

# 11장. Load Balancing



Ribbon 개요 및 특징

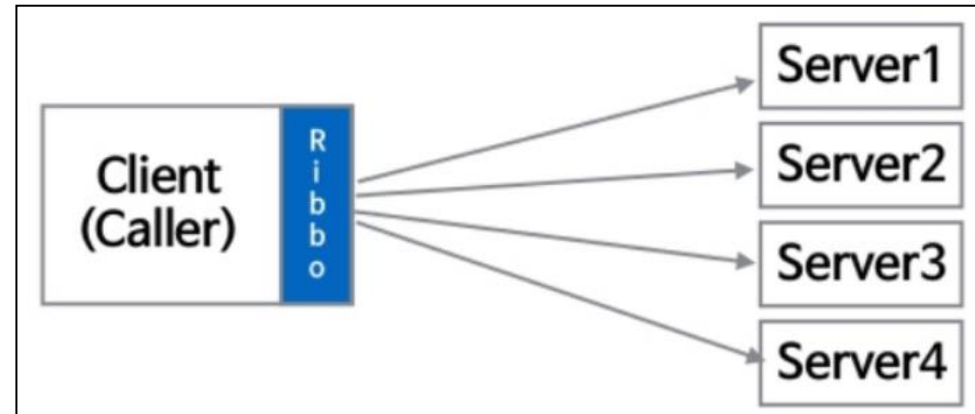
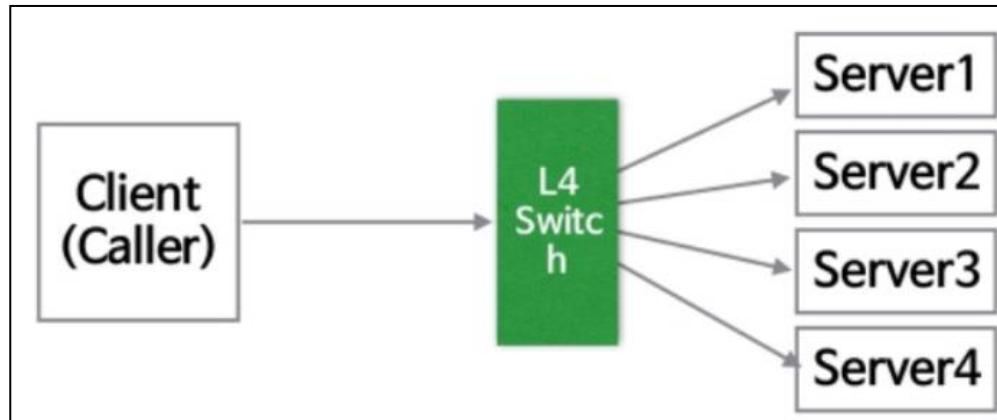
Ribbon 구성요소

Ribbon 적용

# 1. Client LoadBalancer – Ribbon 개요

## 1. Ribbon 개요

Ribbon은 Client에 탑재할 수 있는 소프트웨어 기반의 Load Balancer 이다.  
일반적으로 사용하는 하드웨어적인 L4 Switch(Server-Side 로드밸런싱)를 사용하지만,  
MSA에서는 소프트웨어적으로 구현된 Client-Side 로드밸런싱을 주로 사용한다.  
Ribbon은 분산 처리 방법으로 여러 서버(서비스)를 round-robin (RR)방식으로 부하  
분산 기능을 제공한다.



## 2. Client LoadBalancer – Ribbon 특징/요소

### 2. Ribbon 특징

- REST API를 호출하는 서비스에 탑재되는 S/W 모듈
- 주어진 서버 목록에 대해서 Load Balancing 수행
- 매우 다양한 설정이 가능(서버선택, 실패시 Skip 시간, Ping 처리 등)
- Ribbon에는 retry 기능이 내장되어 있음
- Eureka와 함께 사용되어 매우 강력한 기능 발휘 가능.
- [서버IP:포트번호] 대신에 [서비스명]으로 요청이 가능해진다. (논리적 이름)

### 3. Ribbon 구성요소

#### 가. Rule – 요청을 보낼 서버(서비스)를 선택하는 방법

- Round Robin: 하나의 서버(서비스)씩 돌아가면 요청 (기본)
- Available Filtering: 에러가 많은 서버(서비스) 제외
- Weighted Response Time: 서버(서비스)별 응답 시간에 따라 확률 조절

#### 나. Ping – 서버가 살아 있는지 체크하는 방법

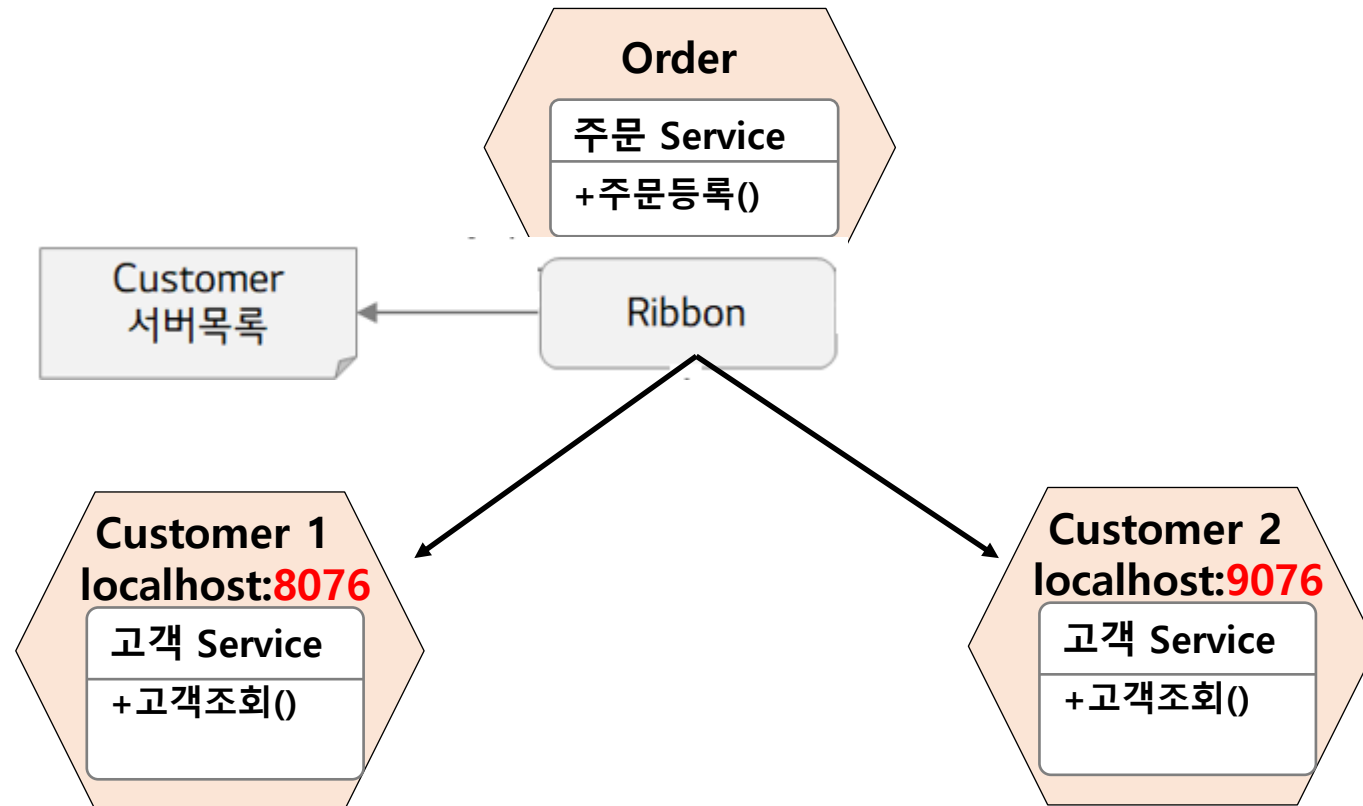
#### 다. Server List – 로드밸런싱 대상 서버(서비스) 목록

- Configuration 을 통해서 static 하게 설정 가능
- Eureka 등을 기반으로 dynamic 하게 설정 가능

### 3. Load Balancing 적용 시나리오

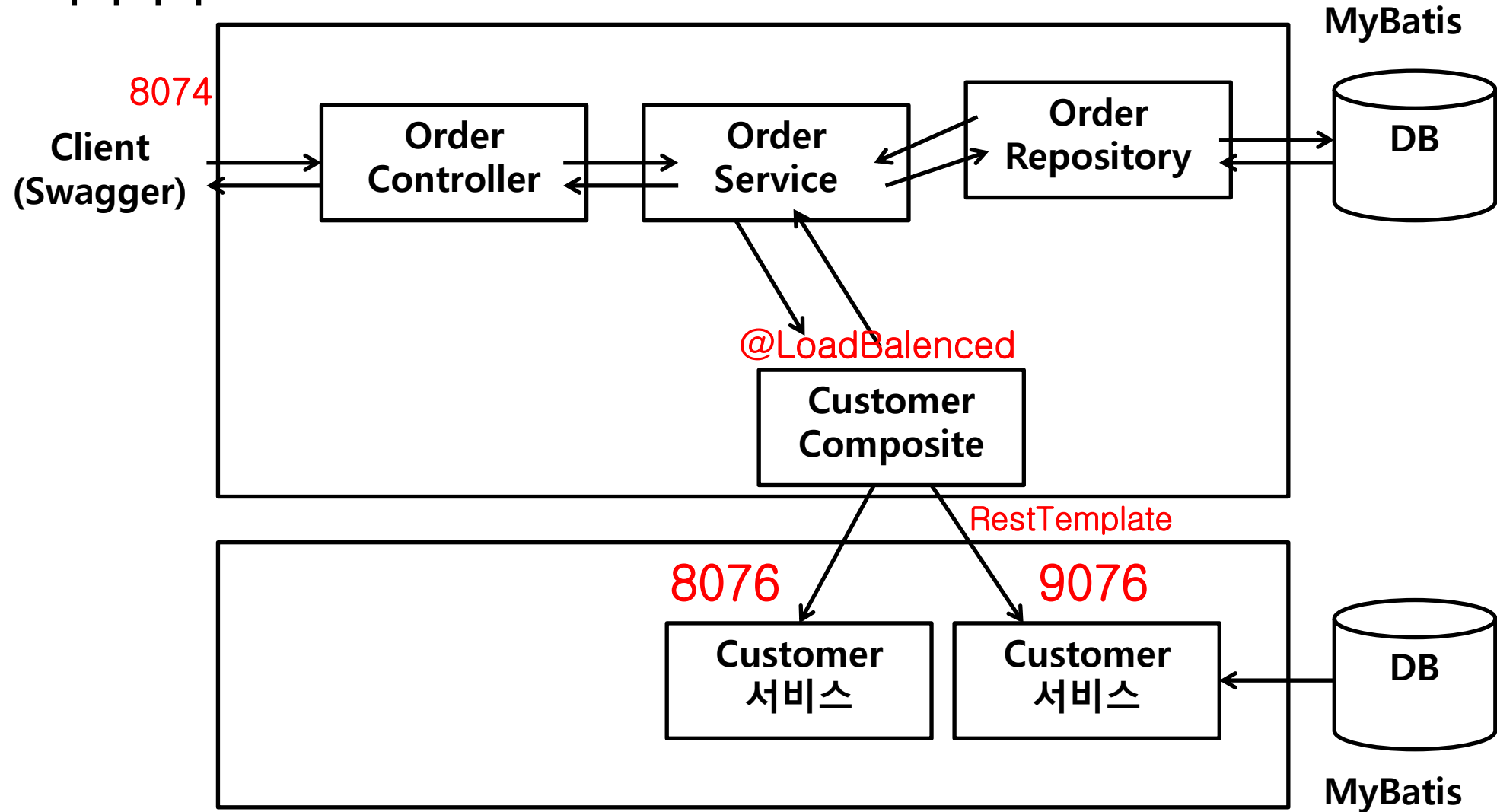
#### 시나리오

Order 서비스에서 주문등록 시 호출하는 Customer 서비스가 이중화로 운영되는 상황에서 Client Load Balancer를 구현한다.



### 3. Load Balancing 적용 시나리오

#### 아키텍처



## 4. Client LoadBalancer – Ribbon 적용

### Ribbon 라이브러리 적용

#### 가. Client 서비스에서 라이브러리 등록

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

#### 나. RestTemplate 사용하는 서비스에서 @LoadBalanced 추가

```
@Component
public class CustomerCompositeImpl implements CustomerComposite {

    @Value("${customer.api.url}")
    private String CUSTOMER_API_URL;

    @LoadBalanced
    @Bean
    public RestTemplate getRestTemplate() {
        return new RestTemplate();
    }

    @Autowired
    RestTemplate restTemplate;
```

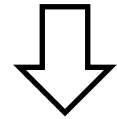
## 4. Client LoadBalancer – Ribbon 적용

### 다. application.properties 수정

```
# ribbon 설정
ecommerce-customer.ribbon.eureka.enabled=false
ecommerce-customer.ribbon.listOfServers=localhost:8076,localhost:9076
ecommerce-customer.ribbon.ServerListRefreshInterval=15000
ecommerce-customer.ribbon.MaxAutoRetries=0

#RestTemplate
#customer.api.url=http://${CUSTOMER}:8076/ecommerce/customer
customer.api.url=http://ecommerce-customer/ecommerce/customer
```

서버IP:포트번호



서비스명

**<clientName>.ribbon.eureka.enabled**  
eureka 사용여부(eureka 사용시 동적 서버 목록 사용 가능 )

**<clientName>ribbon.listOfServers**  
정적 서버 목록

**<clientName>ribbon.ServerListRefreshInterval**  
서버 목록 새로 고침 internal time

**<clientName>ribbon.MaxAutoRetries**  
첫 번째 호출 실패 시 같은 서버로 재시도 하는 횟수

## 4. Client LoadBalancer – Ribbon 적용

application.properties 파일의 server.port 값을 번갈아 수정하여 실행

```
server.port=8076
server.servlet.context-path=/ecommerce/customer
#spring
spring.application.name=ecommerce-customer
```



```
main] c.m.c.MinibankCustomerApplication : Starting MinibankCustomerApplication on DESKTOP-6
main] c.m.c.MinibankCustomerApplication : No active profile set, falling back to default pr
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8076 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.43]
```

```
server.port=9076
server.servlet.context-path=/ecommerce/customer
#spring
spring.application.name=ecommerce-customer
```



```
main] c.m.c.MinibankCustomerApplication : Starting MinibankCustomerApplication on DESKTOP-6
main] c.m.c.MinibankCustomerApplication : No active profile set, falling back to default pr
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9076 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.43]
```



## 4. Client LoadBalancer – Ribbon 적용

## 라. CustomerController 수정 ( 확인용 )

```
@RestController
public class CustomerController {

    @Value("${server.port}")
    String port;

    @Autowired
    CustomerService customerService;

    @ApiOperation(value = "고객 조회", httpMethod = "GET", notes = "고객 조회")
    @GetMapping(value="/rest/customers/{userid}")
    public Customer selectCustomerByUserId(@PathVariable("userid") String userid)

        // 응답지연 코드 추가 ( 10 초 지연 )
        //Thread.sleep(10000);

        System.out.println("현재 사용중인 port: " + port);

        return customerService.selectCustomerByUserId(userid);
}
```

[illegible][illegible]

실습은 RestTemplate 방식만 확인하고  
Feign Client는 Eureka 적용 후 확인한다.

# 12장. Service Registry



적용 시나리오

Eureka 개요 및 아키텍처

Eureka 구성요소

EurekaServer 설정

EurekaClient 설정

# 1. Eureka 개요

## 1) Eureka 개요

서비스 인스턴스 목록과 그 위치(host,port)가 동적으로 변하는 환경에서 사용자가 그 위치를 모두 관리하는 것은 불가능하다.

Eureka를 사용하면 등록된 모든 서비스의 정보를 registry로 관리하고, 이에 대한 접근 정보를 요청하는 서비스에게 목록을 제공한다.

Eureka는 MSA의 장점 중 하나인 동적인 서비스 증설 및 축소를 위하여 필수적으로 필요한 서비스로서 자가 등록, 탐색 및 부하 분산에 사용될 수 있는 라이브러리이다.

Eureka는 Eureka 서버와 Eureka 클라이언트로 구성된다.

### Eureka 서버

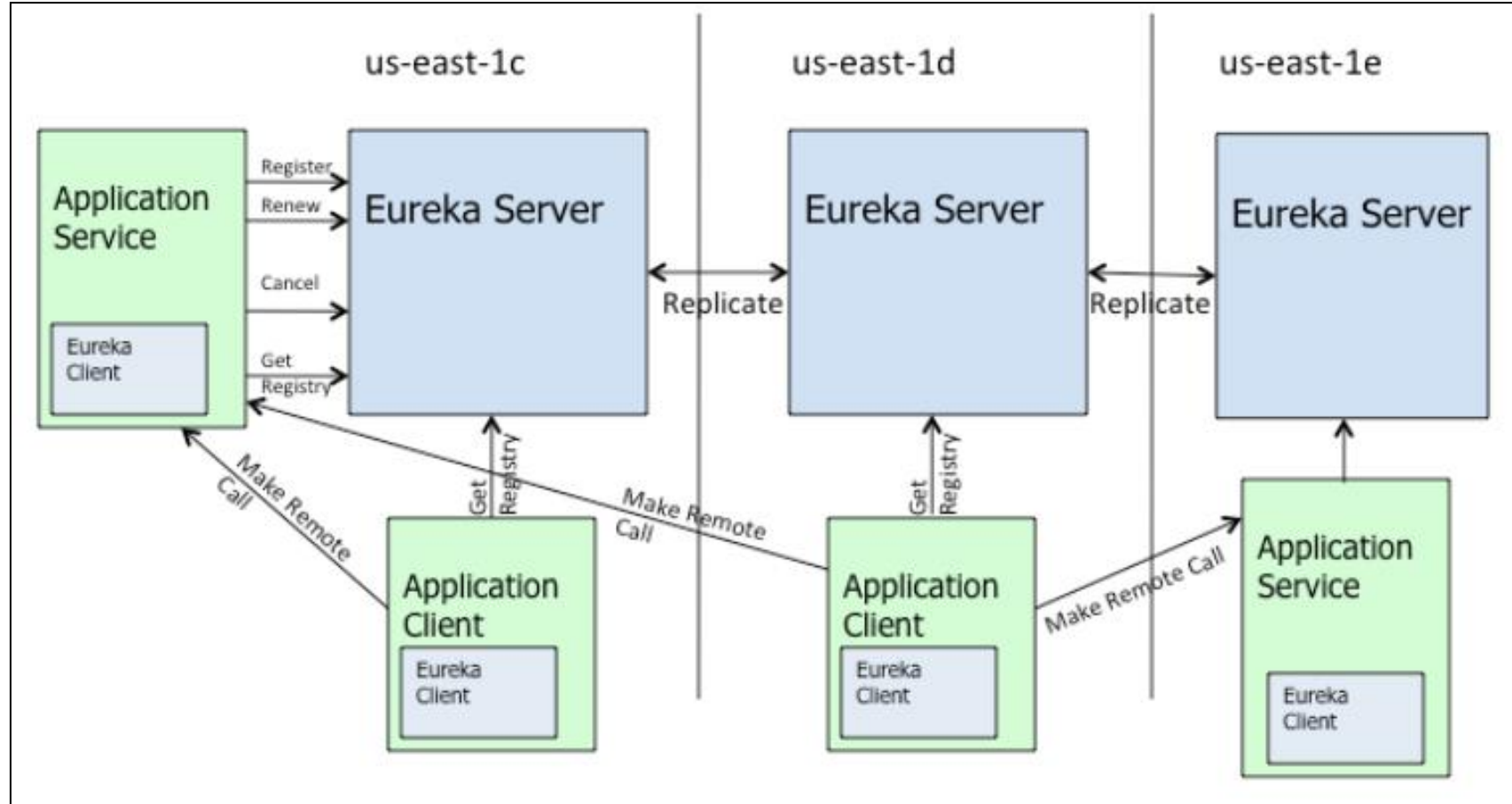
Eureka Service가 자기 자신을 등록(Service Registration)하는 서버이자 Eureka Client가 가용한 서비스 목록(Service Registry)을 요청하는 서버.

### Eureka 클라이언트

Eureka 클라이언트는 서비스가 시작 될 때 Eureka 서버에 자신의 정보를 등록하고 이후 주기적으로 자신의 가용상태(health check)를 알리며, 일정 횟수  
이 상의 ping이 확인되지 않으면 Eureka 서버에서 해당 서비스를 제외 시킨다.

## 2. Eureka 아키텍처

### 2) Eureka 아키텍처



### 3. Eureka 구성

#### 가) Register

eureka.instance, eureka.client 설정값을 바탕으로 Eureka에 등록하기 위한 Eureka instance 정보를 만든다. Client가 eureka 서버로 첫 heartbeat 전송 시 Eureka instance 정보를 등록한다.  
등록된 instance 정보는 eureka dashboard 를 통해서 확인 가능하다.

#### 나) Renew

Client는 eureka에 등록된 이후에 설정된 주기마다 heartbeat를 전송하여 자신의 존재를 알린다.  
속성값: eureka.instance.lease-renewal-interval-in-seconds ( 기본:30)  
설정된 시간 동안 heartbeat를 받지 못하면 해당 Eureka instance를 제거한다.  
속성값: eureka.instance.lease-expiration-duration-in-seconds (기본:90)

#### 다) Fetch Registry

Client는 Server로부터 Registry(서버에 등록된 인스턴스 목록)정보를 가져와서 로컬에 캐시한다. 캐시된 정보는 설정된 주기마다 업데이트 된다.  
속성값: eureka.client.registryFetchIntervalSeconds (기본:30)

### 3. Eureka 구성

#### 라) Cancel

Client가 shutdown될 때 cancel 요청을 eureka 서버로 보내서 registry에서 제거하게 된다.

#### 마) Time Lag

Eureka Server와 client의 registry 관련 캐시 사용으로 인해 client가 호출하려는 다른 instance 정보가 최신으로 갱신되는데 약간의 시간 차가 있다.

#### 바) Self-Preservation mode ( 자가보존모드 )

Eureka Server는 등록된 instance로부터 heartbeat를 주기적으로 받는다. 하지만 네트워크 단절등의 상황으로 heartbeat를 받을 수 없는 경우 일반적으로 registry에서 해당 instance를 제거한다.

Eureka로의 네트워크는 단절되었지만, 해당 서비스 API를 호출하는데 문제가 없는 경우가 있을 수 있는데, 이때 Self-Preservation을 사용하여 registry에서 문제된 instance를 정해진 기간 동안 제거하지 않을 수 있다.

( expected heartbeat수와 actual heartbeat 수 비교 )

### 3. Eureka 구성

#### 사) Peering ( clustering )

고가용성을 위하여 여러 대의 eureka server를 사용하여 서로 peering 구성이 가능하다. Eureka Server는 설정에 정의된 peer nodes를 찾아서 Registry 정보등 Sync 맞추는 작업을 한다.

#### -standalone 구성

속성값: eureka.client.register-with-eureka:false

-Peer nodes로부터 registry를 갱신할 수 없을 때 재시도 횟수

속성값: eureka.server.registry-sync-retries (기본:5)

```
server.port=8761

spring.application.name=ecommerce-eureka

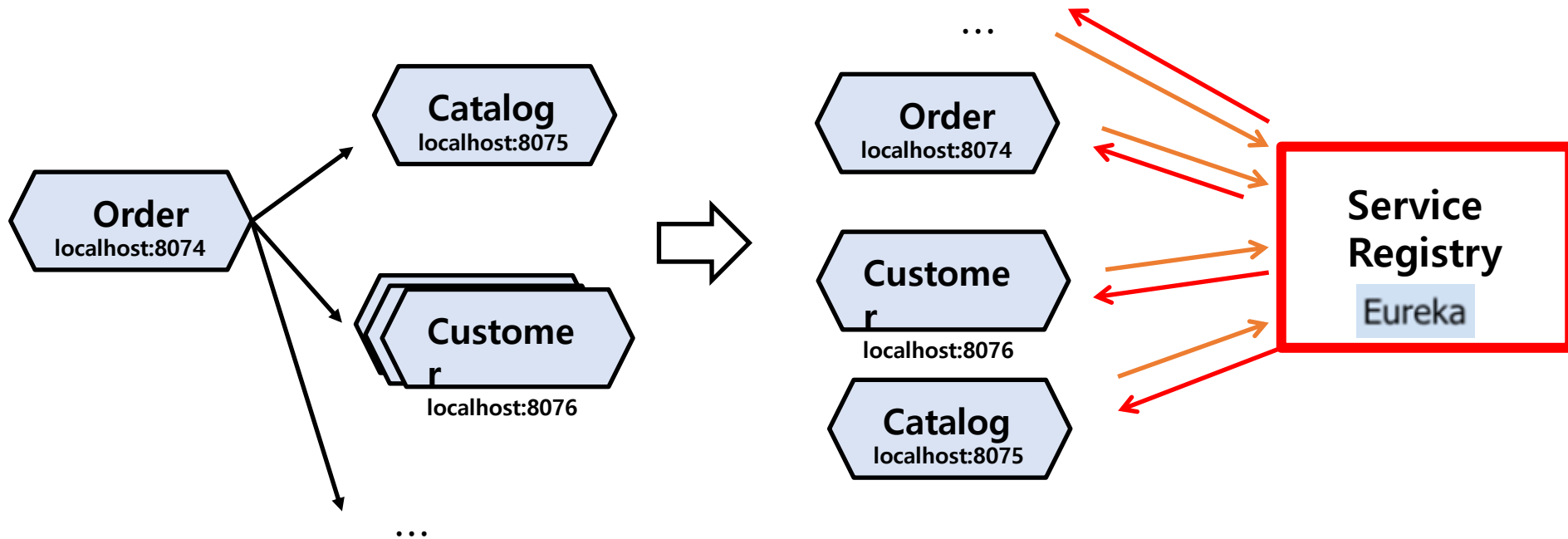
# eureka
#기본값은 30초이나 개발 편의를 위해 1초로 설정
eureka.server.response-cache-update-interval-ms: 1000
# 개발모드
eureka.server.enableSelfPreservation: false
# 개발모드
eureka.client.register-with-eureka: false
# 개발모드
eureka.client.fetch-registry: false
#기본값 설정
eureka.client.service-url.defaultZone: http://${EUREKA}:8761/eureka/,http://${EUREKA}:8762/eureka/
# 각 서버별 접근을 IP로 하겠다는 의미
eureka.instance.prefer-ip-address: true

# env
EUREKA=localhost
```

## 4. Eureka 적용 시나리오

### 시나리오

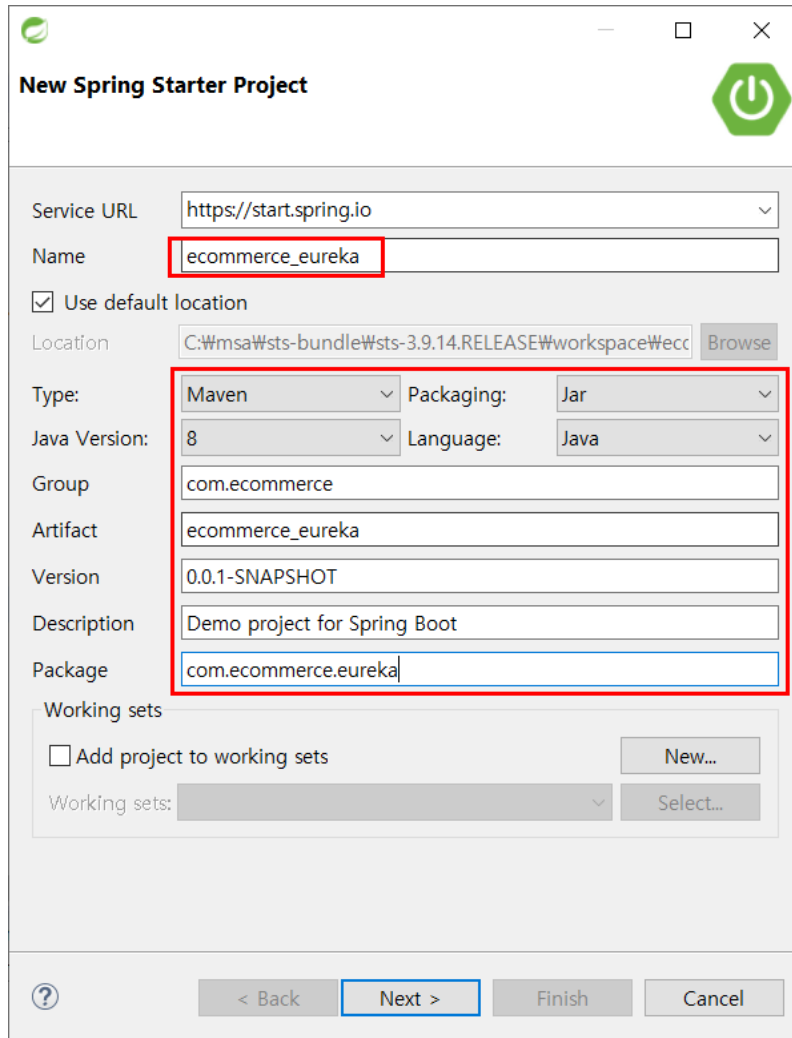
Order 서비스의 주문등록 작업시 동적으로 다중화된 Customer서비스 및 Catalogs 서비스의 위치정보(host, port)를 인식하기가 쉽지 않다.  
따라서 동적으로 다중화된 서비스 정보를 Eureka 서버에 등록하고 필요할 때 Eureka 서버에서 참조하여 사용한다.





## 5. EurekaServer 설정

### 가. Eureka 서버용 마이크로서비스 생성



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

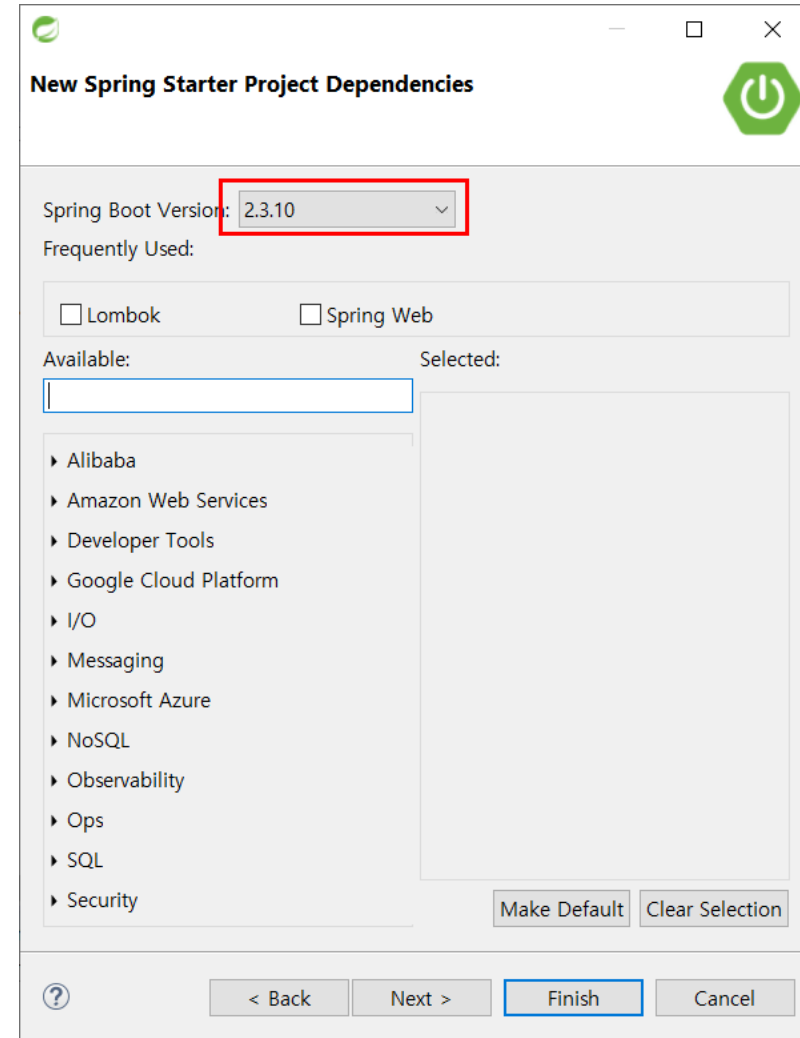
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



**New Spring Starter Project Dependencies**

Spring Boot Version:

Frequently Used:

☐ Lombok ☐ Spring Web

Available:

Selected:

- Alibaba
- Amazon Web Services
- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- SQL
- Security

## 5. EurekaServer 설정

### 나. Eureka 서비스에서 라이브러리 등록

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
<dependency>
```

### 다. Application.java 에서 @EnableEurekaServer 추가

```
@SpringBootApplication
@EnableEurekaServer
public class EcommerceEurekaApplication {

    public static void main(String[] args) {
        SpringApplication.run(EcommerceEurekaApplication.class, args);
    }
}
```

## 5. EurekaServer 설정

### 라. application.properties 설정

```
application.properties
1
2server.port=8761
3
4spring.application.name=ecommerce-eureka
5
6# eureka
7eureka.client.register-with-eureka=false
8eureka.client.fetch-registry=false
9eureka.client.service-url.defaultZone=http://${EUREKA}:8761/eureka
10eureka.instance.prefer-ip-address=true
11eureka.server.enable-self-preservation=false
12
13# env
14EUREKA=localhost
```

eureka.server.response-cache-update-interval-ms  
cache된 registry 정보를 반환. Cache 업데이트 주기의 기본값은 30초.

eureka.server.enable-self-preservation  
eureka server 자기 보존 모드 설정

eureka.client.register-with-eureka  
자신의 서비스 정보 eureka 등록 여부

eureka.client.fetch-registry  
eureka의 서비스 목록 정보를 local에 caching하는 설정

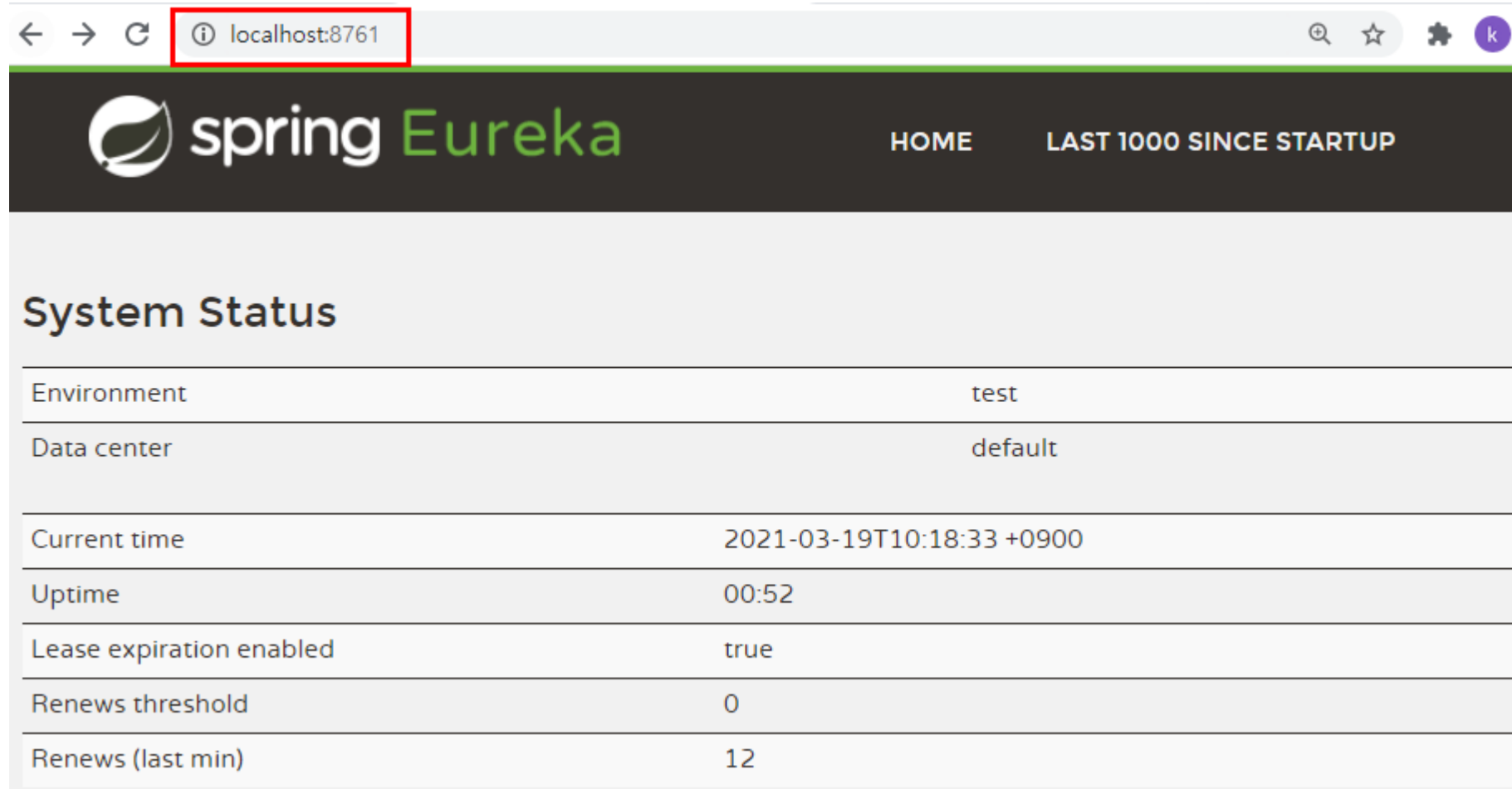
eureka.client.service-url.defaultZone  
client 서비스가 등록된 eureka server 주소

eureka.instance.prefer-ip-address  
등록시 ip 주소 우선 설정. 기본적으로 instance는 host name으로 등록된다.  
그러나 DNS가 없는 경우에는 host 파일에 ip를 등록하지 않으면 instance를 찾지 못한다.

## 5. EurekaServer 설정

마. eureka 서버 시작후 관리 페이지에 접속

http://localhost:8761



Environment	test
Data center	default
Current time	2021-03-19T10:18:33 +0900
Uptime	00:52
Lease expiration enabled	true
Renews threshold	0
Renews (last min)	12

## 6. EurekaClient 설정 – Order 서비스

### 1) EurekaClient 설정 라이브러리 적용

#### 가. 클라이언트 서비스에서 라이브러리 등록

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

#### 나. 클라이언트 Application.java 에서 @EnableEurekaClient 추가

```
@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
@EnableEurekaClient
public class EcommerceOrderApplication {

    public static void main(String[] args) {
        SpringApplication.run(EcommerceOrderApplication.class, args);
    }
}
```

## 6. EurekaClient 설정 – Order 서비스

### 다. application.properties 수정 및 추가

```
# ribbon 설정
ecommerce-customer.ribbon.eureka.enabled=false
ecommerce-customer.ribbon.listOfServers=localhost:8076,localhost:9076
```



```
ecommerce-customer.ribbon.eureka.enabled=true
# eureka
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone=http://${EUREKA}:8761/eureka
eureka.instance.prefer-ip-address=true
```

### 라. Feign Client 사용하는 서비스 수정 (논리적 이름으로 서비스 접근 가능)

```
@FeignClient(name="ecommerce-customer",
             //url = "http://localhost:8076/ecommerce/customer",
             fallbackFactory = CustomerFeignClientFallbackFactory.class
)
public interface CustomerFeignClient {
    @GetMapping("/ecommerce/customer/rest/customers/{userid}")
    public Customer retrieveCustomer(@PathVariable String userid) throws Exception;
}
```

## 6. EurekaClient 설정 – Order 서비스

마. 클라이언트 서버 시작 후 관리 페이지에 접속

<http://localhost:8761>

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
ECOMMERCE-CATALOGS	n/a (1)	(1)	UP (1) - DESKTOP-6AC8BA3:ecommerce-catalogs:8075
ECOMMERCE-CUSTOMER	n/a (1)	(1)	UP (1) - DESKTOP-6AC8BA3:ecommerce-customer:8076
ECOMMERCE-ORDER	n/a (1)	(1)	UP (1) - DESKTOP-6AC8BA3:ecommerce-order:8074

# 13장. API Gateway



API Gateway 개요 및 기능

API Gateway 구현방법

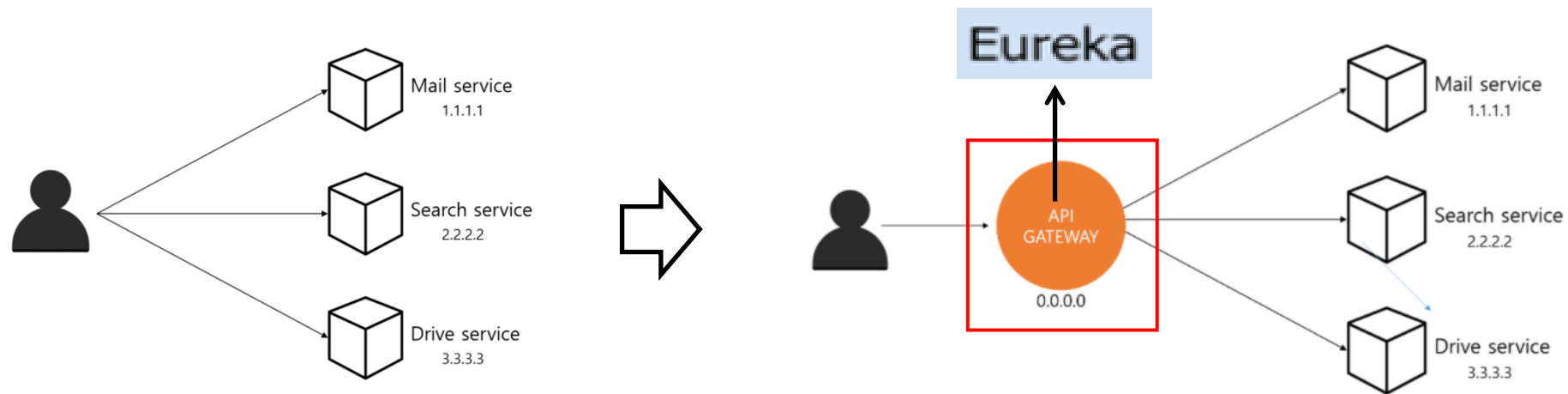
Spring Cloud Gateway (SCG)구축 예



# 1. API Gateway 개요

MSA는 큰 서비스를 잘게 쪼개어 개발/운영하는 방식이다. 하나의 큰 서비스는 수십 ~ 수백 개의 작은 서비스로 나뉘어지며, 만약 이를 클라이언트에서 서비스를 개별적으로 직접 호출하는 형태라면 다음과 같은 문제점이 발생할 수 있다.

- 각각의 서비스마다 인증/인가 등 공통된 로직을 구현해야 된다.
- 수 많은 API 호출을 기록하고 관리하기 어렵다.
- 내부의 비즈니스 로직이 드러나게 되어 보안에 취약해진다.



API Gateway의 scale-out 적용이 유연하지 않을 경우, 병목지점이 되어 성능이 저하되거나 추가적인 계층이 만들어지는 것이기 때문에 그만큼 네트워크 latency가 증가하게 된다.

## 2. API Gateway 구현방법

### 1. API Gateway 의존성 종류

가. spring-cloud-starter-gateway (SCG 방식) - 권장

나. spring-cloud-starter-netflix-zuul

### 2. SCG 와 zuul 차이점

가. Zuul

Spring Cloud의 초창기 버전으로서 Netflix OSS(Open Source Software)에 포함된 컴포넌트이다. 서블릿 프레임워크 기반으로 만들어졌기 때문에 동기(synchronization), 블로킹(Blocking)방식으로 서비스를 처리한다.  
(이후의 Zuul2는 비동기와 논블로킹 지원)

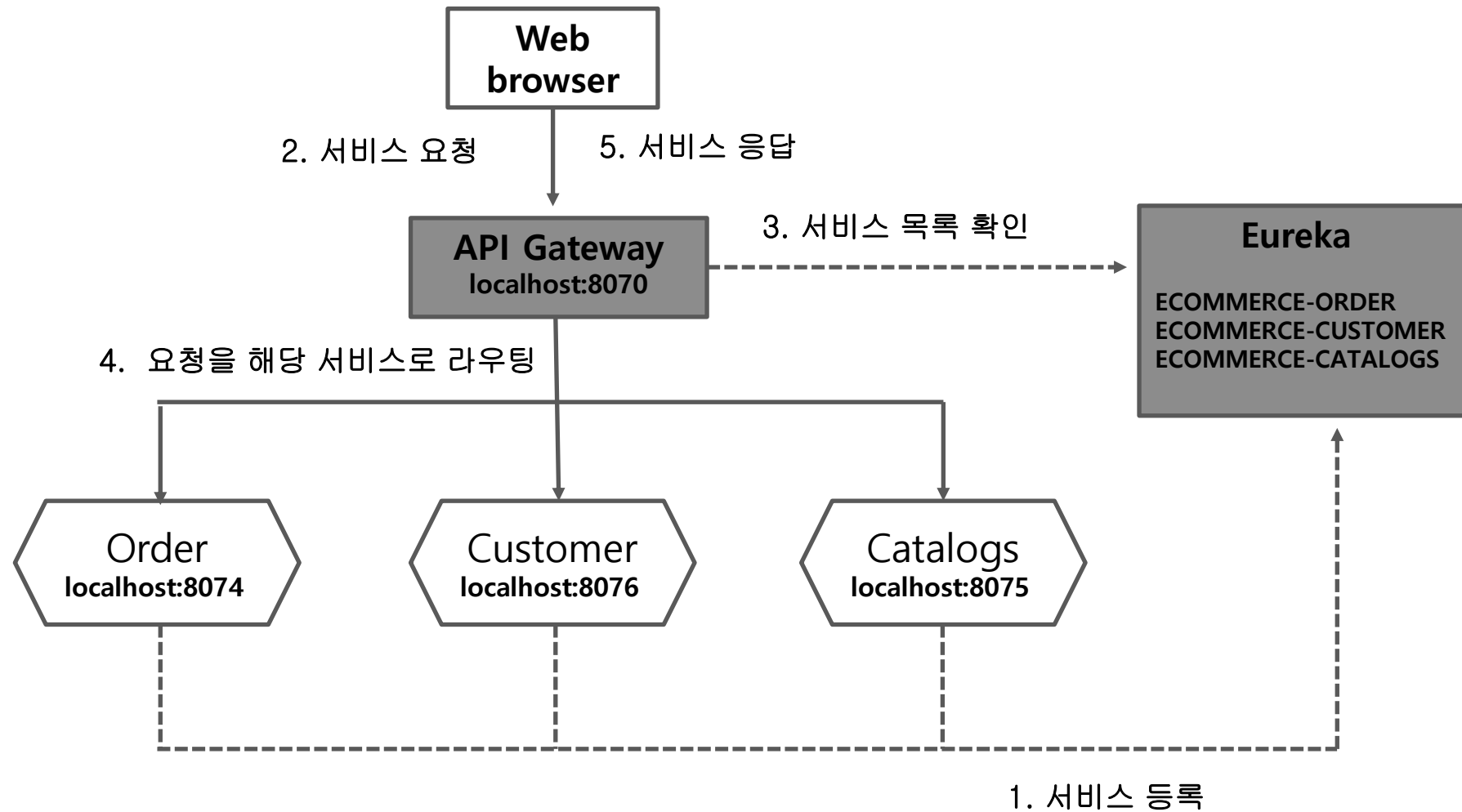
나. SCG

비동기와 논블로킹 지원하며 Spring기반으로 만들어졌기 때문에 Spring 생태계의 여러 프로젝트와 호환이 잘 되며 성능도 더 좋다는 의견도 있다.

하지만 Netty런타임 기반으로 동작하기 때문에 서블릿 컨테이너나 WAR로 빌드된 경우 동작 안될 수도 있다.

### 3. SCG 구축

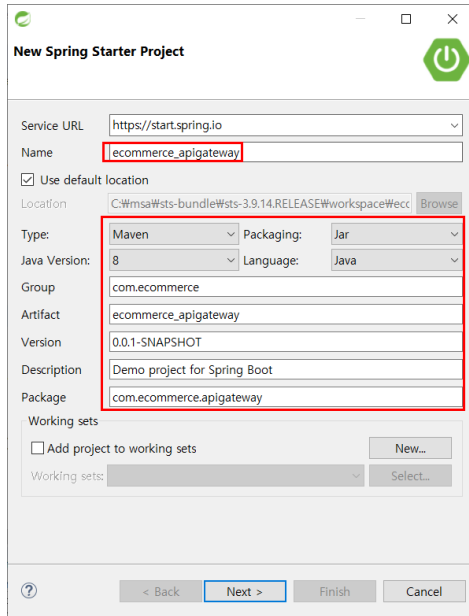
#### 아키텍처



### 3. SCG 구축

## spring-cloud-starter-gateway 라이브러리 적용

#### 가. Gateway 서비스 프로젝트 생성



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

#### 나. Gateway 서비스에서 라이브러리 등록

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>2.2.8.RELEASE</version>
</dependency>
```

### 3. SCG 구축

#### 다. application.yml 설정

```
1 server:
2   port: 8070
3
4 spring:
5   application:
6     name: ecommerce-apigateway
7   cloud:
8     gateway:
9       routes:
10        - id: ecommerce-CUSTOMER
11          uri: lb://ECOMMERCE-CUSTOMER
12          predicates:
13            - Path=/ecommerce/customer/**
14        - id: ecommerce-catalogs
15          uri: lb://ECOMMERCE-CATALOGS
16          predicates:
17            - Path=/ecommerce/catalogs/**
18        - id: ecommerce-order
19          uri: lb://ECOMMERCE-ORDER
20          predicates:
21            - Path=/ecommerce/order/**
22 # eureka
23 eureka:
24   client:
25     register-with-eureka: true
26     fetch-registry: true
27     service-url:
28       defaultZone: http://${EUREKA}:8761/eureka
29   instance:
30     prefer-ip-address: true
31 # env
32 EUREKA: localhost
```

라우트 설정은  
자바코드  
또는  
설정파일(yml)에서 할 수 있다.

http://localhost:8070/ecommerce/customer/swagger-ui.html

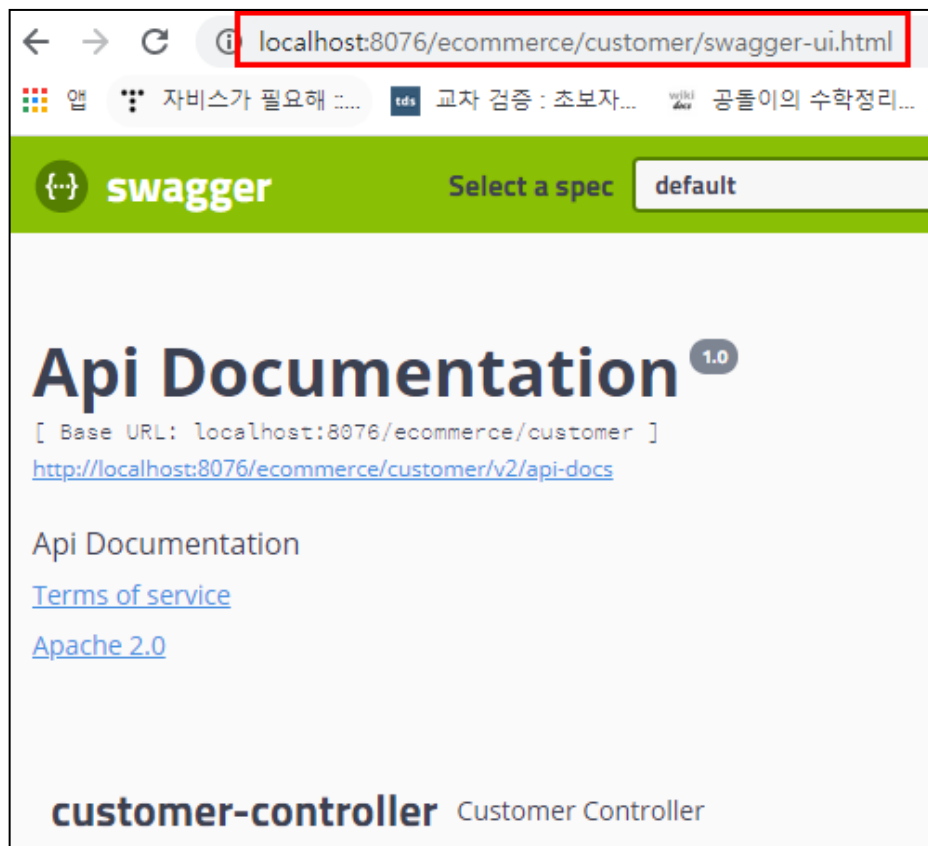
라우팅

http://localhost:8076/ecommerce/customer/swagger-ui.html

### 3. SCG 구축

#### 라. Gateway 서비스 시작 요청

localhost:8076



localhost:8070

