



Istio tutorial

<https://www.oreilly.com/library/view/introducing-istio-service/9781491988770/>

김잠제/김병민

Index

001. Background

002. Install

003. Traffic Control

004. Service Resiliency



Background

Story of Istio background

1. Agile 의 등장으로 더 빠르고 잦은 서비스 배포 프로세스가 발생
 - a. MSA(Micro Service Architecture) 등장
 2. MSA 를 위해 팀과 서비스들을 작은 단위로 나누었을때 문제
 - a. MSA 에서는 서비스간 네트워크가 더욱 중요해지고, 서비스 추가에 따른 네트워크 복잡도는 **exponentially** 하게 증가
 - b. 몇달에 한번씩 배포하던 서비스를 일주일에 수십개의 서비스를 배포한다면 **dev to production** 운영
 3. Best Practice Company
 - a. Netflix - Ribbon, Hystrix, Eureka
 4. Container 의 등장
 - a. Docker, Kubernetes 의 등장으로 DevOps 팀을 위한 기반이 마련됨
 - b. well-automated pipeline 과 빠르게 pipeline 을 통과하는 **immutable image** 이 중요해짐
 5. **polyglot services** 를 배포하고 관리하는건 잘되고 있지만 각 서비스들간 **interaction** 에 대해 자동화 하는데에는 문제가 있음
 - a. 이에 대한 해결책으로 Istio 출현
- Kubernetes = helmsman or ship's pilot = 키잡이 또는 항해사
 - Istio = sail = 항해하다
 - helm = (배의)키

Istio feature

1. 대표기능
 - a. 서비스들 간 connection
2. 주요 특징
 - a. traffic control
 - i. declarative service discovery and routing
 1. not only round-robin
 - b. service discovery
 - i. observability of network flows (tracing)
 - c. load balancing
 - d. resilience
 - i. retry, timeout, circuit-breaker
 - e. observability
 - f. security

Istio Basic

1. Data Plane
 - a. Service proxy - **Envoy Proxy**
 - i. **retry, timeout, circuit breaker, service discovery, security** 와 같은 기능을 처리하는 proxy
 - b. **Sidecar**
 - i. “istio-proxy” container
 - ii. intercept all inbound and outbound network traffic
 - iii. reroute the traffic
 - iv. apply policies (ACL, rate limits, monitoring and tracing data)
2. Control Plane
 - a. pilot
 - i. up-to-date “routing table”
 - ii. support for RouteRule (**VirtualService**) and DestinationPolicy (**DestinationRule**)
 - b. mixer
 - i. istio-proxy 들로부터 **telemetry** 를 전달 받아 정책을 관리
 - ii. create ACL (whitelist and blacklist)
 - iii. rate-limit rule 조절
 - iv. plug-in 을 통해 custom metrics monitor 가능
 - c. auth (= Istio CA (certificate authority))
 - i. encrypting all traffic

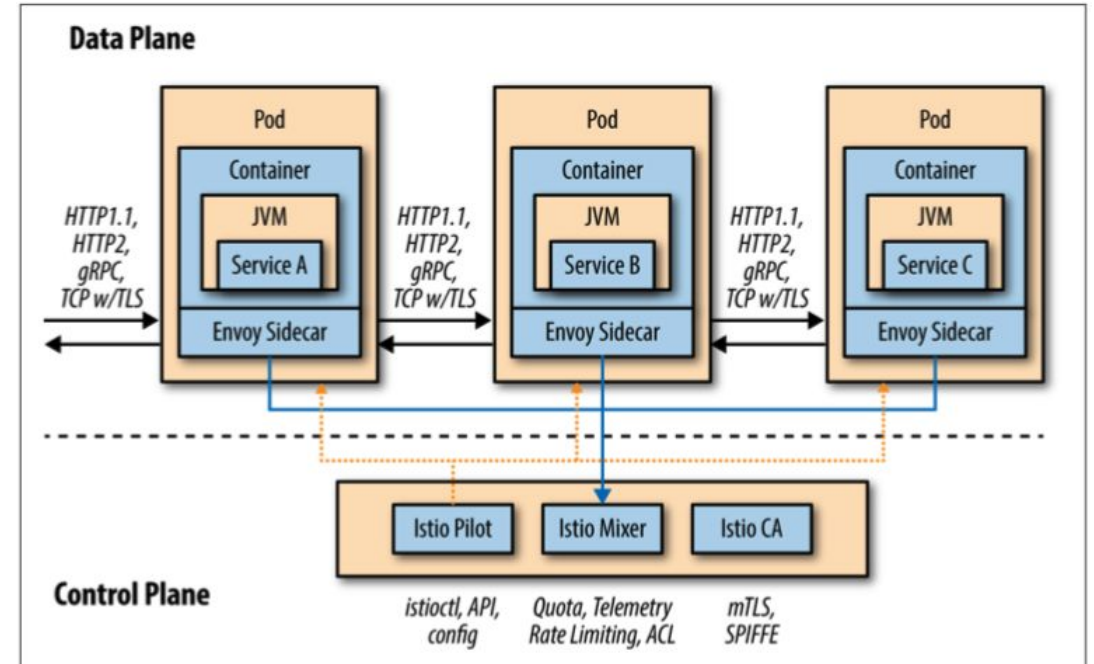


Figure 1-1. Data plane versus control plane

Install

Istio install

1. download link
 - a. [https://github.com/istio/istio/releases/download/\\${VERSION}/istio-\\${VERSION}-\\${OSEXT}.tar.gz](https://github.com/istio/istio/releases/download/${VERSION}/istio-${VERSION}-${OSEXT}.tar.gz)
 - b. 현재 최신버전은 1.1.2 (2019년 4월 7일)
 - c. 테스트 버전은 1.0.2 (kiali)
2. 실제 가이드는 책보다 공식 문서를 참고
 - a. <https://istio.io/docs/setup/kubernetes/install/helm/>
3. kubernetes 환경이 갖추어진 상태에서
 - a. `kubectl apply -f install/kubernetes/helm/helm-service-account.yaml`
 - b. `helm init --service-account tiller`
 - c. 버전 1.1 부터
 - i. `helm install install/kubernetes/helm/istio-init --name istio-init --namespace istio-system`
 - d. `helm upgrade --install istio ./temp/istio/istio-1.0.2/install/kubernetes/helm/istio --namespace istio-system --values ./custom-values.yaml`
 - i. custom-values.yaml : <https://github.com/istiokrsg/handson/blob/master/custom-values.yaml>

`git clone https://github.com:istiokrsg/handson.git`

`git clone https://github.com:redhat-developer-demos/istio-tutorial.git`

Istio install - result pods

```
> kubectl get po,svc -n istio-system
```

NAME	READY	STATUS	RESTARTS	AGE
pod/grafana-75485f89b9-p7nzt	1/1	Running	0	1m
pod/istio-citadel-84fb7985bf-q9hlp	1/1	Running	0	1m
pod/istio-egressgateway-bd9fb967d-drlgz	1/1	Running	0	1m
pod/istio-galley-655c4f9ccd-4qrnk	1/1	Running	0	1m
pod/istio-ingress-56795fd96c-w4gxr	1/1	Running	0	1m
pod/istio-ingress-56795fd96c-xkqxz	1/1	Running	0	21s
pod/istio-ingressgateway-688865c5f7-xl75g	1/1	Running	0	1m
pod/istio-pilot-6cd69dc444-zm5rk	2/2	Running	0	1m
pod/istio-policy-6b9f4697d-tss6k	2/2	Running	0	1m
pod/istio-sidecar-injector-8975849b4-4qp8s	1/1	Running	0	1m
pod/istio-statsd-prom-bridge-7f44bb5ddb-f745n	1/1	Running	0	1m
pod/istio-telemetry-6b5579595f-5r7nn	2/2	Running	0	1m
pod/istio-tracing-ff94688bb-82kx8	1/1	Running	0	1m
pod/kiali-644f5dd546-jp2g9	1/1	Running	0	1m
pod/prometheus-84bd4b9796-7m6p8	1/1	Running	0	1m
pod/servicegraph-749b5b897c-cldr4	1/1	Running	0	1m

Istio install - result services

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/grafana	ClusterIP	10.100.148.83	<none>	3000/TCP
service/istio-citadel	ClusterIP	10.106.49.149	<none>	8060/TCP,9093/TCP
service/istio-egressgateway	ClusterIP	10.103.236.161	<none>	80/TCP,443/TCP
service/istio-galley	ClusterIP	10.105.92.253	<none>	443/TCP,9093/TCP
service/istio-ingress	LoadBalancer	10.107.216.63	<pending>	80:32000/TCP,443:31717/TCP
service/istio-ingressgateway	LoadBalancer	10.102.128.44	localhost	80:31380/TCP,443:31390/TCP,31400:31400
service/istio-pilot	ClusterIP	10.103.198.84	<none>	15010/TCP,15011/TCP,8080/TCP,9093/TCP
service/istio-policy	ClusterIP	10.100.75.63	<none>	9091/TCP,15004/TCP,9093/TCP
service/istio-sidecar-injector	ClusterIP	10.109.77.29	<none>	443/TCP
service/istio-statsd-prom-bridge	ClusterIP	10.111.242.236	<none>	9102/TCP,9125/UDP
service/istio-telemetry	ClusterIP	10.108.154.55	<none>	9091/TCP,15004/TCP,9093/TCP,42422/TCP
service/jaeger-agent	ClusterIP	None	<none>	5775/UDP,6831/UDP,6832/UDP
service/jaeger-collector	ClusterIP	10.102.226.36	<none>	14267/TCP,14268/TCP
service/jaeger-query	ClusterIP	10.111.145.80	<none>	16686/TCP
service/kiali	ClusterIP	10.96.109.144	<none>	20001/TCP
service/prometheus	ClusterIP	10.108.213.84	<none>	9090/TCP
service/servicegraph	ClusterIP	10.110.140.45	<none>	8088/TCP
service/tracing	ClusterIP	10.111.236.40	<none>	80/TCP
service/zipkin	ClusterIP	10.99.157.163	<none>	9411/TCP

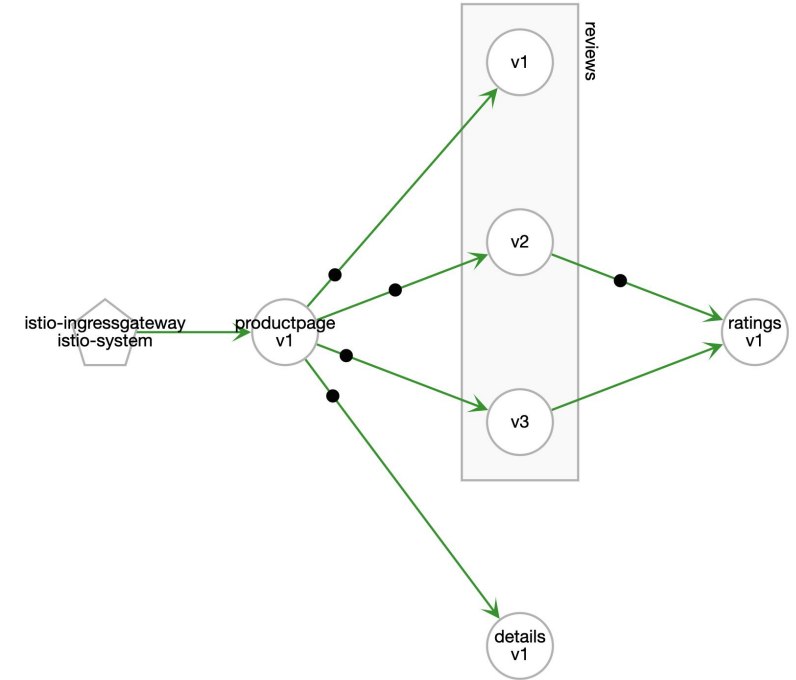
Istio uninstall

1. helm 으로 설치했으니 helm 으로 삭제
 - a. helm del --purge [istio](#)
 - b. 버전 1.1 부터
 - i. helm del --purge istio-init
 - c. kubectl delete crds {istio.io crds LIST}
 - d. kubectl delete namespace [istio-system](#)

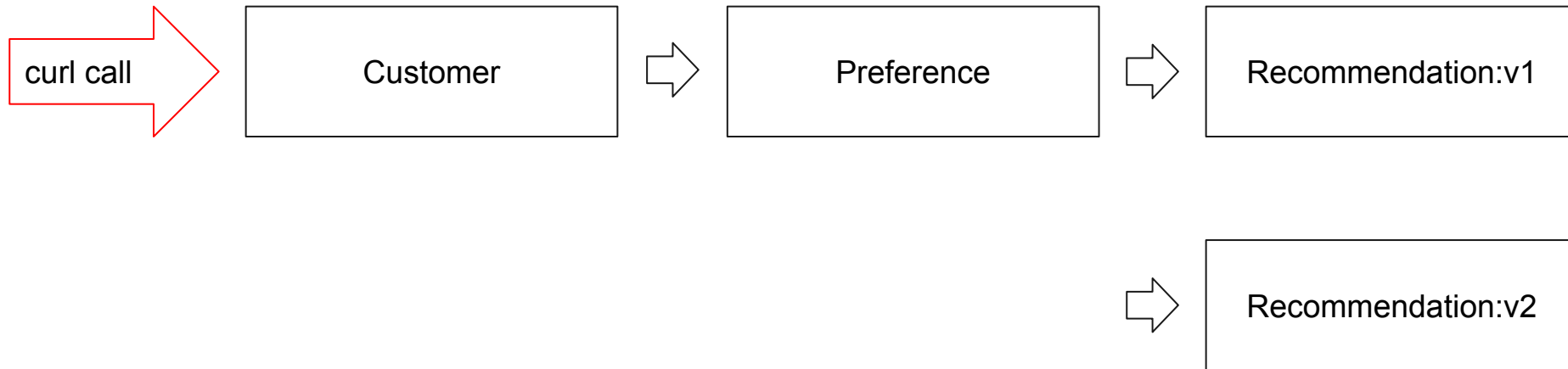
Sample application (bookinfo)

Istio 공식 샘플

1. # auto injection
 - a. `kubectl label namespace default istio-injection=enabled`
2. # install pod, service
 - a. `kubectl apply -f {ISTIOHOME}/samples/bookinfo/platform/kube/bookinfo.yaml`
3. # set network resources (gateway, virtualservices)
 - a. `kubectl apply -f {ISTIOHOME}/samples/bookinfo/networking/bookinfo-gateway.yaml`



Original 예제



customer

1. 책 예제를 git clone
 - a. <https://github.com/redhat-developer-demos/istio-tutorial>
 - b. git clone [git@github.com:redhat-developer-demos/istio-tutorial](https://github.com/redhat-developer-demos/istio-tutorial).git
2. customer 소스코드 위치로 이동 후 빌드
 - a. cd istio-tutorial/customer/java/quarkus
 - b. mvn clean package -DskipTests
3. package 의 결과 target directory 가 만들어졌음을 확인후 docker image 생성
 - a. docker build -t example/customer .
 - b. 확인 : docker images | grep example
4. injection
 - a. injection 확인 : istioctl kube-inject -f ../../kubernetes/Deployment.yml
 - b. kubectl create namespace tutorial
 - c. kubectl apply -f <(istioctl kube-inject -f ../../kubernetes/Deployment.yml) -n tutorial
5. service apply
 - a. oc 로 expose 할거라면
 - i. kubectl apply -f ../../kubernetes/Service.yml -n tutorial
 - b. kubectl 의 nodeport 로 expose 할거라면
 - i. kubectl expose deployment customer -n tutorial --type=NodePort --name=customer
6. check
 - a. kubectl get po,svc -n tutorial
 - b. curl localhost:32722
 - i. NodePort 에서 할당된 포트 확인

preference

1. preference 소스코드 위치로 이동 후 빌드
 - a. `cd istio-tutorial/preference/java/quarkus`
 - b. `mvn clean package -DskipTests`
2. package 의 결과 target directory 가 만들어졌음을 확인후 docker image 생성
 - a. `docker build -t example/preference:v1 .`
 - b. 확인 : `docker images | grep example`
3. injection
 - a. injection 확인 : `istioctl kube-inject -f ../../kubernetes/Deployment.yml`
 - b. `kubectl apply -f <(istioctl kube-inject -f ../../kubernetes/Deployment.yml) -n tutorial`
4. service apply
 - a. `kubectl apply -f ../../kubernetes/Service.yml -n tutorial`
5. check
 - a. `kubectl get po,svc -n tutorial`

recommendation

1. recommendation 소스코드 위치로 이동 후 빌드
 - a. `cd istio-tutorial/preference/java/quarkus`
 - b. `mvn clean package -DskipTests`
2. package 의 결과 target directory 가 만들어졌음을 확인후 docker image 생성
 - a. `docker build -t example/recommendation:v1 .`
 - b. 확인 : `docker images | grep example`
3. injection
 - a. injection 확인 : `istioctl kube-inject -f ../../kubernetes/Deployment.yml`
 - b. `kubectl apply -f <(istioctl kube-inject -f ../../kubernetes/Deployment.yml) -n tutorial`
4. service apply
 - a. `kubectl apply -f ../../kubernetes/Service.yml -n tutorial`
5. check
 - a. `kubectl get po,svc -n tutorial`

customer, preference, recommendation

```
~/git/github/study/istio-tutorial/recommendation/java/quarkus master
```

```
> kubectl get po,svc -n tutorial
```

NAME	READY	STATUS	RESTARTS	AGE
pod/customer-5c77b74458-zp2ns	2/2	Running	0	1h
pod/preference-v1-74499588c9-7nx4l	2/2	Running	0	13m
pod/recommendation-v1-579db4dcb9-rwpfr	2/2	Running	0	20s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/customer	NodePort	10.98.95.80	<none>	8080:32722/TCP,8778:31495/TCP,9779:30216/TCP	1h
service/preference	ClusterIP	10.105.37.219	<none>	8080/TCP	11m
service/recommendation	ClusterIP	10.111.167.131	<none>	8080/TCP	14s

```
~/git/github/study/istio-tutorial/recommendation/java/quarkus master
```

```
> curl localhost:32722
```

```
customer => preference => recommendation v1 from '579db4dcb9-rwpfr': 1
```

Traffic Control

Smarter Canaries

1. 광부들이 탄광에 들어갈때 위험한 가스누출을 알아챌수 있도록 카나리아 새를 데리고 들어가는것에서 유래
2. 운영환경에 새로운 버전을 배포할때 문제점을 미리 알아챌수 있도록 배포하는 subset 을 의미
3. 카나리 배포 방식
 - a. kubernetes 는 service 에 있는 모든 pod 에 대해 round-robin load balancing 으로 동작한다.
 - b. 만약 10%의 사용자에게만 새버전을 제공하여 테스트 하고 싶다면
 - c. pod 10개중 1개를 새버전의 container 로 배포한다.
 - d. Istio 를 사용하면 더 상세하게 설정할 수 있다.
 - e. 트래픽 2%만 새로배포된 3개의 pod 로 route 되도록 설정할 수 있다.
 - f. 그리고 서서히 트래픽을 늘려가면서 이전 버전을 제거할 수 있다.

Traffic Routing

1. recommendation v2 를 배포
 - a. recommendationResource.java 에서 `STRING_FORMAT` 변경해서 재빌드
 - i. `private static final String RESPONSE_STRING_FORMAT = "recommendation v2 from '%s': %d\n";`
 - b. `mvn clean package -DskipTests`
 - c. `docker build -t example/recommendation:v2 .`
 - d. `kubectl apply -f <(istioctl kube-inject -f ../../kubernetes/Deployment-v2.yml) -n tutorial`
 - e. `kubectl get pods -w -n tutorial`

Traffic Routing

```
~/git/github/study/istio-tutorial/recommendation/java/quarkus master* 39s
> kubectl get po,svc -n tutorial
NAME                                     READY   STATUS    RESTARTS   AGE
pod/customer-5c77b74458-zp2ns           2/2     Running   0           2h
pod/preference-v1-74499588c9-7nx4l      2/2     Running   0           47m
pod/recommendation-v1-579db4dcb9-rwpfr   2/2     Running   0           34m
pod/recommendation-v2-86c9458d5c-v5k9g  2/2     Running   0           44s

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)                                     AGE
service/customer    NodePort    10.98.95.80     <none>           8080:32722/TCP,8778:31495/TCP,9779:30216/TCP 1h
service/preference  ClusterIP   10.105.37.219   <none>           8080/TCP                                     46m
service/recommendation ClusterIP   10.111.167.131 <none>           8080/TCP                                     34m

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v1 from '579db4dcb9-rwpfr': 11

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 1

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v1 from '579db4dcb9-rwpfr': 12

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 2

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v1 from '579db4dcb9-rwpfr': 13

~/git/github/study/istio-tutorial/recommendation/java/quarkus master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 3
```

Traffic Routing using Istio

1. v2 로만 패킷이 가도록 설정
 - a. `kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yml -n tutorial`
 - b. `kubectl create -f istiofiles/virtual-service-recommendation-v2.yml -n tutorial`
 - c.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v2
        weight: 100
```

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


Traffic Routing using Istio

```
~/git/github/study/istio-tutorial master* 8s
> kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yml -n tutorial
destinationrule.networking.istio.io/recommendation created

~/git/github/study/istio-tutorial master*
> kubectl create -f istiofiles/virtual-service-recommendation-v2.yml -n tutorial
virtualservice.networking.istio.io/recommendation created

~/git/github/study/istio-tutorial master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 4

~/git/github/study/istio-tutorial master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 5

~/git/github/study/istio-tutorial master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 6

~/git/github/study/istio-tutorial master*
> curl localhost:32722
customer => preference => recommendation v2 from '86c9458d5c-v5k9g': 7
```

Traffic Routing using Istio

1. v1 로만 패킷이 가도록 재설정
 - a. `kubectl replace -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial`
2. virtual service 삭제
 - a. `kubectl delete -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial`

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
        weight: 100
```


canary deployment

1. scenario
 - a. v2 를 새 버전이라고 가정한다
 - b. v1 : 90%, v2 : 10% 의 비율로 트래픽을 조절
2. virtual service 적용
 - a. `kubectl create -f istiofiles/virtual-service-recommendation-v1_and_v2.yml -n tutorial`
3. 부하
 - a. `while(true){ curl localhost:32722; sleep 2 }`
4. virtual service 교체 (트래픽 비율 수정 75:25)
 - a. `kubectl replace -f istiofiles/virtual-service-recommendation-v1_and_v2_75_25.yml -n tutorial`

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

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
        weight: 75
    - destination:
        host: recommendation
        subset: version-v2
        weight: 25
```

canary deployment based on user-agent header (advanced)

1. customer 에서 request header 에 baggage-user-agent 를 추가
 - a. Safari 가 포함된 문자열이면 v2 로 트래픽을 route
2. test 방법
 - a. Mac 에서는 크롬으로 접속해도 Safari 임
 - b. `curl -A Safari localhost:32722`

같은 방법으로 모바일로 접근한 유저만 v2 로 보낼수 있음

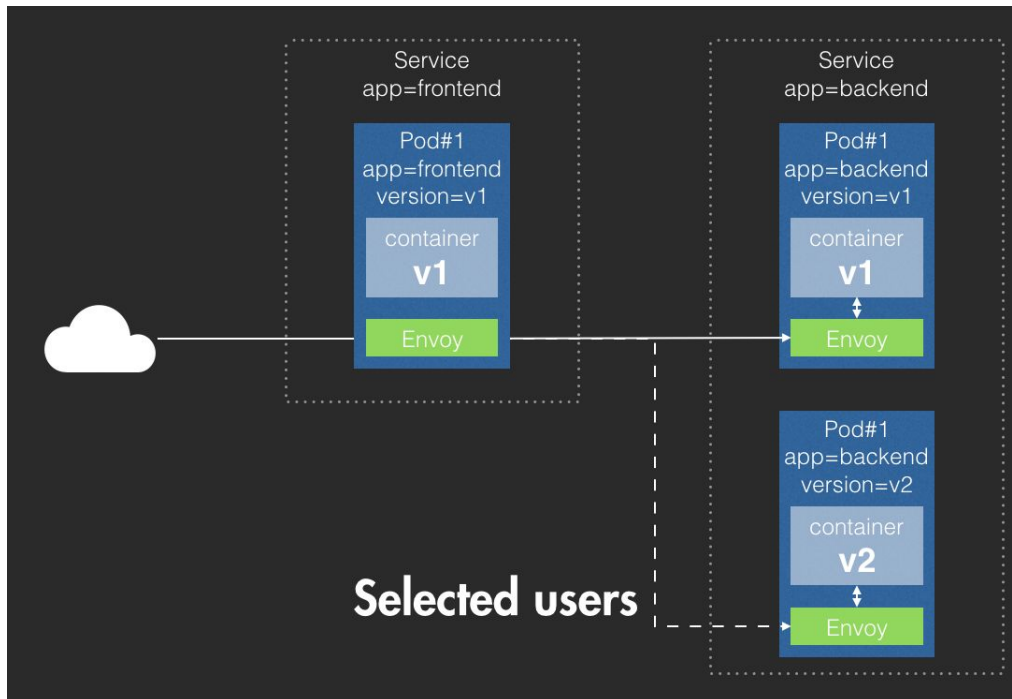
```
public class BaggageHeadersFactory implements ClientHeadersFactory {  
    @Override  
    public MultivaluedMap<String, String> update(MultivaluedMap<String, String> in  
        MultivaluedHashMap<String, String> headers = new MultivaluedHashMap<>();  
        String userAgent = incomingHeaders.getFirst("user-agent");  
        headers.putSingle("baggage-user-agent", userAgent);  
        return headers;  
    }  
}
```

```
apiVersion: networking.istio.io/v1alpha3  
kind: VirtualService  
metadata:  
  name: recommendation  
spec:  
  hosts:  
  - recommendation  
  http:  
  - match:  
    - headers:  
      baggage-user-agent:  
        regex: .*Mobile.*  
    route:  
    - destination:  
        host: recommendation  
        subset: version-v2  
  - route:  
    - destination:  
        host: recommendation  
        subset: version-v1
```

```
apiVersion: networking.istio.io/v1alpha3  
kind: VirtualService  
metadata:  
  name: recommendation  
spec:  
  hosts:  
  - recommendation  
  http:  
  - match:  
    - headers:  
      baggage-user-agent:  
        regex: .*Safari.*  
    route:  
    - destination:  
        host: recommendation  
        subset: version-v2  
  - route:  
    - destination:  
        host: recommendation  
        subset: version-v1
```

Dark Launch

1. 운영환경에 배포하지만 일부 사용자(내부직원 등)에게만 새버전을 노출 하는 방식
 - a. facebook 에서는 Gatekeeper (<https://www.facebook.com/notes/facebook-engineering/building-and-testing-at-facebook/10151004157328920/>) 라는 dark launch tool 이 있음
 - b. 실제 고객은 새버전의 존재를 모름
 - c. duplication or mirror production traffic 을 새버전으로 routing 하여 새버전을 운영환경에서 테스트할 수 있음
2. Istio 를 가지고 mirror traffic 을 사용할 수 있음
 - a. 클러스터 안에서 mirrored request 를 async 하게 보내기 때문에 이전 버전에는 영향이 없음
3. `kubectl apply -f istiofiles/virtual-service-recommendation-v1-mirror-v2.yml -n tutorial`



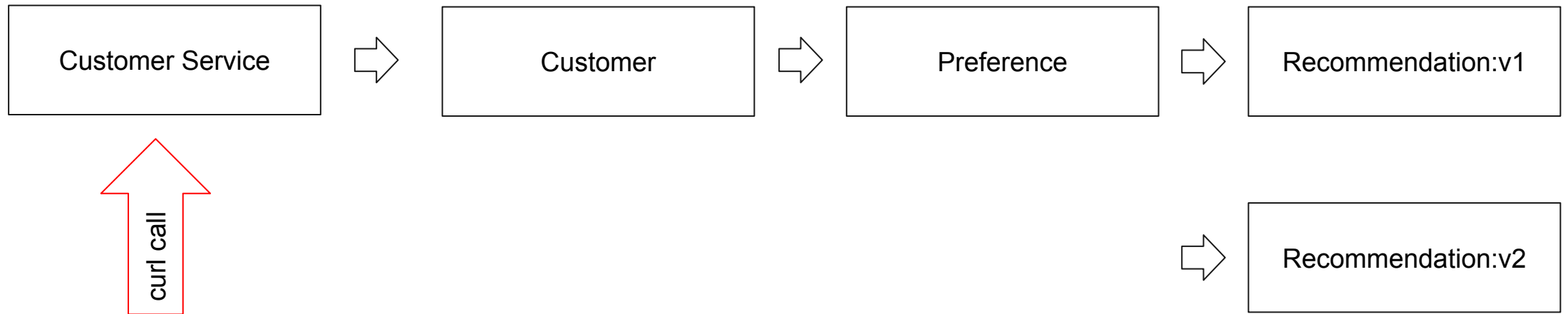
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
    mirror:
      host: recommendation
      subset: version-v2
```

Egress

1. Istio 는 클러스터 밖으로 나가는 모든 outbound 를 block 하는게 default
2. 명시적으로 routing rule 을 만들어줘야 외부로 트래픽이 나갈수 있음
3. Resource name : ServiceEntry
4. 1.1.0 버전 이후 부터 config map 의 설정이 추가되었음 (**global.outboundTrafficPolicy.mode**)
 - a. <https://istio.io/docs/tasks/traffic-management/egress/>
 - b. ALLOW_ALL, REGISTRY_ONLY
 - c. REGISTRY_ONLY 하고 service entry 설정해서 사용하는방법을 권장

Service Resiliency

<https://www.oreilly.com/library/view/introducing-istio-service/9781491988770/ch04.html>



Service Resiliency

1. 서비스의 복원력을 구현하기 위해 다음과 같은 기능들을 제공하고 있다
 - a. Client-side load balancing
 - i. 쿠버네티스의 load balancing 을 강화
 - b. Timeout
 - i. 응답을 기다린후 N (초) 후에 give up
 - c. Retry
 - i. 하나의 pod 가 에러 나면, 다른 pod 로 재시도
 - d. Circuit breaker
 - i. 서비스가 내려가는 대신, 더이상의 request 를 reject 하는 circuit 제공
 - e. Pool ejection
 - i. 에러가 발생하는 pod를 load balancing pool 에서 제거

Load Balancing

1. Client side load balancing(<http://blog.leekyoungil.com/?p=259>)
 - a. L4, L7 같은 고가의 장비가 필요없다 (no more HAProxy)
 - b. client 에서 로드밸런싱 알고리즘이 동작해야 한다
 - c. 오류가 발생했을때 적절하게 동작해야한다 (timeout, retry)
2. Istio 에서 지원하는 LB 기능들
 - a. ROUND_ROBIN
 - i. 순서대로 돌아가며 한번씩(default)
 - b. RANDOM
 - i. 랜덤
 - c. LEAST_CONN
 - i. weighted least request load balancing 을 구현
 - ii. https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/load_balancing/load_balancers
 - iii. TL;DR : 랜덤하게 2개(default) 골라서 active request 가 적은 서버로 call

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: recommendation
spec:
  host: recommendation
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
```

```
claud@claud-500R5N-500R5Y-501R5N:~/git/openshift/istio-tutorial/istiofiles$ istioctl create -f destination-rule-recommendation-random.yml
Created config destination-rule/default/recommendation at revision 67948
claud@claud-500R5N-500R5Y-501R5N:~/git/openshift/istio-tutorial/istiofiles$ while true; do curl http://192.168.99.96:80/customer; sleep .5; done
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 34
customer => preference => recommendation v2 from '66bd89d66c-wqzs6': 32
customer => preference => recommendation v2 from '66bd89d66c-wqzs6': 33
customer => preference => recommendation v2 from '66bd89d66c-wqzs6': 34
customer => preference => recommendation v2 from '66bd89d66c-sncsd': 11
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 35
customer => preference => recommendation v2 from '66bd89d66c-wqzs6': 35
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 36
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 37
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 38
customer => preference => recommendation v2 from '66bd89d66c-sncsd': 12
customer => preference => recommendation v1 from '679dfdf957-ctkfr': 39
customer => preference => recommendation v2 from '66bd89d66c-wqzs6': 36
```


Timeout

1. 서비스에 타임아웃을 설정하여 사용자에게 응답성을 빠르게 하는 기능
2. 무한 대기 방지

```
claud@claud-500R5N-500R5Y-501R5N:~/git/istio-tutorial/recommendation/java/
.5; done
customer => preference => recommendation v2 from '7fb655b9bf-s2lp1': 3

real    0m3.031s
user    0m0.003s
sys     0m0.005s
customer => preference => recommendation v1 from '6b4df7d9d9-74wpb': 85

real    0m0.054s
user    0m0.017s
sys     0m0.006s
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
      timeout: 1.000s
```

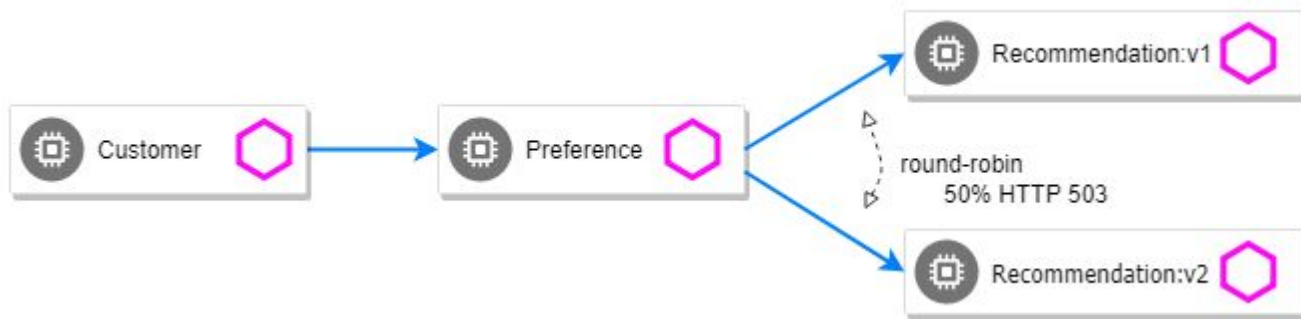
```
claud@claud-500R5N-500R5Y-501R5N:~/git/istio-tutorial/recommendation/java/quark
.5; done
customer => preference => recommendation v1 from '6b4df7d9d9-74wpb': 87

real    0m0.033s
user    0m0.004s
sys     0m0.004s
customer => Error: 503 - preference => Error: 504 - upstream request timeout

real    0m1.045s
user    0m0.001s
sys     0m0.014s
customer => preference => recommendation v1 from '6b4df7d9d9-74wpb': 88
```

HTTP Error 503

1. 간헐적인 오류일 때, **Retry** 설정으로 서비스의 견고성을 갖출 수 있다.
2. **DestinationRule** 설정 : **Recommendation v1, v2 round-robin** 설정
3. **VirtualService** 설정으로 **503 에러를 50% 설정함**



```
customer => preference => recommendation v1 from '6b4df7d9d9-s2hkh': 5
real    0m0.076s
user    0m0.016s
sys     0m0.009s
customer => Error: 503 - preference => Error: 503 - fault filter abort

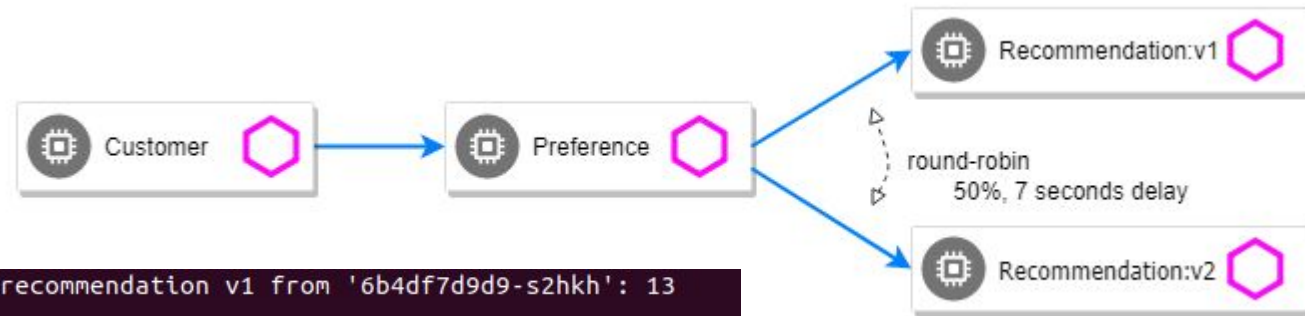
real    0m0.039s
user    0m0.019s
sys     0m0.005s
customer => Error: 503 - preference => Error: 503 - fault filter abort

real    0m0.045s
user    0m0.017s
sys     0m0.004s
customer => preference => recommendation v2 from '7fb655b9bf-s2lpl': 53
real    0m3.039s
```

```
kind: VirtualService
metadata:
  name: recommendation
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: app-recommendation
    fault:
      abort:
        httpStatus: 503
        percent: 50
```

Delay

1. 분산환경에서 **fault** 된 상황은 서비스의 **down**이 아니라 늦은 네트워크 응답이 서비스가 계단식으로 실패 하게 됩니다.
2. **DestinationRule** 설정 : **Recommendation v1, v2 round-robin** 설정
3. **VirtualService** 설정으로 30% 확률로 7초 딜레이를 설정

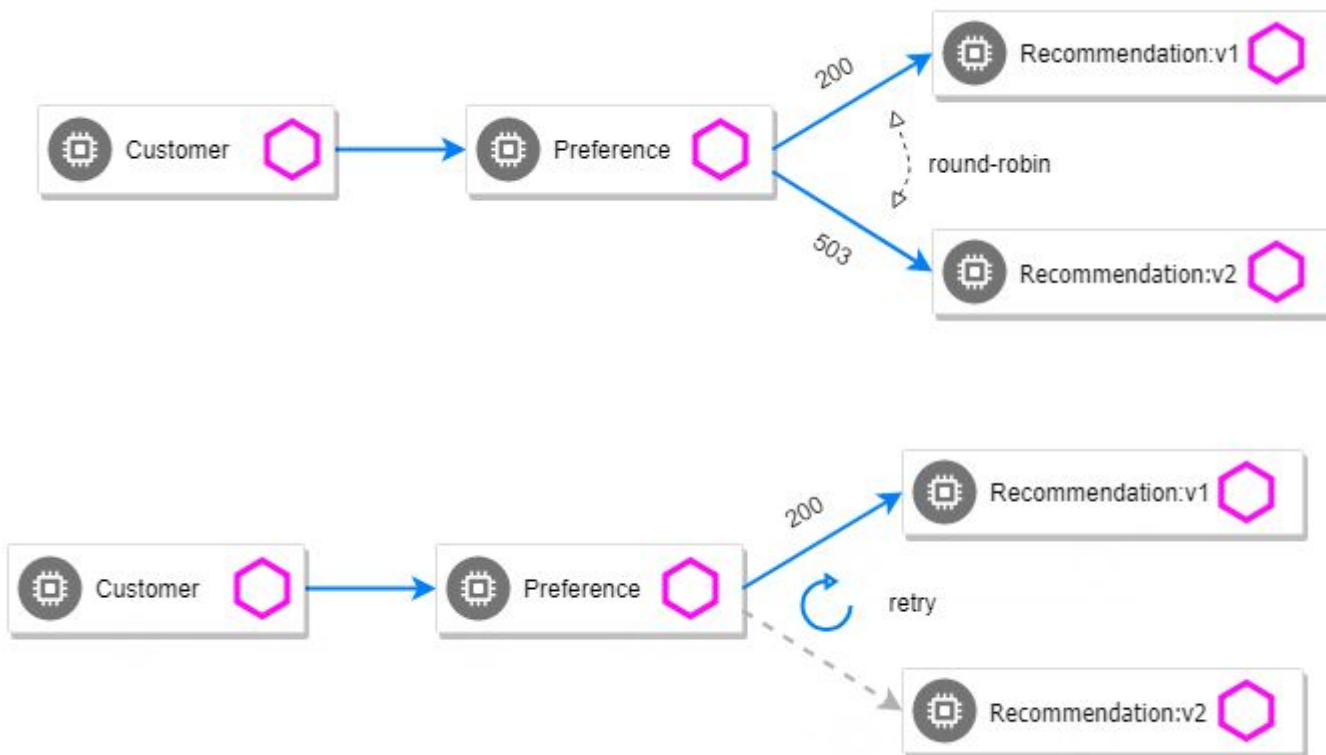


```
customer => preference => recommendation v1 from '6b4df7d9d9-s2hkh': 13
real    0m0.056s
user    0m0.008s
sys     0m0.015s

customer => preference => recommendation v2 from '7fb655b9bf-s2lpl': 62
real    0m10.062s
user    0m0.013s
sys     0m0.015s
customer => preference => recommendation v1 from '6b4df7d9d9-s2hkh': 14
real    0m0.017s
user    0m0.000s
sys     0m0.006s
customer => preference => recommendation v2 from '7fb655b9bf-s2lpl': 63
real    0m10.017s
user    0m0.004s
sys     0m0.004s
customer => preference => recommendation v1 from '6b4df7d9d9-s2hkh': 15
real    0m7.041s
user    0m0.006s
sys     0m0.006s
```

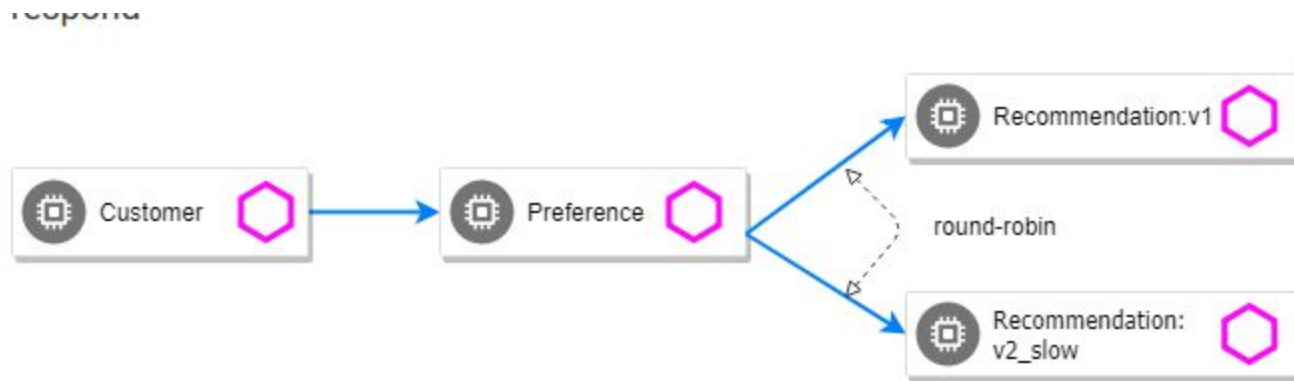
Retry

1. 간헐적인 오류일 때, **Retry** 설정으로 서비스의 견고성을 갖출 수 있다.
2. **recommendation-v2** 100% 실패로 설정
3. **VirtualService** 설정 **retry rule add**



Timeout- 2

1. recommendation-v2 deployment 를 조정하여 3초를 추가함
2. 라운드-로빈 동작시 3초인것을 확인
3. VirtualService 설정 timeout 을 1초로 설정
4. v2 로 갈때, 1초 안에 에러응답이 오는 것을 확인



Circuit Breaker

1. 느린 서비스를 뺀다라는 개념으로 이해-> 느린 서비스를 룰에 의해 제거 됨으로써 서비스의 탄력성을 지키게 됨.
2. Netflix의 Hystrix 라이브러리가 2012 년에 출시되면서 circuit Breaker패턴이 대중화됨.
3. Eureka (서비스 검색), 리본 (로드 밸런싱) 및 Hystrix (circuit Breaker 및 bulk head)와 같은 Netflix 라이브러리는 빠르게 대중화됨.
4. Netflix OSS는 Kubernetes / OpenShift가 출시되기 전에 만들어졌으며 몇 가지 단점이 있습니다.
5. 하나는 Java 전용이고 두 가지는 응용 프로그램 개발자가 포함 라이브러리를 올바르게 사용해야한다는 것입니다..

Circuit Breaker

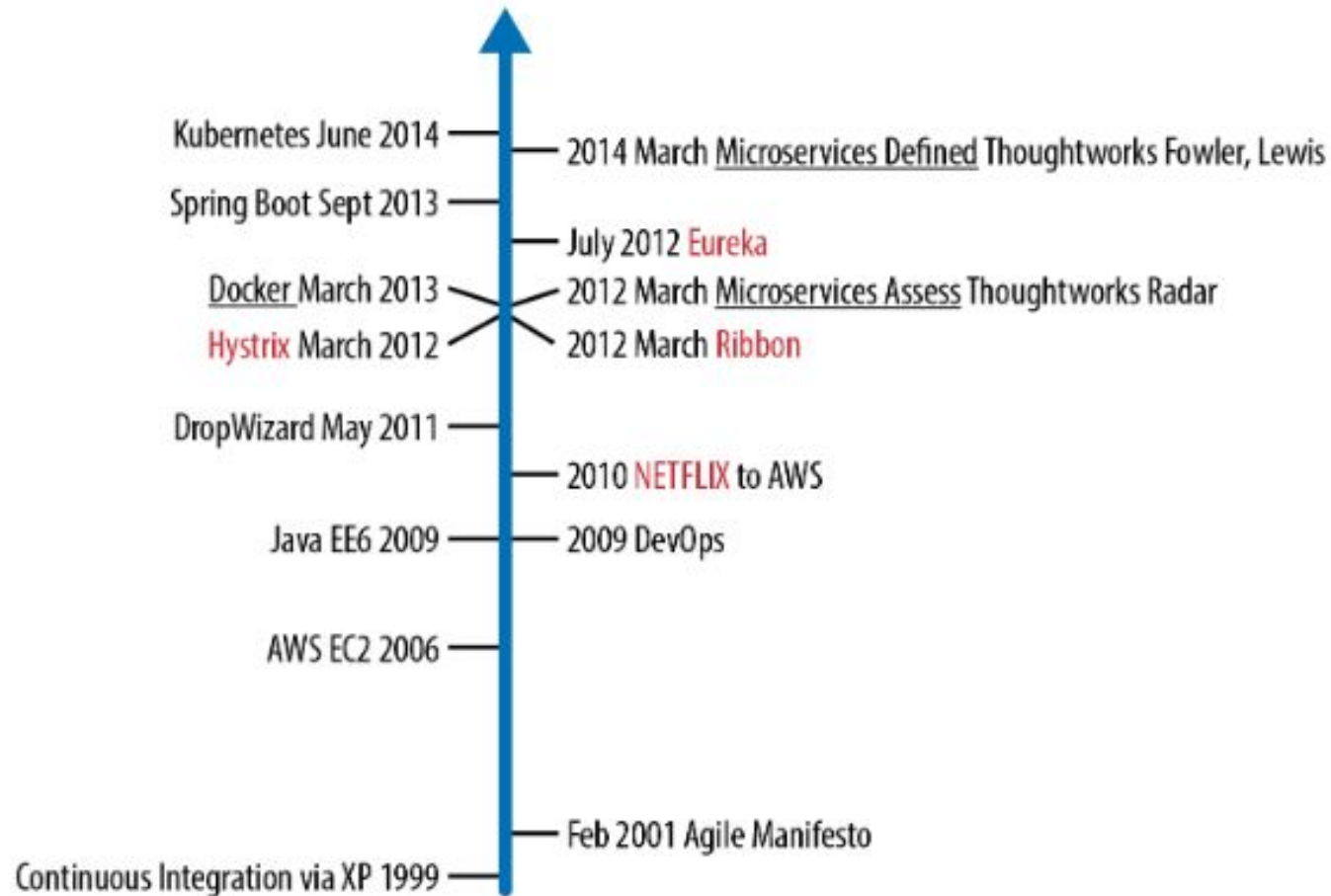


Figure 4-1. Microservices timeline

Circuit Breaker

1. 시나리오
 - a. v1 과 slow v2 recommendation 에 동등한 분배되게 부하 분산 설정함
2. 절차
 - a. v2 slow 설정
 - b. v1, v2 destination-rule 설정-라벨링
 - c. 부하분산 트래픽 설정 virtual service

Circuit Breaker

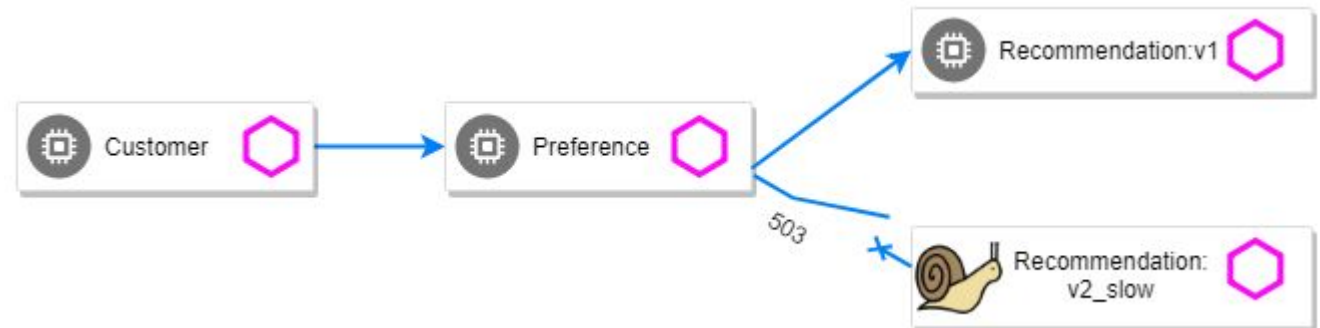
Test 1: Load test without circuit breaker - siege 를 이용한 부하 테스트

1. 모든 요청은 성공함
2. v2 인스턴스가 느린 성능이었기 때문에 테스트를 실행하는 데 약간의 시간이 걸림

Test 2: Load test with circuit breaker - - siege 를 이용한 부하 테스트

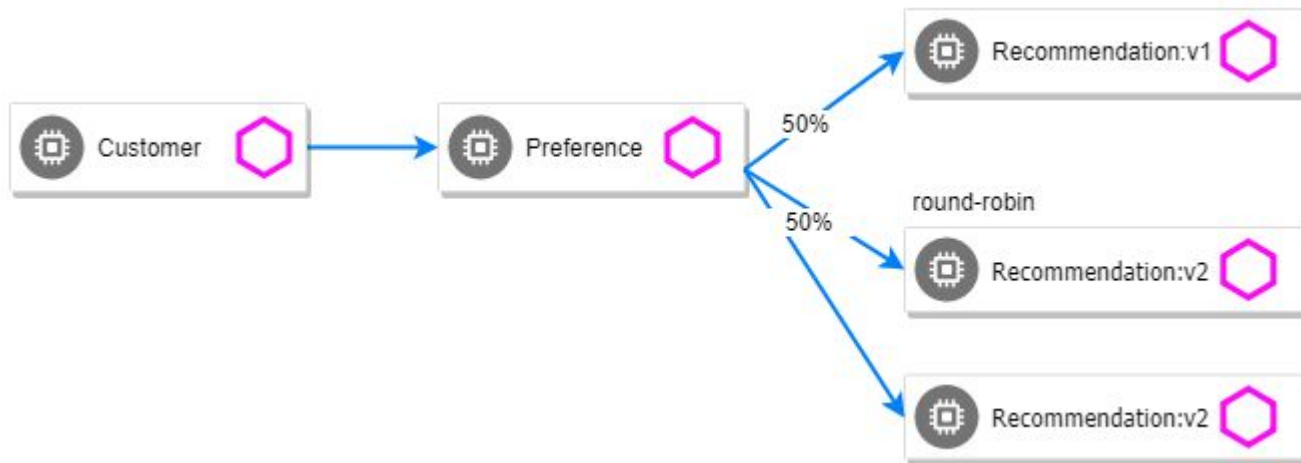
1. 프로덕션 시스템에서는 3초 지연이 동일한 인스턴스에 대한 너무 많은 동시 요청으로 인하여 발생한다고 가정.
2. 요청에 대하여 대기열(큐)추가하여 느린 인스턴스가 요청에 대하여 적게 할당 되게 하는 것이 가능함

DestinationRule 위의 사항을 적용



Pool Ejection

1. 잘못 동작 하는 서비스를 Envoy 를 통해 풀에서 빼는 동작하는 하는 것이 기본개념
2. Pool Ejection 또는 이상 값 탐지는 클라이언트 요청을 처리 할 instance pods 의 요청을 처리할 pool이 있을 때 마다 복원하는 전략
3. 요청이 특정 instance pods 전달 되고, 실패할 경우(50x 오류 코드 반환) istio는 특정 시간동안 풀에서 인스턴스를 꺼내고, 이것은 15초동안 유지됨.
4. 상태가 좋은 인스턴스 pods 만 풀에 유지 함으로 전체 가용성을 높임.



Pool Ejection

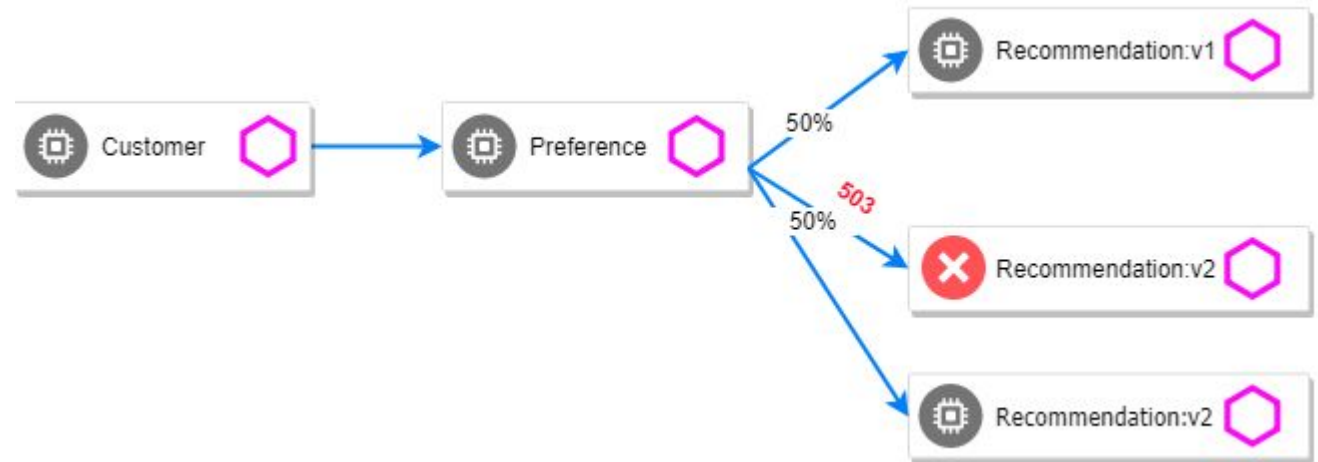
Test 1: Load test without failing instances

Test 2: Load test with failing instance

에러 확률은 25%,

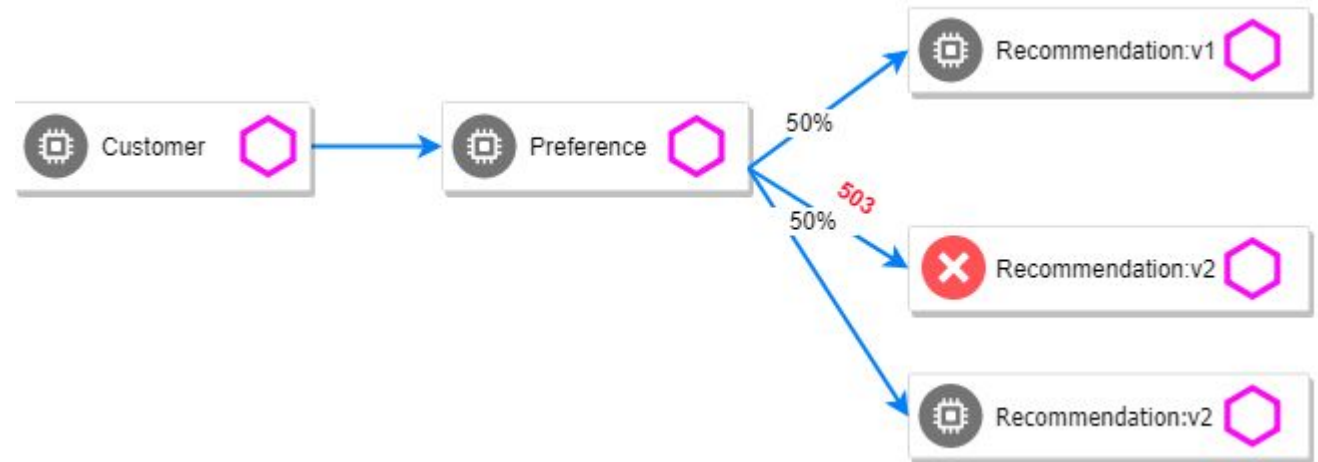
50% v1

50% v2 중에서 1/2



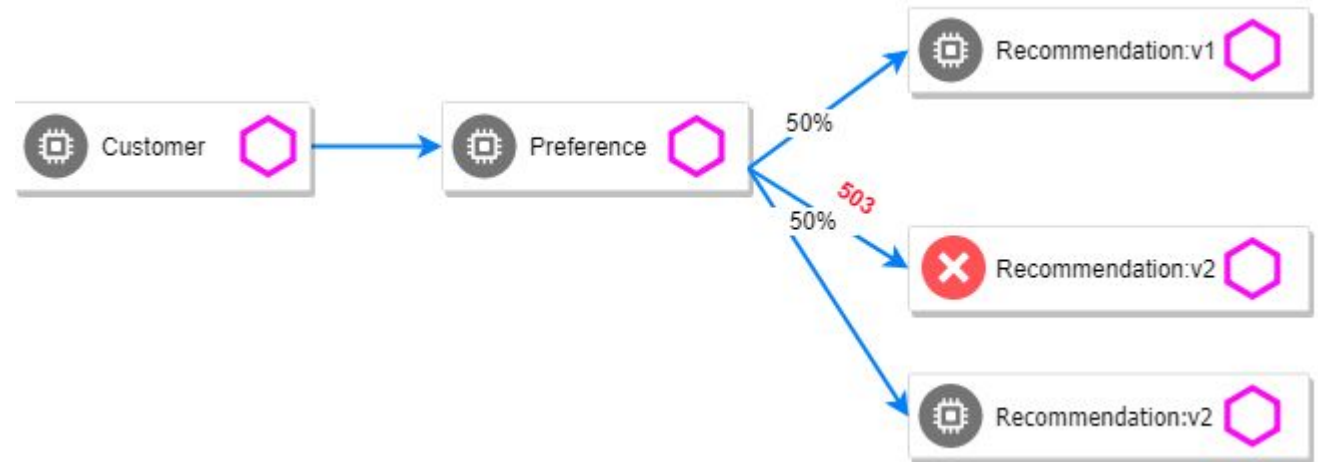
Pool Ejection

Test 3: Load test with failing instance and with pool ejection



Pool Ejection

Test 3: Load test with failing instance and with pool ejection



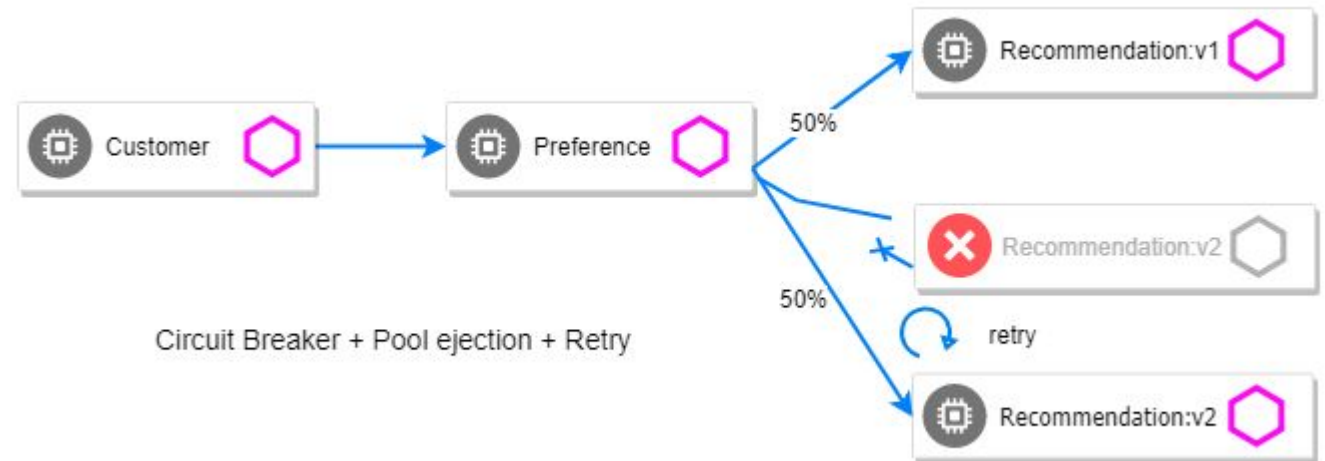
Circuit Breaker + Pool Ejection + Retries

Test 4: Ultimate resilience with retries, circuit breaker, and pool ejection

Circuit Breaker: 인스턴스에 대한 여러개의 동시 요청을 방지

Pool Ejection: 응답 인스턴스 풀에서 실패한 인스턴스를 제거

Retries: 풀에 배출 발생한 경우를 대비하여 다른 인스턴스로 요청을 전달



로컬환경 설정 및 테스트

https://github.com/claude-kim/istio-tutorial/blob/master/minikube_istio.md

학습자료 : <http://learn.openshift.com/servicemesh>



To be continue...

<https://www.oreilly.com/library/view/introducing-istio-service/9781491988770/>

next