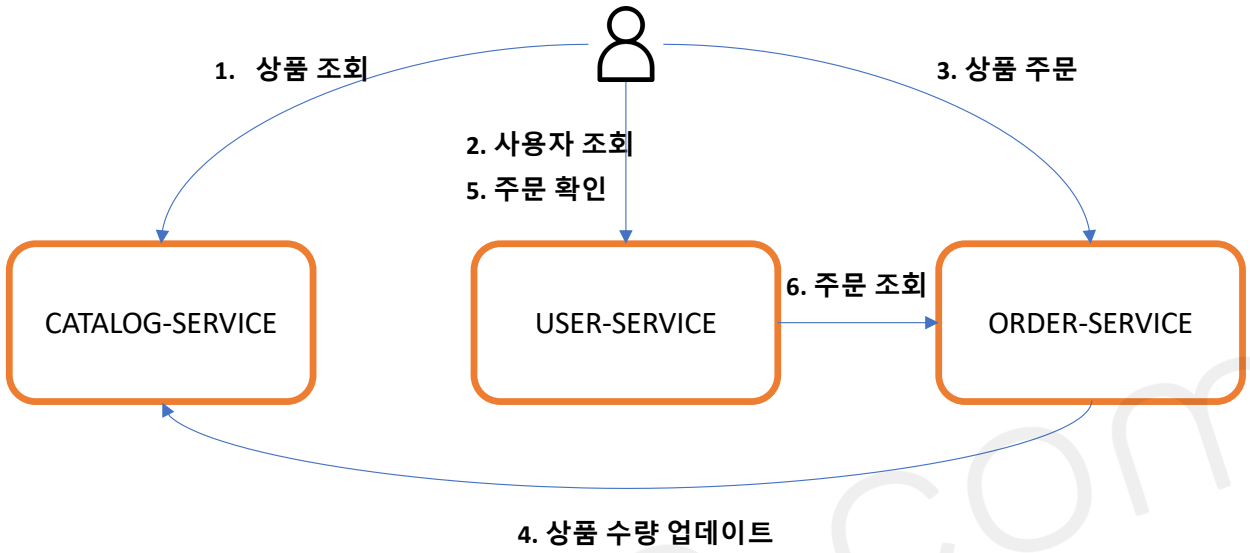
Spring Cloud로 개발하는 마이크로서비스 애플리케이션

Section 3.

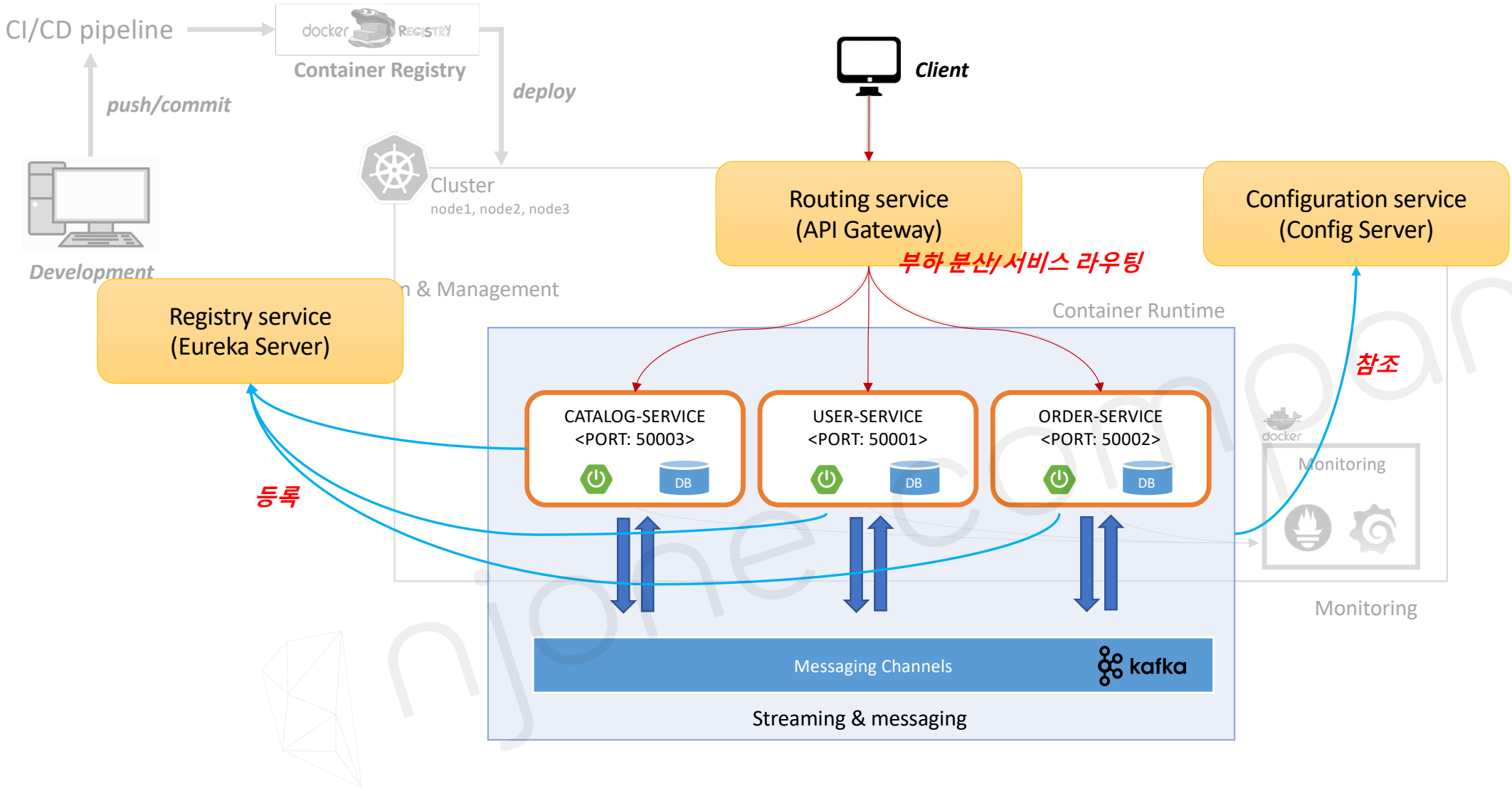
Outer Architecture

- 애플리케이션 개요
- 애플리케이션 구성
- 애플리케이션 APIs
- Service Discovery
- API Gateway

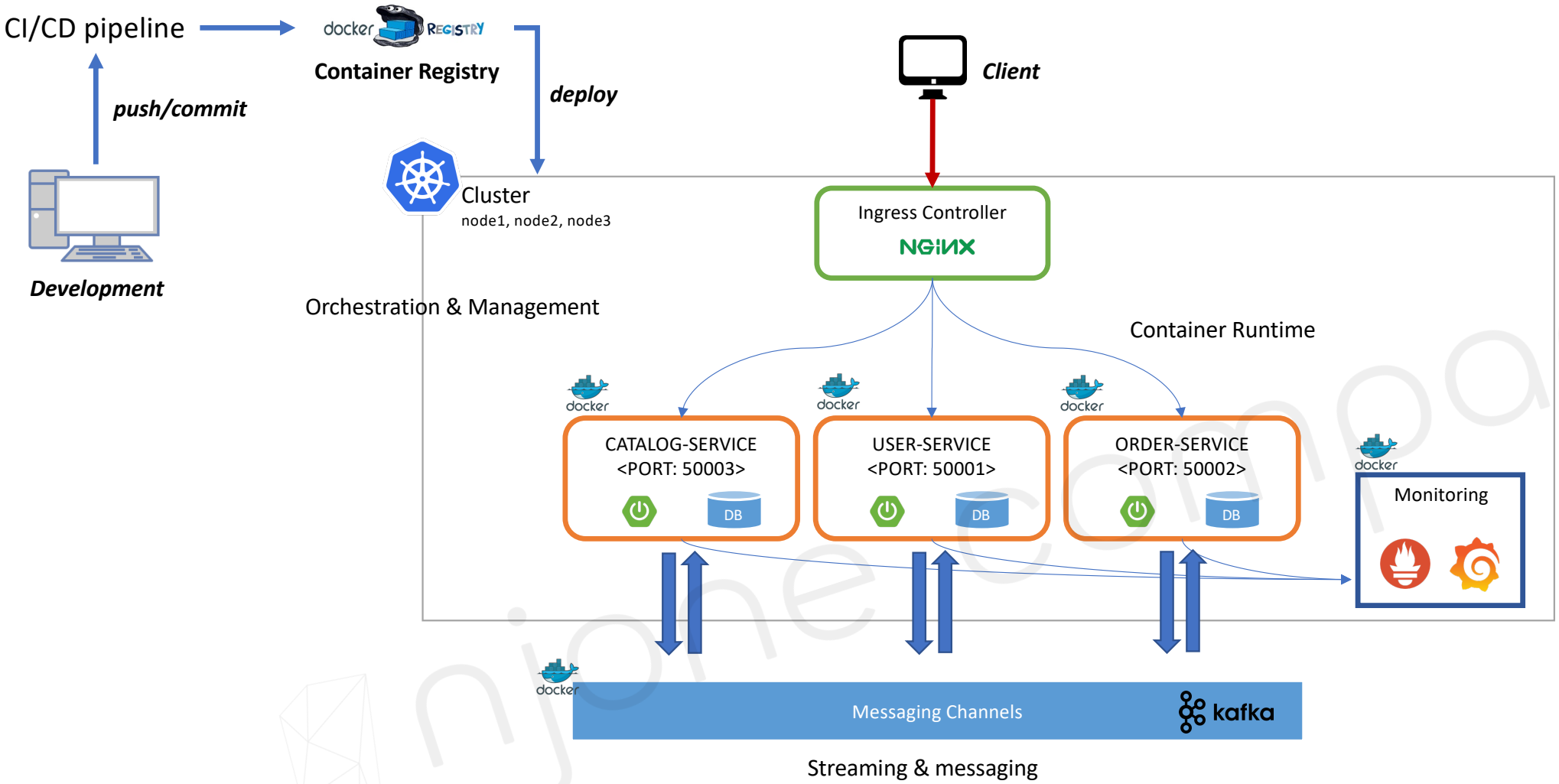
전체 애플리케이션 개요



전체 애플리케이션 구성


CI/CD pipeline
docker REGISTRY
Container Registry
push/commit
deploy
Client
Development
Cluster
node1, node2, node3
Routing service
(API Gateway)
부하 분산/서비스 라우팅
Configuration service
(Config Server)
Registry service
(Eureka Server)
& Management
Container Runtime
CATALOG-SERVICE
<PORT: 50003>
USER-SERVICE
<PORT: 50001>
ORDER-SERVICE
<PORT: 50002>
DB
DB
DB
등록
참조
docker
Monitoring
Monitoring
Messaging Channels
kafka
Streaming & messaging

전체 애플리케이션 구성


CI/CD pipeline
Container Registry
push/commit
deploy
Client
Development
Cluster
node1, node2, node3
Ingress Controller
NGINX
Orchestration & Management
Container Runtime
CATALOG-SERVICE
<PORT: 50003>
USER-SERVICE
<PORT: 50001>
ORDER-SERVICE
<PORT: 50002>
DB
DB
DB
Monitoring
Messaging Channels
kafka
Streaming & messaging

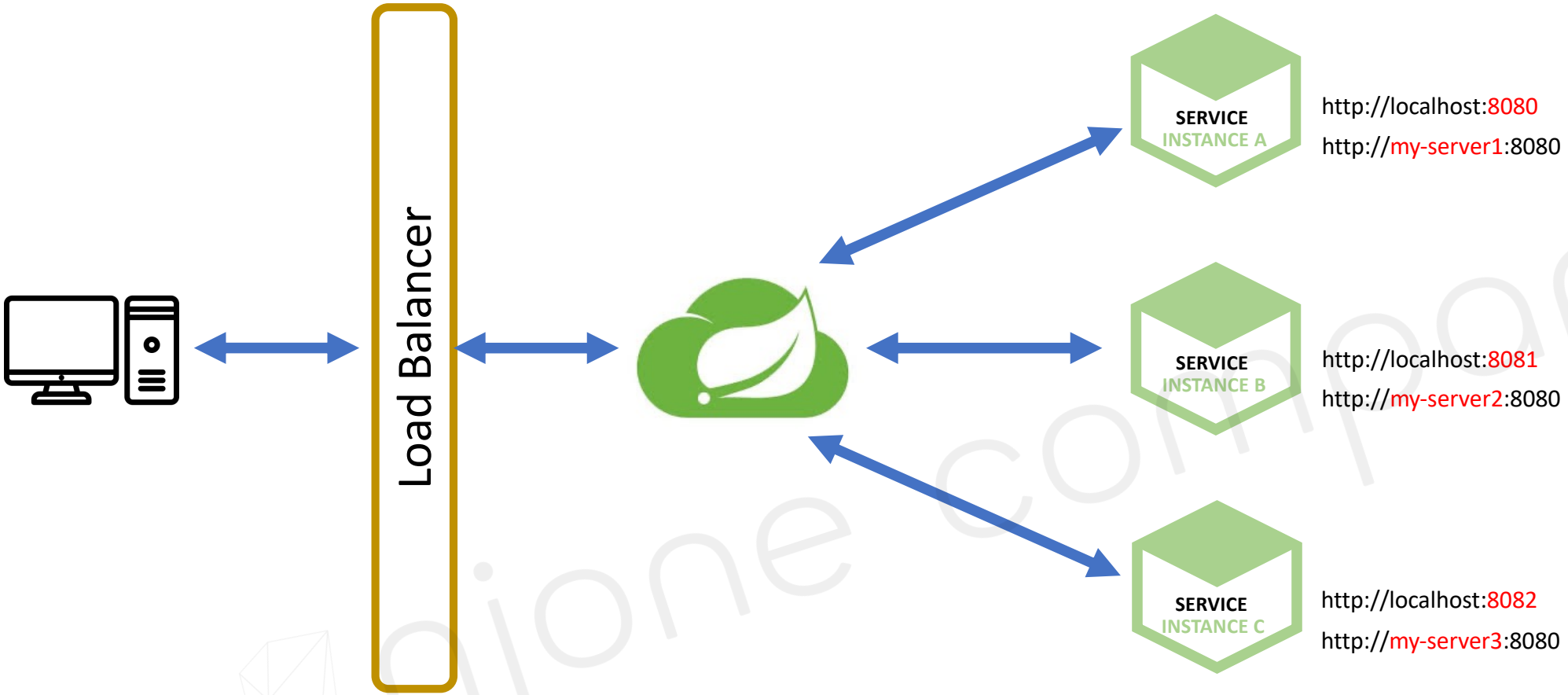
전체 애플리케이션 구성요소

구성요소	설명	Inner / Outer Architecture
Git Repository	마이크로서비스 소스 관리 및 프로파일 관리	-
Config Server	Git 저장소에 등록된 프로파일 정보 및 설정 정보	Outer
Eureka Server	마이크로서비스 등록 및 검색	Outer
API Gateway Server	마이크로서비스 부하 분산 및 서비스 라우팅	Outer
Microservices	회원 MS, 주문 MS, 상품(카테고리) MS	Inner
Queuing System	마이크로서비스 간 메시지 발행 및 구독	Outer

애플리케이션 APIs

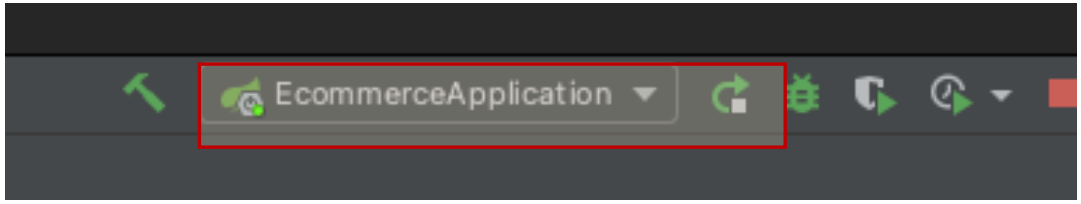
마이크로서비스	RESTful API	HTTP Method
Catalog Service	/catalog-service/catalogs : 상품 목록 제공	GET
User Service	/user-service/users : 사용자 정보 등록 /user-service/users : 전체 사용자 조회 /user-service/users/***{user_id}*** : 사용자 정보, 주문 내역 조회	POST GET GET
Order Service	/order-service/users/***{user_id}***/orders : 주문 등록 /order-service/users/***{user_id}***/orders : 주문 확인	POST GET

Spring Cloud Netflix Eureka



ServiceDiscovery 예제

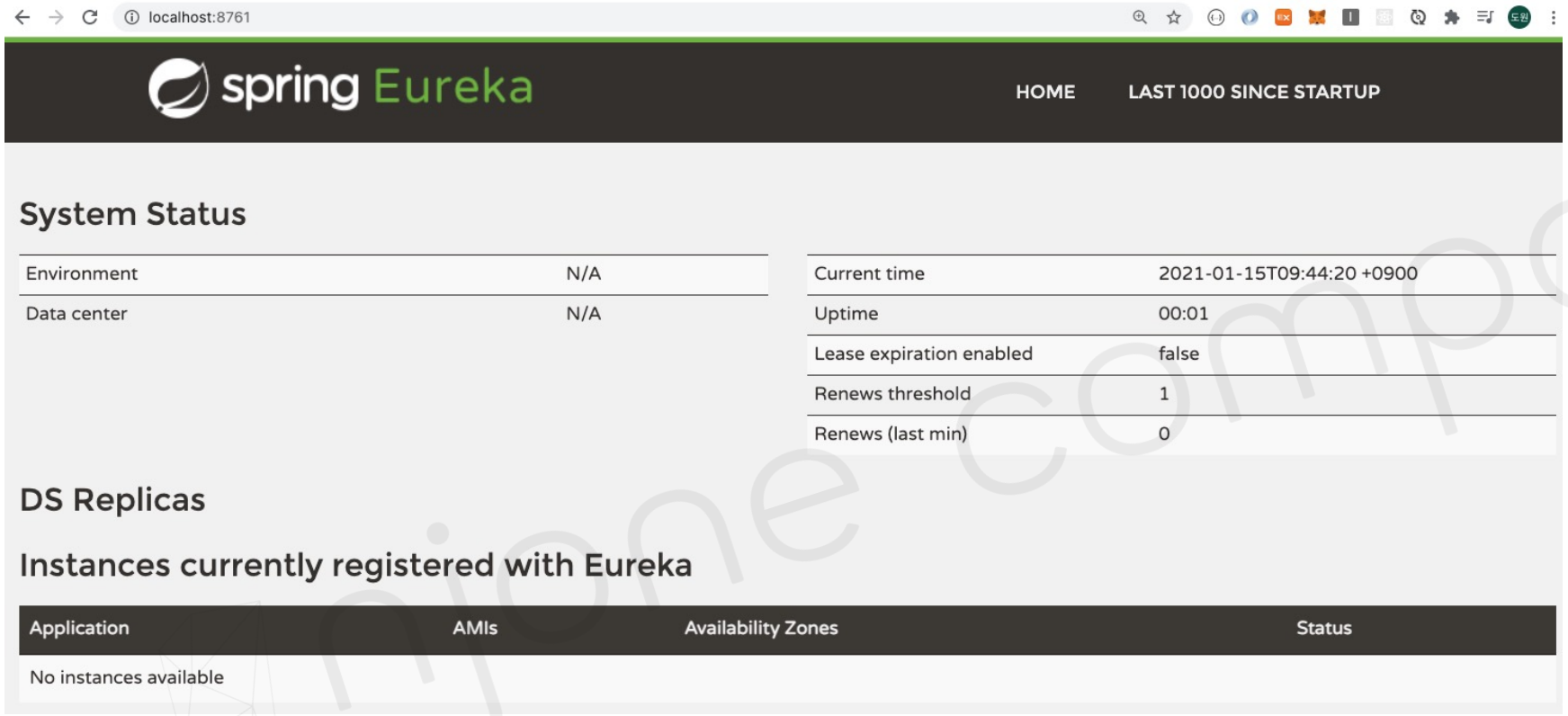
- 실행




```
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8761 (http) with context path ''
.s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
c.e.ecommerce.EcommerceApplication       : Started EcommerceApplication in 3.252 seconds (JVM running for 3.799)
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
```

ServiceDiscovery 예제

- 실행화면
  - http://localhost:8761



Client Service

- application.yml

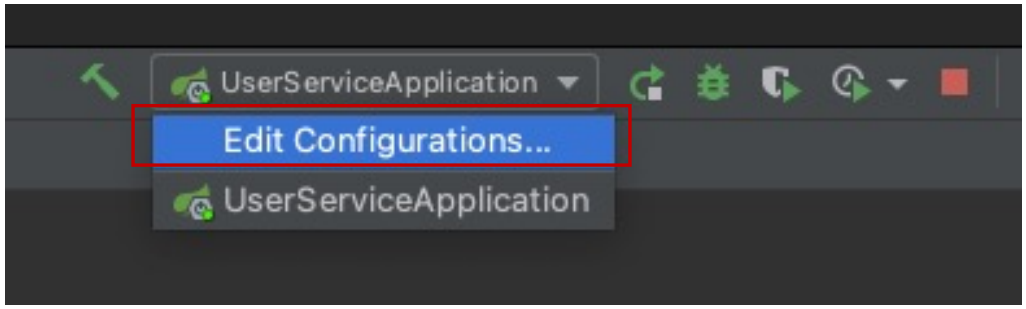
```yaml
server:
  port: 9001

spring:
  application:
    name: user-service

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

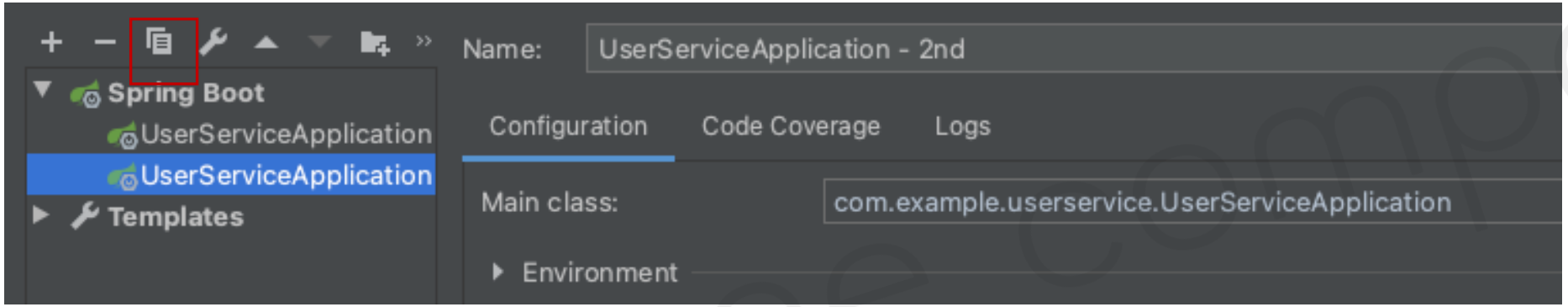
Instances currently registered with Eureka

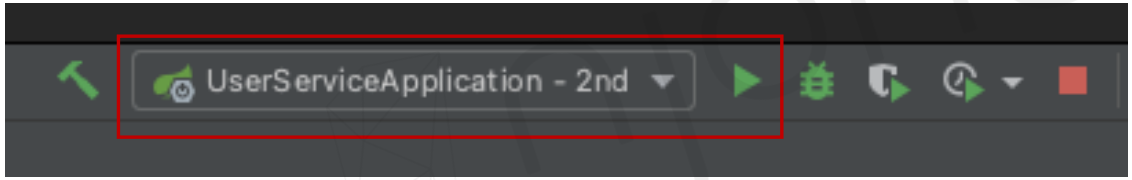
Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (1)	(1)	UP (1) - 10.90.1.91:user-service:9001

Client Service

- Scaling – 같은 서비스 추가 실행 #1







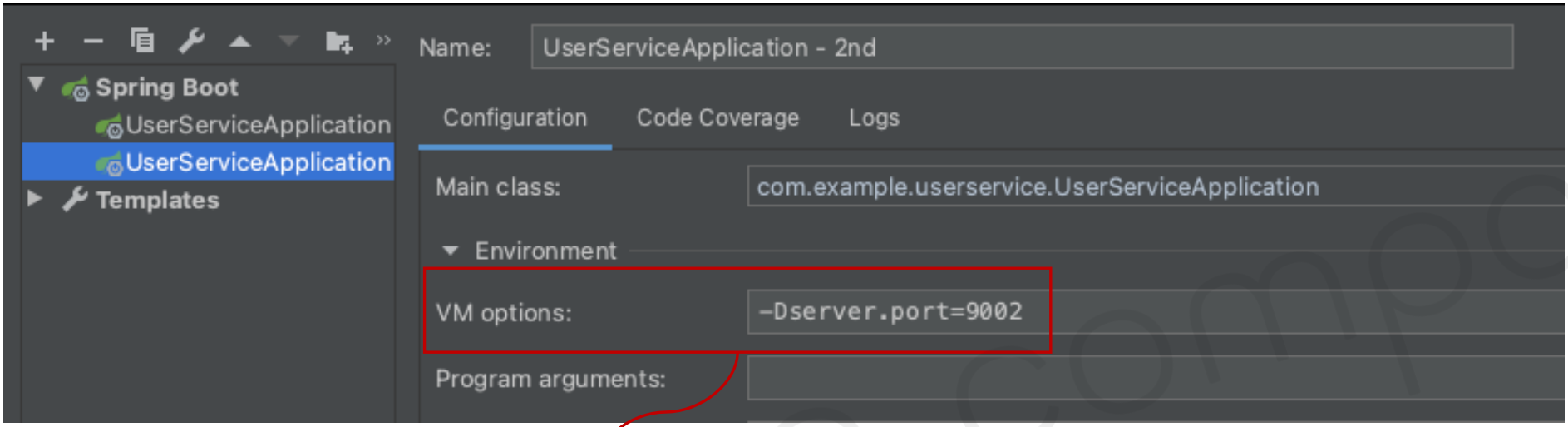
Client Service

- Scaling – 같은 서비스 추가 실행 #1
  - ***Port 충돌***

```
Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2021-01-15 10:54:05.701 ERROR 68628 --- [  restartedMain] o.s.b.d.LoggingFailureAnalysisReporter   :

***************************
APPLICATION FAILED TO START
***************************

Description:

Web server failed to start. Port 9001 was already in use.
```

Client Service

- Scaling – 같은 서비스 추가 실행 #1
  - VM Options → -Dserver.port=[다른포트]




```
: Tomcat started on port(s): 9002 (http) with context path ''
: Updating port to 9002
: DiscoveryClient_USER-SERVICE/10.90.1.91:user-service:9002 - registration
: Started UserServiceApplication in 2.881 seconds (JVM running for 3.455)
```

Client Service

- Scaling – 같은 서비스 추가 실행 #1

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (2)	(2)	UP (2) - 10.90.1.91:user-service:9001 , 10.90.1.91:user-service:9002

- Scaling - 같은 서비스 추가 실행 #2

```
$ mvn spring-boot:run -Dspring-boot.run.jvmArguments='-Dserver.port=9003'
```

- Scaling - 같은 서비스 추가 실행 #3

```
$ mvn clean compile package
$ java -jar -Dserver.port=9004 ./target/user-service-0.0.1-SNAPSHOT.jar
```

Client Service

- Scaling – 같은 서비스 추가 실행

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (4)	(4)	UP (4) - 10.90.1.91:user-service:9003 , 10.90.1.91:user-service:9004 , 10.90.1.91:user-service:9001 , 10.90.1.91:user-service:9002

UP (4) - 10.90.1.91:user-service:9003 , 10.90.1.91:user-service:9004 , 10.90.1.91:user-service:9001 , 10.90.1.91:user-service:9002

Client Service

- Scaling – Random 포트 사용으로 같은 서비스 추가 실행
  - application.yml

```yaml
server:
  port: 0

spring:
  application:
    name: user-service
```

```
Spring Cloud LoadBalancer is currently working with the default
Tomcat started on port(s): 55002 (http) with context path ''
Updating port to 55002
```

$ mvn spring-boot:run

```
Spring Cloud LoadBalancer is currently working with the default

Tomcat started on port(s): 55019 (http) with context path ''
Updating port to 55019
```

Client Service

- Scaling – Random 포트 사용으로 같은 서비스 추가 실행
  - 같은 서비스 명으로 인해 Eureka에 하나의 앱만 등록

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (1)	(1)	UP (1) - 10.90.1.91:user-service:0

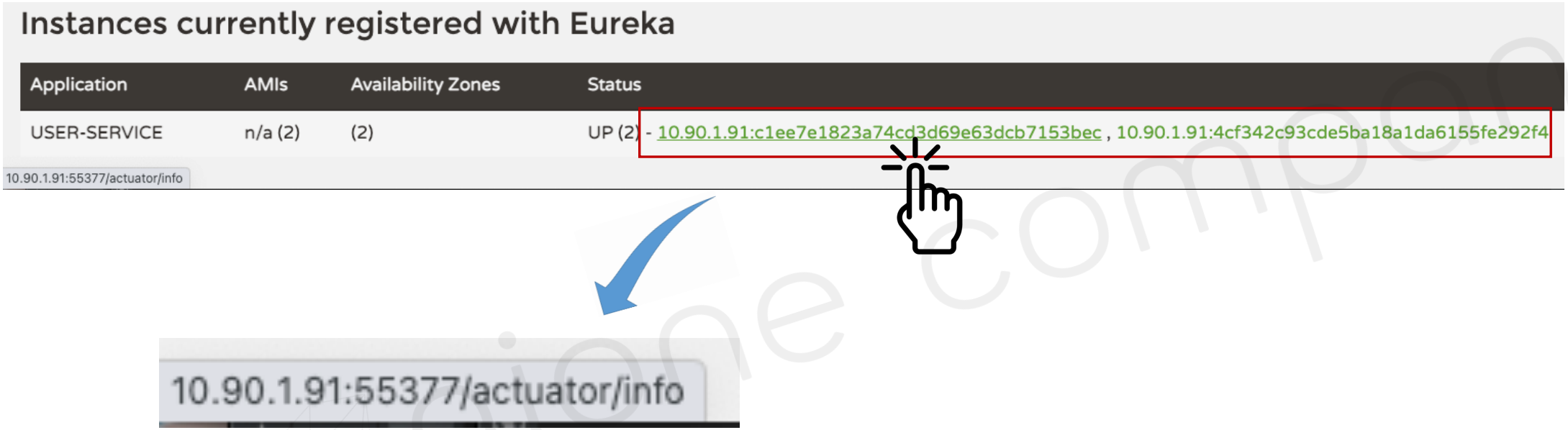
Client Service

- Scaling – Random 포트 사용으로 같은 서비스 추가 실행
  - application.yml에 instance 정보 등록
    - ***eureka.instance.instanceId***

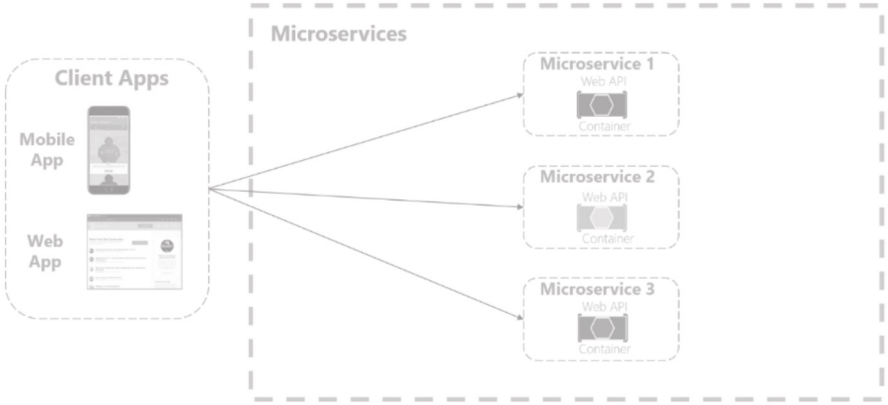
```yaml
server:
  port: 0

spring:
  application:
    name: user-service


eureka:
  instance:
    instanceId: ${spring.cloud.client.hostname}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
```

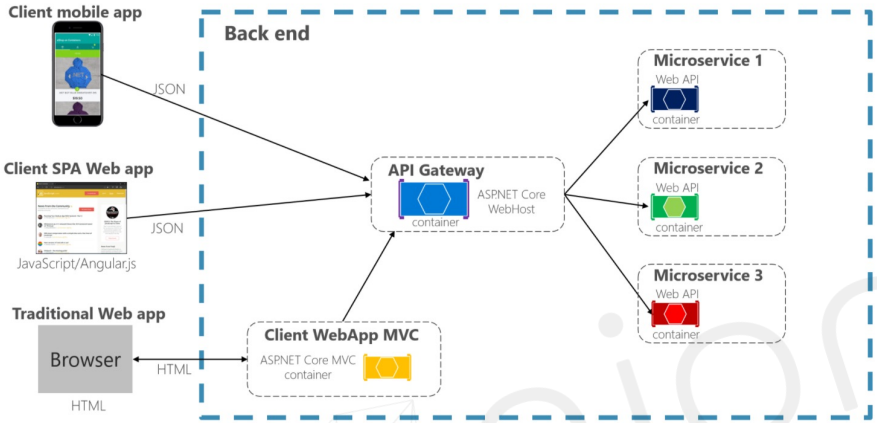
Client Service

- Scaling – Random 포트 사용으로 같은 서비스 추가 실행
  - application.yml에 instance 정보 등록
    - ***eureka.instance.instanceId***



API Gateway Service



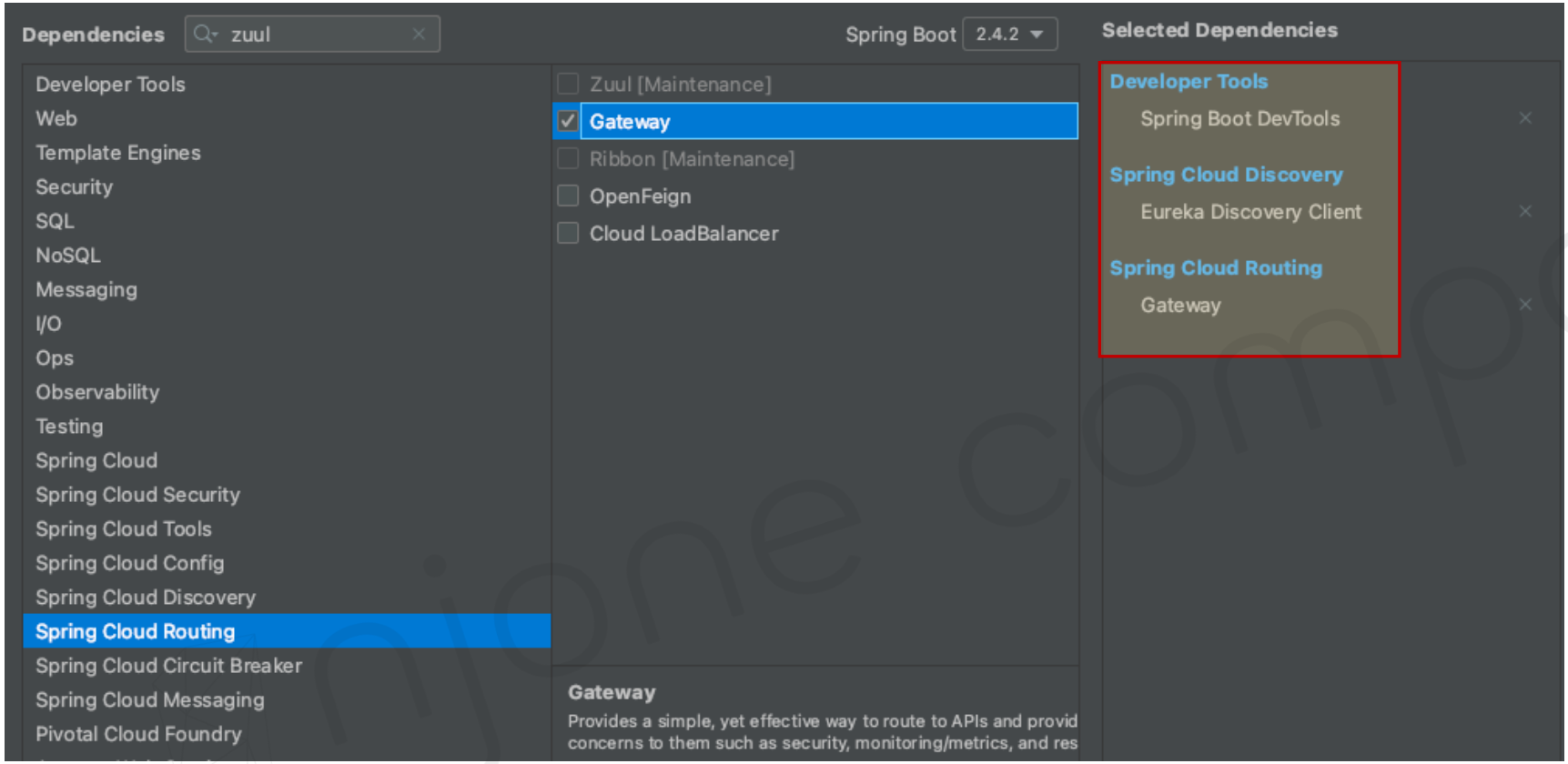


- 인증 및 권한 부여
- 서비스 검색 통합
- 응답 캐싱
- 정책, 회로 차단기 및 QoS 다시 시도
- 속도 제한
- 부하 분산
- 로깅, 추적, 상관 관계
- 헤더, 쿼리 문자열 및 청구 변환
- IP 허용 목록에 추가

참조: https://docs.microsoft.com/ko-kr/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

Spring Cloud Gateway – 기본

- Step1) Dependencies
  - DevTools, Eureka Discovery Client, ***Gateway***



Spring Cloud Gateway – 기본

- Step2) application.properties (or application.yml)

```yaml
server:
  port: 8000

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://localhost:8761/eureka
```

```yaml
spring:
  application:
    name: gateway-service
  cloud:
    gateway:
      routes:
        - id: first-service
          uri: http://localhost:8081/
          predicates:
            - Path=/first-service/**
        - id: second-service
          uri: http://localhost:8082/
          predicates:
            - Path=/second-service/**
```

Spring Cloud Gateway – 기본

- Step3) Test

```
@RestController
@RequestMapping("/first-service")
public class FirstServiceController {
```

127.0.0.1:8000/first-service/welcome

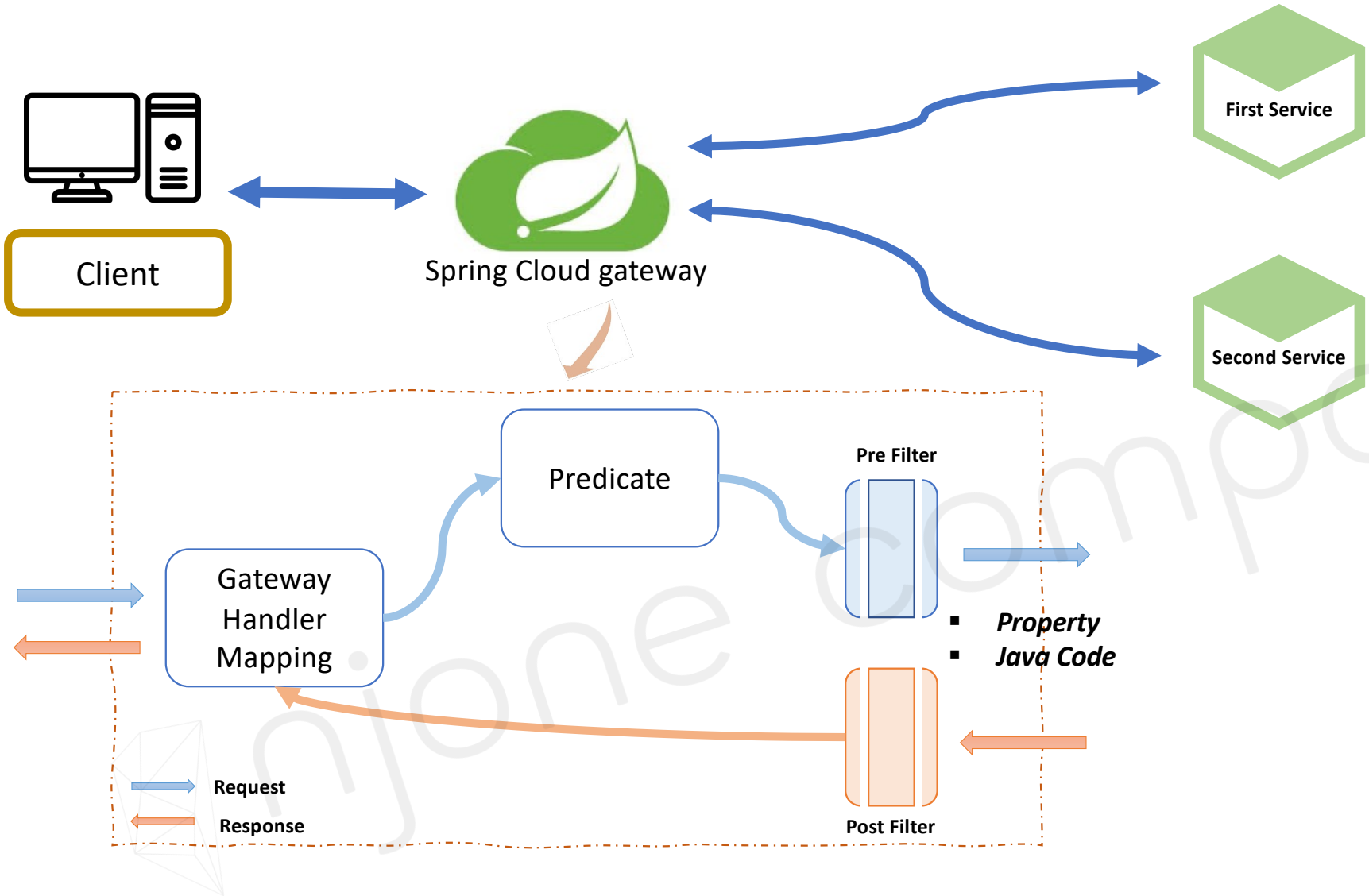
Welcome to the First service.

```
@RestController
@RequestMapping("/second-service")
public class SecondServiceController {
```

127.0.0.1:8000/second-service/welcome

Welcome to the Second service.

Spring Cloud Gateway – Filter



Spring Cloud Gateway – Filter

- Step4) Filter using Java Code – FilterConfig.java

```java
@Configuration
public class FilterConfig {
    @Bean
    public RouteLocator gatewayRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(r -> r.path( ...patterns: "/first-service/**")
                        .filters(f -> f.addRequestHeader( headerName: "first-request",  headerValue: "first-request-header")
                                .addResponseHeader( headerName: "first-response",  headerValue: "first-response-header"))
                        .uri("http://localhost:8081/"))
                .route(r -> r.path( ...patterns: "/second-service/**")
                        .filters(f -> f.addRequestHeader( headerName: "second-request",  headerValue: "second-request-header")
                                .addResponseHeader( headerName: "second-response",  headerValue: "second-response-header"))
                        .uri("http://localhost:8082/"))
                .build();
    }
}
```

Spring Cloud Gateway – Filter

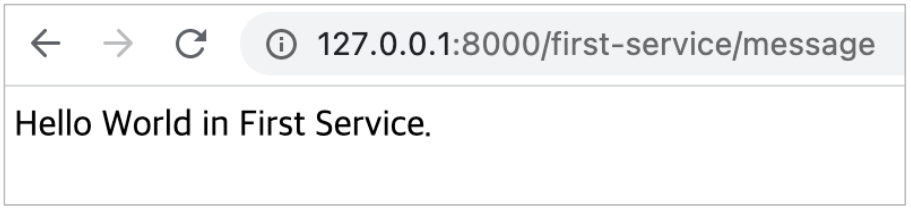
- Step4) Filter using Java Code – FirstServiceController.java, SecondServiceController.java

```java
@RestController
@RequestMapping("/first-service")
public class FirstServiceController {

    @GetMapping("/welcome")
    public String welcome() { return "Welcome to the First service."; }

    @GetMapping("/message")
    public String message(@RequestHeader("first-request") String header) {
        System.out.println(header);
        return "Hello World in First Service.";
    }
}
```

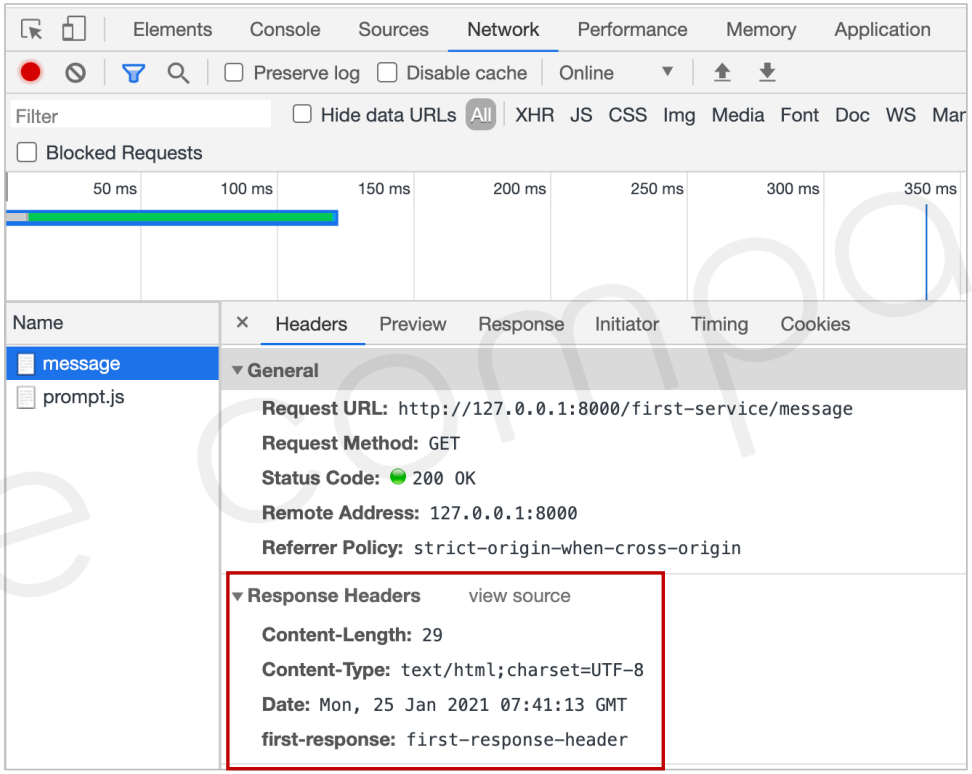
```java
@RestController
@RequestMapping("/second-service")
public class SecondServiceController {

    @GetMapping("/welcome")
    public String welcome() { return "Welcome to the Second service."; }

    @GetMapping("/message")
    public String message(@RequestHeader("second-request") String header) {
        System.out.println(header);
        return "Hello World in Second Service.";
    }
}
```

Spring Cloud Gateway – Filter

- Step4) Test




```
2021-01-25 16:41:13.638  INFO 52232 --- [nio-8081-exec-1] o.s.
2021-01-25 16:41:13.645  INFO 52232 --- [nio-8081-exec-1] o.s.
first-request-header
```



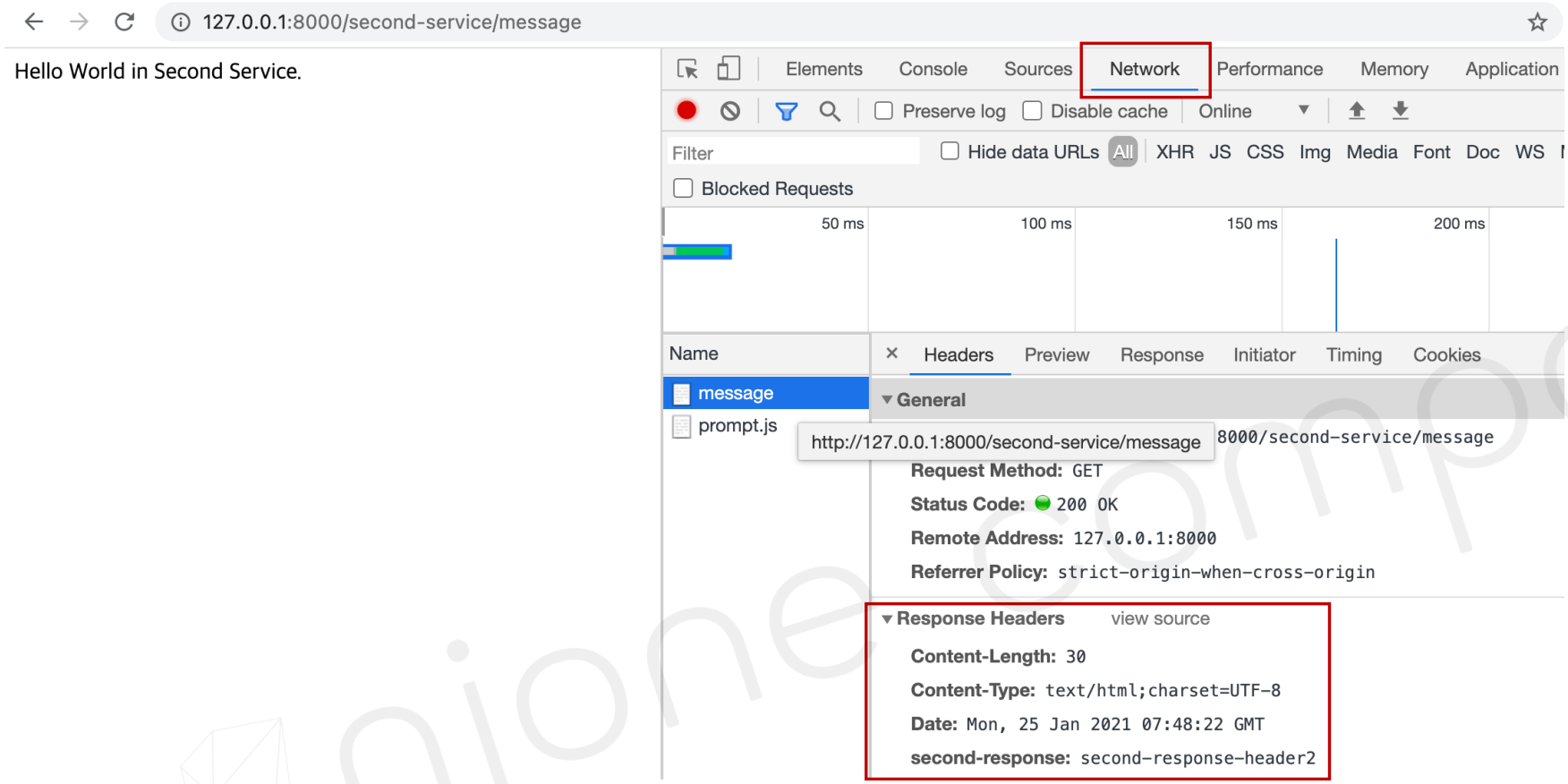
Spring Cloud Gateway – Filter

- Step5) Filter using Property – application.yml

```yaml
routes:
  - id: first-service
    uri: http://localhost:8081/
    predicates:
      - Path=/first-service/**
    filters:
      - AddRequestHeader=first-request, first-request-header2
      - AddResponseHeader=first-response, first-response-header2
  - id: second-service
    uri: http://localhost:8082/
    predicates:
      - Path=/second-service/**
    filters:
      - AddRequestHeader=second-request, second-request-header2
      - AddResponseHeader=second-response, second-response-header2
```

Spring Cloud Gateway – Filter

- Step5) Test



Spring Cloud Gateway – Custom Filter

- Step6) Custom Filter – CustomFilter.java

```java
@Component
@Slf4j
public class CustomFilter extends AbstractGatewayFilterFactory<CustomFilter.Config> {
    public CustomFilter() {
        super(Config.class);
    }

    @Override
    public GatewayFilter apply(Config config) {
        // Custom Pre Filter. Suppose we can extract JWT and perform Authentication
        return (exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest();
            ServerHttpResponse response = exchange.getResponse();

            log.info("Custom PRE filter: request uri -> {}", request.getId());
            // Custom Post Filter.Suppose we can call error response handler based on error code.
            return chain.filter(exchange).then(Mono.fromRunnable(() -> {
                log.info("Custom POST filter: response code -> {}", response.getStatusCode());
            }));
        };
    }
```

Spring Cloud Gateway – Custom Filter

- Step6) Custom Filter – application.yml

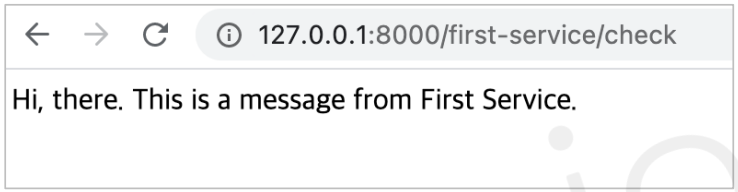
```yaml
      routes:
        - id: first-service
          uri: http://localhost:8081/
          predicates:
            - Path=/first-service/**
          filters:
#           - AddRequestHeader=first-request, first-request-header2
#           - AddResponseHeader=first-response, first-response-header2
            - CustomFilter
        - id: second-service
          uri: http://localhost:8082/
          predicates:
            - Path=/second-service/**
          filters:
#           - AddRequestHeader=second-request, second-request-header2
#           - AddResponseHeader=second-response, second-response-header2
            - CustomFilter
```

Spring Cloud Gateway – Custom Filter

- Step6) Custom Filter – FirstServiceController.java, SecondServiceController.java

```java
@GetMapping("/check")
public String check() {
    return "Hi, there. This is a message from First Service.";
}
```

```java
@GetMapping("/check")
public String check() {
    return "Hi, there. This is a message from Second Service.";
}
```




```
INFO 67665 --- [ctor-http-nio-2] c.e.a.filter.CustomFilter                : Custom PRE filter: request id -> 406df49b-1
INFO 67665 --- [ctor-http-nio-5] c.e.a.filter.CustomFilter                : Custom POST filter: response code -> 200 OK
```

Spring Cloud Gateway – Global Filter

- Step7) Global Filter – GlobalFilter.java

```java
@Component
@Slf4j
public class GlobalFilter extends AbstractGatewayFilterFactory<GlobalFilter.Config> {
    public GlobalFilter() { super(Config.class); }

    @Override
    public GatewayFilter apply(Config config) {
        return ((exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest();
            ServerHttpResponse response = exchange.getResponse();

            log.info("Global Filter baseMessage: {}", config.getBaseMessage());
            if (config.isPreLogger()) {
                log.info("Global Filter Start: request id -> {}", request.getId());
            }
            return chain.filter(exchange).then(Mono.fromRunnable(()->{
                if (config.isPostLogger()) {
                    log.info("Global Filter End: response code -> {}", response.getStatusCode());
                }
            }));
        });
    }
```

```java
    @Data
    public static class Config {
        private String baseMessage;
        private boolean preLogger;
        private boolean postLogger;
    }
```

Spring Cloud Gateway – Global Filter

- Step7) Global Filter – application.yml

```yaml
spring:
  application:
    name: gateway-service
  cloud:
    gateway:
      default-filters:
        - name: GlobalFilter
          args:
            baseMessage: Spring Cloud Gateway GlobalFilter
            preLogger: true
            postLogger: true
      routes:
        - id: first-service
          uri: http://localhost:8081/
          predicates:
            - Path=/first-service/**
          filters:
#           - AddRequestHeader=first-request, first-request-header2
#           - AddResponseHeader=first-response, first-response-header2
            - CustomFilter
```

Spring Cloud Gateway – Global Filter

- Step7) Global Filter – Test

```
INFO 67815 --- [ctor-http-nio-2] c.e.a.filter.GlobalFilter          : Global Filter baseMessage: Spring Cloud Gateway GlobalFilter
INFO 67815 --- [ctor-http-nio-2] c.e.a.filter.GlobalFilter          : Global Filter Start: request id -> e5494608-1
INFO 67815 --- [ctor-http-nio-2] c.e.a.filter.CustomFilter          : Custom PRE filter: request id -> e5494608-1
INFO 67815 --- [ctor-http-nio-5] c.e.a.filter.CustomFilter          : Custom POST filter: response code -> 200 OK
INFO 67815 --- [ctor-http-nio-5] c.e.a.filter.GlobalFilter          : Global Filter End: response code -> 200 OK
```

```
log.info("Global Filter baseMessage: {}", config.getBaseMessage());
if (config.isPreLogger()) {
    log.info("Global Filter Start: request id -> {}", request.getId());
}
```

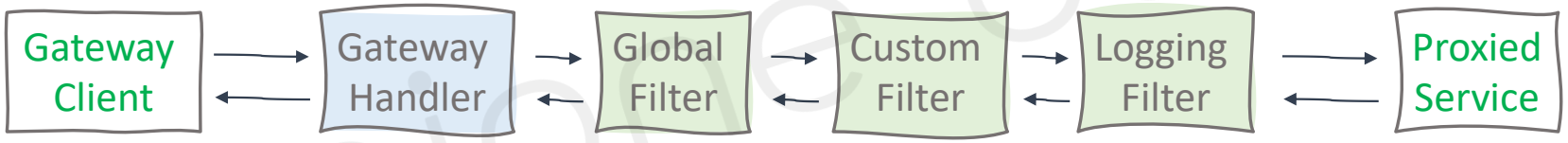
```
@Data
public static class Config {
    private String baseMessage;
    private boolean preLogger;
    private boolean postLogger;
}
```

```
default-filters:
  - name: GlobalFilter
    args:
      baseMessage: Spring Cloud Gateway GlobalFilter
      preLogger: true
      postLogger: true
```

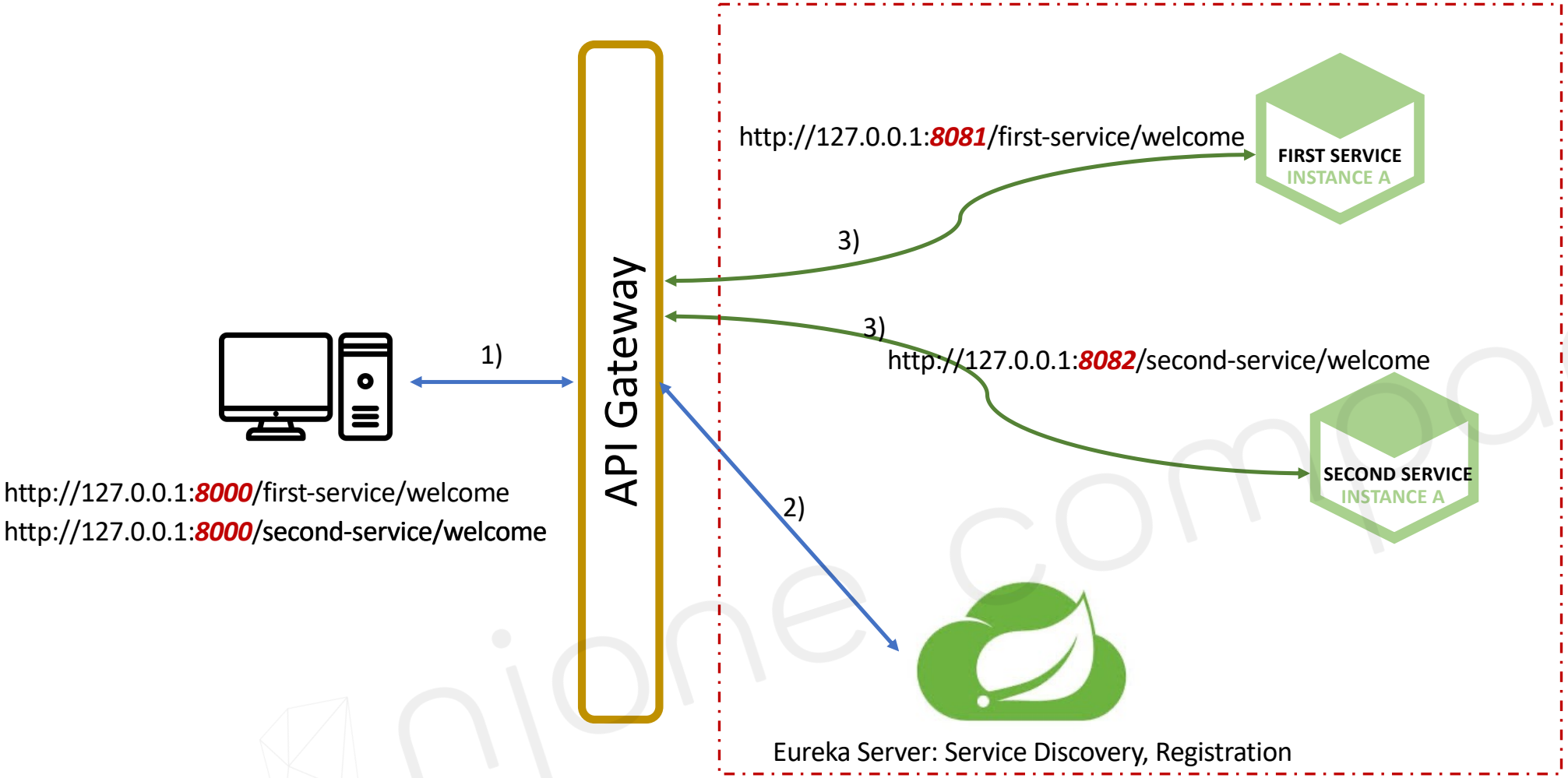
Spring Cloud Gateway – Custom Filter (Logging)

- Step8) Logging Filter – Test

```
: Global Filter baseMessage: Spring Cloud Gateway GlobalFilter
: Global Filter Start: request id -> 7690b2fe-1
: Custom PRE filter: request id -> 7690b2fe-1
: Logging filter baseMessage: Hi, there.
: Logging PRE filter: request uri -> http://127.0.0.1:8000/second-service/welcome
: Logging fPOST filter: response code -> 200 OK
: Custom POST filter: response code -> 200 OK
: Global Filter End: response code -> 200 OK
```



Spring Cloud Gateway – Eureka 연동



Spring Cloud Gateway – Eureka 연동

- Step1) Eureka Client 추가 – pom.xml, application.yml
  - Spring Cloud Gateway, First Service, Second Service

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

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

```yaml
eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

Spring Cloud Gateway – Eureka 연동

- Step2) Eureka Client 추가 –application.yml
  - Spring Cloud Gateway

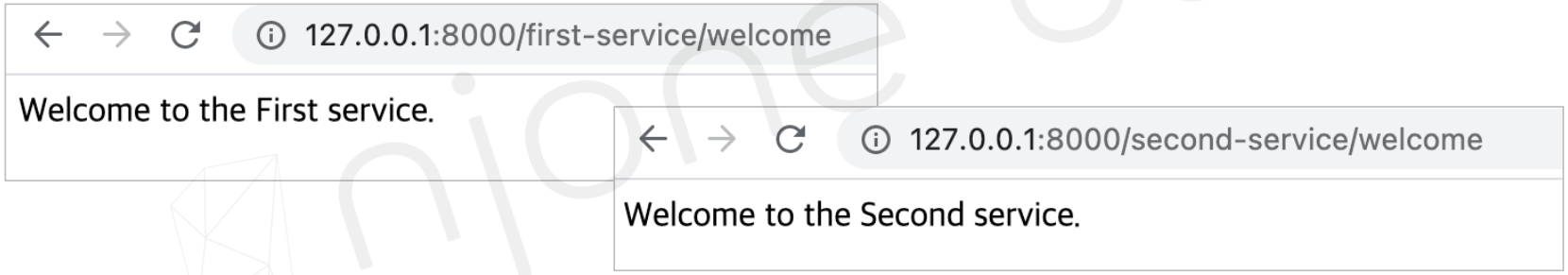
```
cloud:
  gateway:
    default-filters: <1 item>
    routes:
      - id: first-service
        uri: lb://MY-FIRST-SERVICE
        predicates:
          - Path=/first-service/**
        filters: <1 item>
      - id: second-service
        uri: lb://MY-SECOND-SERVICE
        predicates:
          - Path=/second-service/**
        filters: <2 items>
```

Spring Cloud Gateway – Eureka 연동

- Step3) Eureka Server – Service 등록 확인
  - Spring Cloud Gateway, First Service, Second Service

Instances currently registered with Eureka

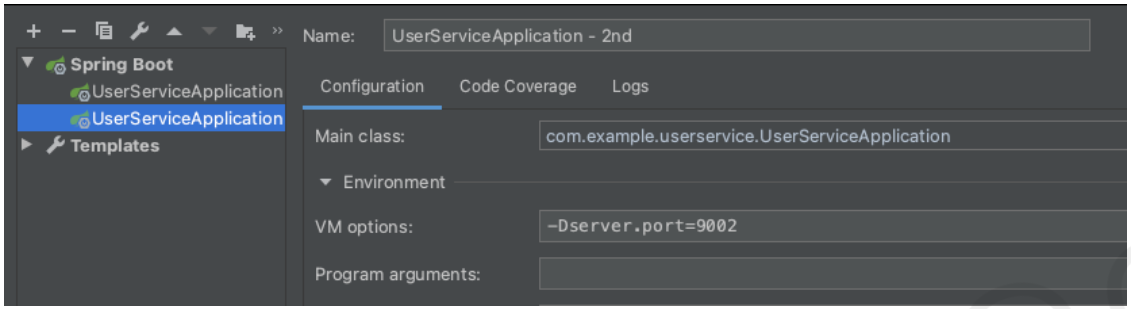
Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000
MY-FIRST-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:my-first-service:8081
MY-SECOND-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:my-second-service:8082



Spring Cloud Gateway – Load Balancer

- Step4) First Service, Second Service를 각각 2개씩 기동

1) VM Options → -Dserver.port=[다른포트]



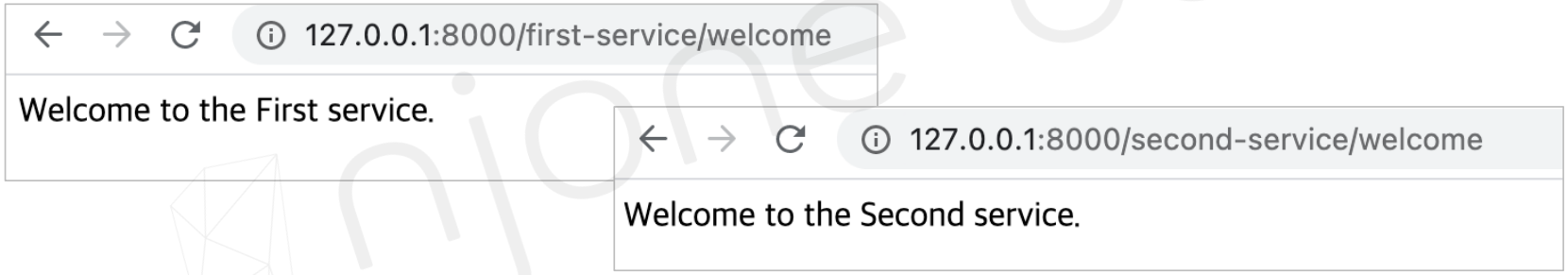
2) $ mvn spring-boot:run -Dspring-boot.run.jvmArguments='-Dserver.port=9003'

3) $ mvn clean compile package
$ java -jar -Dserver.port=9004 ./target/user-service-0.0.1-SNAPSHOT.jar

Spring Cloud Gateway – Load Balancer

- Step5) Eureka Server – Service 등록 확인
  - Spring Cloud Gateway, First Service, Second Service

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000
MY-FIRST-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.8:my-first-service:50001 , 192.168.0.8:my-first-service:50002
MY-SECOND-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.8:my-second-service:60002 , 192.168.0.8:my-second-service:60001



Spring Cloud Gateway – Load Balancer

- Step6) Random port 사용
  - Spring Cloud Gateway, First Service, Second Service

```yaml
server:
  port: 0

spring:
  application:
    name: my-first-service

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka

  instance:
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
```

Spring Cloud Gateway – Load Balancer

- Step6) Post 확인 – FirstController.java

```java
@Slf4j
public class FirstServiceController {
    @Value("local.server.port")
    private String serverPort;

    @GetMapping("/welcome")
    public String welcome() { return "Welcome to the First service."; }

    @GetMapping("/message")
    public String message(@RequestHeader("first-request") String header) {...}

    @GetMapping("/check")
    public String check(HttpServletRequest request) {
        log.info("Server port={}", request.getServerPort());

        return String.format("Hi, there. This is a message from First Service on PORT %s.", serverPort);
    }
}
```

Spring Cloud Gateway – Load Balancer

- Step6) Random port 사용
  - Spring Cloud Gateway, First Service, Second Service

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.8:gateway-service:8000
MY-FIRST-SERVICE	n/a (2)	(2)	UP (2) - my-first-service:c2452c34ef152ac050fc1da40eb4330e , my-first-service:eb69bfb9f798c4109f861fc2272fc854
MY-SECOND-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.8:my-second-service:60002 , 192.168.0.8:my-second-service:60001

192.168.0.8:52888/actuator/info

192.168.0.8:52887/actuator/info

127.0.0.1:8000/first-service/check

Hi, there. This is a message from First Service on PORT 52888.

127.0.0.1:8000/first-service/check

Hi, there. This is a message from First Service on PORT 52887.