

Cloud Service

V.1.0.0



Copyright©2018 by SK CLOUDZ LABS All rights reserved.

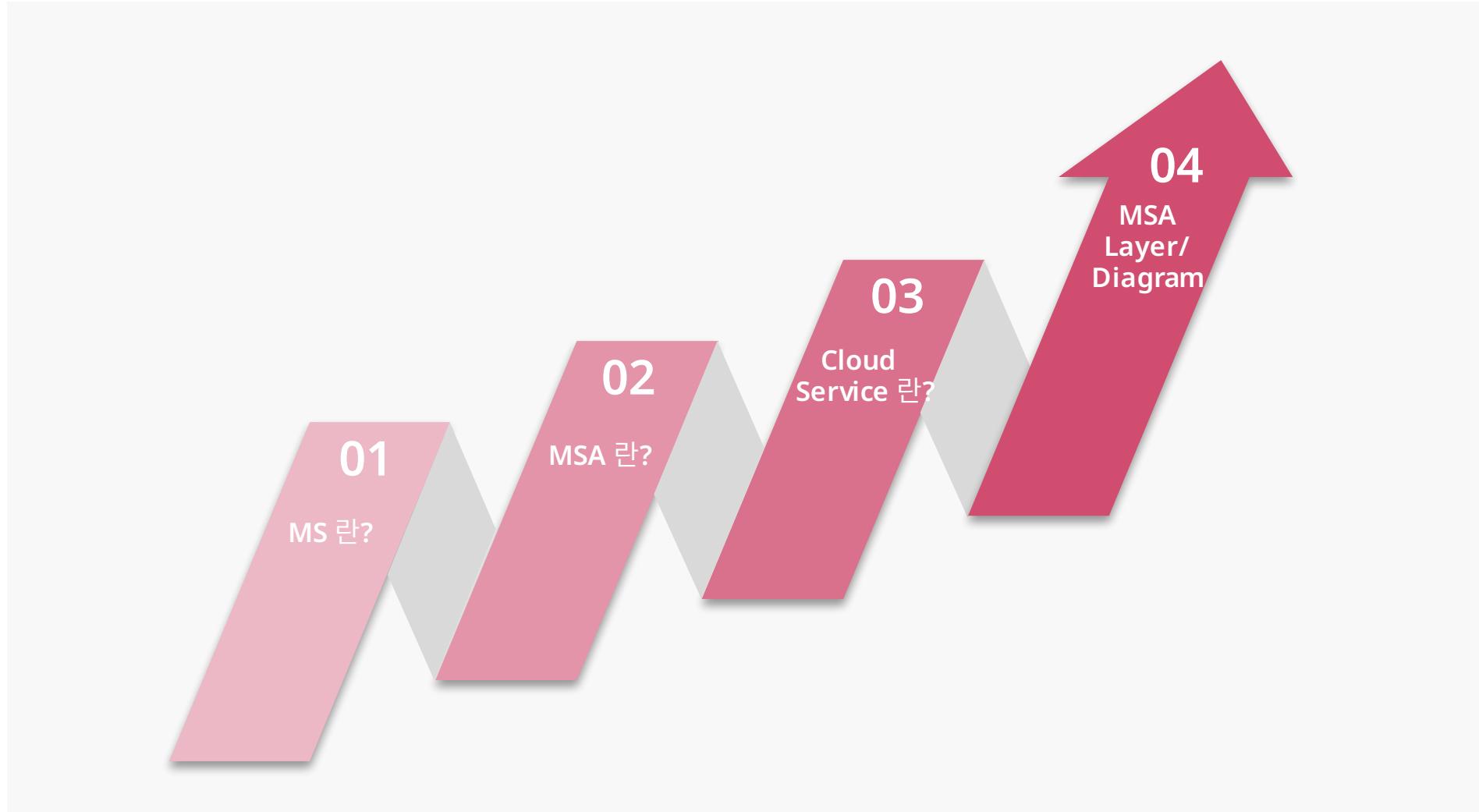


Table of Contents

01 | About Cloud Service

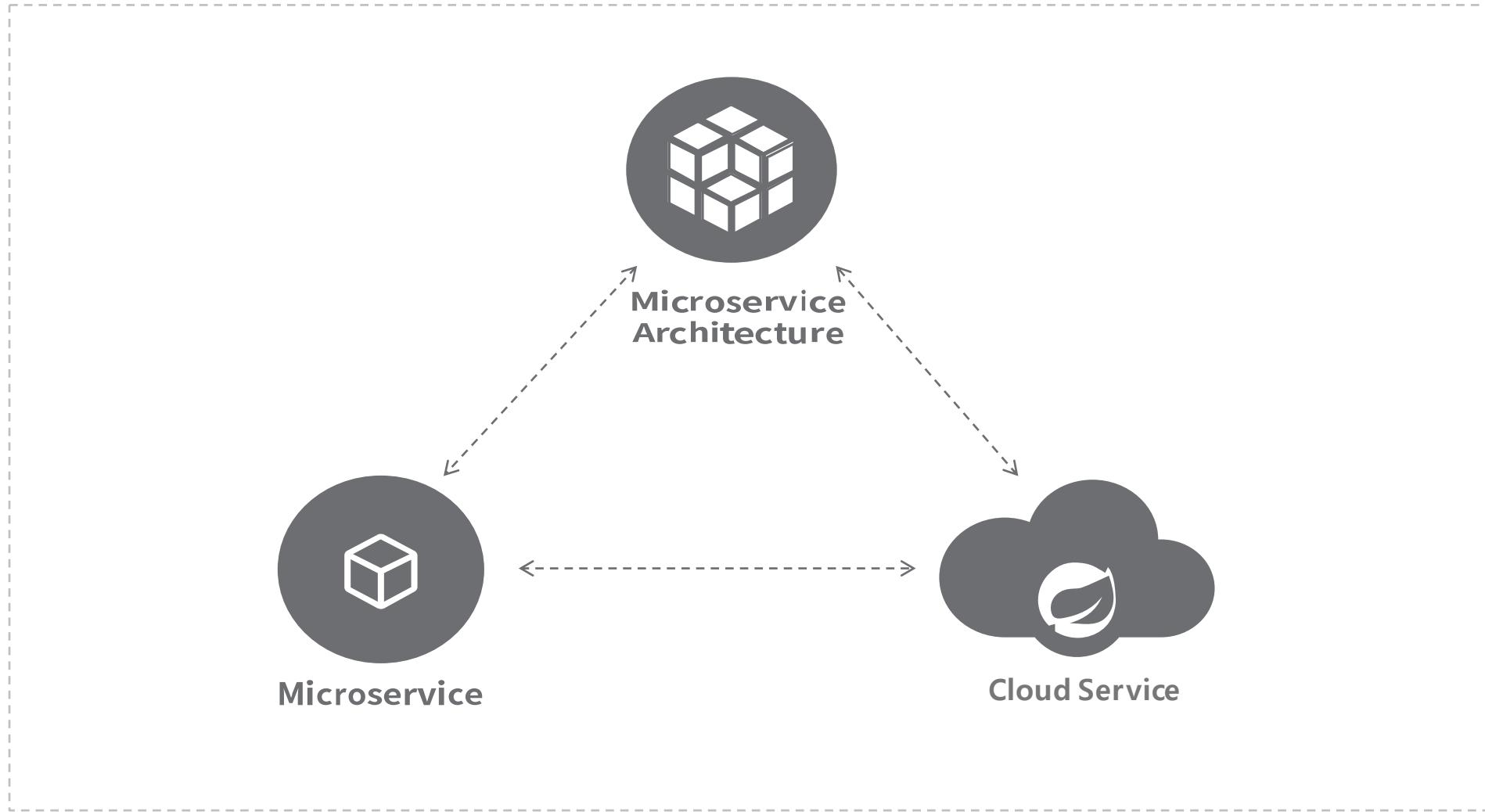
02 | Cloud Service Detail

Part 01. About Cloud Service



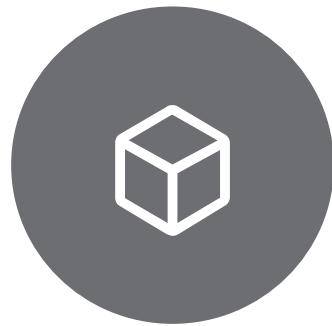
Overview

Cloud Service를 알기 위해 MicroService와 MicroService Architecture의 개념과 관계를 간략히 알아봅니다.



MicroService 란 ?

MicroService에 대해 알아봅니다.



- **MicroService 란 ?**
 - 단독으로 실행 가능하고
 - 가벼운 통신 프로토콜을 사용하며
 - 독립적으로 배치될 수 있는
 - 작은 단위로 기능이 분해된 서비스

MicroService 란?

MicroService에 대해 알아봅니다.



Microservice

단독으로 실행 가능하고	Stand Alone 구축 환경
가벼운 통신 프로토콜을 사용하며	API FIRST REST API
독립적으로 배치될 수 있는	Loosely Coupled Scalability, Stateless
작은 단위로 기능이 분해된 서비스	MS 분리 워크샵 Bounded Context

MicroService Architecture 란?

MicroService Architecture에 대해 알아봅니다.



Microservice
Architecture

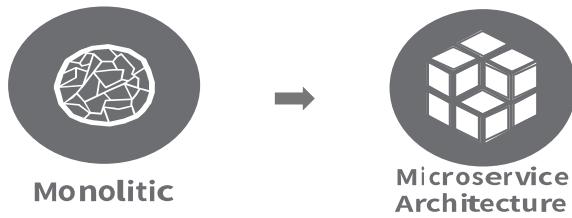
1. MicroService Architecture 란?

- MicroService로 이루어진 시스템 구성
 - Biz. Logic이 구현된 Application
 - 시스템 관리/편의 용도의 Application
 - Backing Service

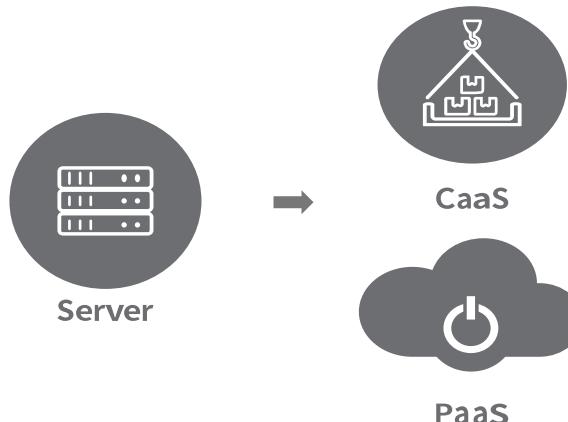
MicroService Architecture 란 ?

MicroService Architecture에 대해 알아봅니다.

2. 왜 MSA를 고려해야 할까 ?



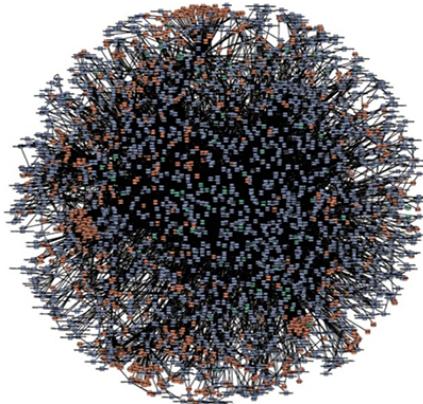
Monolithic의 단점 -> MSA 이점



PaaS, CaaS 등 시스템 구축 환경 변화

MicroService Architecture 란 ?

MicroService Architecture를 적용시 달라지는 부분은 ?



amazon.com®



NETFLIX

3. MSA runtime 고려사항

- Tens to thousands of MicroService Instances
- Inner Communication



[Backup]

MicroService Architecture의 새로운 요구사항과 달성 방법

요구사항	달성 방법
Configuration Management	 
Service Discovery	 
Load Balancing	 
API Gateway	 
Service Security	 
Centralized Logging	 
Centralized Metrics	 
Distributed Tracing	 
Resilience & Fault Tolerance	
Auto Scaling & Self Healing	
Packaging/Deployment	 
Job Management	 

Cloud Service 란?

Cloud Service에 대해 알아봅니다.

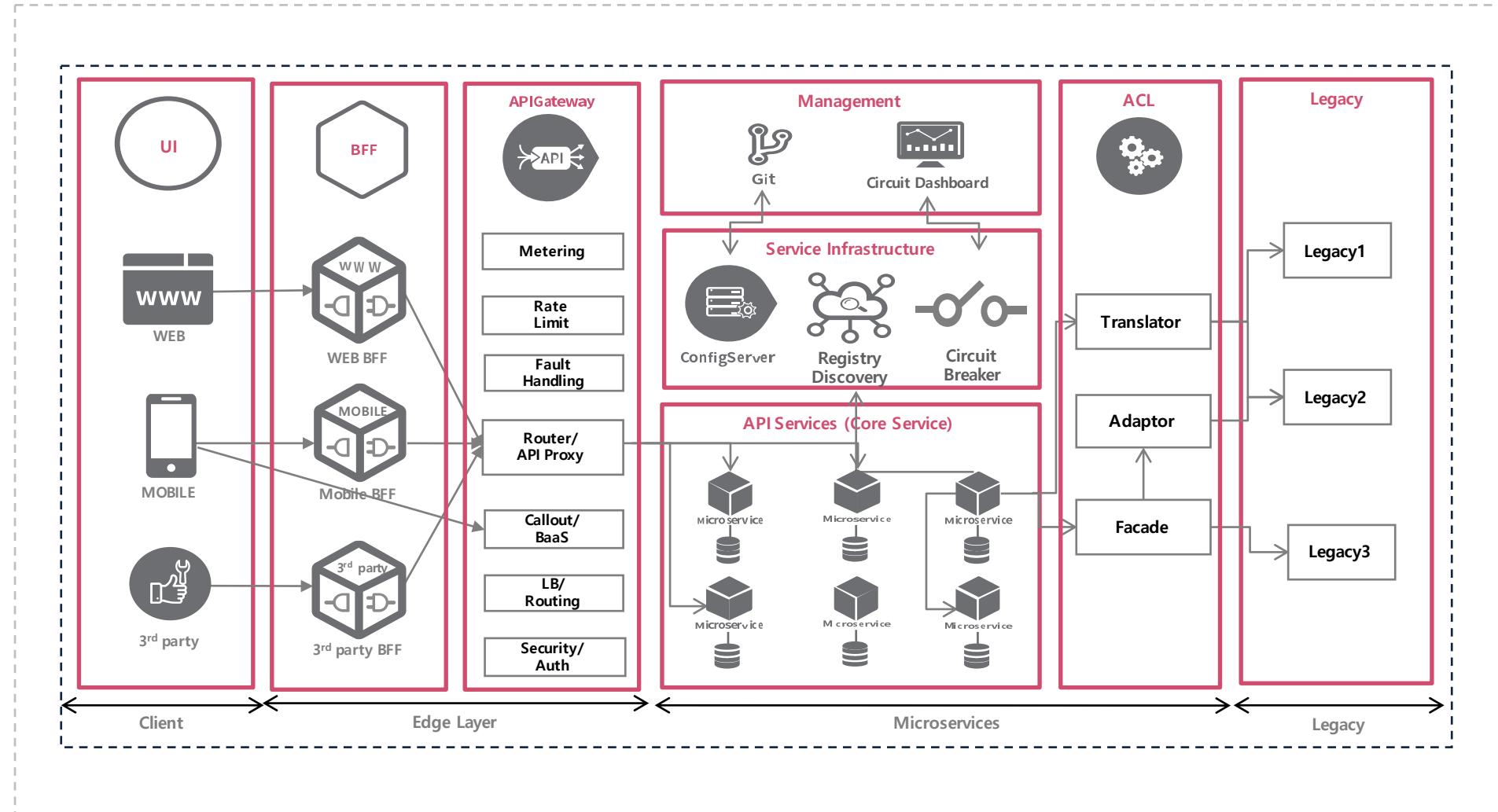


- **Cloud Service 란 ?**
 - MSA로 구현된 시스템의 관리/편의/연계 용도의 기능
 - 다양한 패턴 및 구현
 - BFF
 - API G/W
 - Config Server
 - Circuit Breaker
 - Registry / Discovery
 - ACL(Anti-Corruption Layer)

PART 01. About Cloud Service

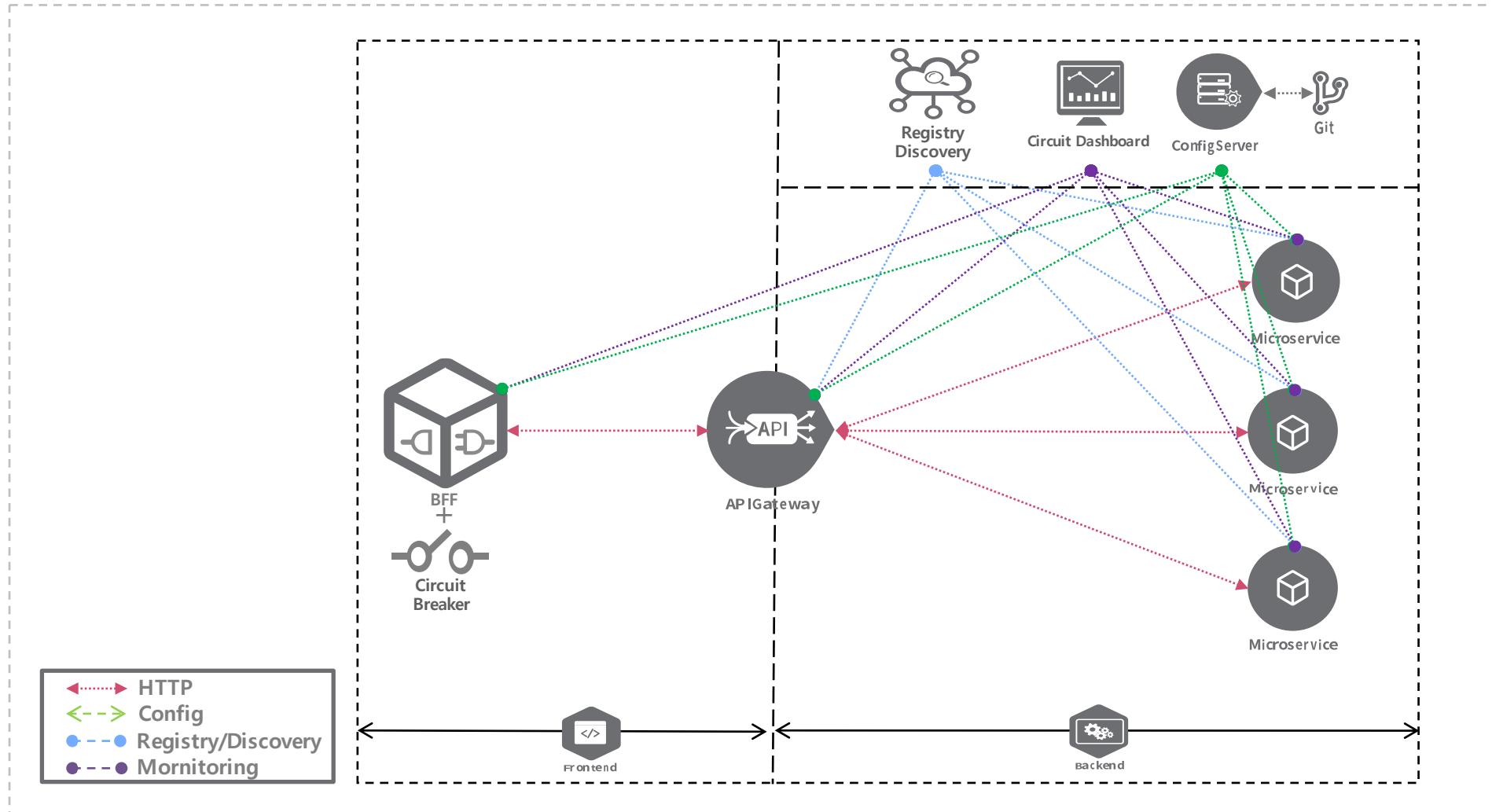
MSA WEB SYSTEM Layer

MicroService Architecture가 적용된 웹 시스템의 기본 Layer를 알아봅니다.

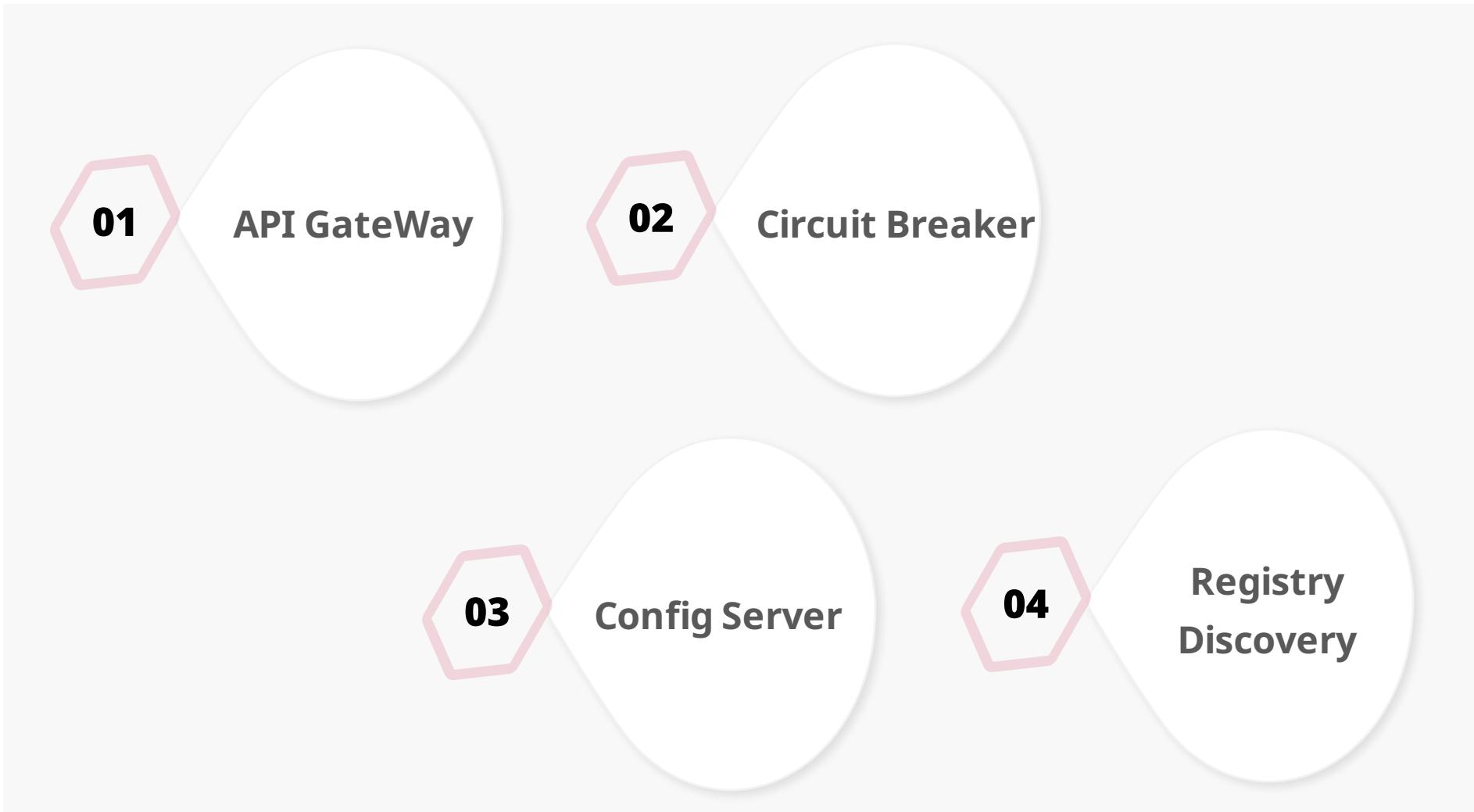


MSA WEB SYSTEM Diagram

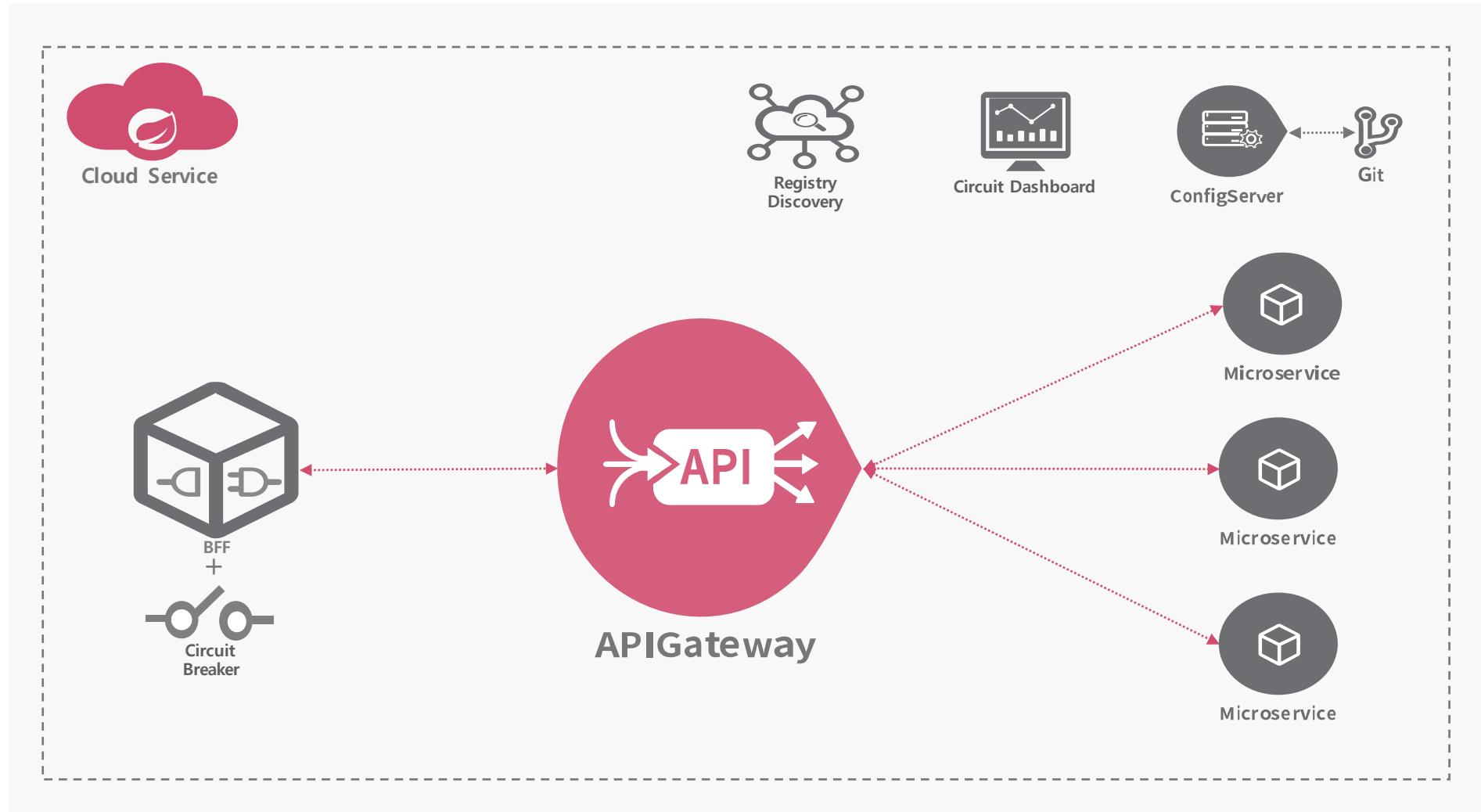
MicroService Architecture가 적용된 웹 시스템의 기본 Diagram을 알아봅니다.



PART 02. Cloud Service Detail

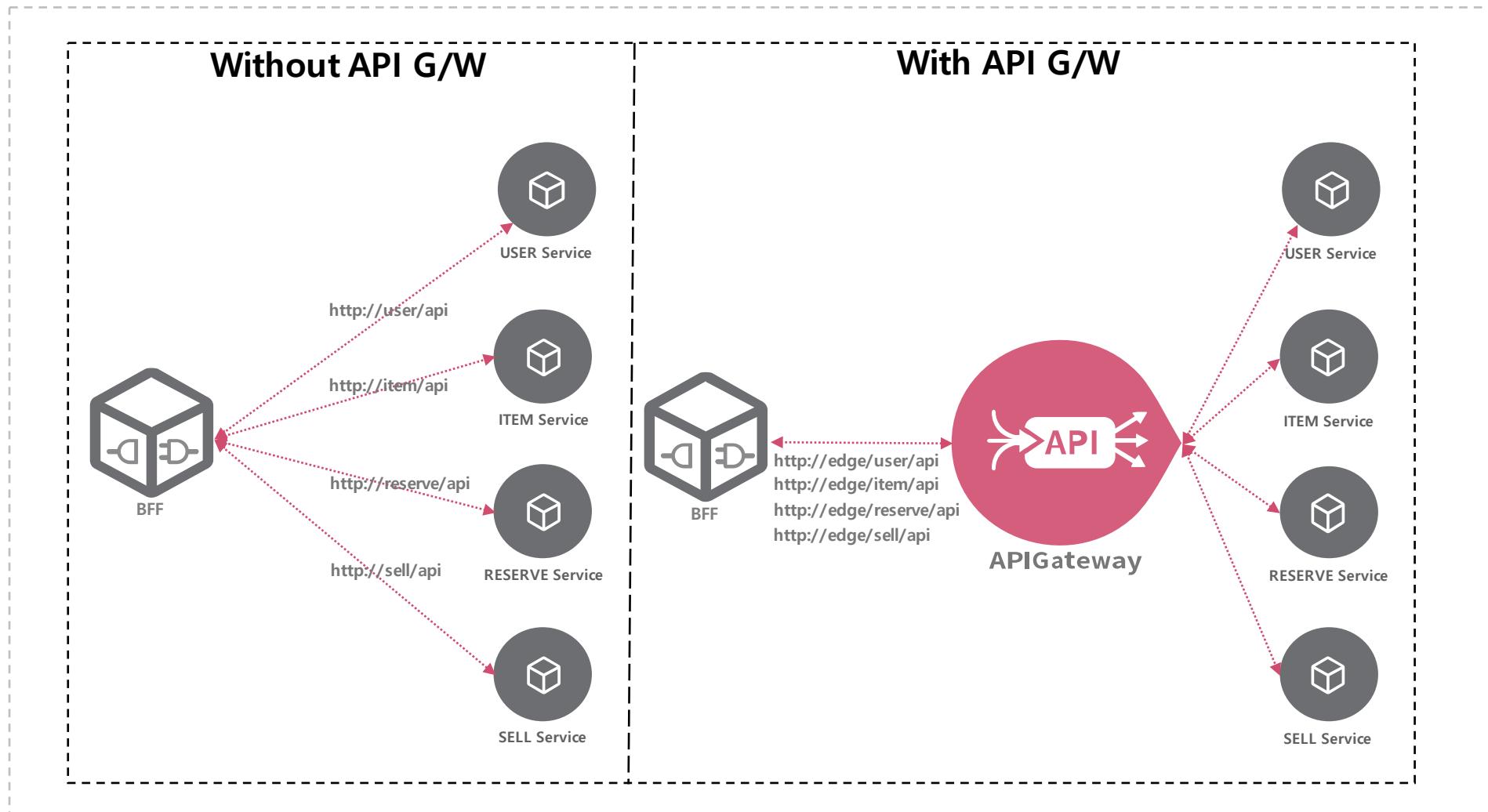


Cloud Service Diagram – API Gateway



API Gateway

Cloud Service 中 API Gateway에 대해 알아봅니다.



API Gateway

Cloud Service 中 API Gateway에 대해 알아봅니다.

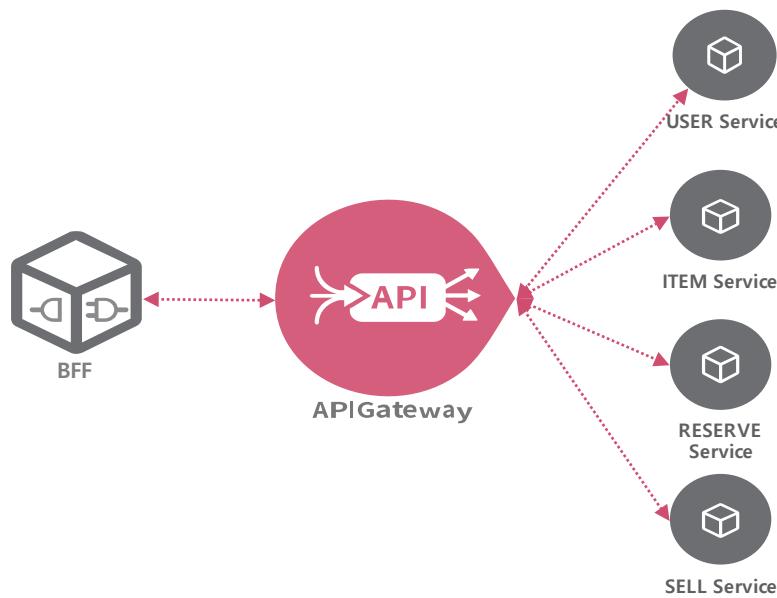
• API G/W

- Edge-Service
- Proxy, Routing, LoadBalancing
- SYSTEM 내 API호출 통제
 - 빌링
 - 미터링
 - 사용 제한
 - 모니터링
 - 트랜잭션
 - 요청/응답 횟수 Count
- 공통기능
 - 보안
 - 로깅
 - 캐싱

⚠ WARNING

API G/W는 서비스 요청을 수용하는 항구 역할을 하기에 과도한 기능의 추가는 성능상의 병목지점이 될 수 있습니다.

API G/W를 구현할 때는 MicroService Architecture(MSA)에 대한 깊은 이해(성능, 기능, 확장성 등)가 필요합니다.



Zuul

Spring Cloud Netflix OSS – Zuul에 대해 알아봅니다.



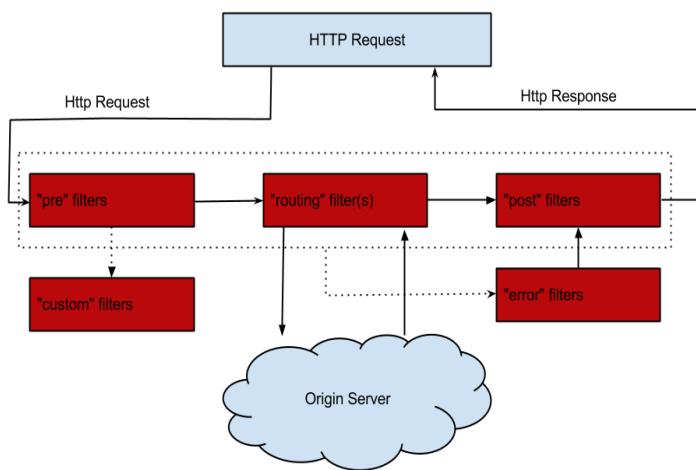
- **Zuul 이란 ?**
 - Spring Cloud Netflix OSS 中 API G/W 구현에 관련된 기능을 제공 ([Docs Link](#))
- **Zuul 기능**
 - 인증 및 보안
 - 통계 및 모니터링
 - 동적 라우팅
 - 스트레스 테스팅
 - 부하차단
 - 정적 응답 처리
 - 다중 영역 복원력

Zuul

Zuul은 내부적으로 Zuul Filters를 통해 다양한 기능을 제공합니다.

• Zuul Filters

- Pre Filters
 - 인증요청
 - 라우팅 할 Origin Server 선택
 - Debug 정보 로깅
- Routing Filters
 - Apache HttpClient or Netflix Ribbon을 사용, HTTP 요청을 만들어 Origin Server로 전송
- Post Filters
 - Origin Server -> Client로 응답 return
 - 응답에 기본 HTTP 헤더를 추가
 - 통계 정보 획득
- Error Filters
 - 각 단계 수행 중 오류 발생 시 동작
- Custom Filters
 - 사용자 정의 Filter 작성 가능



Zuul

Zuul은 다른 Spring Cloud Netflix OSS와 연계할 때 더욱 풍부한 기능을 제공합니다.

- Zuul 연계



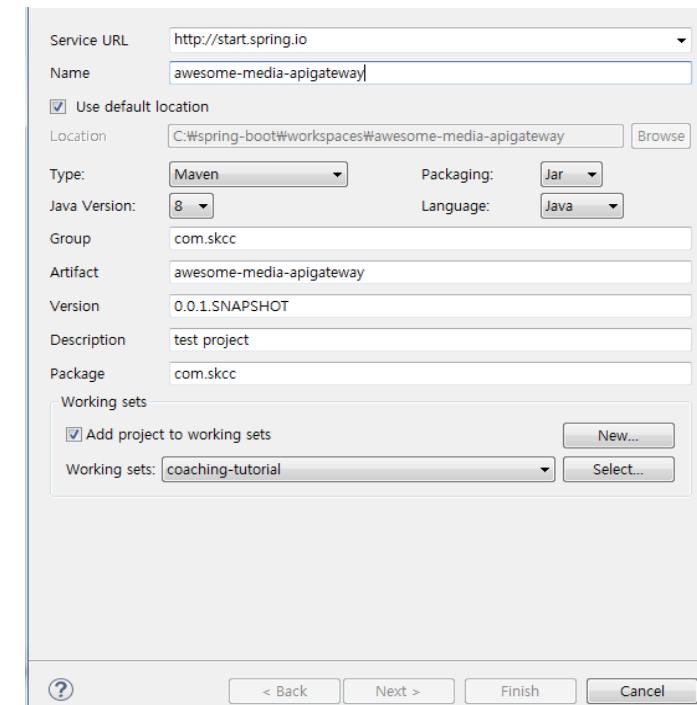
- Hystrix
 - 내부적으로 routing 시 장애가 발생시, request를 HystrixCommand로 Wrapping
- Ribbon
 - 부하 분산을 위한 로드 밸런싱 뿐만아니라, 네트워크 성능과 오류에 대한 자세한 정보를 제공
- Turbine
 - 실시간으로 Application Cluster의 metrics 집계
- Eureka
 - 라우팅 설정시 서비스명과 매핑된 url pattern만 적용해 간편하게 세팅

Zuul 실습

Spring Boot + Zuul을 사용해 AWESOME-MEDIA에 API GATEWAY Routing 기능을 구현해 봅시다.

1. Project를 신규로 생성합니다.

- Project Spec.
 - Project 명 : awesome-media-apigateway
 - Spring Boot Version: 1.5.10
 - Build Type : Maven
 - Packaging : Jar
 - Java Version : 8

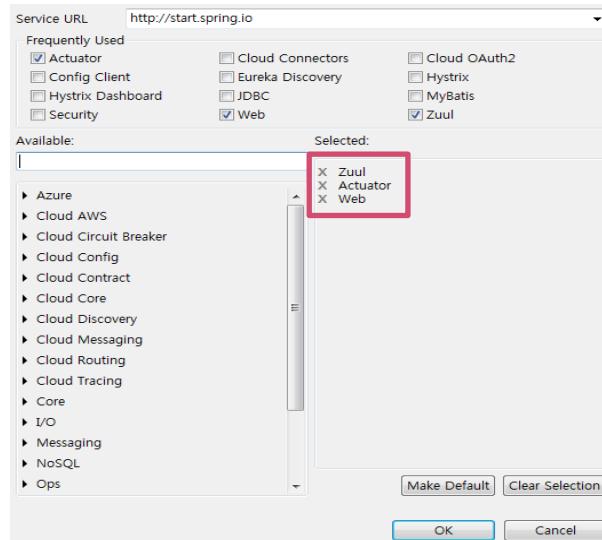


Zuul 실습

Spring Boot + Zuul을 사용해 AWESOME-MEDIA에 API GATEWAY Routing 기능을 구현해 봅시다.

2. spring-cloud-starter-zuul Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-boot-starter-zuul</artifactId>
</dependency>
```

3. application.yml에 설정을 추가합니다.

```
zuul:
  ignored-services: "*"
  routes:
    awesome-media-backend:
      path: /awesome-media/**
      url: http://localhost:8090
```

```
server:
  port: 9999
```

```
management:
  security:
    enabled: false
```

```
Spring:
  application:
    name: "awesome-media-apigateway"
```

Zuul 실습

Spring Boot + Zuul을 사용해 AWESOME-MEDIA에 API GATEWAY Routing 기능을 구현해 봅시다.

4. @EnableZuulProxy 어노테이션을 추가합니다.

```
@SpringBootApplication  
@EnableZuulProxy  
public class AwesomeMediaApigatewayApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(AwesomeMedia ApigatewayApplication.class, args);  
    }  
}
```

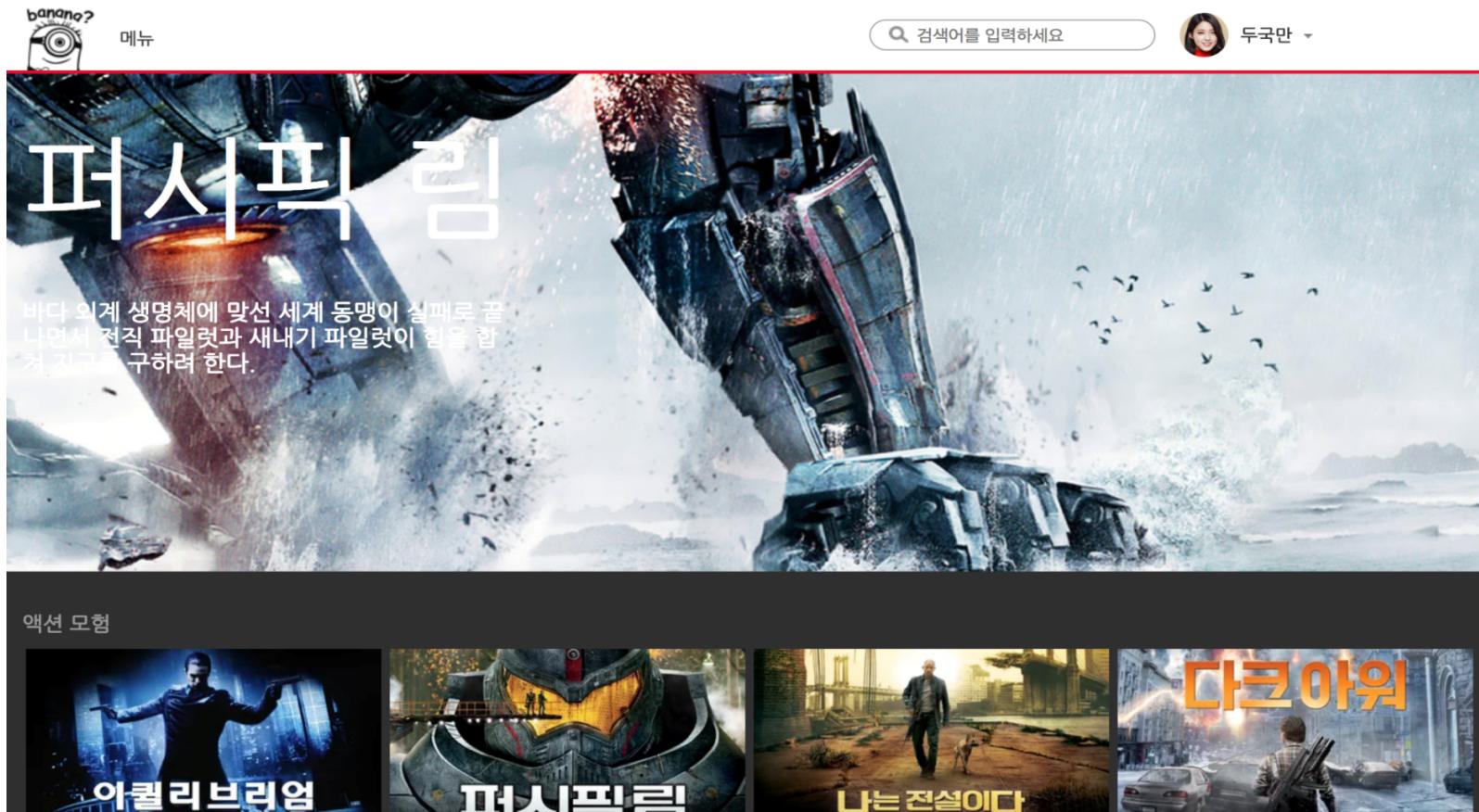
5. AWESOME-MEDIA-FRONTEND의 호출 url을 변경합니다.

```
api:  
  bff:  
    path: /  
    services:  
      url: http://localhost:9999/awesome-media  
    # url: http://localhost:8090
```

Zuul 실습

awesome-media-frontend를 구동해 awesome-media-apigateway에서 정상적으로 라우팅 된 것을 확인합니다.

<http://localhost:8080>



[Backup]Zuul Routing UseCases

Zuul을 사용하는 경우에 따라 다양한 설정을 적용할 수 있습니다.

i Zuul Routing Use Cases

Zuul route는 Use Cases 에 따라 다양한 설정이 가능합니다.

서비스ID와 url path를 다르게 지정할 경우

`zuul.routes.<serviceID>/<url path>/**`

```
zuul:  
routes:  
user: /myusers/** -- service ID: url path
```

route name과 서비스ID를 다르게 지정할 경우

```
zuul:  
routes:  
user:           -- route name  
path: /myusers/**      -- url path  
serviceId: users_service -- service ID
```

특정 서비스만 라우팅 할 경우

`zuul.ignored-services` 에 제외할 서비스 ID 패턴 목록을 설정

```
zuul:  
ignoredServices:"*"  
routes:  
user: /myusers/**
```

서비스가 아니라 특정 url을 사용할 경우

eureka를 사용하지 않는 route일 때, 물리 주소 명시

`zuul.routes.<serviceID>.path=<url path>`

`zuul.routes.<serviceID>.url=<physical location>`

`zuul:`

`routes:`

<code>user:</code>	<code>-- route name</code>
<code>path: /myusers/**</code>	<code>-- url path</code>
<code>url: http://example.com/users_service</code>	<code>-- physical location</code>

특정 url 패턴을 라우팅하지 않을 경우

`zuul.ignored-patterns` 에 제외한 url path 패턴을 설정

```
zuul:  
ignoredPatterns: /**/api/**      -- "**/api/**"가 포함된 요청은 라우팅 X  
routes:  
user: /myusers/**           -- url path
```

Zuul 서버 내 구현된 endpoint로 포워딩 하는 경우

`zuul.routes.<serviceID>.url=forward:</endpoint>` 설정

```
zuul:  
routes:  
user:           -- route name  
path: /myusers/**      -- url path  
url: forward:/users   -- "/users/**"를 zuul 서버 내 /users로 포워딩
```

Zuul 실습

Spring Boot + Zuul을 사용해 AWESOME-MEDIA 시스템에 API GATEWAY LoadBalancing 기능을 구현해 봅시다.



- **Ribbon이란 ?**

- Spring Cloud Netflix OSS 中 Client Side LoadBalancing 기능을 제공
- Zuul은 내부에 Ribbon을 포함하고 있어, Zuul Dependency를 추가한 경우 별도 설정없이 Ribbon 사용 가능

Zuul 실습

Spring Boot + Zuul을 사용해 AWESOME-MEDIA 시스템에 API GATEWAY LoadBalancing 기능을 구현해 봅시다.

6. AWESOME-MEDIA-APIGATEWAY 에 Ribbon 설정 추가

<route name>.ribbon.listOfServers에 서비스 url 목록을 정의
eureka가 적용되어 있는 경우, ribbon.eureka.enable=false 설정

```
zuul:  
  ignored-services: "*"  
  routes:  
    awesome-media-backend:  
      path: /awesome-media/**  
      url: http://localhost:8090  
      service-id: awesome-media-backend  
    ribbon-test:  
      path: /ribbon-test/**
```

```
ribbon-test:  
  ribbon:  
    listOfServers: google.com, naver.com
```

```
#ribbon without eureka
```

```
ribbon:  
  eureka:  
    enabled: false
```

Zuul 실습

awesome-media-apigateway의 지정된 endpoint를 호출해 LoadBalancing 기능을 수행합니다.

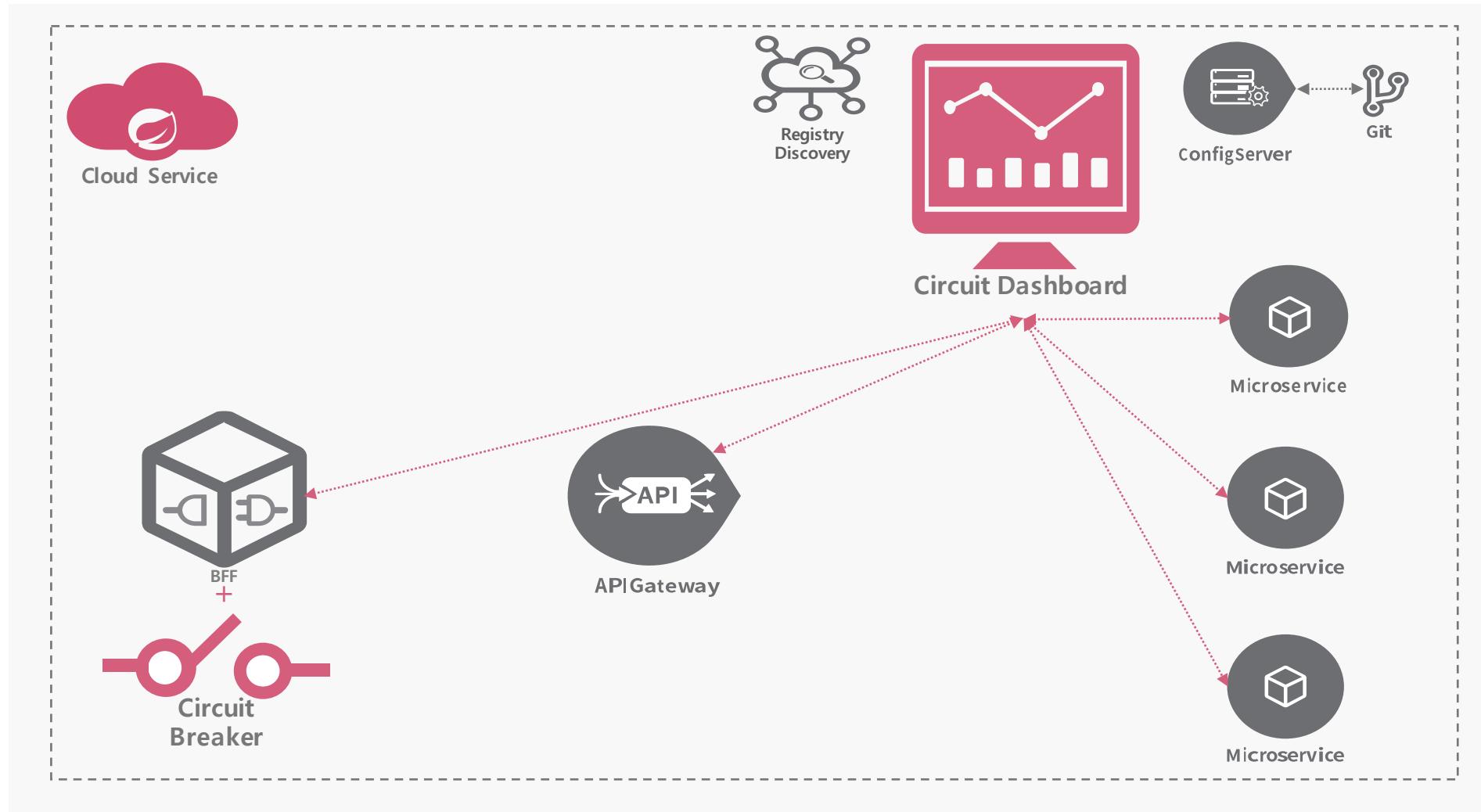
<http://localhost:9999/ribbon-test>



메일 카페 블로그 지식iN 쇼핑 Pay ▶TV 사진 뉴스 증권 부동산 지도 영화 뮤직 책 웹툰 더보기、

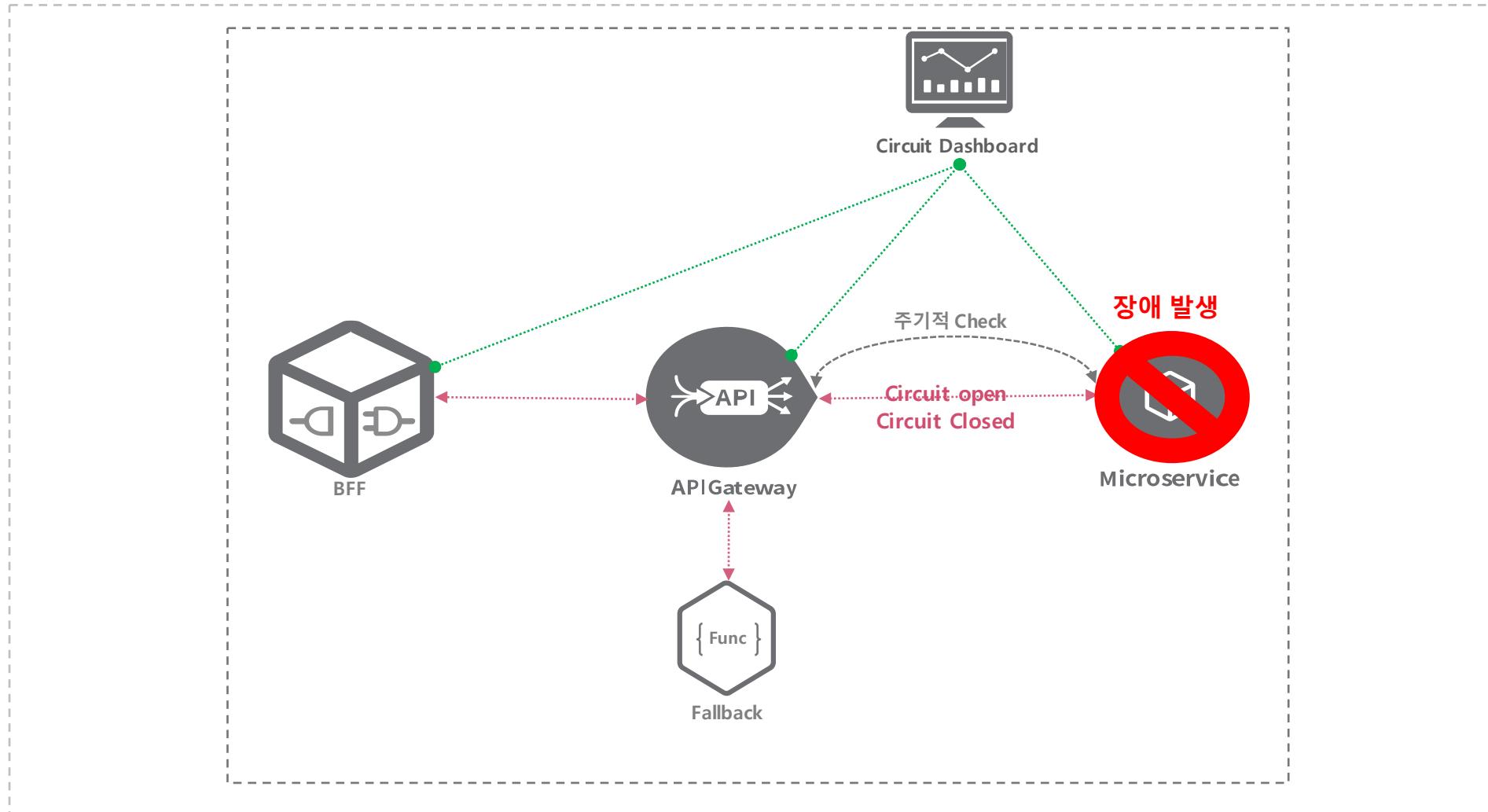


Cloud Service Diagram – Circuit Breaker



Circuit Breaker

Cloud Service 中 Circuit Breaker에 대해 알아봅니다.

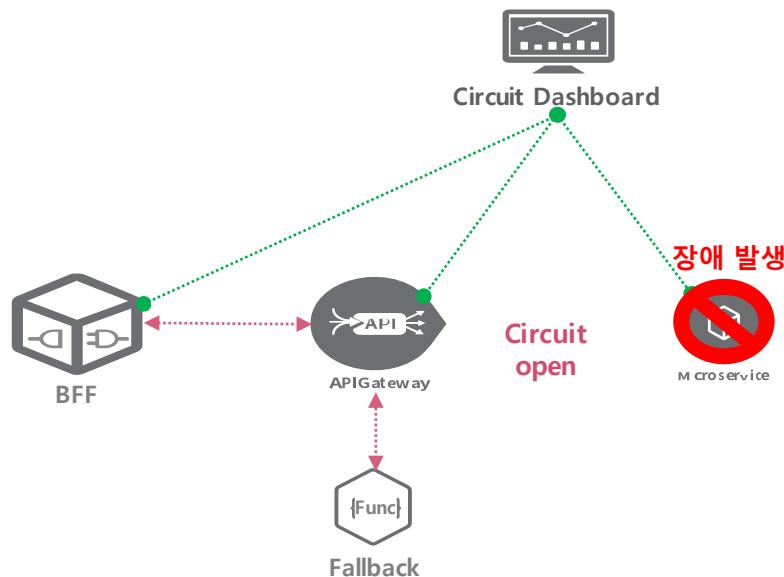


Circuit Breaker

Cloud Service 中 Circuit Breaker에 대해 알아봅니다.

• Circuit Breaker 란 ?

- 하나의 MicroService 장애가 전체 시스템의 장애로 번지지 않도록 하는 MSA 디자인 패턴
1. 임계값(timeout, 에러율 등)을 설정
 2. 임계값을 초과하면 Circuit을 Open해 요청을 차단하고 Fallback 패턴을 수행
 3. Health Check를 통해 자체적으로 장애 지속 여부를 파악
 4. 장애가 해결되면 Circuit이 다시 Close되어 정상적으로 동작



Hystrix

Spring Cloud Netflix OSS – Hystrix에 대해 알아봅니다.

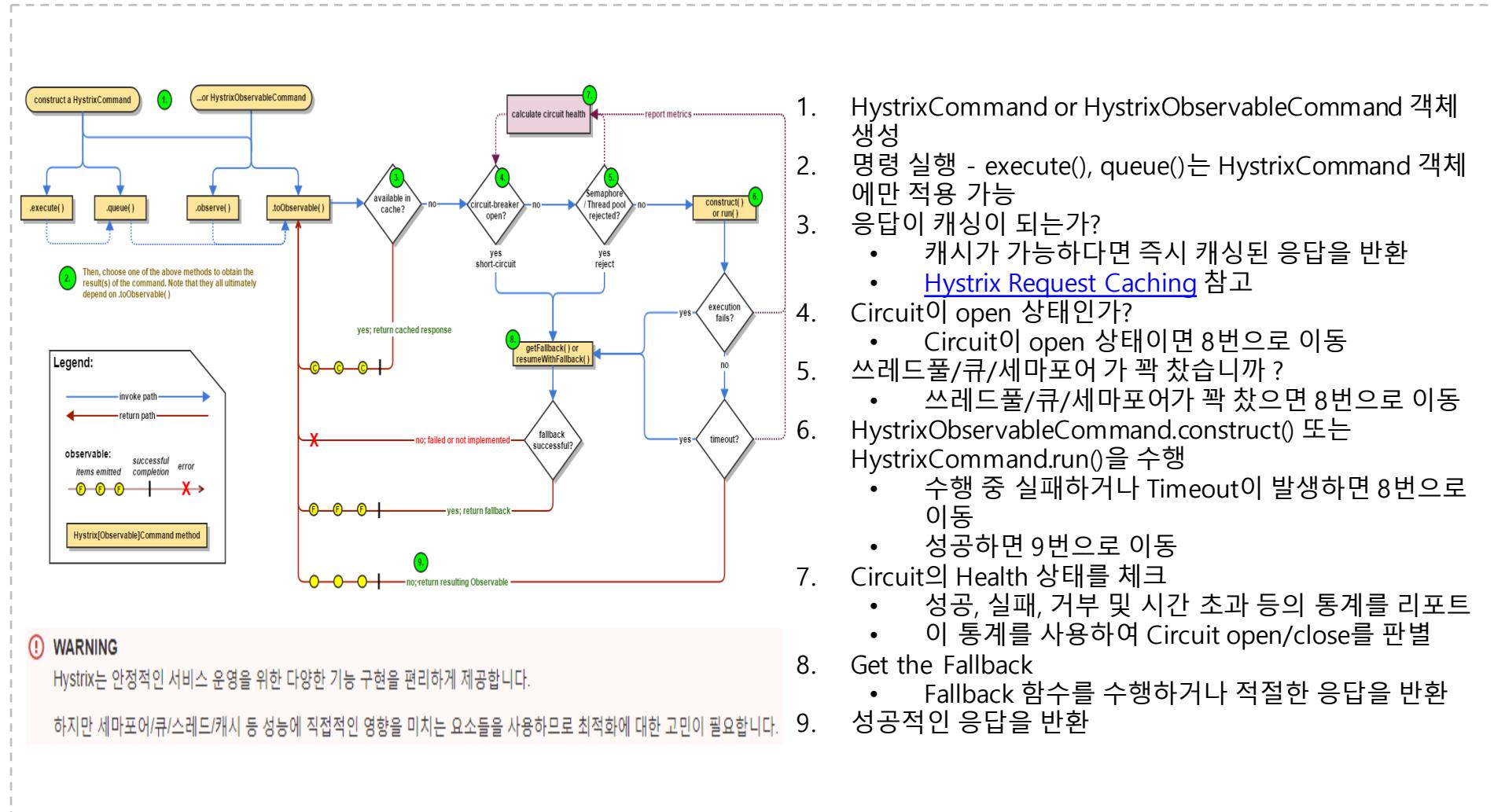


• Hystrix 란 ?

- Spring Cloud Netflix OSS 中 Circuit Breaker 구현에 관련된 기능을 제공 ([Docs Link](#))
- 기본적으로 @HystrixCommand 가 명시된 메소드 단위로 별개의 Circuit Breaker를 갖게 됩니다.
- MicroService Architecture(MSA) 의 분산환경에서 각 서비스를 호출하는 구간을 격리하고 모니터링 할 수 있습니다.
- CommandKey를 명시하여 하나의 Circuit 으로 Grouping 할 수 있고 Group내 메소드 별 metrics 을 함께 집계할 수 있습니다.

Hystrix

Spring Cloud Netflix OSS – Hystrix 의 내부 동작에 대해 알아봅니다.



Hystrix

Spring Cloud Netflix OSS – Hystrix 의 다양한 Fallback 패턴에 대해 알아봅니다.

- **Hystrix Fallback 패턴**

패턴명	설명	시나리오
Fail Fast	Fallback을 구현하지 않고 실패할 경우 에러를 발생	1.대상 : 로그인 2.시나리오 : skb-account-service 다운 3.결과 : 오류 출력
Fail Silent	실패할 경우에러가 나지 않도록 Fallback에서 empty response를 반환	1.대상 : skb-bff-service에서 API Connect를 통해 백엔드 서비스 호출 2.시나리오 : 백엔드 서비스 다운 3.결과 : 화면에 에러 메세지가 출력되지 않고 빈화면이 출력됨
Fail Cache via Network	실패할 경우 오래된 데이터를 Memcached나 Redis와 같은 캐시 서비스에서 가져옴	1.대상 : 메인 화면 상단의 프로모션 조회 2.시나리오 : skb-recommend-service 다운 3.결과 : 백업용 Redis에서 프로모션 결과 조회 (퍼시픽림 -> 셜록)
Client Doesn't Perform Network Access	latency 문제나 감당할 수 없는 스레딩 오버헤드가 발생하는 경우 HystrixCommand에서 세마포어를 사용	1.대상 : 메인 화면의 카테고리 별 모든 영상 정보 조회 2.시나리오 : 가장 빈번한 요청이 있는 서비스 3.결과 : 잦은 호출에도 정상 서비스됨

Hystrix Dashboard

Spring Cloud Netflix OSS – Hystrix Dashboard에 대해 알아봅니다.



Hystrix Dashboard

<http://hostname:port/turbine/turbine.stream>

Cluster via Turbine (default cluster): <http://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [http://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](http://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <http://hystrix-app:port/hystrix.stream>

Delay: ms Title:

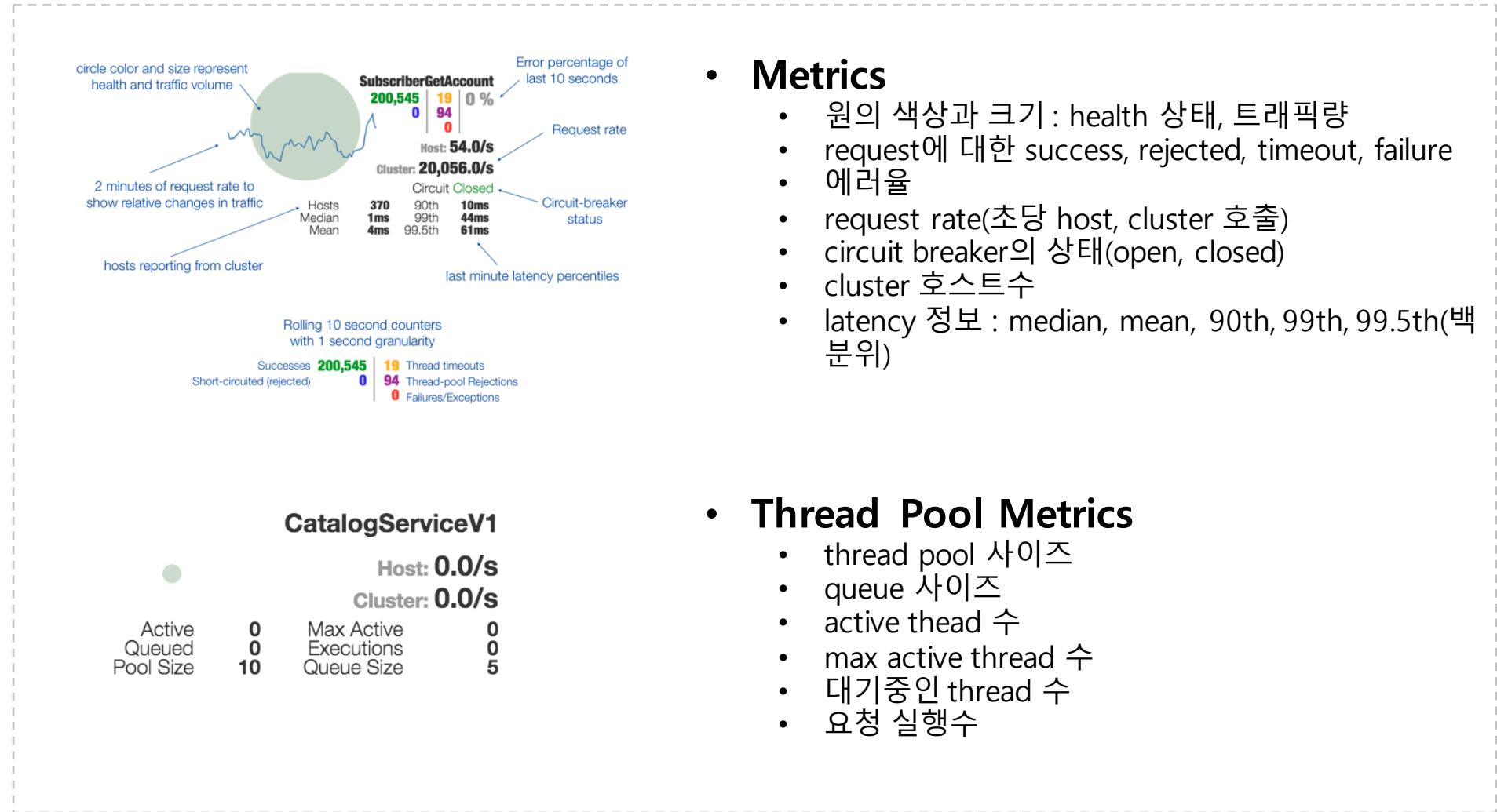
[Monitor Stream](#)

• Hystrix Dashboard 란 ?

- 시스템 내 MicroService의 상태를 확인할 수 있는 모니터링 대시보드 기능을 지원
- 하나의 앱을 모니터링하는 경우 : <http://{ip}:{port}/hystrix.stream>
- 여러 개의 앱을 모니터링하는 경우
 - 디폴트 cluster를 사용 : <http://{ip}:{port}/turbine.stream>
 - 사용자가 지정한 cluster 사용 : [http://{ip}:{port}/turbine.stream?cluster=\[clusterName\]](http://{ip}:{port}/turbine.stream?cluster=[clusterName])

Hystrix Dashboard

Spring Cloud Netflix OSS – Hystrix Dashboard에 대해 알아봅니다.



• Metrics

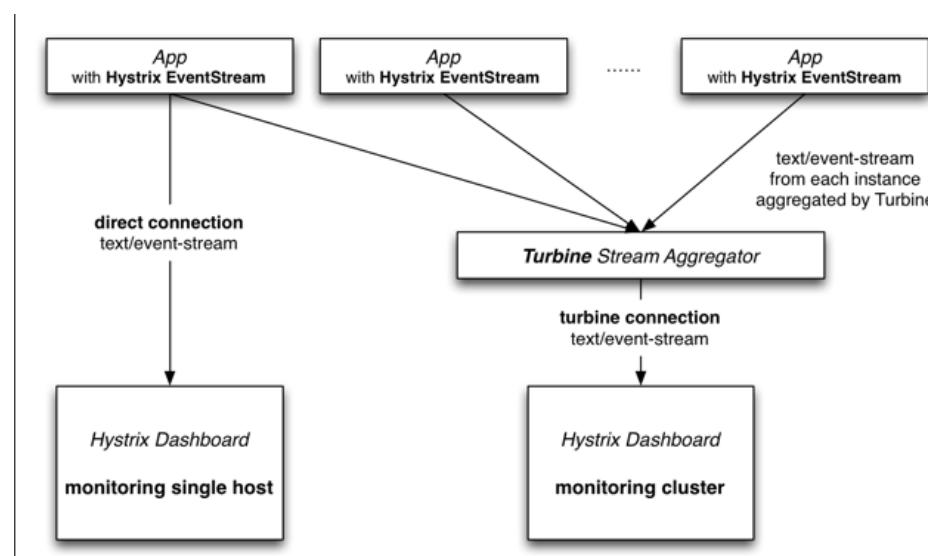
- 원의 색상과 크기 : health 상태, 트래픽량
- request에 대한 success, rejected, timeout, failure
- 에러율
- request rate(초당 host, cluster 호출)
- circuit breaker의 상태(open, closed)
- cluster 호스트수
- latency 정보 : median, mean, 90th, 99th, 99.5th(백분위)

• Thread Pool Metrics

- thread pool 사이즈
- queue 사이즈
- active thread 수
- max active thread 수
- 대기중인 thread 수
- 요청 실행수

Turbine

Spring Cloud Netflix OSS – Turbine 에 대해 알아봅니다.



- **Turbine 이란 ?**

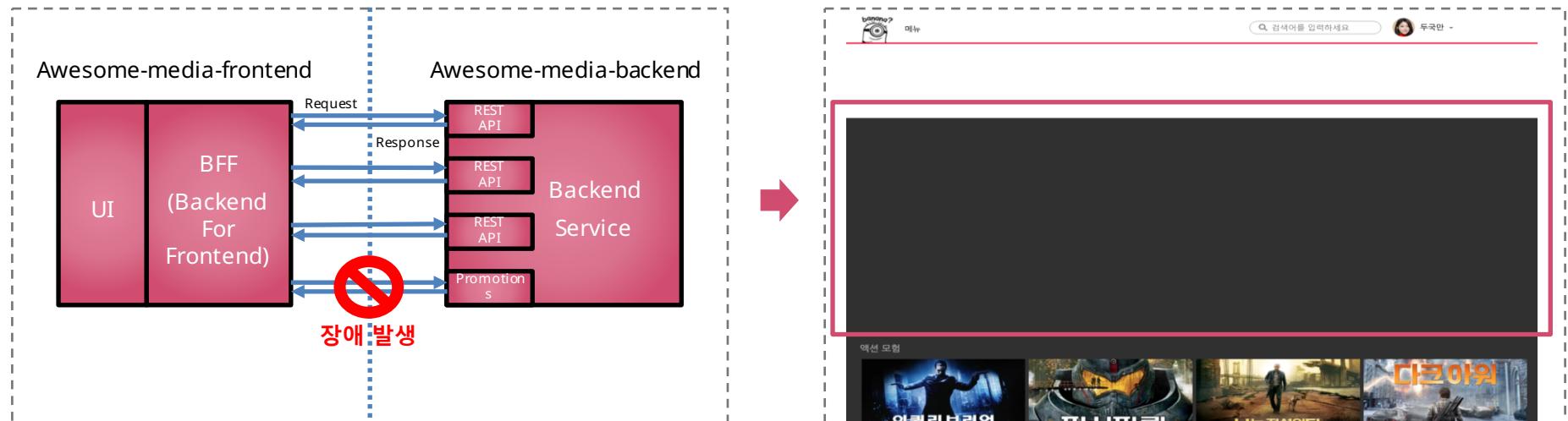
- Spring Cloud Netflix OSS 中 개별 Hystrix metrics를 모아 하나의 Dashboard 상에서 모니터링 기능을 제공([Docs Link](#))
- Cluster 단위로 모니터링 가능
 - eureka 사용시 turbine과 eureka 서버에 연동된 application들을 기본 cluster로 모니터링 가능

Hystrix 실습

Spring Boot + Hystrix를 사용해 AWESOME-MEDIA에 Circuit Breaker 패턴을 구현해 봅시다.

- 실습 시나리오

- AWESOME-MEDIA-FRONT에서 정상적으로 PROMOTION 기능을 호출 못하는 경우, Fallback을 수행



- FallBack 패턴

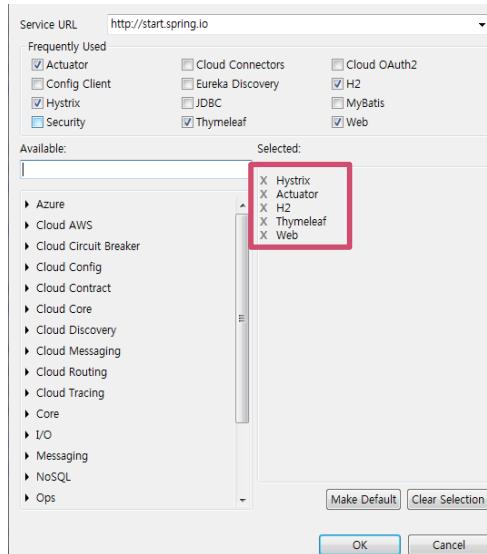
적용 패턴	설명
Fail Silent	실패할 경우 에러가 나지 않도록 Fallback에서 empty response를 반환
Stubbed	실패할 경우 에러가 나지 않도록 Fallback에서 Default response를 반환
Via Network	실패할 경우 에러가 나지 않도록 다른 네트워크를 수행해 response를 반환

Hystrix 실습

Spring Boot + Hystrix를 사용해 AWESOME-MEDIA에 Circuit Breaker 패턴을 구현해 봅시다.

1. spring-cloud-starter-hystrix Dependency를 추가합니다.
2. application.yml에 설정을 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

```
hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMilliseconds: 60000
```

✓ <https://github.com/Netflix/Hystrix/wiki/configuration> 참고

Hystrix 실습

Spring Boot + Hystrix를 사용해 AWESOME-MEDIA에 Circuit Breaker 패턴을 구현해 봅시다.

3. @EnableHystrix 어노테이션을 추가합니다.

```
@SpringBootApplication  
@EnableHystrix  
public class AweSomeMediaFrontendApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(AweSomeMediaFrontendApplication.class, args);  
    }  
}
```

Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}") 을 추가하고 fallbackMethod를 구현합니다.

4-1. Fallback – Stubbed

```
@HystrixCommand(fallbackMethod="getPromotionFallBackStubbed")
public Content getPromotion() {
    return restTemplate.getForObject(String.format("%s/v1/promotion", serviceUrl), Content.class);
}

public Content getPromotionFallBackStubbed() {
    log.info("exec getPromotionFallBackStubbed");

    Content content = new Content();
    content.setCategory("action");
    content.setGrade(5);
    content.setHasEpisodes(false);
    content.setId(70);
    content.setPoster("https://occ-0-1007-10091.nflxso.net/art/45d17/fcfe82644f60b74072a0214a28c1f6d88aa45d17.webp");
    content.setRate("12");
    content.setRegDate("2018-04-18");
    content.setRuntime(148);
    content.setStillcut("https://occ-0-1007-10091.nflxso.net/art/c59d1/cae8d57e1483ba8d427575903d26be411f9c59d1.webp");
    content.setSummary("정부의 개입이 불러온 어벤져스의 분열. 슈퍼히어로의 독립성을 위해 싸울 것인가. 아니면 정부의 입장에 따를 것인가.  
한 치의 양보도 없는 양면, 이 팽팽한 대립의 결말은?");

    content.setTitle("캡틴아메리카: 시빌 워");
    content.setVideo("dKrVegVI0Us");
    content.setView(13);
    content.setYear(2018);

    return content;
}
```

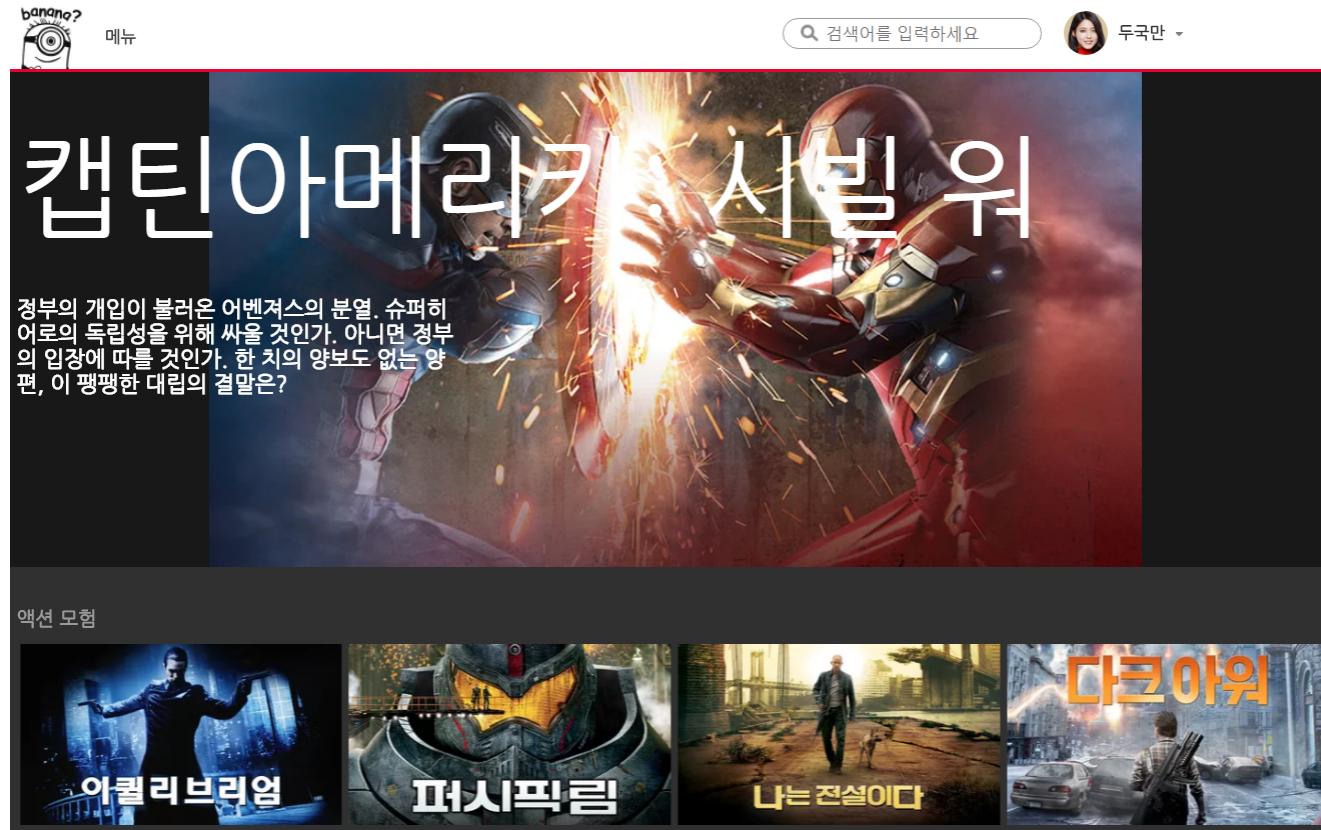
↑ 장애 발생 point "%s/v1/promotions"

Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}")을 추가하고 fallbackMethod를 구현합니다.

4-1. Fallback – Stubbed

<http://localhost:8080>



Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}") 을 추가하고 fallbackMethod를 구현합니다.

4-2. Fallback – ViaNetwork

```
@HystrixCommand(fallbackMethod="getPromotionFallBackViaNetwork")
public Content getPromotion() {
    return restTemplate.getForObject(String.format("%s/v1/promotion", serviceUrl), Content.class);
}

public Content getPromotionFallBackViaNetwork() {
    log.info("exec getPromotionFallBackViaNetwork");

    return restTemplate.getForObject(String.format("%s/v1/promotions", serviceUrl), Content.class);
}
```

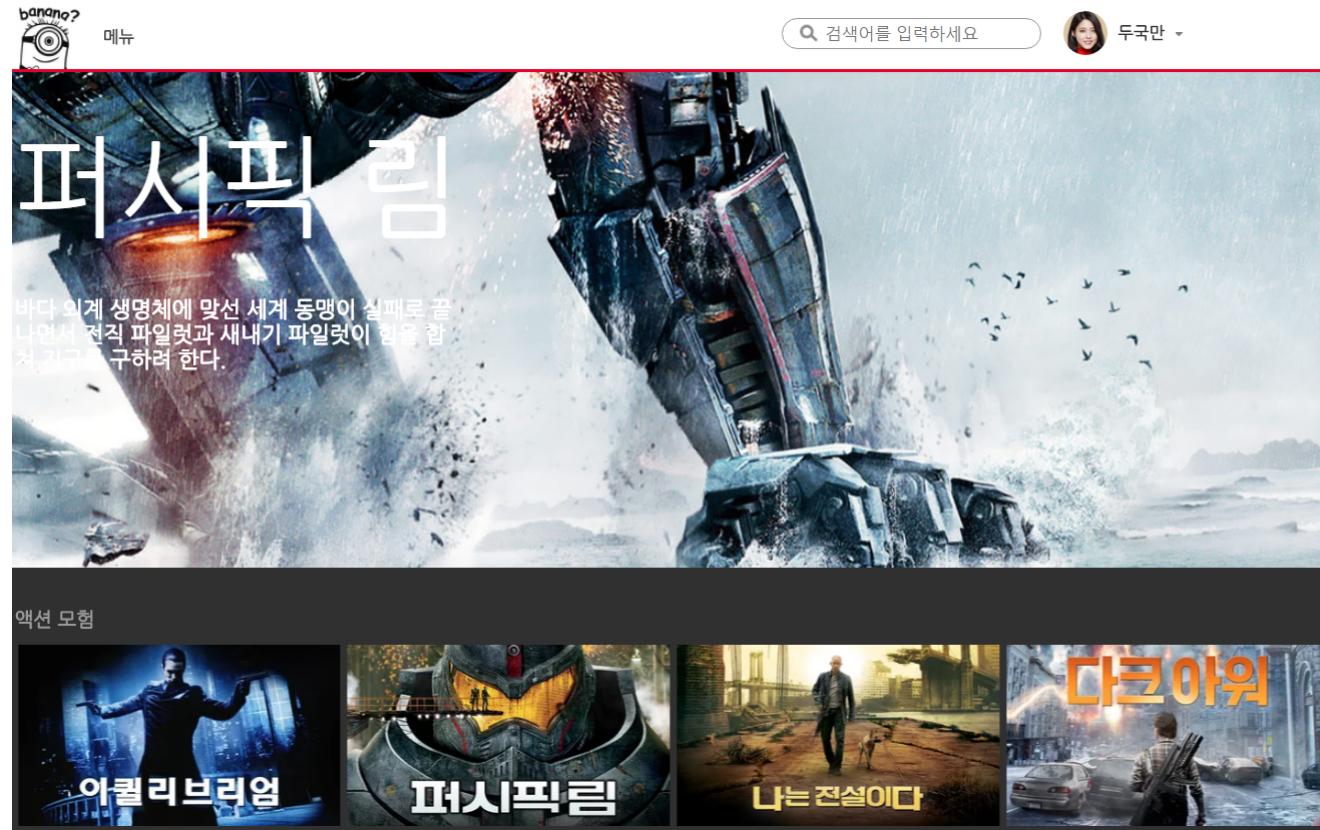
↑ 장애 발생 point "%s/v1/promotions"

Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}")을 추가하고 fallbackMethod를 구현합니다.

4-2. Fallback – ViaNetwork

<http://localhost:8080>



Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}") 을 추가하고 fallbackMethod를 구현합니다.

4-3. Fallback – ViaNetwork 中 추가 장애

```
@HystrixCommand(fallbackMethod="getPromotionFallBackViaNetwork")
public Content getPromotion() {
    return restTemplate.getForObject(String.format("%s/v1/promotion", serviceUrl), Content.class);
}

@HystrixCommand(fallbackMethod="getPromotionFallBackStubbed")
public Content getPromotionFallBackViaNetwork() {
    log.info("exec getPromotionFallBackViaNetwork");

    return restTemplate.getForObject(String.format("%s/v1/promotion", serviceUrl), Content.class);
}

public Content getPromotionFallBackStubbed() {
    log.info("exec getPromotionFallBackStubbed");

    Content content = new Content();
    content.setCategory("action");
    content.setGrade(5);
    ..중략..
    content.setYear(2018);

    return content;
}
```

↑ 장애 발생 point "%s/v1/promotions"

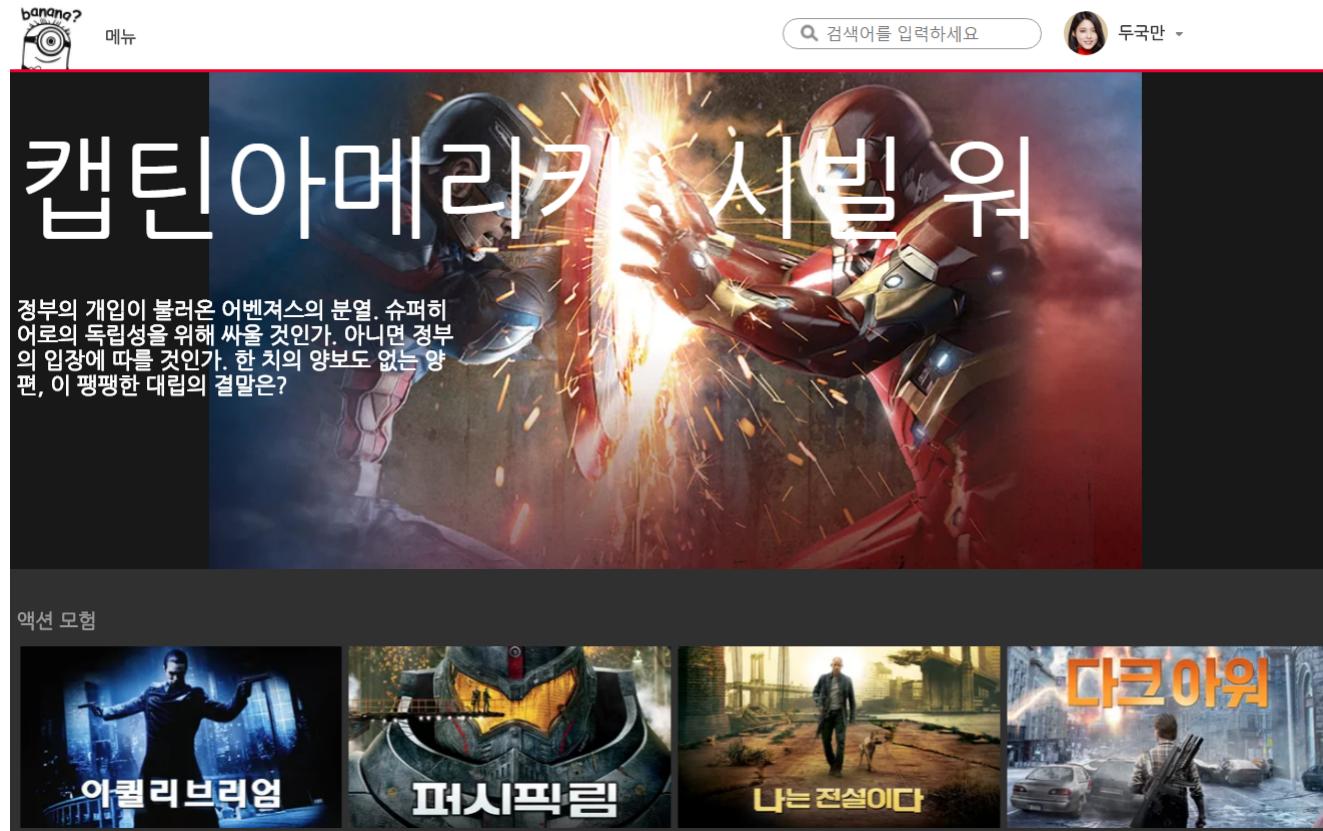
↑ 장애 발생 point "%s/v1/promotions"

Hystrix 실습

@HystrixCommand(fallbackMethod="{fallbackMethod 명}")을 추가하고 fallbackMethod를 구현합니다.

4-3. Fallback – ViaNetwork 中 추가 장애

<http://localhost:8080>

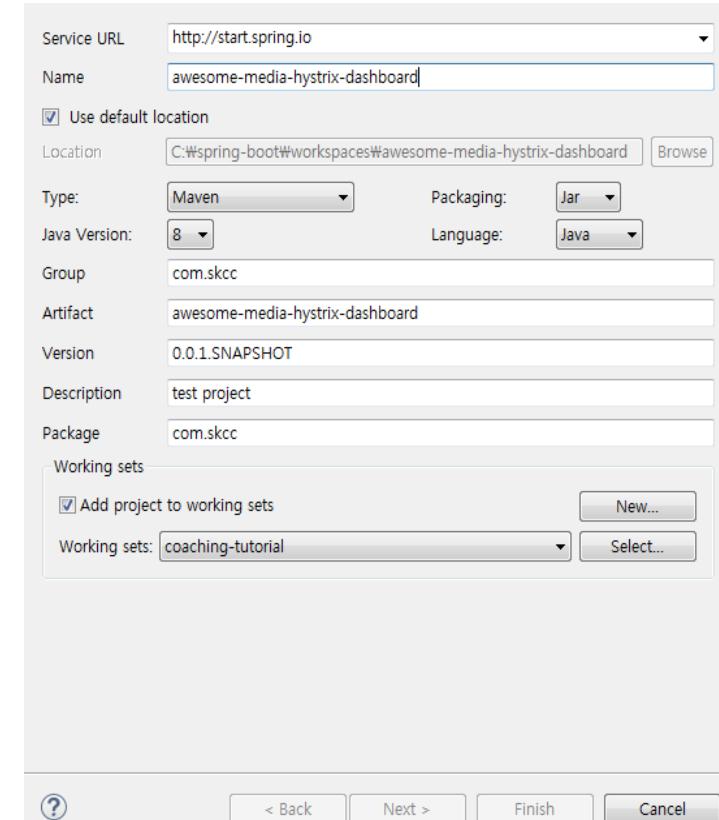


Hystrix Dashboard 실습

Spring Boot + Hystrix Dashboard를 사용해 AWESOME-MEDIA에 Application 모니터링 기능을 구현해 봅시다.

1. Project를 신규로 생성합니다.

- Project Spec.
 - Project 명 : awesome-media-hystrix-dashboard
 - Spring Boot Version: 1.5.10
 - Build Type : Maven
 - Packaging : Jar
 - Java Version : 8

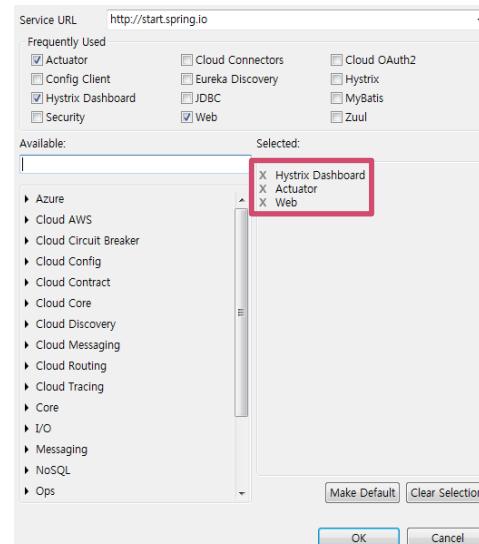


Hystrix Dashboard 실습

Spring Boot + Hystrix Dashboard를 사용해 AWESOME-MEDIA에 Application 모니터링 기능을 구현해 봅시다.

2. spring-cloud-starter-hystrix-dashboard Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
</dependency>
```

3. application.yml에 설정을 추가합니다.

```
server:
  port: 7979

management:
  security:
    enabled: false

spring:
  application:
    name: awesome-media-hystrix-dashboard
```

Hystrix Dashboard 실습

Spring Boot + Hystrix Dashboard를 사용해 AWESOME-MEDIA에 Application 모니터링 기능을 구현해 봅시다.

4. @EnableHystrixDashboard 어노테이션을 추가합니다.

```
@SpringBootApplication  
@EnableHystrixDashboard  
public class AweSomeMediaHystrixDashboardApplication {  
    public static void main(String[] args){  
        SpringApplication.run(AweSomeMediaHystrixDashboardApplication.class, args);  
    }  
}
```

Hystrix Dashboard 실습

AWESOME-MEDIA-HYSTRIX-DASHBOARD를 구동해 AWESOME-MEDIA-FRONT를 모니터링합니다.

<http://localhost:7979/hystrix>



Hystrix Dashboard

Cluster via Turbine (default cluster): http://turbine-hostname:port/turbine.stream

Cluster via Turbine (custom cluster): http://turbine-hostname:port/turbine.stream?cluster=[clusterName]

Single Hystrix App: http://hystrix-app:port/hystrix.stream

Delay: ms

Title:

- ⓘ Hystrix Dashboard의 모니터링은 대상 Application의 1번 이상 호출 수행 후 정상적으로 모니터링수행이 가능합니다.

Application 구동 직후 동작없이 Hystrix Dashboard에서 접근을 하면 해당 metrics를 정상적으로 조회하지 못합니다.

- 확인방법 : <application_host>:<application_port>/hystrix.stream → 'ping: ' 만 반복되면 application 동작을 한번 수행한 후 다시 모니터링 합니다.

Hystrix Dashboard 실습

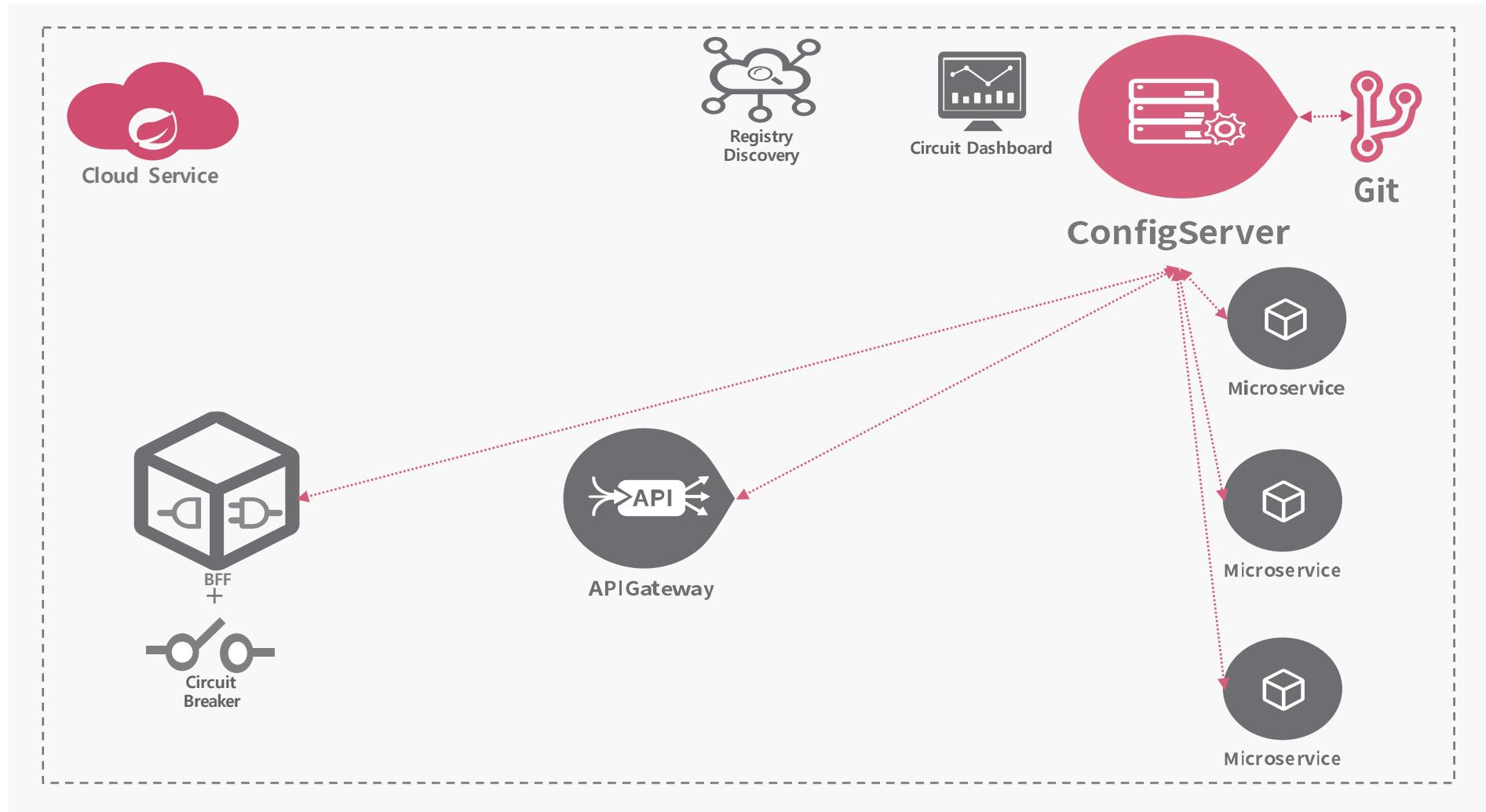
AWSOME-MEDIA-HYSTRIX-DASHBOARD를 구동해 AWSOME-MEDIA-FRONT를 모니터링 합니다.

<http://localhost:8080/hystrix.stream>

The screenshot shows the Hystrix Stream interface at <http://localhost:8080/hystrix.stream>. At the top, there is a JSON dump of the Hystrix configuration for the 'getPromotion' command. Below it, the main dashboard displays two circuit breakers and a thread pool:

- Circuit Breaker Status:**
 - getPromotion:** 100.0% failure rate, Host: 1.3/s, Cluster: 1.3/s, Circuit Closed. Latency: 47ms Median, 48ms Mean.
 - getPromotionFallBackViaNetwork:** 100.0% failure rate, Host: 1.3/s, Cluster: 1.3/s, Circuit Closed. Latency: 21ms Median, 25ms Mean.
- Thread Pools:**
 - PromotionService:** Active Queue Size: 0, Pool Size: 10, Max Active Executions: 28, Queue Size: 5. Host: 2.8/s, Cluster: 2.8/s.

Cloud Service Diagram – Config Server



Config Server

Cloud Service 中 Config Server에 대해 알아봅니다.



Spring Cloud Config Server

- **Config Server**

- Spring Cloud 中 Config Server 구현에 관련된 기능을 제공 ([Docs Link](#))
- MSA Application은 하나의 코드베이스를 사용하고 배포환경 별로 설정을 분리해서 관리합니다. (12-factors)
- Application의 Profile 별 설정을 중앙(Git Repo, File 시스템)에 모아두고 관리포인트를 일원화합니다.
- Git Repo를 사용하는 경우 설정 History를 관리할 수 있습니다.
- Config Server는 외부에서 설정을 조회/갱신할 수 있는 EndPoint를 제공합니다.
- 설정 변경 후 Application의 재빌드 없이 변경사항을 적용 할 수 있습니다.

Config Server

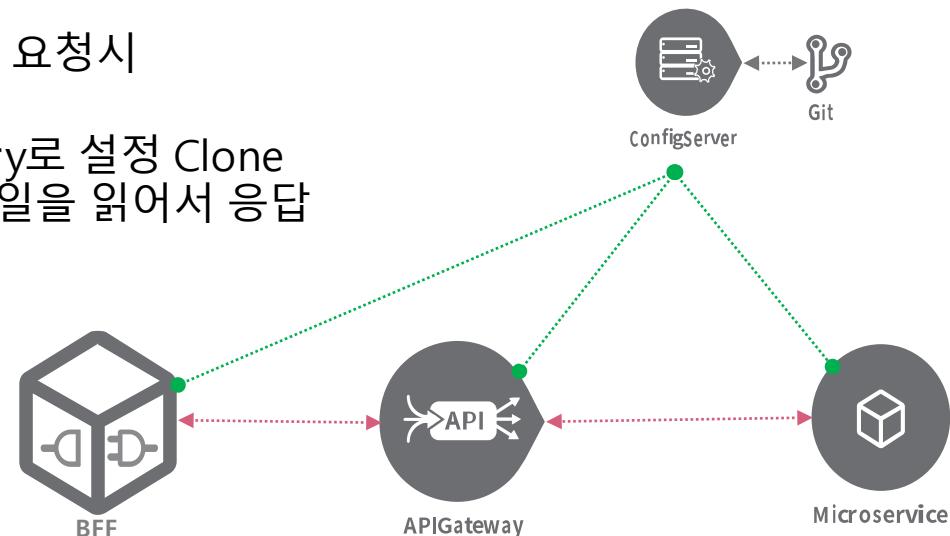
Cloud Service 中 Config Server에 대해 알아봅니다.

- **Config Server 동작**

1. Client Application으로부터 첫번째 요청시
2. Local Working Directory 생성
3. 원격 저장소에서 Working Directory로 설정 Clone
4. Working Directory에 있는 설정 파일을 읽어서 응답

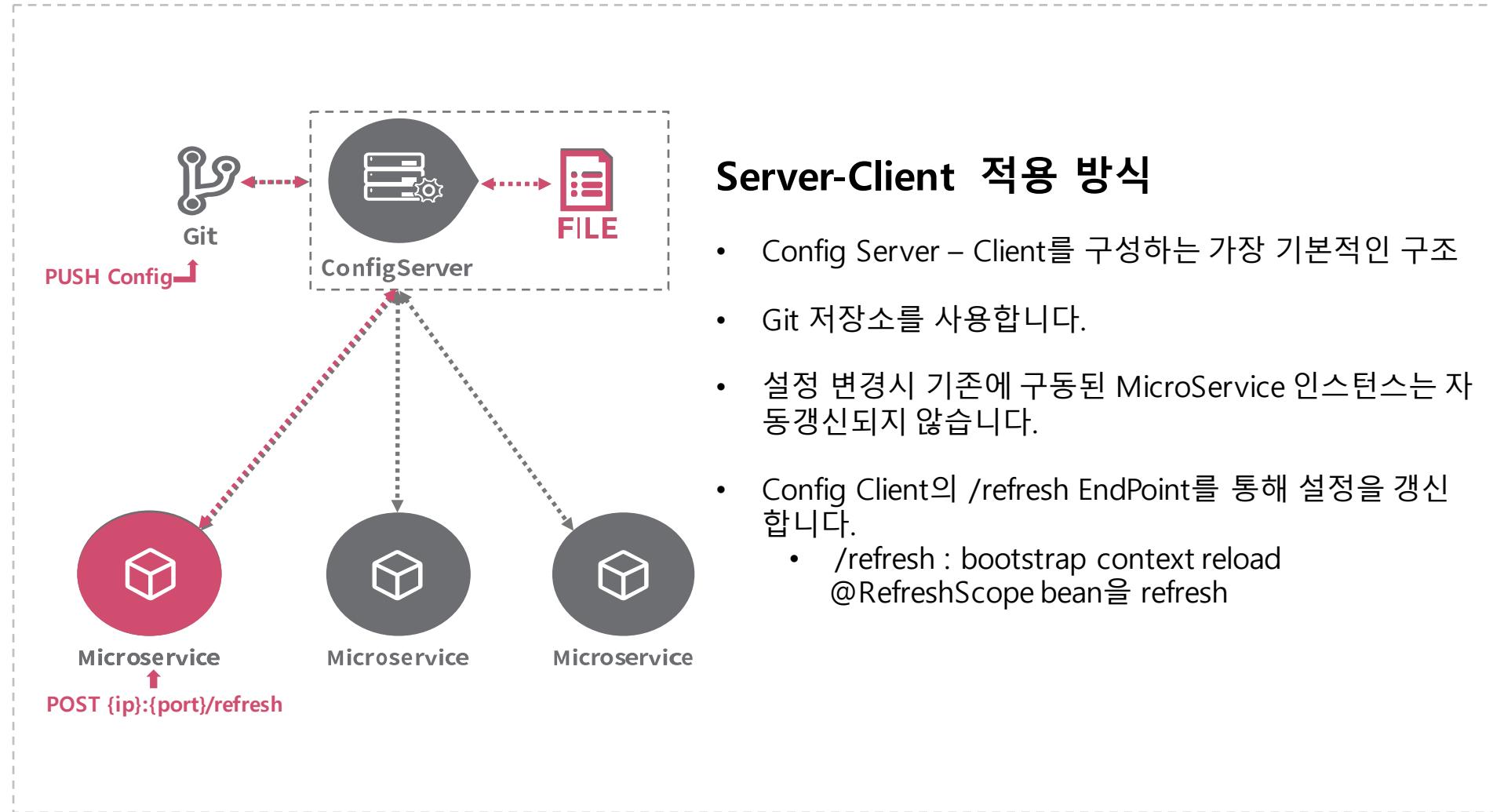
- **Config Server 구성 유형**

1. Server-Client 적용 방식
2. MQ 활용 방식
3. Webhook 활용 방식



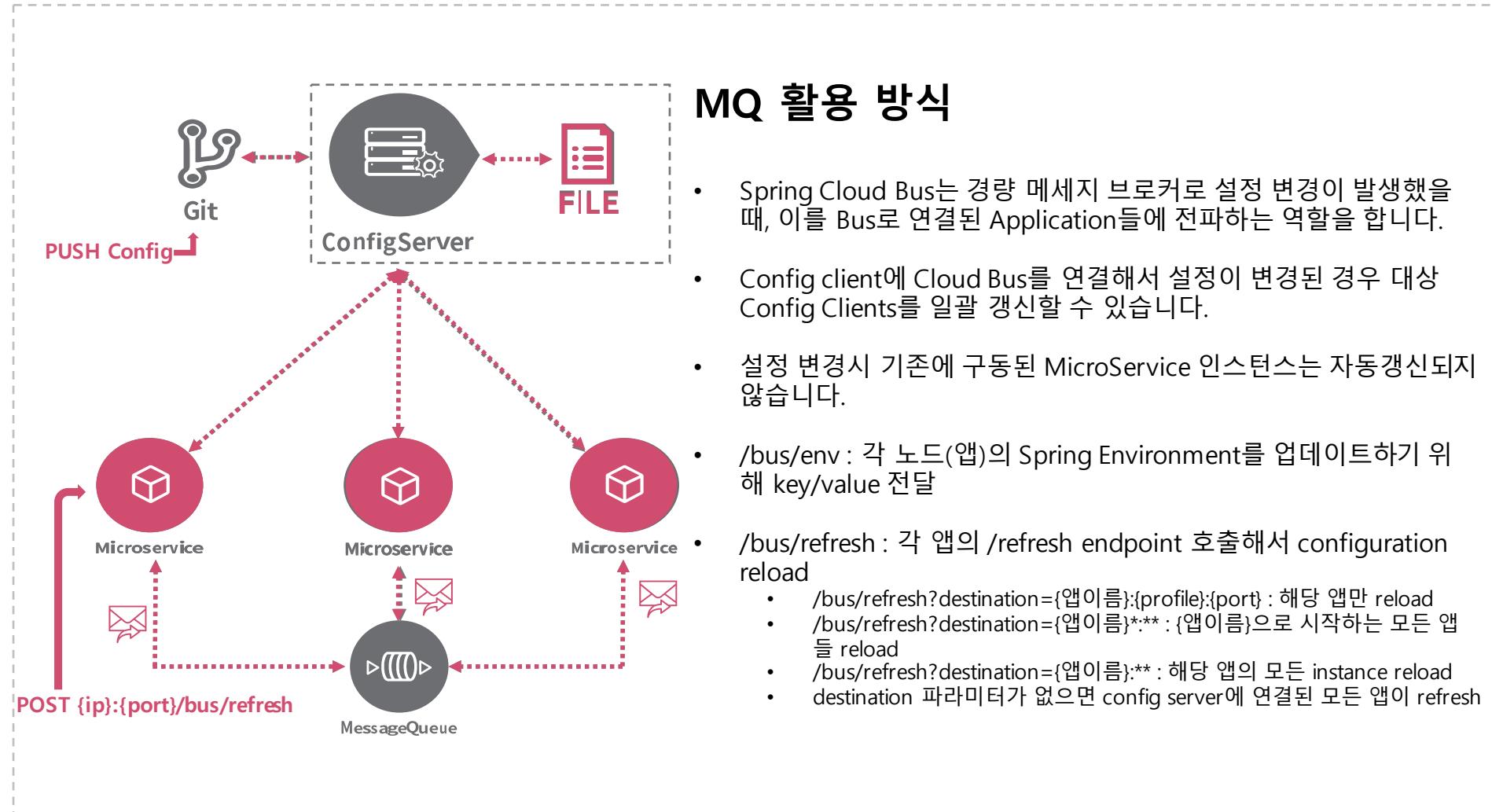
Config Server

Config Server 구성 유형 中 Server-Client 적용 방식을 알아봅니다.



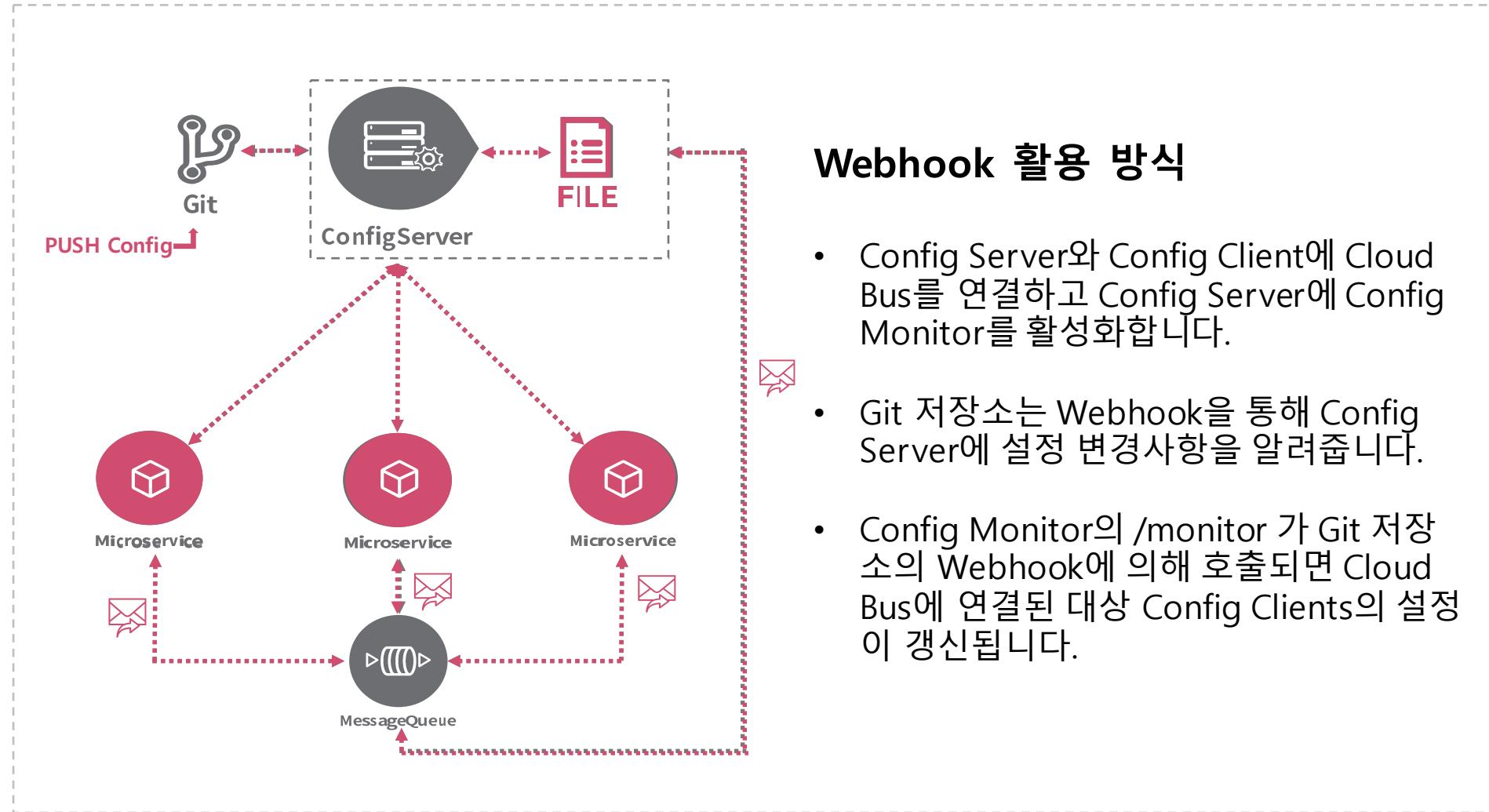
Config Server

Config Server 구성 유형 中 MQ 활용 방식을 알아봅니다.



Config Server

Config Server 구성 유형 中 Webhook 활용 방식을 알아봅니다.



Config Server

Spring Cloud - Config 참조 순서에 대해 알아봅니다.

- **Config** 참조 순서

- config Server를 사용하지 않을 경우
 - bootstrap.yml -> application.yml -> application-{profile}.yml
- config Server를 사용할 경우
 - bootstrap.yml -> application.yml -> application-{profile}.yml
-> application.yml -> {앱이름}.yml -> application-{profile}.yml -> {앱이름}-{profile}.yml
 - ✓ 녹색은 Application 내의 설정파일, 파란색은 Config Server 내의 설정 파일

```
#example
```

```
spring:  
  application:  
    name: foo  
  profiles:  
    active: dev,mysql
```

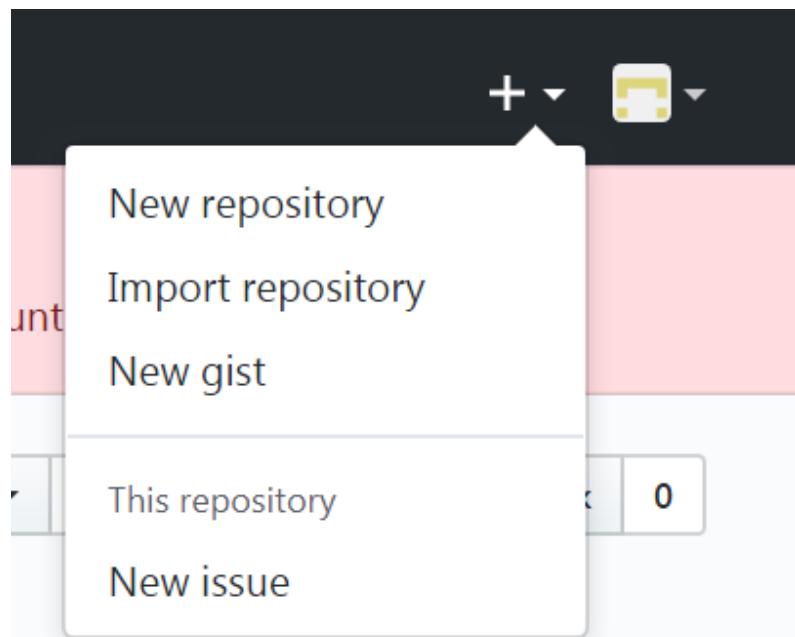
```
// 위와 같이 설정되어 있다면  
// property의 오버라이딩 순서는 다음과 같다. 같은 key값이면 뒤쪽 설정을 가져감  
// application.yml → foo.yml → application-dev.yml → application-mysql.yml → foo-dev.yml → foo-mysql.yml
```

Config Server 실습

Application의 설정 파일들을 Github 저장소에 업로드 합니다.

1. Github 계정 준비 및 Repository 생성

- + New repository 클릭



- Repository 명 입력 후 Create repository 클릭
Repository name : awesome-media-config

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: jisangYun / Repository name: awesome-media-config ✓

Great repository names are short and memorable. Need inspiration? How about urban-octo-fortnight.

Description (optional):

Public: Anyone can see this repository. You choose who can commit.

Private: You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Create repository

Config Server 실습

Application의 설정 파일들을 Github 저장소에 업로드 합니다.

2. README를 클릭해 Repository 초기화

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#)

or

HTTPS

SSH

<https://github.com/jisangYun/awesome-media-config.git>

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

3. Application 설정파일 업로드

Branch: master ▾ New pull request

Create new file Upload files Find file Clone or download ▾

File	Description	Time
jisangYun Create awesome-media-backend-prd.yml	Latest commit 61d888a 3 minutes ago	
README.md	Create README.md	5 minutes ago
awesome-media-backend-default.yml	Create awesome-media-backend-default.yml	4 minutes ago
awesome-media-backend-dev.yml	Create awesome-media-backend-dev.yml	4 minutes ago
awesome-media-backend-k8s.yml	Create awesome-media-backend-k8s.yml	4 minutes ago
awesome-media-backend-prd.yml	Create awesome-media-backend-prd.yml	3 minutes ago
awesome-media-backend-stg.yml	Create awesome-media-backend-stg.yml	3 minutes ago
awesome-media-backend.yml	Create awesome-media-backend.yml	5 minutes ago

✓ 설정 파일 구성 패턴은 프로젝트 개발 표준에 따라 다양한 구성이 가능합니다.

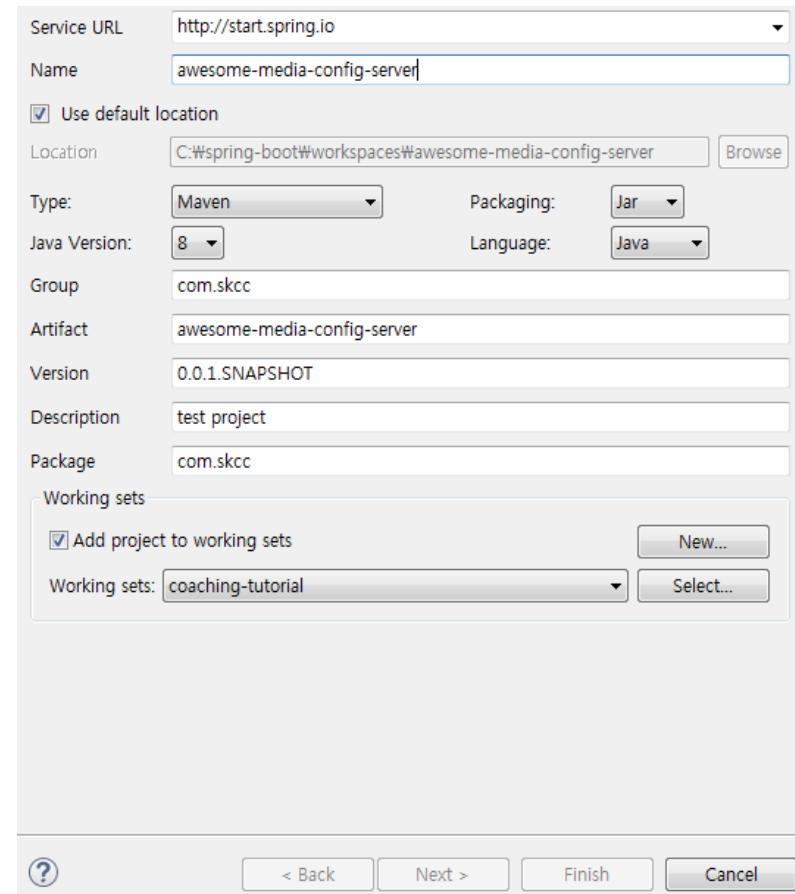
- {Application명}-{profile}.yml
- {Application명}.yml # .yml 파일 내부에서 spring.profiles에 따른 설정 구분
- {Application명} 폴더 내부에 {Application명}-{profile}.yml

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

4. Project를 신규로 생성합니다.

- Project Spec.
 - Project 명 : awesome-media-config-server
 - Spring Boot Version: 1.5.10
 - Build Type : Maven
 - Packaging : Jar
 - Java Version : 8

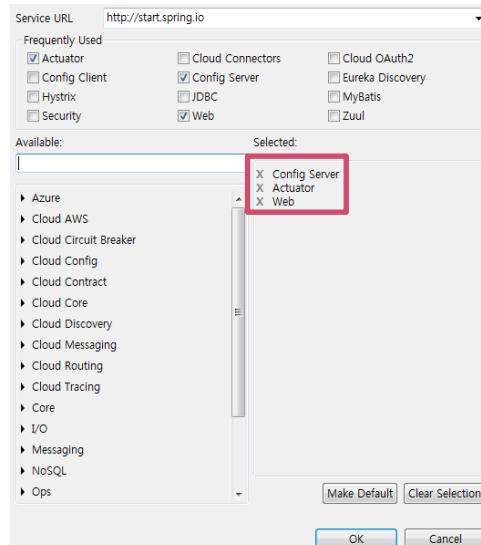


Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

5. spring-cloud-config-server Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

6. application.yml에 설정을 추가합니다.

```
server:
  port: 8888

management:
  security:
    enabled: false

spring:
  application:
    name: awesome-media-config-server
```

7. bootstrap.yml에 설정을 추가합니다.

```
spring:
  cloud:
    config:
      server:
        git:
          uri: https://mygit.skcc.com/scm/cna/awesmoe-media-config.git
          uri: https://github.com/jisangYun/awesome-media-config.git
```

✓ bootstrap.yml은 Application 구동시 참고하는 설정들을 기입하는 파일입니다.

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

8. @EnableConfigServer 어노테이션을 추가합니다.

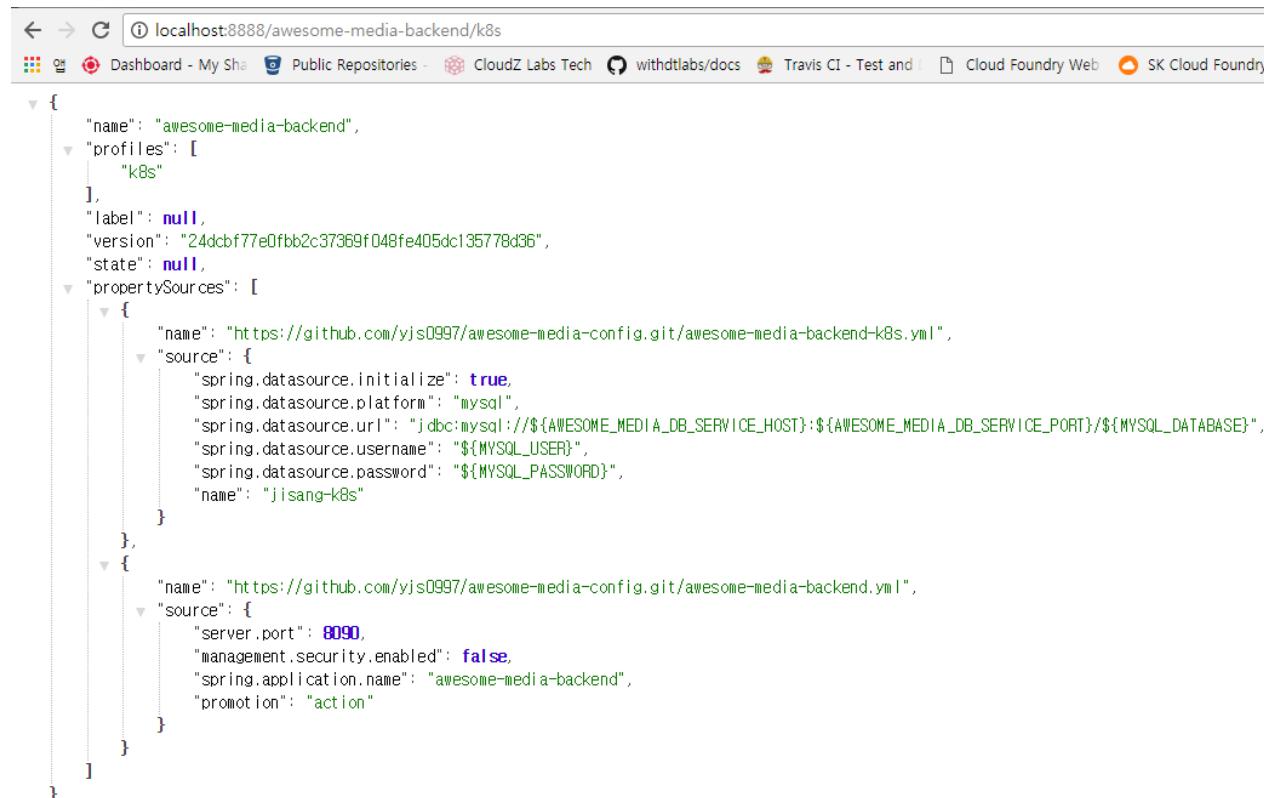
```
@SpringBootApplication  
@EnableConfigServer  
public class AwesomeMediaConfigServerApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(AwesomeMediaConfigServerApplication.class,  
        args);  
    }  
}
```

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

9. {Config Server IP}:{Config Server PORT}/{설정을 가져올 Application명}/{profile}을 확인합니다.

<http://localhost:8888/awesome-media-backend/k8s>



The screenshot shows a browser window with the URL `localhost:8888/awesome-media-backend/k8s`. The page displays a JSON object representing the configuration for the `awesome-media-backend` application. The JSON structure includes fields for name, profiles (k8s), label (null), version (24dcbf77e0fb2c37369f048fe405dc135778d36), state (null), and propertySources. There are two property sources: one from a GitHub repository (`https://github.com/yjs0997/awesome-media-config.git/awesome-media-backend-k8s.yml`) containing MySQL database settings, and another local source (`server.port: 8090`, `management.security.enabled: false`, `spring.application.name: awesome-media-backend`, `promotion: action`). The JSON is color-coded with green for strings and variables like `AWESOME_MEDIA_DB_SERVICE_HOST`.

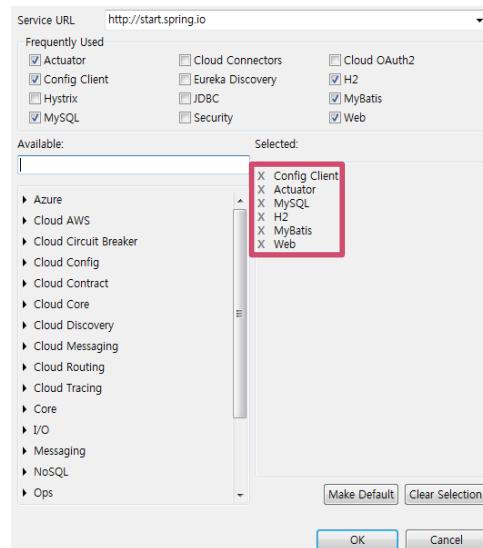
```
{
  "name": "awesome-media-backend",
  "profiles": [
    "k8s"
  ],
  "label": null,
  "version": "24dcbf77e0fb2c37369f048fe405dc135778d36",
  "state": null,
  "propertySources": [
    {
      "name": "https://github.com/yjs0997/awesome-media-config.git/awesome-media-backend-k8s.yml",
      "source": {
        "spring.datasource.initialize": true,
        "spring.datasource.platform": "mysql",
        "spring.datasource.url": "jdbc:mysql://${AWESOME_MEDIA_DB_SERVICE_HOST}:${AWESOME_MEDIA_DB_SERVICE_PORT}/${MYSQL_DATABASE}",
        "spring.datasource.username": "${MYSQL_USER}",
        "spring.datasource.password": "${MYSQL_PASSWORD}",
        "name": "jisang-k8s"
      }
    },
    {
      "name": "https://github.com/yjs0997/awesome-media-config.git/awesome-media-backend.yml",
      "source": {
        "server.port": 8090,
        "management.security.enabled": false,
        "spring.application.name": "awesome-media-backend",
        "promotion": "action"
      }
    }
  ]
}
```

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

10. AWESOME-MEDIA-BACKEND에
spring-cloud-config-server Dependency를 추가합니다

1) spring boot starters를 사용해서 추가 가능



11. bootstrap.yml에 설정을 추가합니다.

```
spring:
  profiles: default
  cloud:
    config:
      uri: http://localhost:8888
      name: awesome-media-backend
```

#http://localhost:8888/awesome-media-backend/default의
의미

2) pom.xml에 작성해서 추가 가능

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

Config Server 실습

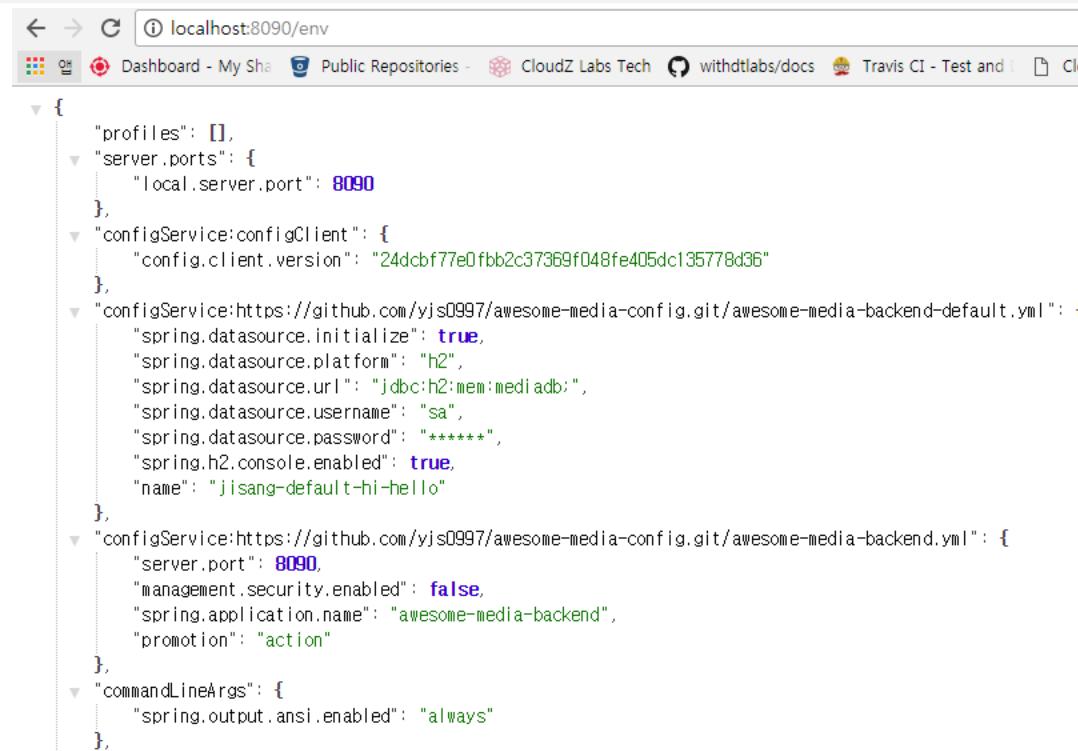
Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

12. AWESOME-MEDIA-BACKEND 내 설정을 제거하고 서버를 재기동합니다.

- Config Server를 통해 설정을 가져오는 Log를 확인합니다.

```
c.c.c.ConfigServicePropertySourceLocator : Fetching config from server at: http://localhost:8888
c.c.c.ConfigServicePropertySourceLocator : Located environment: name=awesome-media-backend, profiles=[default], label=null, version=24dcbf77e0fb2c
b.c.PropertySourceBootstrapConfiguration : Located property source: CompositePropertySource [name='configService', propertySources=[MapPropertySour
```

- AWESOME-MEDIA-BACKEND의 "/env" EndPoint를 조회해 설정이 적용된 것을 확인합니다.



```
{
  "profiles": [],
  "server.ports": {
    "local.server.port": 8090
  },
  "configService:configClient": {
    "config.client.version": "24dcbf77e0fb2c37369f048fe405dc135778d36"
  },
  "configService:https://github.com/yjs0997/awesome-media-config.git/awesome-media-backend-default.yml": {
    "spring.datasource.initialize": true,
    "spring.datasource.platform": "H2",
    "spring.datasource.url": "jdbc:h2:mem:mediadb;",
    "spring.datasource.username": "sa",
    "spring.datasource.password": "*****",
    "spring.h2.console.enabled": true,
    "name": "jisang-default-hi-hello"
  },
  "configService:https://github.com/yjs0997/awesome-media-config.git/awesome-media-backend.yml": {
    "server.port": 8090,
    "management.security.enabled": false,
    "spring.application.name": "awesome-media-backend",
    "promotion": "action"
  },
  "commandLineArgs": {
    "spring.output.ansi.enabled": "always"
  }
}
```

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

13. 재기동 없이 설정 변경이 적용되는지 확인을 위해 NameController 구현과 설정을 추가합니다.

- Package : com.awesome.api.name.controller
- Controller 명 : NameController

NameController.java

```
@RestController
@RefreshScope
public class NameController{
    @Value("${name}")
    private String name;

    @GetMapping(value="/name")
    public String name(){
        return "this is " + name;
    }
}
```

yjs0997 Update awesome-media-backend-default.yml

1 contributor

13 lines (11 sloc) | 187 Bytes

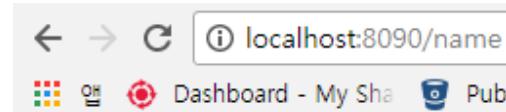
```
1 spring:
  2   datasource:
  3     initialize: true
  4     platform: h2
  5     url: jdbc:h2:mem:mediadb;
  6     username: sa
  7     password: null
  8   h2:
  9     console:
 10       enabled: true
 11
 12   name: jisang-default
```

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

14. AWESOME-MEDIA-BACKEND를 재기동하고 "/name" EndPoint를 확인합니다.

<http://localhost:8090/name>



15. Github에서 설정을 변경합니다.

yjs0997 Update awesome-media-backend-default.yml
1 contributor
13 lines (11 sloc) | 190 Bytes

```
1 spring:  
2   datasource:  
3     initialize: true  
4     platform: h2  
5     url: jdbc:h2:mem:mediadb;  
6     username: sa  
7     password: null  
8   h2:  
9     console:  
10    enabled: true  
11  
12   name: jisang-default-hi
```

The image shows a GitHub commit page for a file named "awesome-media-backend-default.yml". The commit message is "Update awesome-media-backend-default.yml". It shows one contributor and 13 lines of code with 11 source lines. The code itself is a configuration snippet for a Spring application, specifically for a database connection. At the bottom of the code block, the line "name: jisang-default-hi" is highlighted with a red rectangular box.

Config Server 실습

Spring Boot + Spring Cloud Config를 사용해 AWESOME-MEDIA에 Config Server 패턴을 구현해 봅시다.

16. AWESOME-MEDIA-BACKEND의 "/refresh" EndPoint를 호출하여 새로운 설정을 적용합니다.

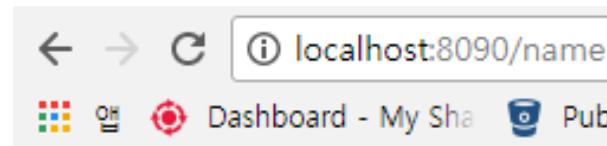
The screenshot shows a Postman interface with the following details:

- Method: POST
- URL: localhost:8090/refresh
- Authorization: No Auth
- Headers: (4)
- Body: JSON
- Status: 200 OK
- Body content (Pretty):

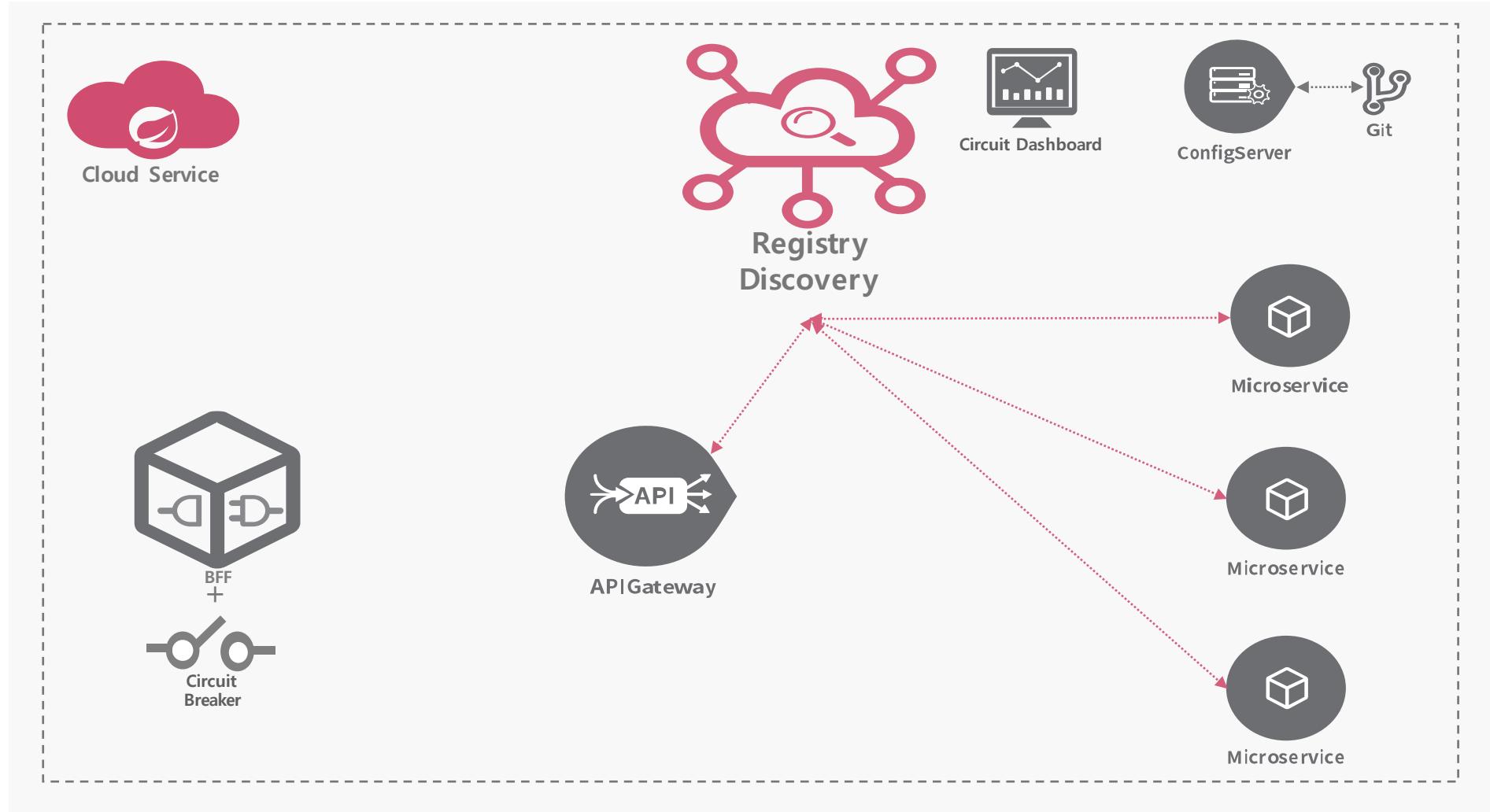
```
1 [  
2   "config.client.version",  
3   "name"  
4 ]
```

17. AWESOME-MEDIA-BACKEND를 재기동 하지 않고, "/name" EndPoint를 호출합니다.

<http://localhost:8090/name>

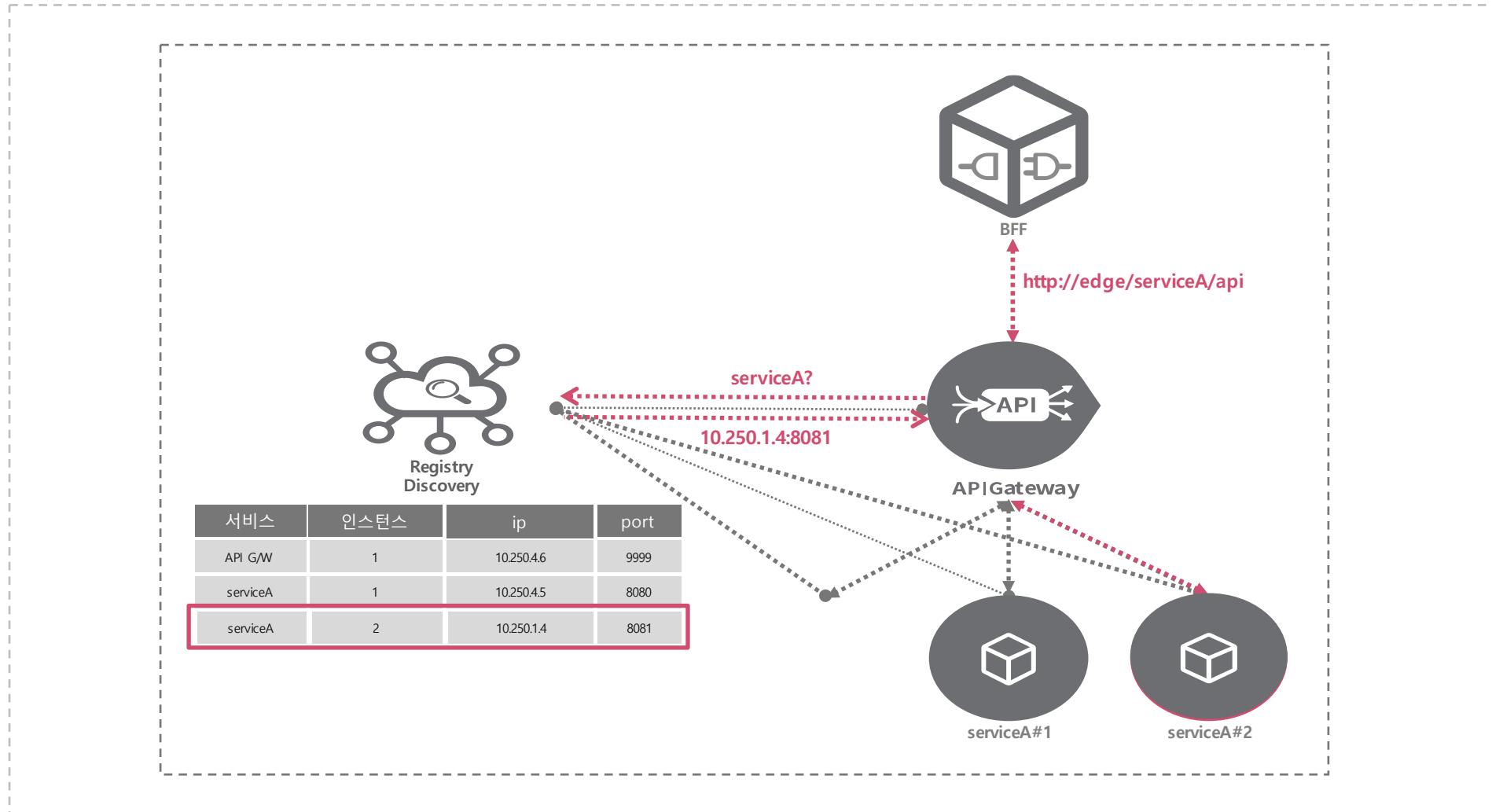


Cloud Service Diagram - Registry/Discovery



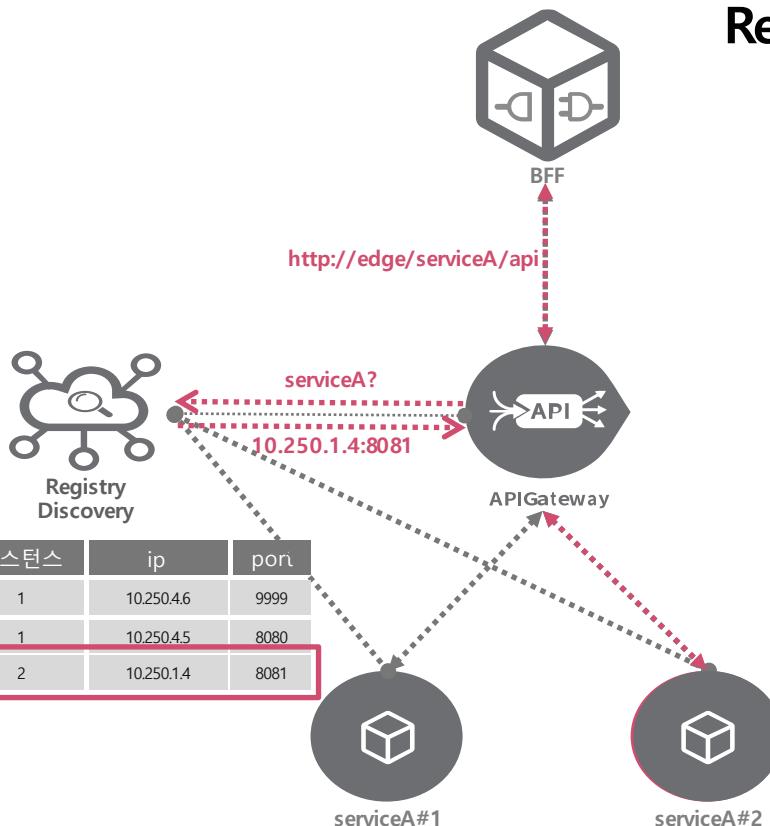
Registry/Discovery

Cloud Service 中 Registry/Discovery에 대해 알아봅니다.



Registry/Discovery

Cloud Service 中 Registry/Discovery에 대해 알아봅니다.



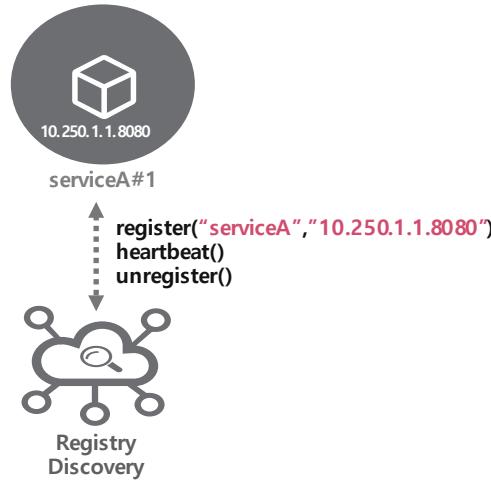
Registry/Discovery

- MSA에서 각 MicroService는 인스턴스의 수와 위치(ip,port)가 동적으로 변하기 때문에 이를 관리하는 매커니즘이 필요합니다.
- 클라이언트는 Service Discovery에 등록된 MicroService 인스턴스의 주소를 호출할 수 있습니다.
- Service Discovery는 서비스 등록/해제/조회/instance health check 등의 API를 제공합니다.
- 기본 동작
 - Regist
 - Discover
 - Load Balancing
 - Health Check

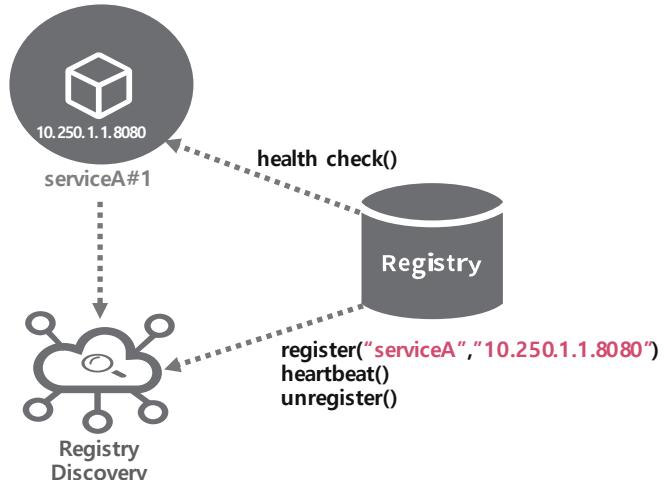
Registry/Discovery

Cloud Service 中 Registry/Discovery에 대해 알아봅니다.

Self-Registration



3rd party Registration



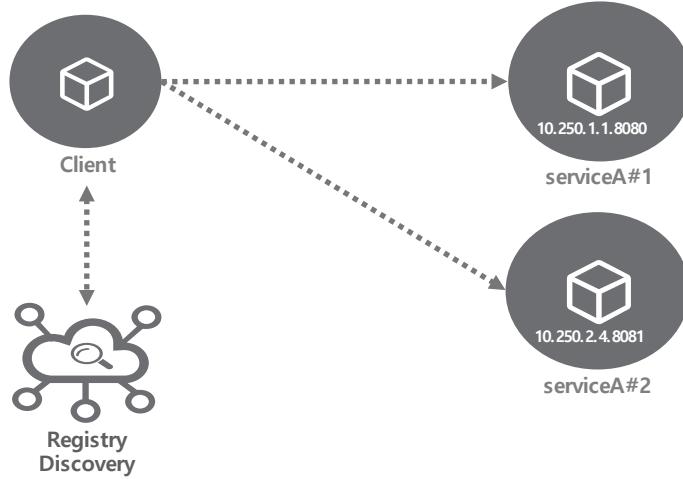
- MicroService 스스로 Service Discovery에 등록/해제 할 책임이 있습니다.
- 등록 정보가 만료되지 않게 Service Registry에 heartbeat를 전송해야 합니다.
- 장점 : 비교적 단순하고 별도의 컴포넌트를 추가할 필요가 없습니다.
- 단점 : MicroService에서 Service Discovery 등록/해제 기능을 구현해야 합니다.
- Spring Netflix Eureka, Spring Netflix Ribbon 등

- MicroService 스스로 Service Discovery에 등록/해제를 하지 않고, Service Registrator가 등록/해제 합니다.
- Service Registrator는 MicroService 인스턴스들에 polling, event 구독 등의 방식으로 변경을 감지하고 등록/해제 합니다.
- 장점 : MicroService에서 추가 기능 구현이 필요 없습니다.
- 단점 : 추가 Component가 필요하고, 고가용성을 유지해야합니다.
- etcd, zookeeper, consul 등

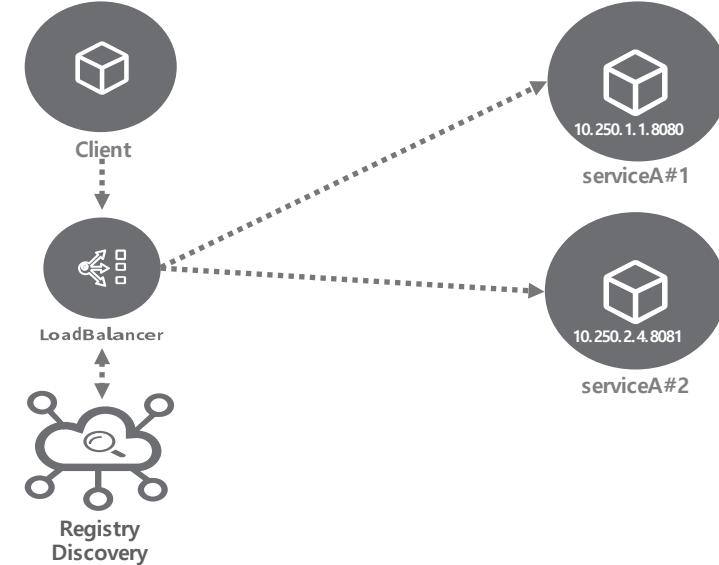
Registry/Discovery

Cloud Service 中 Registry/Discovery에 대해 알아봅니다.

Client-side Discovery & Load Balancing



Server-side Discovery & Load Balancing



- 클라이언트가 Service Discovery에 질의를 해서 네트워크 주소를 얻고, 로드밸런싱 알고리즘을 통해 서비스를 선택 후 요청(request)을 생성합니다.
- 장점 : 비교적 간단하다. 클라이언트가 사용 가능한 서비스를 알기 때문에 개별로 알맞는 Load Balancing 방법을 선택할 수 있습니다.
- 단점 : 클라이언트와 Service Discovery 사이에 의존성이 생긴다. 클라이언트에 Discovery 기능을 구현해야 한다.
- ex. Spring Cloud Netflix Eureka

- 클라이언트는 로드밸런서를 통해 서비스에 요청을 보냅니다. 로드밸런서는 service registry에 서비스의 네트워크 주소를 질의한 뒤, 사용 가능한 서비스로 각 요청을 Routing 합니다.
- 장점 : 클라이언트에서 Discovery 기능을 분리할 수 있습니다. Kubernetes Cluster 등 몇몇 플랫폼에서 해당 기능을 제공합니다.
- 단점 : LoadBalancing을 위한 Component가 배포되어 있어야 합니다. 고 가용성을 유지해야 합니다.
- ex. AWS Elastic Load Balancer(ELB), Kubernetes Cluster 내 Service

Registry/Discovery

Spring Cloud Netflix OSS – Eureka에 대해 알아봅니다.



The screenshot shows the Eureka UI interface. At the top, there's a navigation bar with tabs like HOME, SEARCH, and FILTER. Below it, the main content area has sections for System Status, DS Replicas, and Instances currently registered with Eureka.

System Status:

Environment	test
Data center	default
Current time	2017-05-29T01:16:31 -0800
Uptime	00:04
Lease expiration enabled	true
Renew threshold	1
Renews (last min)	26

DS Replicas:

DS Replicas	eureka-serviceregistry-second
-------------	-------------------------------

Instances currently registered with Eureka:

Application	AMIs	Availability Zones	Status
EUREKA-SERVICEREGISTRY	n/a (1)	(1)	UP (I) - 192.168.105.51:eureka-serviceregistry:8761
EUREKA-SERVICEREGISTRY-SECOND	n/a (1)	(1)	UP (I) - 192.168.111.230:eureka-serviceregistry-second:8761
WEATHERBACKEND	n/a (1)	(1)	UP (I) - 192.168.97.47:weatherbackend:8090
WEATHERSERVICE	n/a (1)	(1)	UP (I) - 192.168.108.18:weatherservice:8095
ZUUL-EDGESERVICE	n/a (1)	(1)	UP (I) - 192.168.111.215:zuul-edgeservice:8080

- **Eureka 란?**

- Spring Cloud Netflix OSS 中 Registry/Discovery 구현에 관련된 기능을 제공 ([Docs Link](#))

- **기능**

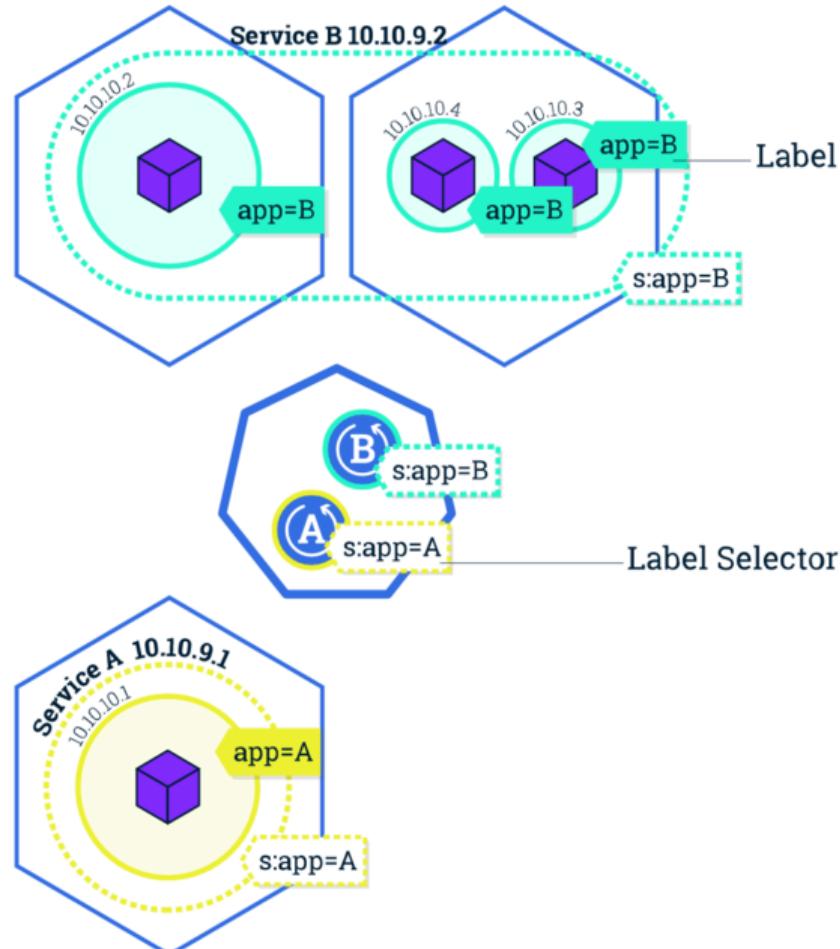
- Registry/Discovery 기능을 Server & Client 형태로 구현
- 등록된 MicroService 인스턴스의 상태를 확인할 수 있는 모니터링 대쉬보드를 지원

- **기본 동작**

- Eureka의 등록/해제 유형은 Self-Registration
- Eureka는 Client-side Discovery & LoadBalancing 을 사용합니다.
- Eureka Server는 Standalone 과 Peer Awareness로 구동할 수 있습니다. ([Eureka Peer 동작 원리](#))

Registry/Discovery

Kubernetes - Service 대해 알아봅니다.

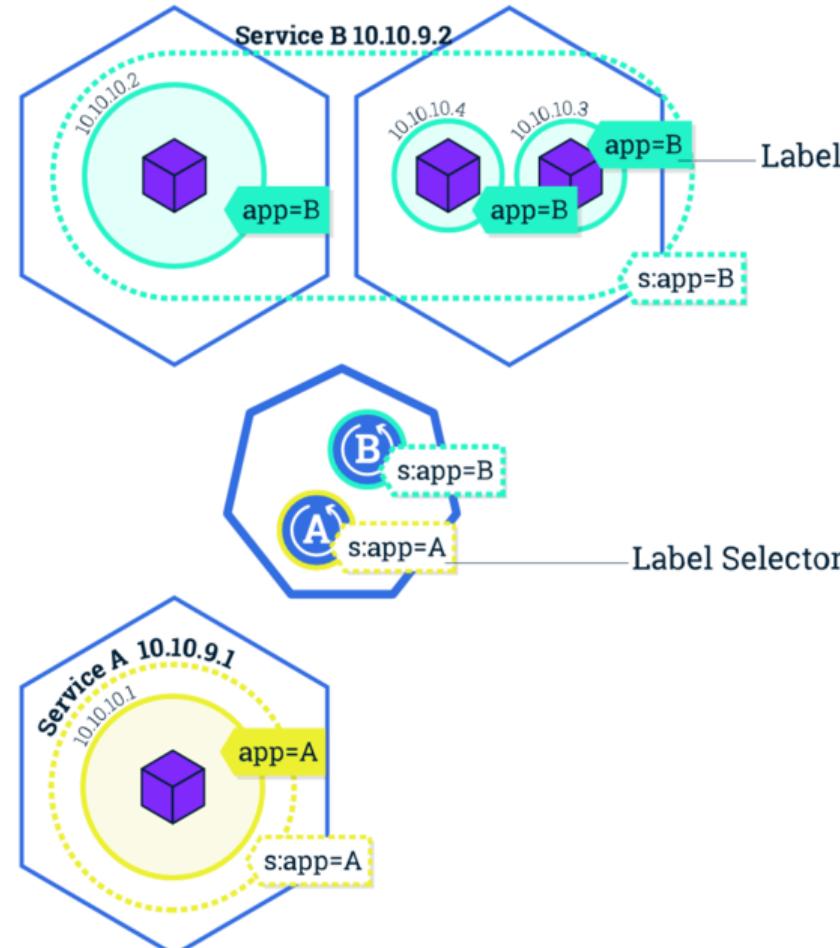


■ Service

- Pod의 논리적 집합과 액세스 정책을 정의하는 추상화 개념
- 서비스가 타겟으로 하는 Pod의 집합은 Label Selector에 의해 결정
 - Label
 - Pod와 같은 Object에 설정되는 key/value 쌍의 별칭
 - Selector
 - 타 Object를 찾기 위한 별칭 검색어
 - 해당 서비스에서 관리하는 Pod의 목록(실제 Pod의 엔드포인트)을 맵핑하고 있는 Endpoint 객체가 자동으로 생성
 - Pod가 죽으면 자동으로 Endpoint에서 제거되고, Selector와 매칭되는 새로운 Pod가 생기면 자동으로 추가됨
- 로드밸런싱, 서비스 디스커버리 등 제공
- 서비스를 통해 외부에서 접속하거나 클러스터 내부에서만 접근하도록 설정 가능

Registry/Discovery

Kubernetes - Service 대해 알아봅니다.



■ 서비스 디스커버리 방식

1. 환경변수 방식

- Pod 생성 시 컨테이너 내부에 모든 서비스들의 HOST, PORT 정보가 환경변수로 주입됨
- 컨테이너에서 환경변수를 통해 다른 서비스에 대한 접속 정보를 알 수 있음
- **redis-master** 서비스가 미리 생성되어 있는 경우, 애플리케이션을 Pod로 띄우면 내부에 아래와 같은 환경 변수가 주입됨

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

2. DNS 서버 방식

- ICP에 이미 구성된 DNS 서비스를 통해 서비스명으로 접근
- 다른 네임스페이스에 있는 서비스도 접근 가능함

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

1. Project를 신규로 생성합니다.

- Project Spec.
 - Project 명 : awesome-media-discovery
 - Spring Boot Version: 1.5.10
 - Build Type : Maven
 - Packaging : Jar
 - Java Version : 8

Service URL: http://start.spring.io

Name: awesome-media-discovery

Use default location

Location: C:\spring-boot\workspaces\awesome-media-discovery

Type: Maven Packaging: Jar

Java Version: 8 Language: Java

Group: com.skcc

Artifact: awesome-media-discovery

Version: 0.0.1.SNAPSHOT

Description: test project

Package: com.skcc

Working sets:

Add project to working sets

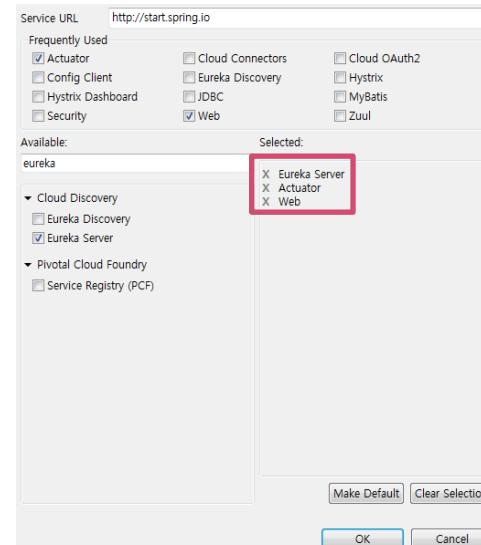
Working sets: coaching-tutorial

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

2. spring-cloud-starter-netflix-eureka-server Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-boot-starter-eureka-server</artifact
Id>
</dependency>
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

3. application.yml에 설정을 추가합니다.

```
server:  
  port: 8761  
  
management:  
  security:  
    enabled: false  
  
spring:  
  application:  
    name: awesome-media-discovery  
  
eureka:  
  instance:  
    instance-id: localhost:${server.port}  
    hostname: localhost  
  client:  
    region: default  
    fetch-registry: false  
    register-with-eureka: false  
    service-url:  
      defaultZone: http://localhost:8761/eureka/
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

4. @EnableEurekaServer 어노테이션을 추가합니다.

```
@SpringBootApplication  
@EnableEurekaServer  
public class AwesomeMediaDiscoveryApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(AwesomeMediaDiscoveryApplication.class, args);  
    }  
}
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

5. AWESOME-MEDIA-DISCOVERY를 기동하고 Eureka Dashboard를 확인합니다.

<http://localhost:8761>

The screenshot shows the Spring Eureka dashboard at <http://localhost:8761>. The top navigation bar includes links for GitHub, OAuth, Developers, Concourse, IBM Bluemix, and Netflix. The main header says "spring Eureka". Below it, there are two tabs: "HOME" and "LAST 1000 SINCE STARTUP".

System Status

Environment	test	Current time	2018-04-06T15:31:39 +0900
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

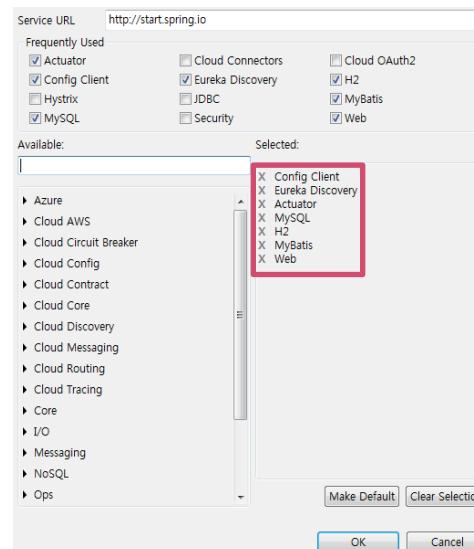
Name	Value
total-avail-memory	329mb
environment	test
num-of-cpus	8
current-memory-usage	171mb (51%)
server-uptime	00:00
registered-replicas	
unavailable-replicas	

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

6. AWESOME-MEDIA-BACKEND에 spring-cloud-starter-eureka Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

7. application.yml에 설정을 추가합니다.

```
eureka:  
  instance:  
    instance-id: localhost:${server.port}  
    hostname: localhost  
  client:  
    region: default  
    fetch-registry: true  
    register-with-eureka: true  
  service-url:  
    defaultZone: http://localhost:8761/eureka/
```

8. @EnableEurekaClient 어노테이션을 추가합니다.

```
@SpringBootApplication  
@EnableEurekaClient  
public class AwesomeMediaBackendApplication {  
  public static void main(String[] args) {  
    SpringApplication.run(AwesomeMediaBackendApplication.class, args);  
  }  
}
```

9. AWESOME-MEDIA-BACKEND를 재기동합니다.

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

10. Eureka Dashboard를 확인합니다.

<http://localhost:8761>

The screenshot shows the Spring Eureka dashboard at <http://localhost:8761>. The top navigation bar includes links for Dashboard, Public Repositories, CloudZ Labs Tech, Travis CI - Test and, Cloud Foundry Web, SK Cloud Foundry, labs admin, GitHub, oauth, Developers - Concur, IBM Bluemix - 클라우드, tower-of-babel, and Spring Cloud Netflix. The main content area has sections for System Status, DS Replicas, and General Info.

System Status

Environment	test	Current time	2018-04-06T15:53:27 +0900
Data center	default	Uptime	00:03
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	4

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AWESOME-MEDIA-BACKEND	n/a (1)	(1)	UP (1) - awesome-media-backend:8090

General Info

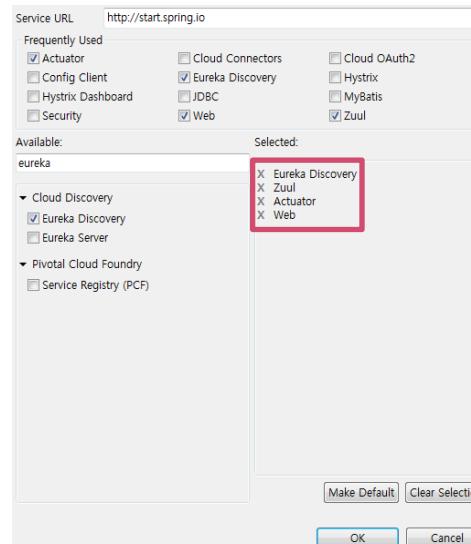
Name	Value
total-avail-memory	347mb
environment	test
num-of-cpus	8
current-memory-usage	84mb (24%)
server-upptime	00:03
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

11. AWESOME-MEDIA-APIGATEWAY에 spring-cloud-starter-eureka Dependency를 추가합니다.

- spring boot starters를 사용해서 추가 가능



- pom.xml에 작성해서 추가 가능

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

12. application.yml에 설정을 추가합니다.

```
eureka:  
  instance:  
    instance-id: localhost:${server.port}  
    hostname: localhost  
  client:  
    region: default  
    fetch-registry: true  
    register-with-eureka: true  
    service-url:  
      defaultZone: http://localhost:8761/eureka/
```

13. application.yml에 routes 설정을 변경합니다.

```
zuul:  
  routes:  
    awesome-media-backend: /awesome-media/**  
# awesome-media-backend:  
# path: /awesome-media/**  
# url: http://localhost:8090
```

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

14. application.yml에 설정을 추가합니다.

```
@SpringBootApplication  
@EnableEurekaClient  
public class AwesomeMediaApigatewayApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(AwesomeMediaApigatewayApplication.class, args);  
    }  
}
```

15. AWESOME-MEDIA-BACKEND를 재기동합니다.

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

16. Eureka Dashboard를 확인합니다.

<http://localhost:8761>

The screenshot shows the Spring Eureka dashboard at <http://localhost:8761>. The top navigation bar includes links for GitHub, OAuth, Developers, and various cloud providers like Cloud Foundry, SK Cloud Foundry, and IBM Bluemix. The main interface has sections for System Status, DS Replicas, and General Info.

System Status:

Environment	test	Current time	2018-04-06T16:02:11 +0900
Data center	default	Uptime	00:12
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	6

DS Replicas:

localhost

Instances currently registered with Eureka:

Application	AMIs	Availability Zones	Status
AWESOME-MEDIA-APIGATEWAY	n/a (1)	(1)	UP (1) - awesome-media-apigateway:9999
AWESOME-MEDIA-BACKEND	n/a (1)	(1)	UP (1) - awesome-media-backend:8090

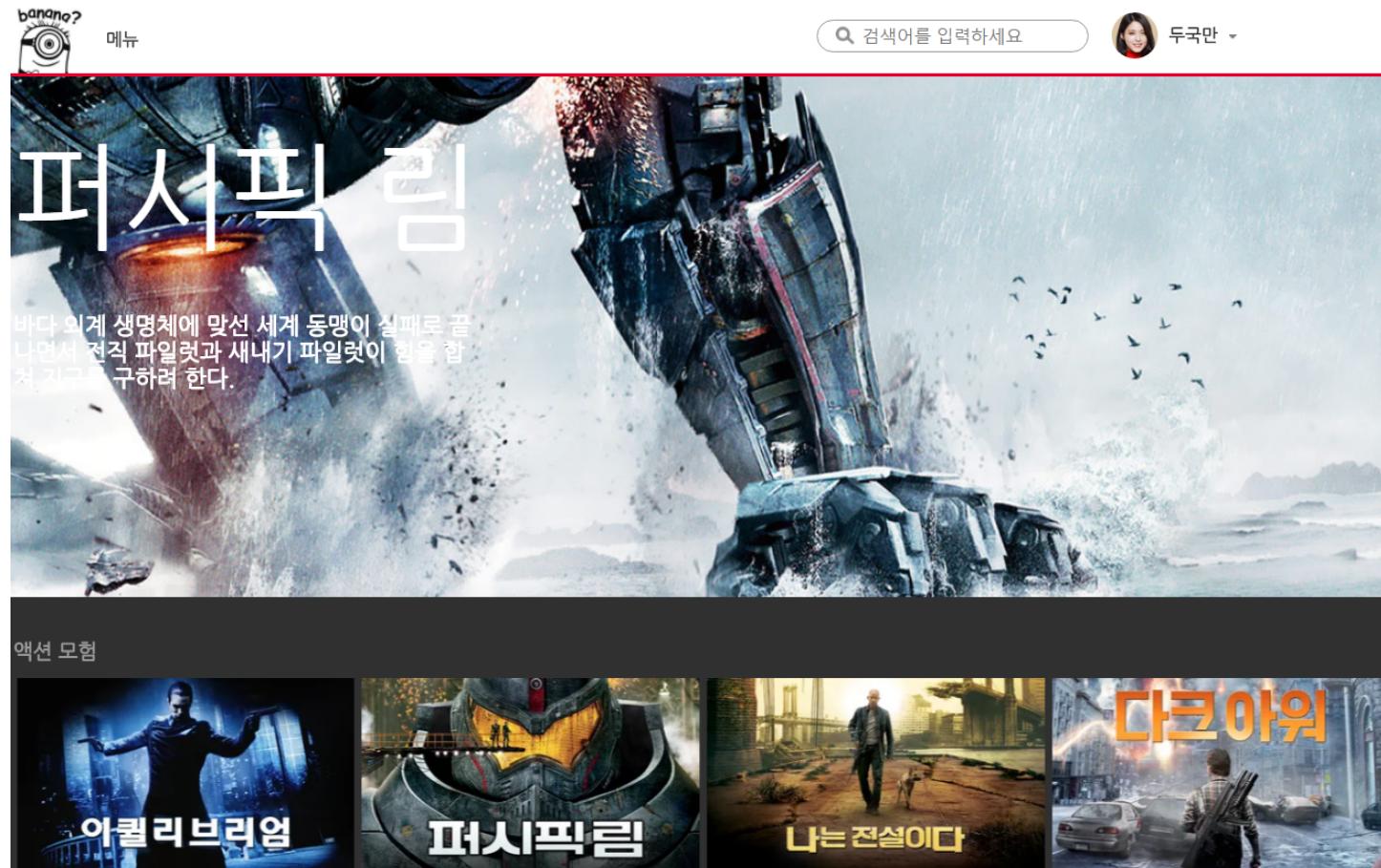
General Info:

Name	Value
total-avail-memory	347mb
environment	test
num-of-cpus	8
current-memory-usage	91mb (26%)
server-upptime	00:12
registered-replicas	http://localhost:8761/eureka/

Registry/Discovery 실습

Spring Boot + Eureka를 사용해 AWESOME-MEDIA에 Registry/Discovery 기능을 구현해 봅시다.

<http://localhost:8080>



[Backup]Eureka의 다양한 설정

Eureka 설정의 의미를 파악합니다.

Eureka 설정

Eureka:

instance:

instance-id: localhost:8090 -- Eureka에 등록될 instance ID입니다.

hostname: localhost:\${server.port} -- discovery service에 등록된 hostname입니다.

prefer-ip-address: true -- false일 경우 hostname이 등록되고, true일 경우 ip-address 값이 등록됩니다.

-- no-route 구성일 때 false를 선택합니다.

ip-address: 169.10.1.2 -- prefer-ip-address 속성을 true로 설정할 경우 eureka에 등록될 ip입니다.

non-secure-port: 80 -- prefer-ip-address 속성을 true로 설정할 경우 eureka에 등록될 port입니다.

lease-renewal-interval-in-seconds: 30 -- 클라이언트가 eureka에 heartbeat를 보내는 주기입니다.

client:

enabled: true -- eureka에 연결여부를 결정하는 설정입니다.

region: default

fetch-registry: false -- eureka client에서 eureka 정보를 가져올지 결정하는 설정입니다.

registry-with-eureka: true -- eureka를 사용할지 결정하는 설정입니다.

service-url:

defaultZone: http://localhost:8761/eureka/ -- eureka의 url입니다.

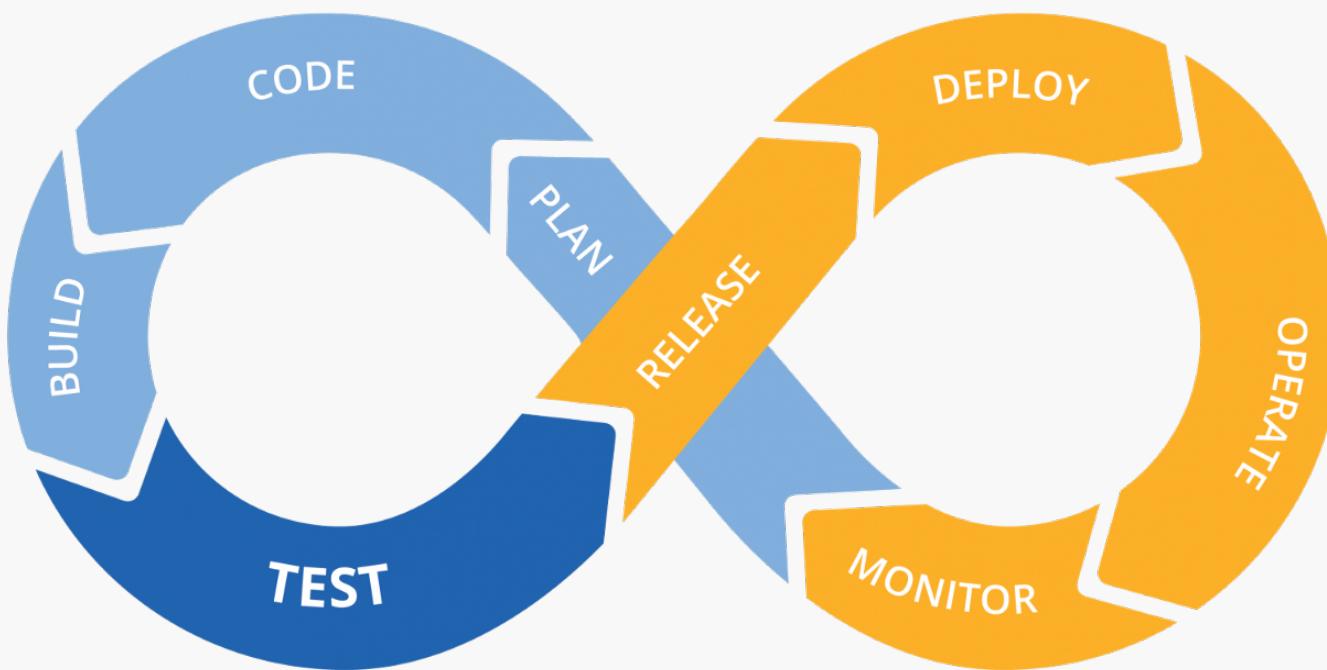
WARNING

Client 설정 시, Discovery Server에 등록될 Application 명에 다음의 기호는 넣지 않습니다.

"~" | "_" | ":" | "!" | "~" | "*" | "^^" | "^^^" | "(" | ")"

해당 기호가 포함될 경우, URI Illegal Syntax Error가 발생합니다.

Deploy



감사합니다

