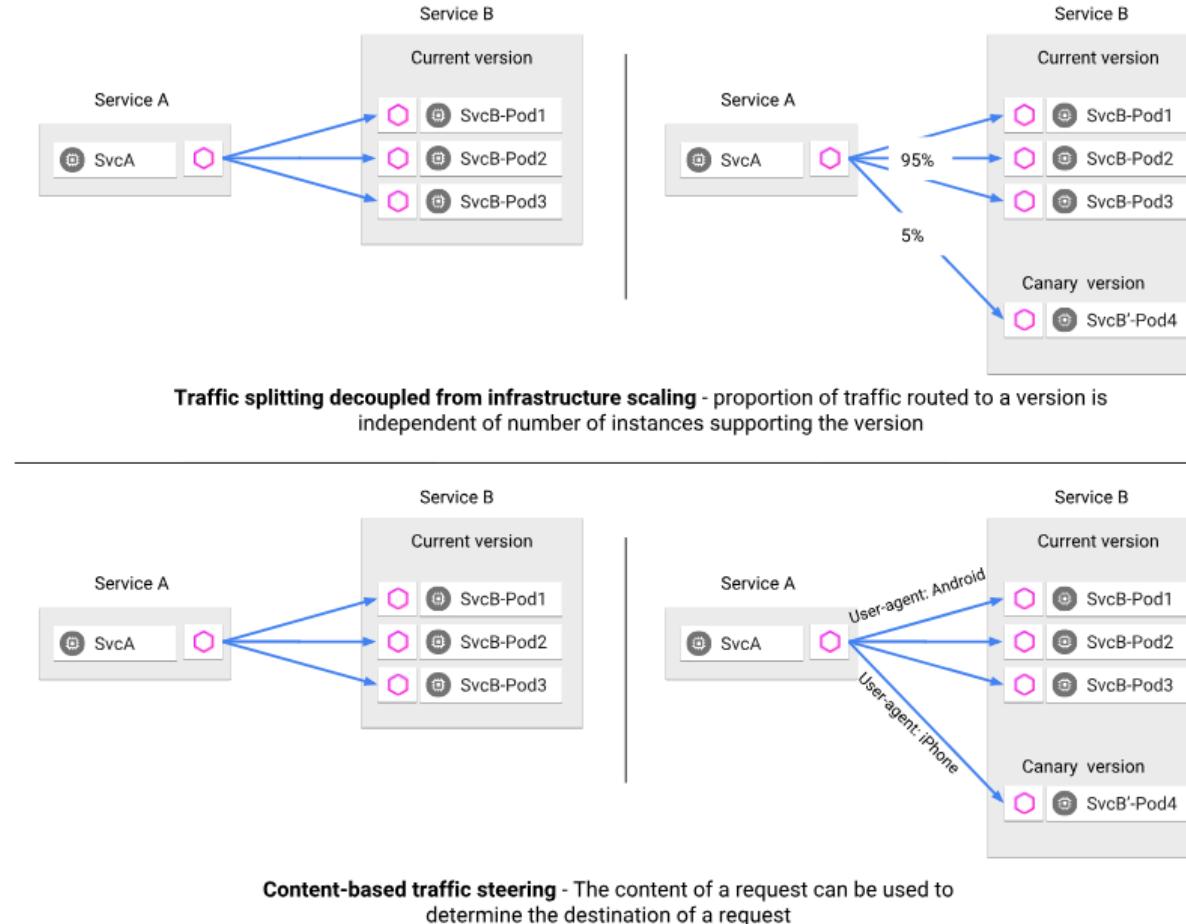
- Istio의 트래픽 관리에 사용되는 핵심 구성 요소
- Istio 서비스 메시에 배포 된 모든 Envoy 프록시 인스턴스를 관리하고 구성하는 역할
- Envoy 인스턴스의 라이프사이클 관리

✓ **Envoy (Proxy)**

- 모든 서비스에 대한 인바운드 및 아웃바운드 트래픽을 중재
- Envoy 인스턴스는 Pilot으로부터 가져온 정보를 기반으로 로드밸런싱 정보를 유지 관리함
- 다른 인스턴스에 대한 서비스 디스커버리 및 **Health check** 수행

## CLOUDZ LABS

Request Routing 설정을 통해 Canary 배포, A/B 테스팅, 점진적 배포 등을 수행합니다.



## ✓ Dynamic Request Routing

### ▪ 기능

- 가중치 기반 트래픽 분리
- 컨텐츠 기반 트래픽 분리
  - 헤더값
  - Request URI prefix

### ▪ 장점

- 여러 조건에 따라 트래픽 분리 가능
- Canary Relaese, A/B Testing 수행
- 트래픽을 미세하게 관리 가능
  - 예를 들어, K8S의 경우 1%로 트래픽을 분리하기 위해 100개 이상의 Pod가 필요함

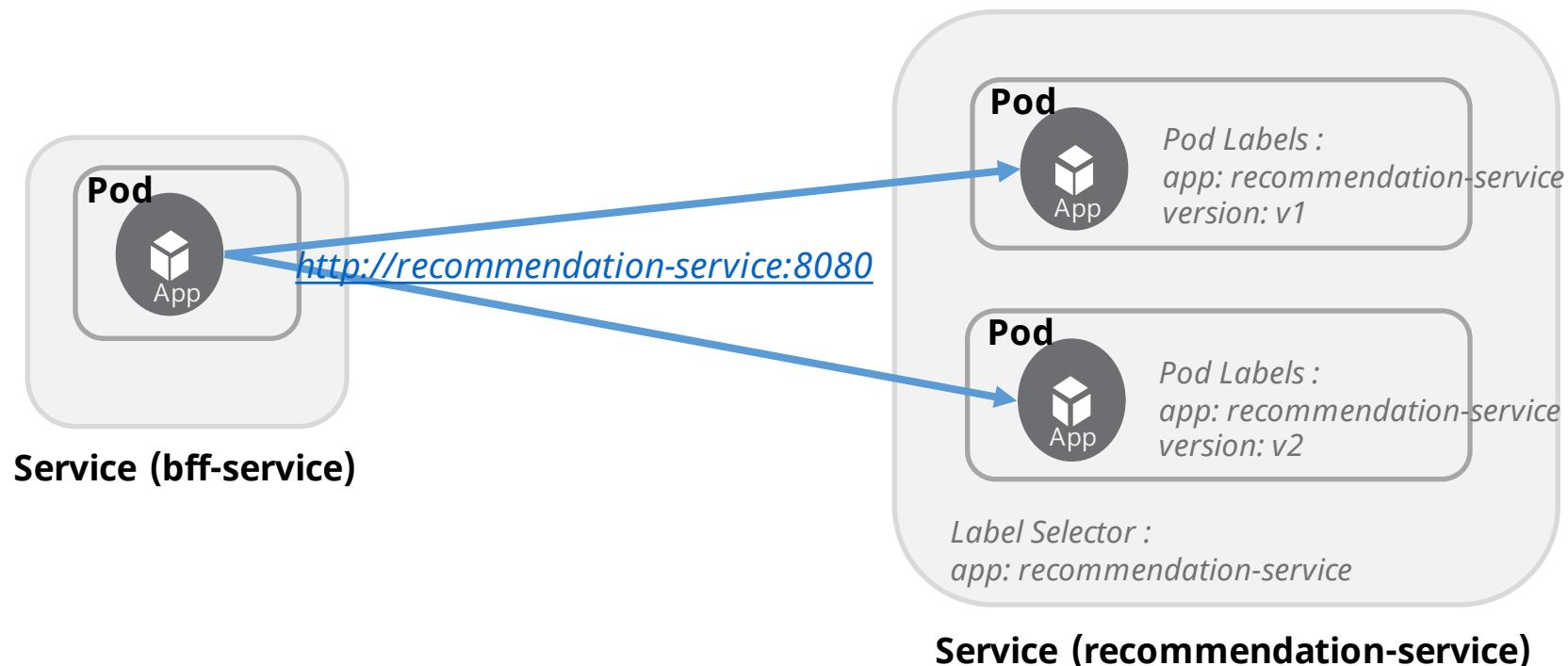
## ✓ 라우팅 룰을 설정하는 핵심 개념

- VirtualService
- DestinationRule

## CLOUDZ LABS

## K8S에서 기본적인 서비스간 호출 방식

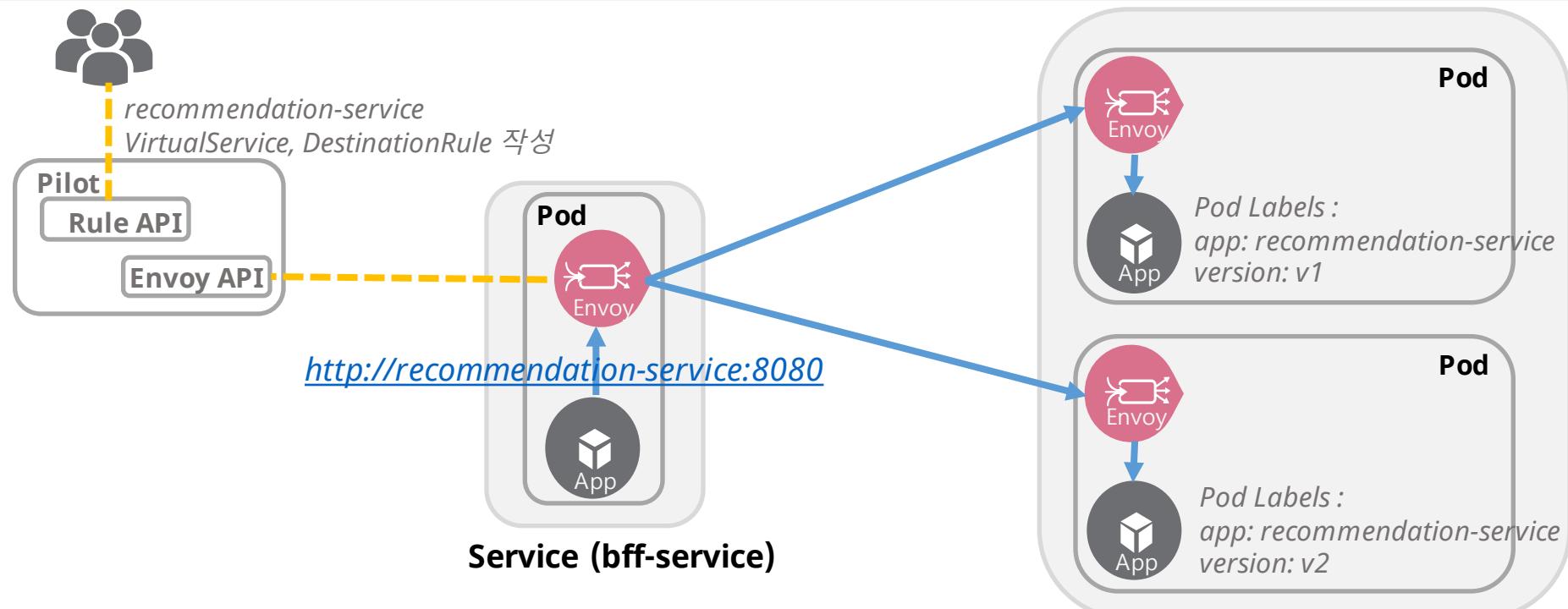
- 호출 방식 예시 (bff -> recommendation)
  1. bff 애플리케이션이 recommendation 서비스를 호출
  2. recommendation 서비스의 label selector와 로드밸런싱 정책에 따라 호출할 recommendation 애플리케이션을 결정
  3. recommendation 애플리케이션이 호출됨



## CLOUDZ LABS

## K8S에 Istio 적용 시 서비스간 호출 방식

- 호출 방식 예시 (bff -> recommendation)
  1. recommendation 서비스에 대한 라우팅 룰 설정 (VirtualService, DestinationRule)
  2. bff 애플리케이션이 recommendation 서비스를 호출
  3. bff envoy가 발생한 outbound 트래픽을 가로챔
  4. bff envoy에서 가지고 있는 라우팅 규칙에 따라 호출할 recommendation 애플리케이션을 결정(버전 별 가중치, 로드밸런싱 정책 등 반영)
  5. recommendation 애플리케이션으로 들어오는 inbound 트래픽을 recommendation envoy가 가로챔
  6. Recommendation envoy가 recommendation 애플리케이션을 호출



## CLOUDZ LABS

## VirtualService와 DestinationRule

- ✓ recommendation 서비스에 대한 VirtualService 와 DestinationRule

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation-service
spec:
  hosts:
    - recommendation-service
  http:
    - route:
        - destination:
            host: recommendation-service
            subset: v1
            weight: 10
        - destination:
            host: recommendation-service
            subset: v2
            weight: 90
```

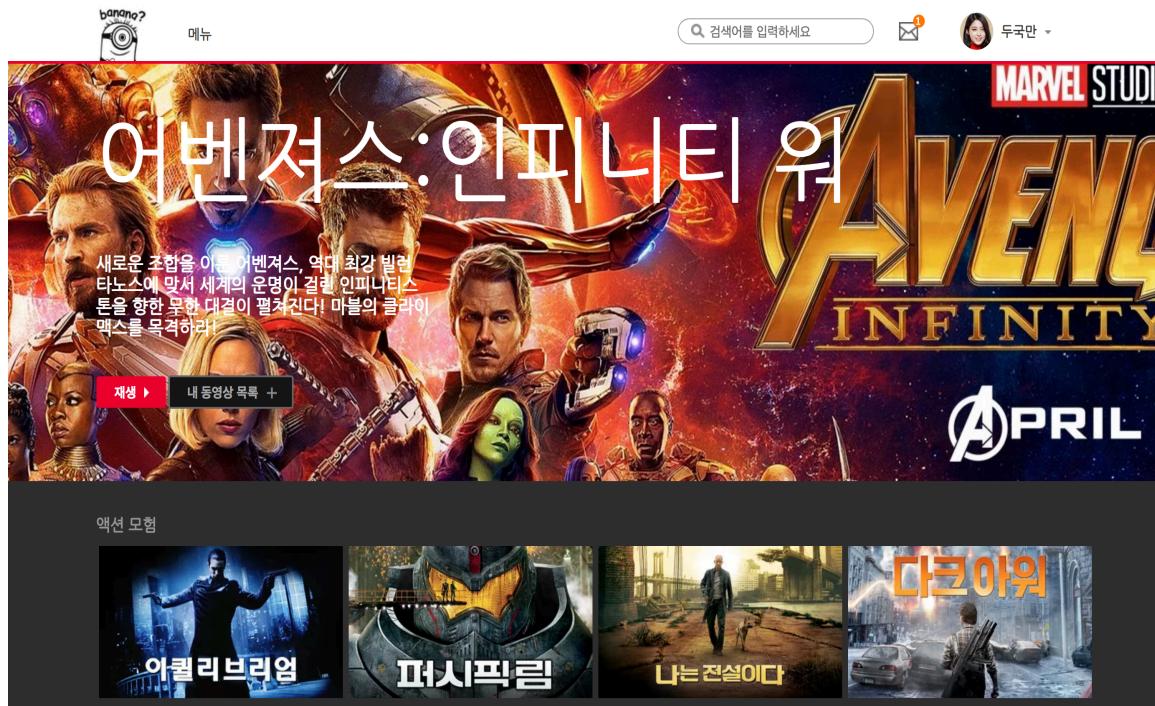
```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: recommendation-service
spec:
  host: recommendation-service
  subsets:
    - name: v1
      labels:
        version: v1
    - name: v2
      labels:
        version: v2
```

\* subset : 서비스 버전 별로 관리하기 위한 개념

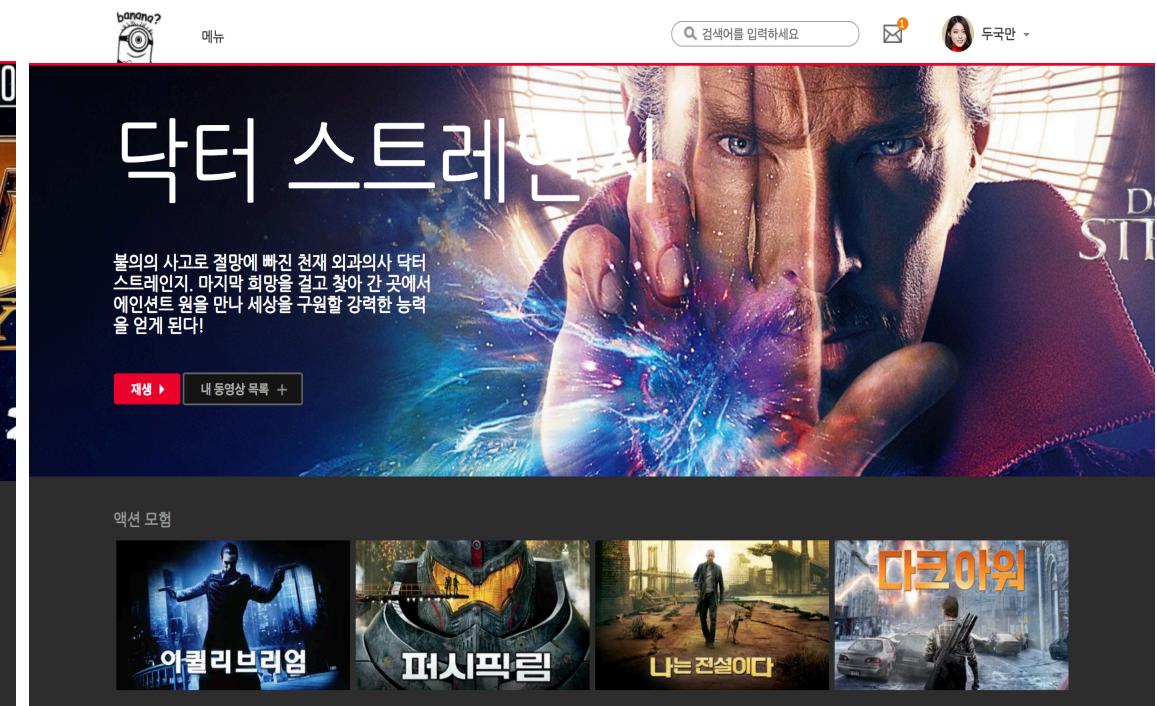
### CLOUDZ LABS

#### Weight Based Routing 실습

- ✓ 실습 전 상태 (recommendation v1, v2 가 비슷한 비율로 호출됨)



recommendation v1 호출 시



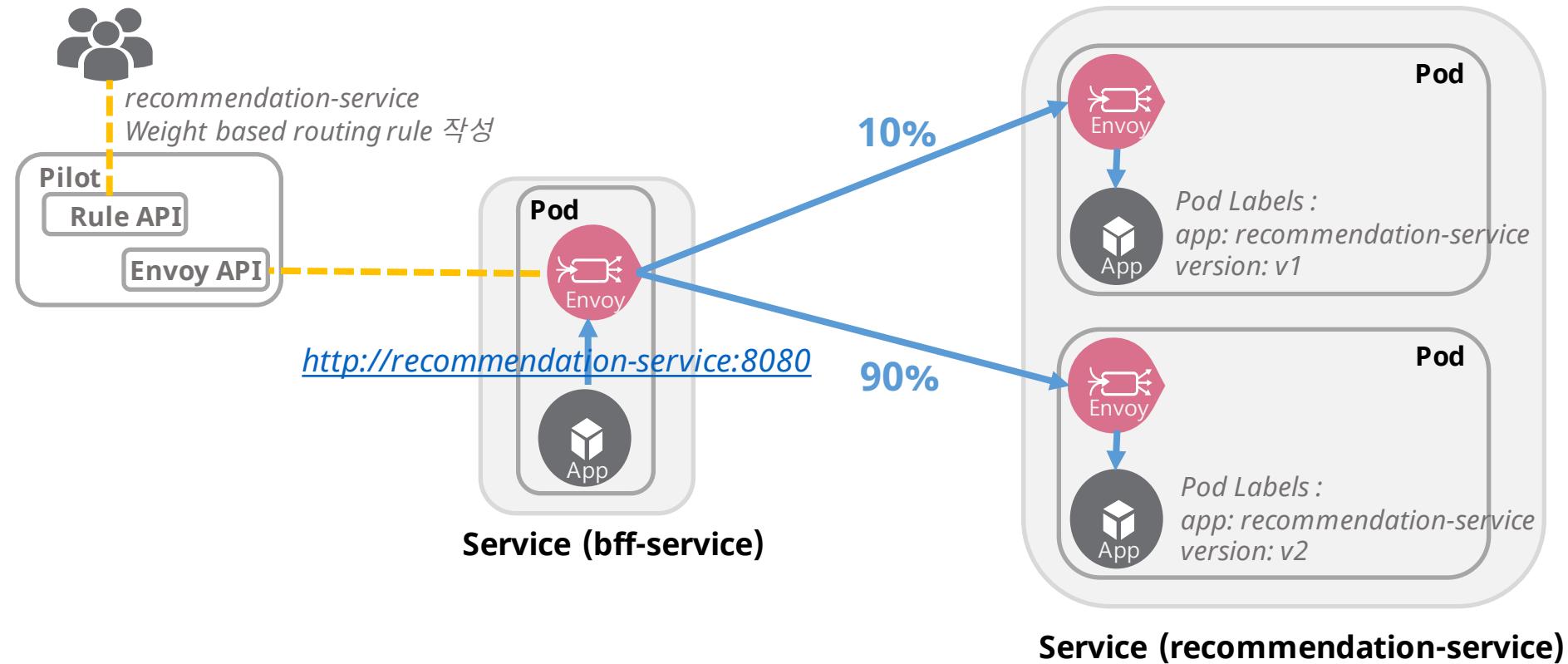
recommendation v2 호출 시

## CLOUDZ LABS

Weight Based Routing 실습 (recommendation v1 : recommendation v2 = 10 : 90)

## ✓ Weight Based Routing

- 애플리케이션의 업데이트 전에 일부 사용자들로 하여금 새 버전의 테스트가 가능하여 리스크를 줄일 수 있습니다.



## CLOUDZ LABS

Weight Based Routing 실습 (recommendation v1 : recommendation v2 = 10 : 90)

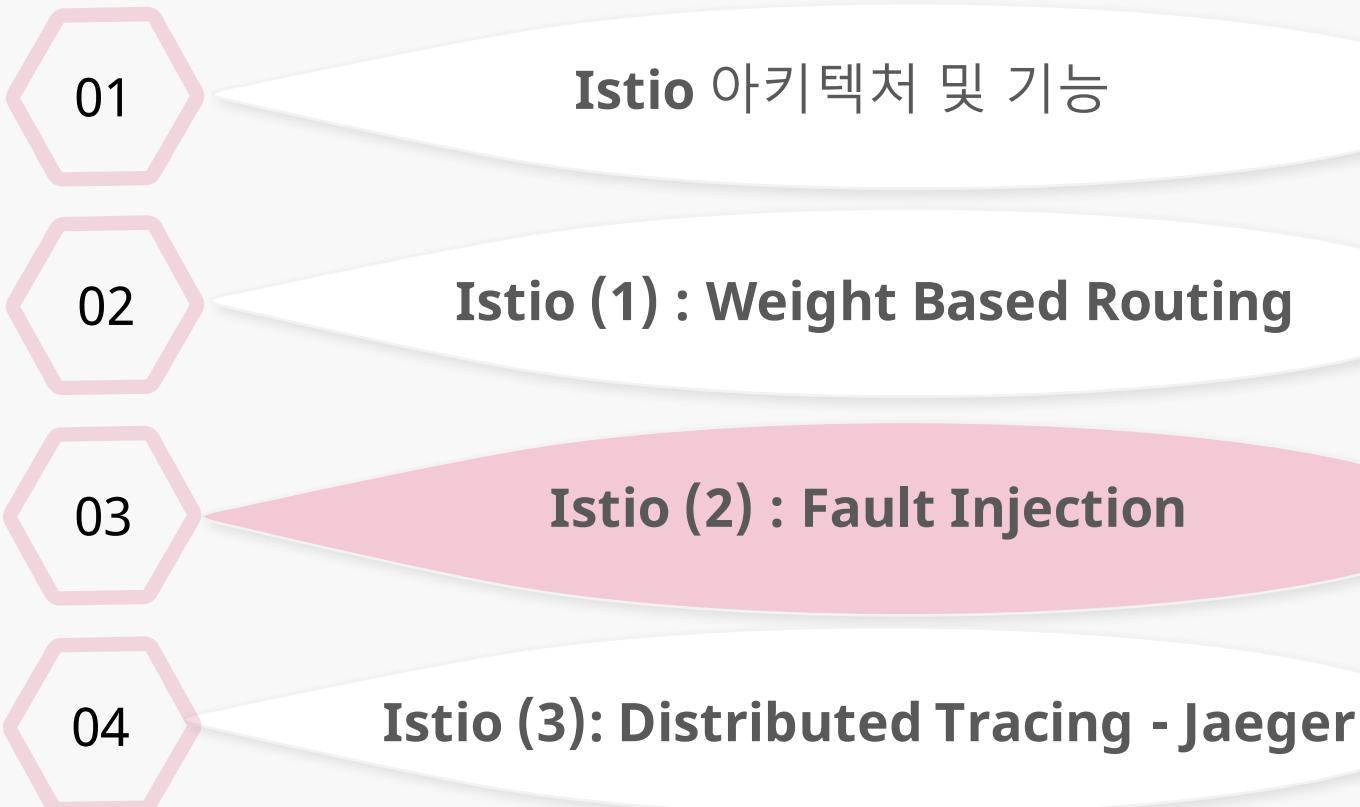
### 1. Weight Based Routing 설정 적용

```
$ kubectl apply -f recommendation-service-weight-based-routing.yaml
```

### 2. bff-service 화면을 통해 테스트

(recommendation v2를 90% 호출하므로 메인 화면에 닥터스트레인지가 많이 노출됨)

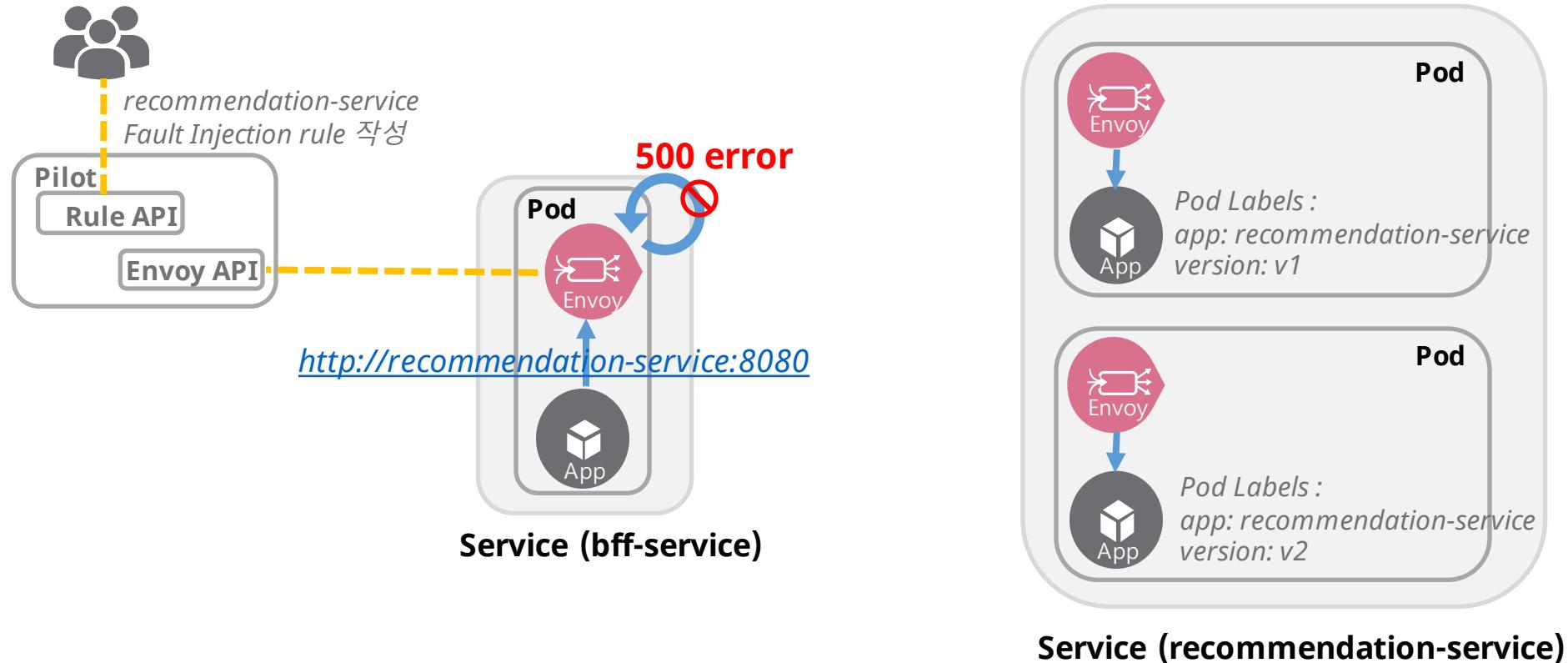
## PART 02. Service Mesh 구현체: Istio



## Fault Injection 실습

## ✓ Fault Injection

- 서비스 호출이 빈번한 마이크로서비스에서 애플리케이션 수정 없이 설정만으로 서비스 장애 상황을 재현하여 대비할 수 있습니다.



## 1. VirtualService의 Fault Injection 설정

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation-service
spec:
  hosts:
    - recommendation-service
  http:
    - route:
        - destination:
            host: recommendation-service
            subset: v1
            weight: 10
        - destination:
            host: recommendation-service
            subset: v2
            weight: 90
      fault:
        abort:
          httpStatus: 500
          percent: 100
```

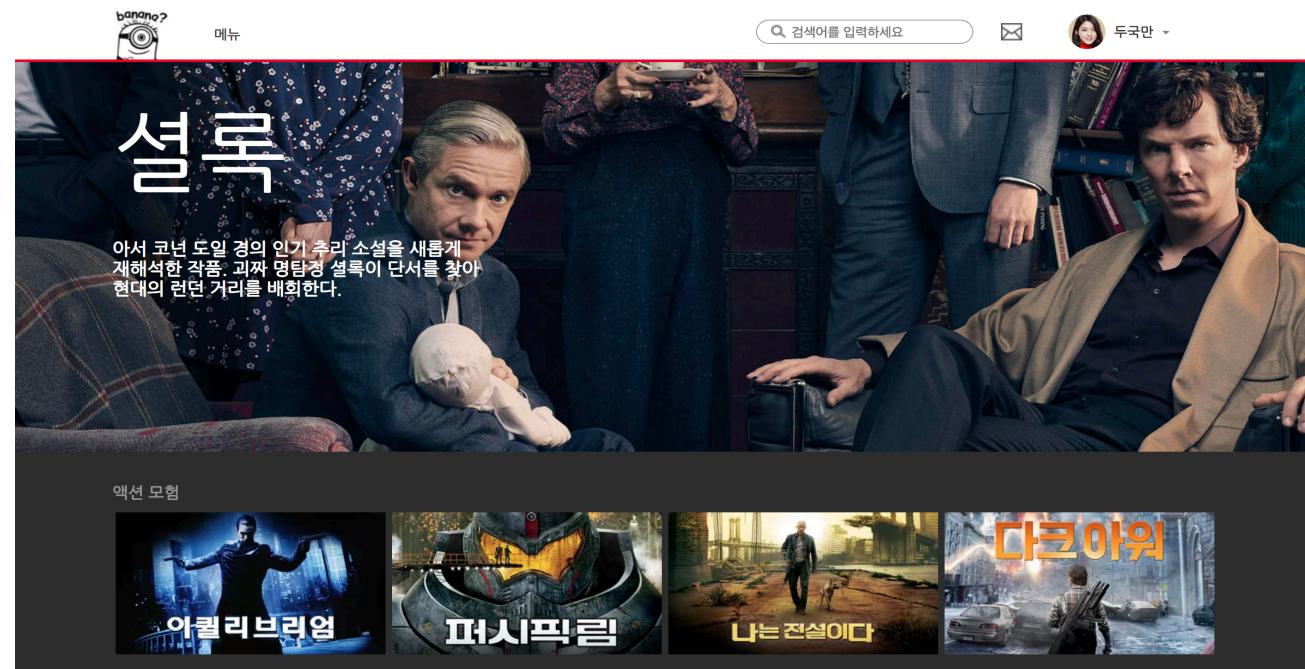
## 2. Weight Based Routing 설정 삭제

```
$ kubectl delete -f recommendation-service-weight-based-routing.yaml
```

## 3. Fault Injection 설정 적용

```
$ kubectl apply -f recommendation-service-fault-injection.yaml
```

## 4. bff-service 화면을 통해 테스트



## PART 02. Service Mesh 구현체: Istio

01

Istio 아키텍처 및 기능

02

Istio (1) : Weight Based Routing

03

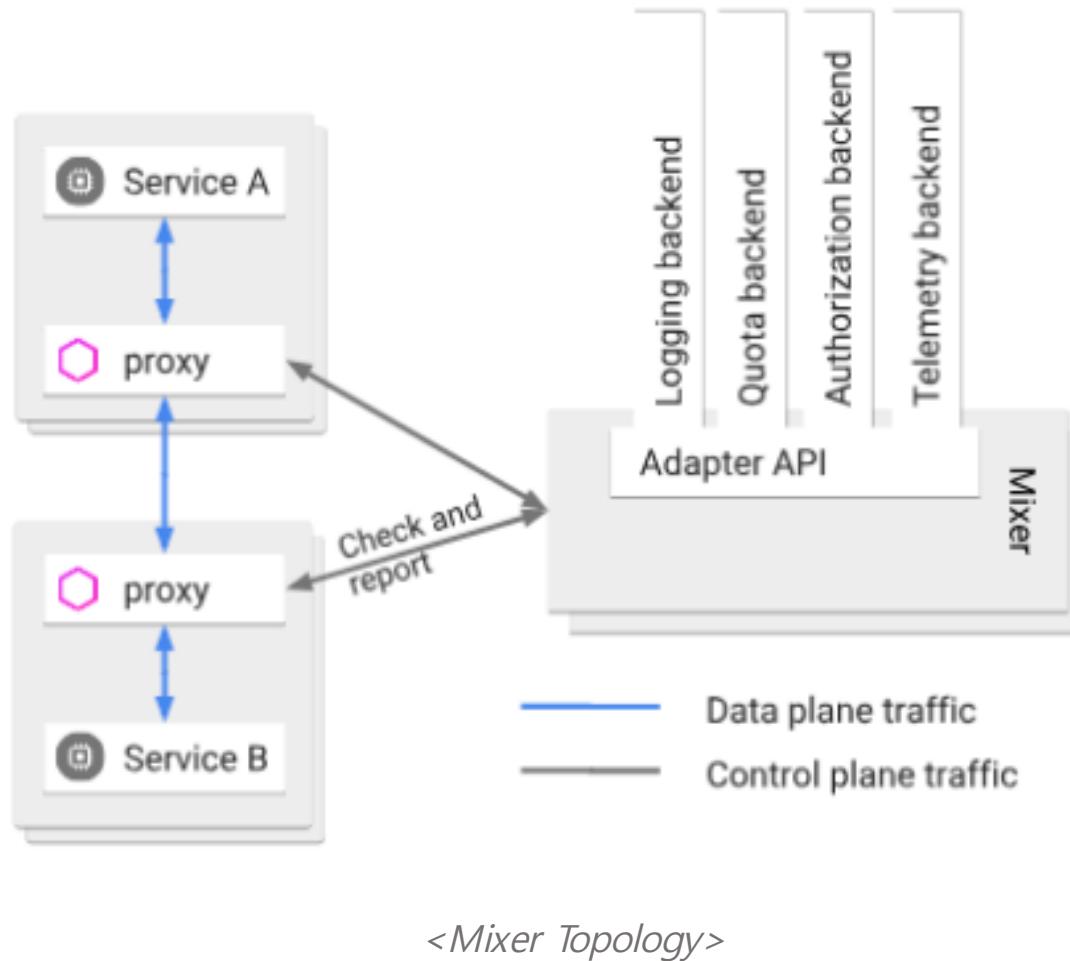
Istio (2) : Fault Injection

04

Istio (3): Distributed Tracing - Jaeger

## CLOUDZ LABS

Istio는 Mesh 내의 서비스에 대한 Telemetry(메트릭, 트래픽, 로그 등)를 수집할 수 있는 유연한 모델을 제공합니다.



✓ **Mixer**

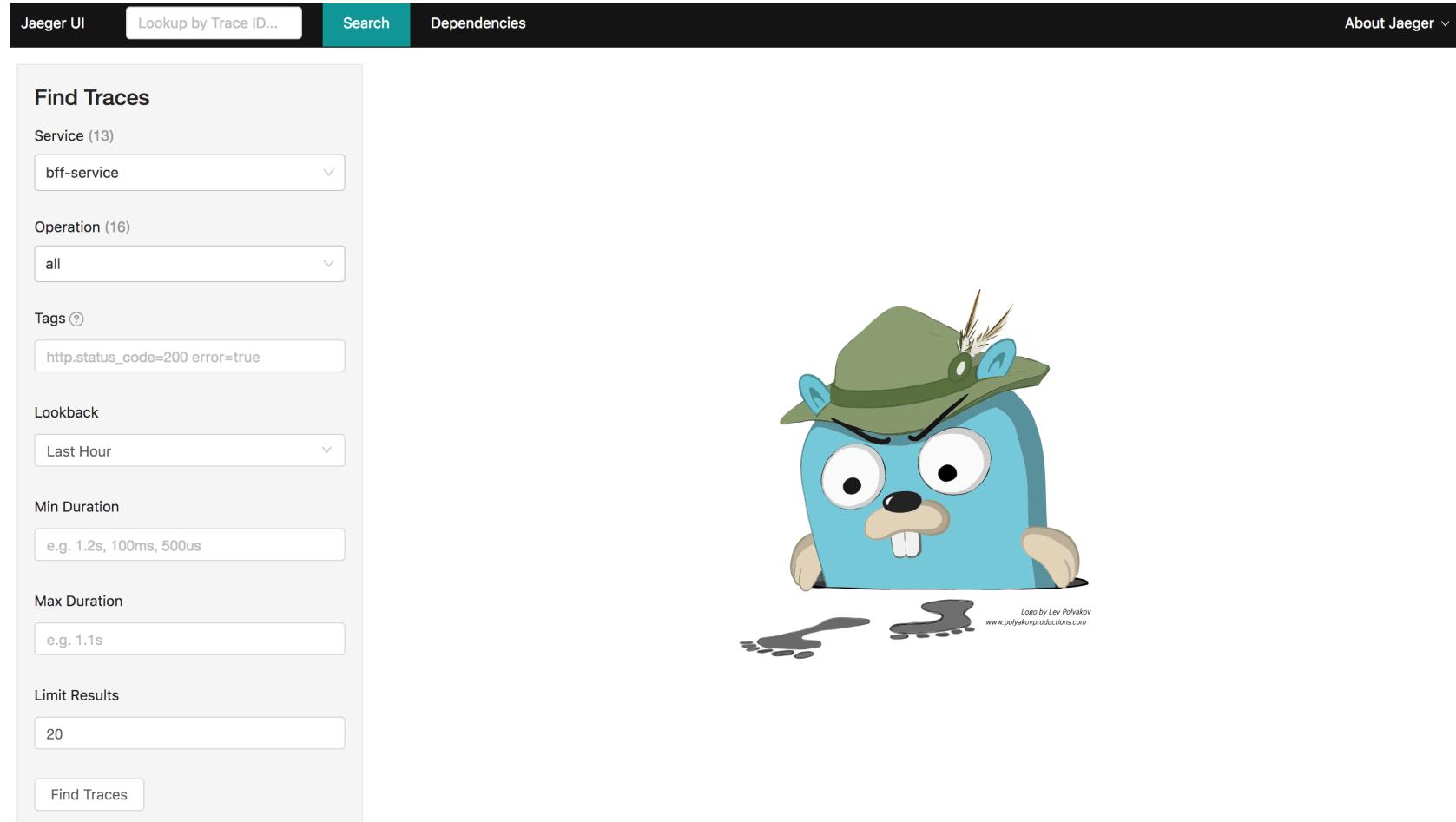
- 정책 제어 및 원격 측정(**Telemetry**) 수집을 제공하는 구성요소
- **Envoy** 사이드카는 각 요청 전에 **Mixer**를 통해 사전 검사를 수행하고 **Telemetry**를 보고하기 위해 **Mixer**를 호출함
- **Adapter API**

- 어댑터는 Mixer와 Prometheus 또는 Stackdriver와 같은 특정 외부 인프라 백엔드를 인터페이스하는 데 필요한 로직을 캡슐화하고, 믹서와 백엔드를 연결해줌

## CLOUDZ LABS

Istio를 구성함과 동시에 설치된 **Jaeger**를 통해 트래픽 분산 추적이 가능합니다. 이를 통하여 어떤 서비스에서 이슈가 발생하고 있는지를 빠르게 파악하여 대비할 수 있습니다.

### 1. http://169.56.107.250:31129 접속 -> Jaeger Dashboard



The image shows the Jaeger UI dashboard. At the top, there is a navigation bar with tabs: 'Jaeger UI' (selected), 'Lookup by Trace ID...', 'Search' (highlighted in teal), 'Dependencies', and 'About Jaeger'. Below the navigation bar is a search interface titled 'Find Traces' with the following fields:

- Service (13):** bff-service
- Operation (16):** all
- Tags (1):** http.status\_code=200 error=true
- Lookback:** Last Hour
- Min Duration:** e.g. 1.2s, 100ms, 500us
- Max Duration:** e.g. 1.1s
- Limit Results:** 20

At the bottom of the search interface is a 'Find Traces' button. To the right of the search interface is a large, friendly blue dog wearing a green hat, which is the official Jaeger logo. The logo has the text 'Logo by Lev Polyakov' and 'www.polyakovproductions.com' below it.

## CLOUDZ LABS

실습 종료 후 리소스 삭제

### 1. Fault Injection 설정 삭제

```
$ kubectl delete -f recommendation-service-fault-injection.yaml
```

### 2. recommendation deployment v1, v2 삭제

```
$ kubectl delete deploy/recommendation-service-deployment deploy/recommendation-service-deployment-v2
```

감사합니다

