Spring Cloud로 개발하는 마이크로서비스 애플리케이션

+

목차

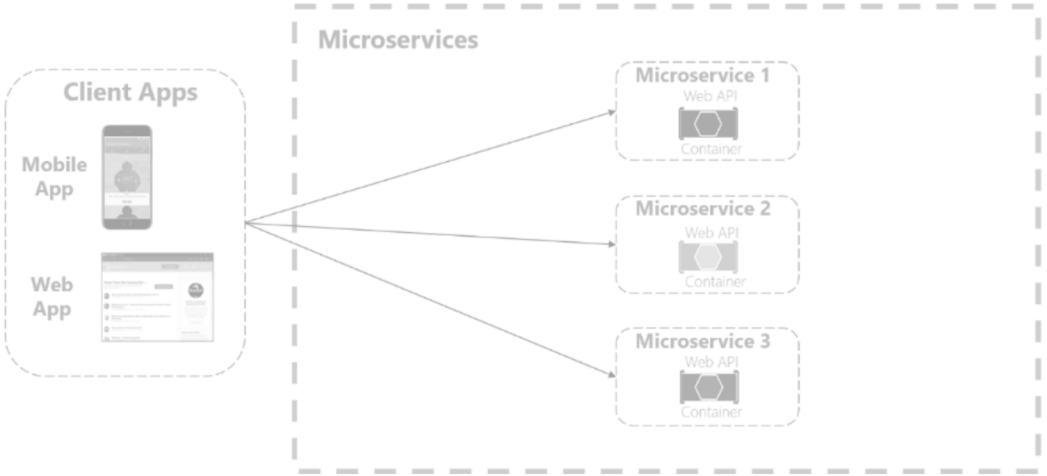
Part 1

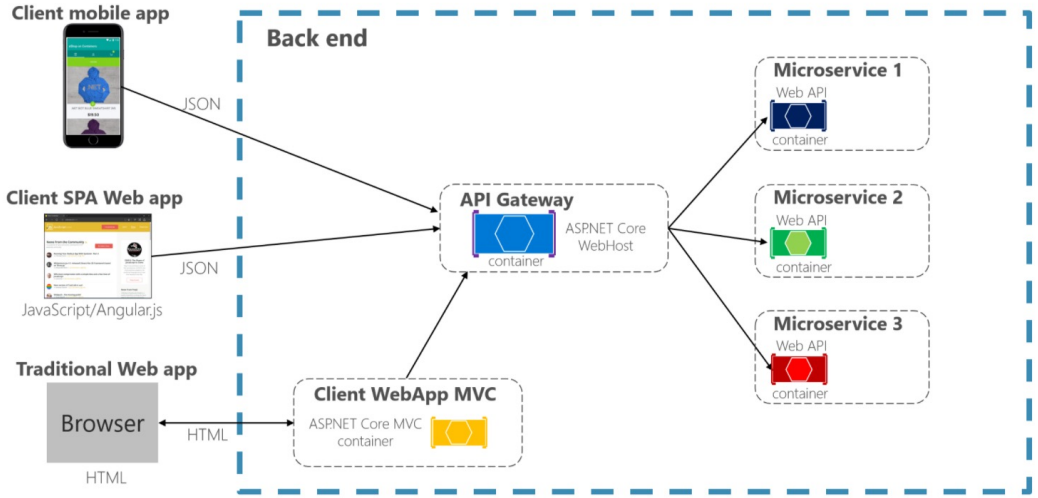
Section 2.

API Gateway Service

- API Gateway Service
- Netflix Ribbon과 Zuul
- Spring Cloud Gateway – 기본
- Spring Cloud Gateway – Filter
- Spring Cloud Gateway – Eureka 연동
- Spring Cloud Gateway – Load Balancer

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

Netflix Ribbon

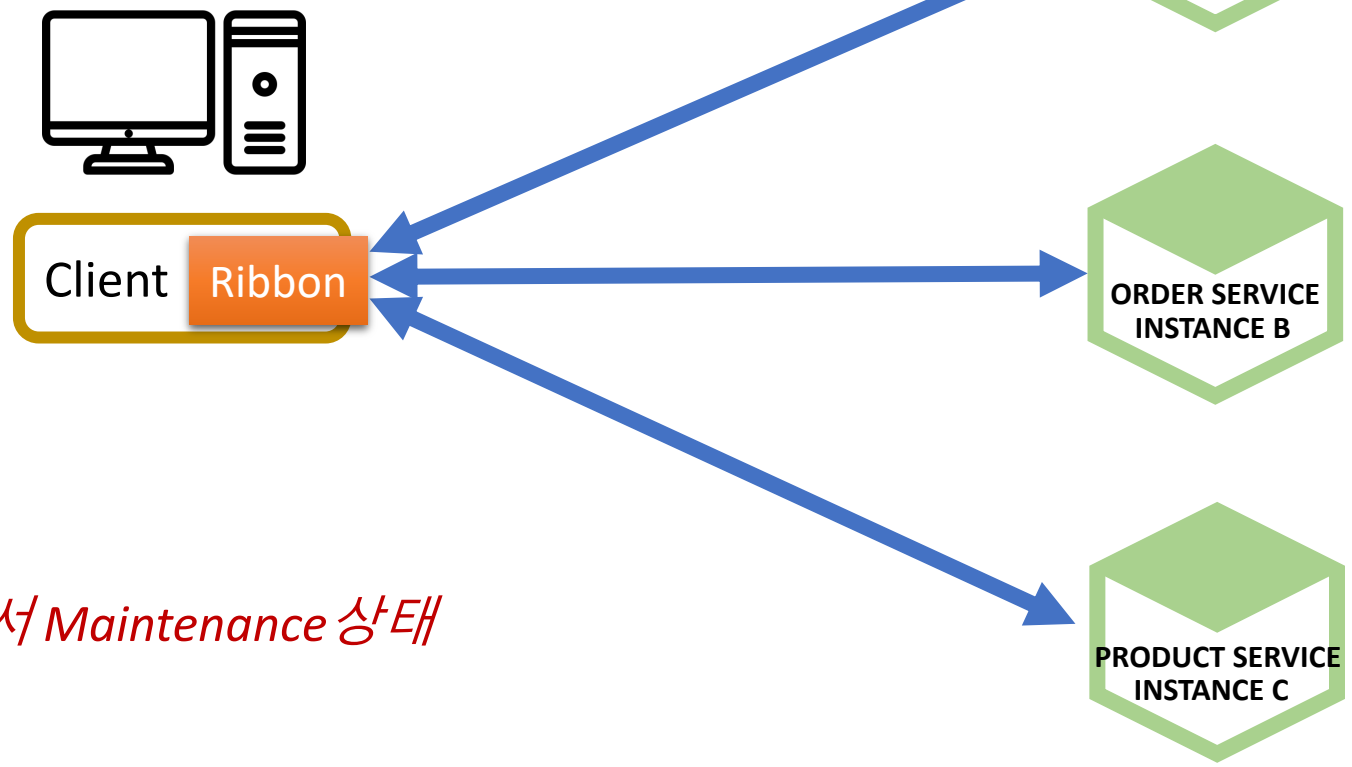
- Spring Cloud에서의 MSA간 통신

  1) *RestTemplate*

  2) *Feign Client*

```
RestTemplate restTemplate = new RestTemplate();
restTemplate.getForObject("http://localhost:8080/", User.class, 200);
```

```
@FeignClient("stores")
public interface StoreClient {
    @RequestMapping(method = RequestMethod.GET, value = "/stores")
    List<Store> getStores();
```

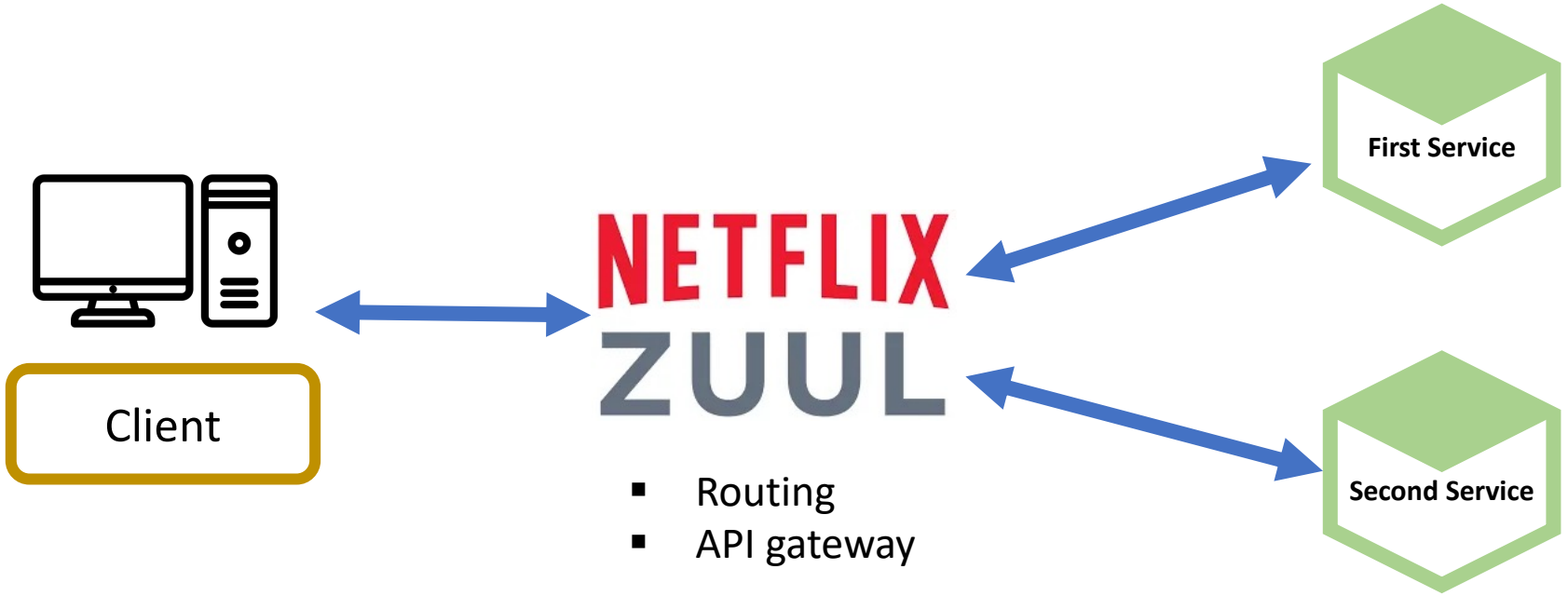


- Ribbon: ***Client side** Load Balancer*
  - *서비스 이름으로 호출*
  - *Health Check*

- *Spring Cloud Ribbon은 Spring Boot 2.4 에서 Maintenance 상태*

Netflix Zuul 구현

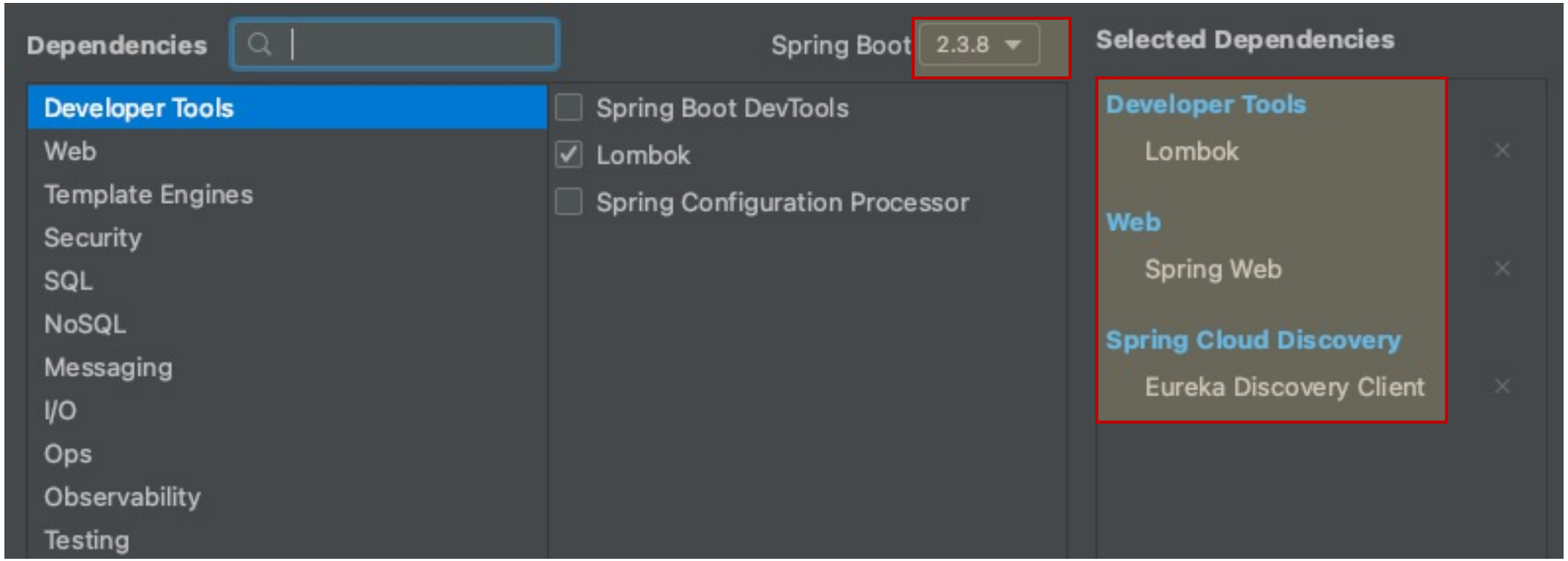
- 구성
  - First Service
  - Second Service
  - Netflix Zuul



- *Spring Cloud Zuul은 Spring Boot 2.4 에서 Maintenance 상태*
  - https://spring.io/blog/2018/12/12/spring-cloud-greenwich-rc1-available-now#spring-cloud-netflix-projects-entering-maintenance-mode

Netflix Zuul 구현

- Step1) First Service, Second Service
  - Spring Boot: 2.3.8
  - Dependencies: Lombok, Spring Web, Eureka Discovery Client



Netflix Zuul 구현

- Step2) First Service, Second Service

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

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

}
```

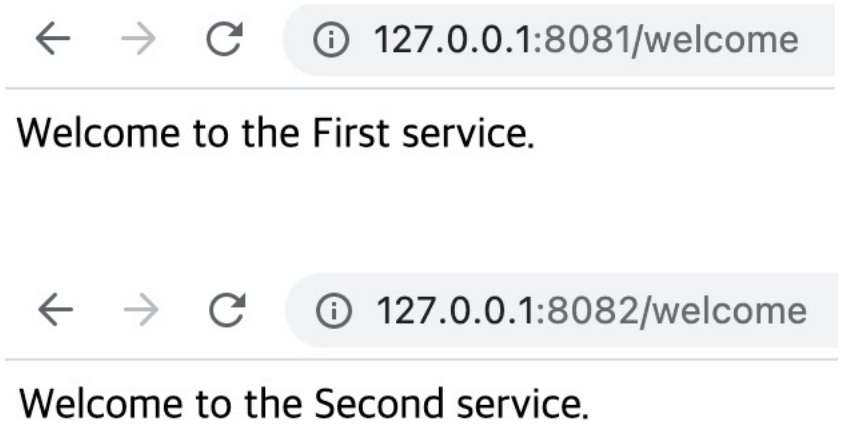
```yaml
server:
  port: 8081

spring:
  application:
    name: my-first-service

eureka:
  client:
    register-with-eureka: false
    fetchRegistry: false
```

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

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

}
```

```yaml
server:
  port: 8082

spring:
  application:
    name: my-second-service

eureka:
  client:
    register-with-eureka: false
    fetchRegistry: false
```

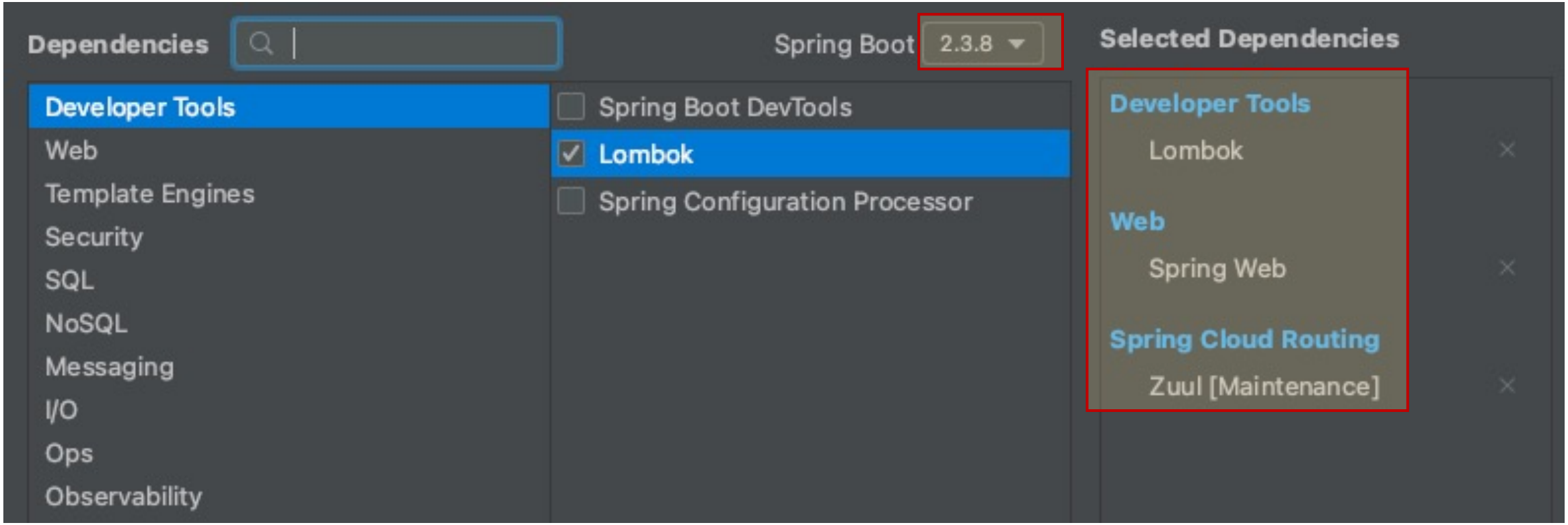
Netflix Zuul 구현

- Step3) Test



Netflix Zuul 구현

- Step4) Zuul Service
  - Spring Boot: 2.3.8
  - Dependencies: Lombok, Spring Web, Zuul

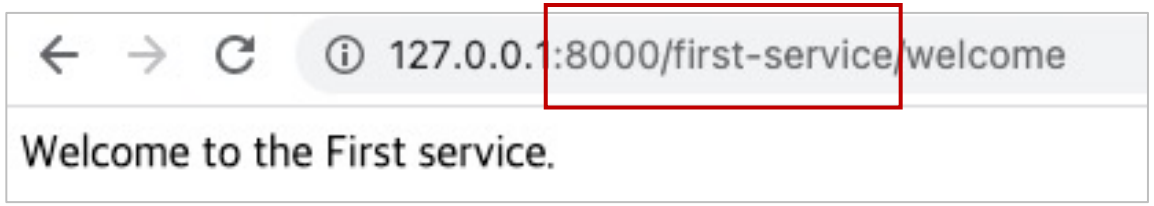


Netflix Zuul 구현

- Step5) Zuul Service

```java
@SpringBootApplication
@EnableZuulProxy
public class ZuulServiceApplication {

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

}
```

```yaml
server:
  port: 8000

spring:
  application:
    name: my-zuul-service

zuul:
  routes:
    first-service:
      path: /first-service/**
      url: http://localhost:8081
    second-service:
      path: /second-service/**
      url: http://localhost:8082
```



Netflix Zuul 구현

- Step6) ZuulFilter

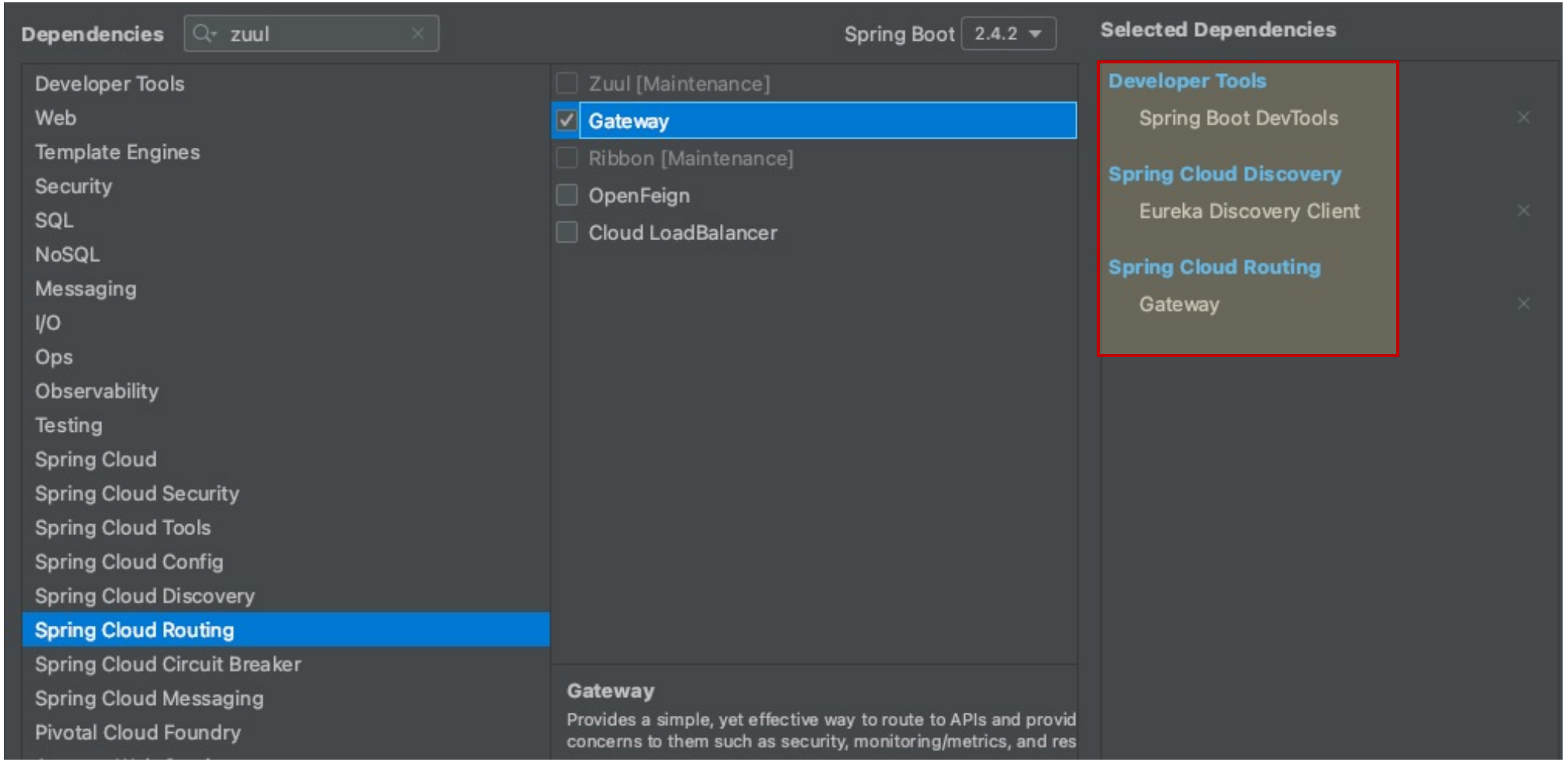
```java
@Component
public class ZuulLoggingFilter extends ZuulFilter {

    Logger logger = LoggerFactory.getLogger(ZuulLoggingFilter.class);
    @Override
    public Object run() throws ZuulException {
        logger.info("********************** printing logs : ");

        RequestContext ctx = RequestContext.getCurrentContext();
        HttpServletRequest request = ctx.getRequest();
        logger.info("********************** " + request.getRequestURI());

        return null;
    }
```

```
: ********************** printing logs :
: ********************** /first-service/welcome
: ********************** printing logs :
: ********************** /second-service/welcome
```

```java
    @Override
    public String filterType() {
        return "pre";
    }
}
```

Spring Cloud Gateway – 기본

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



Spring Cloud Gateway – 기본

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

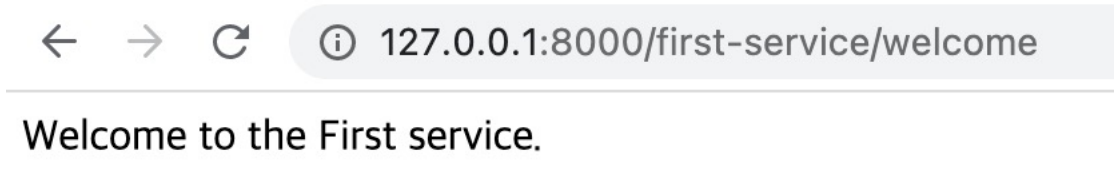
```yaml
server:
  port: 8000

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/**
```

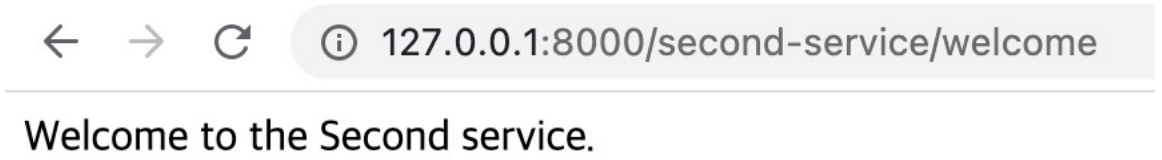
Spring Cloud Gateway – 기본

- Step3) Test

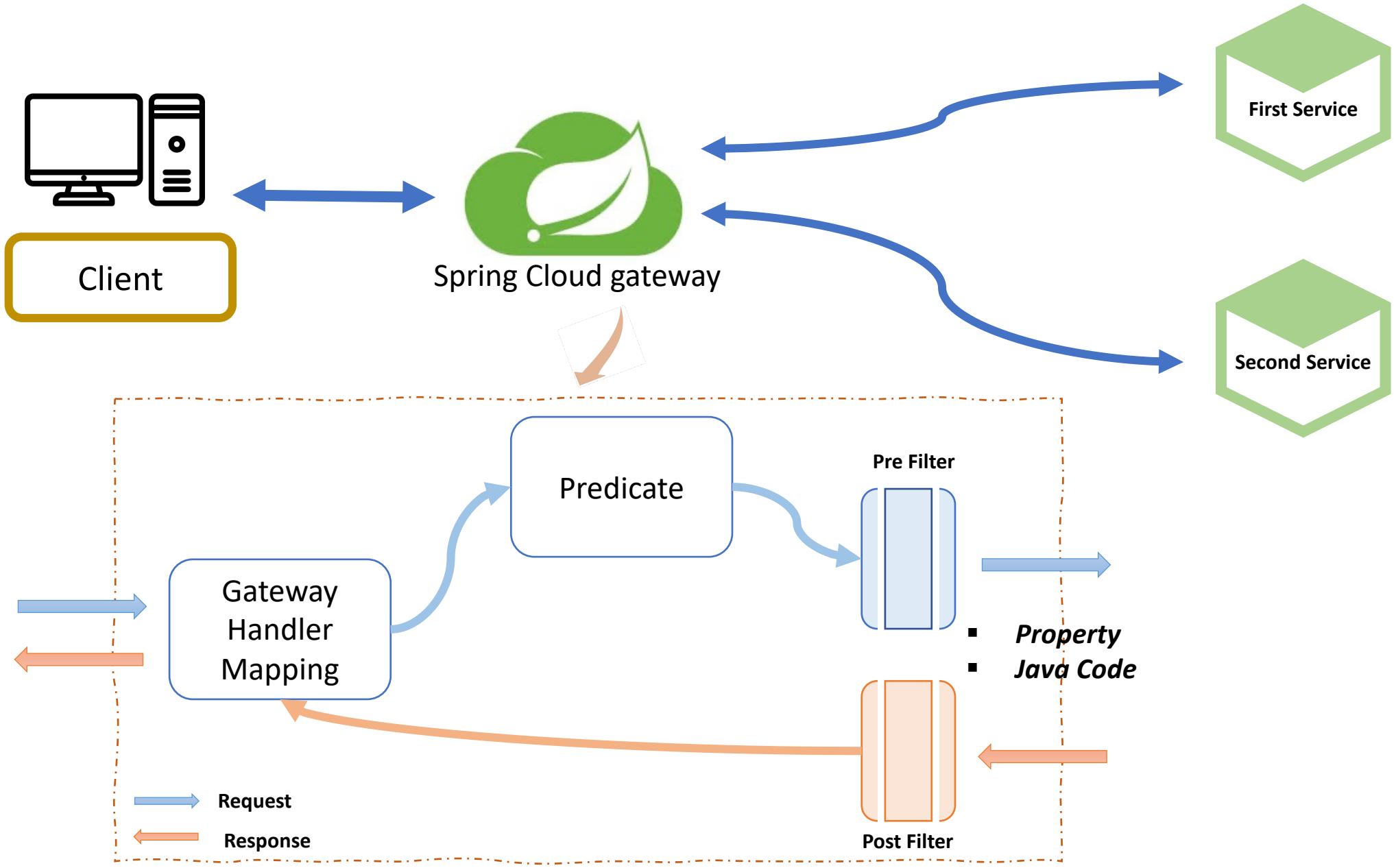
```
@RestController
@RequestMapping("/first-service")
public class FirstServiceController {
```




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



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

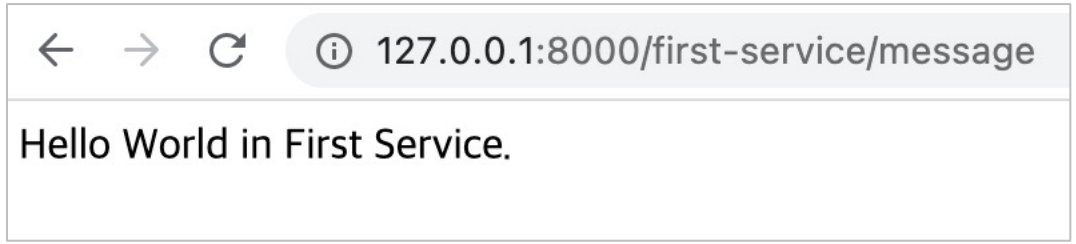
- Step4) Filter using Java Code – FirstServiceController.java, SecondServiceController.java

```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.";
    }
}
```

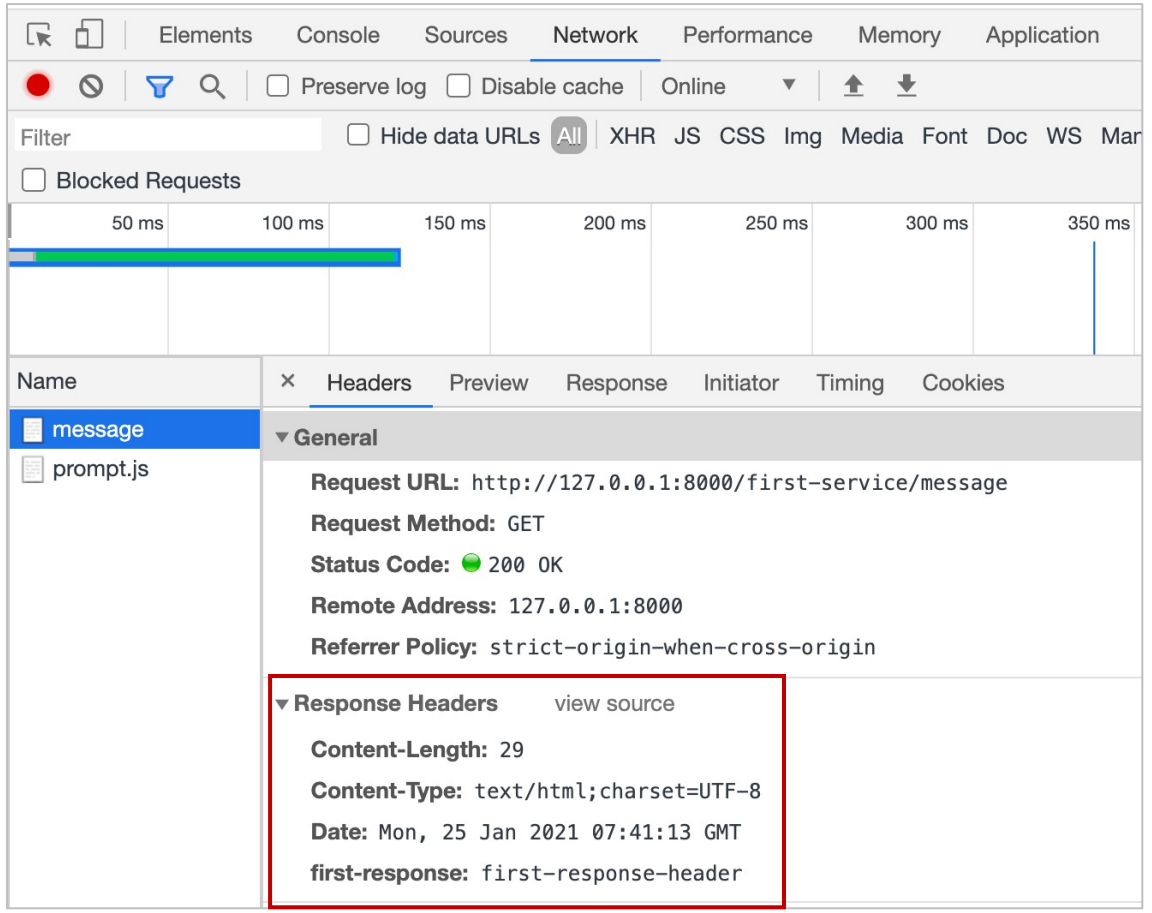
```java
@RestController
@RequestMapping("/second-service")
public class SecondServiceController {

    @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
```



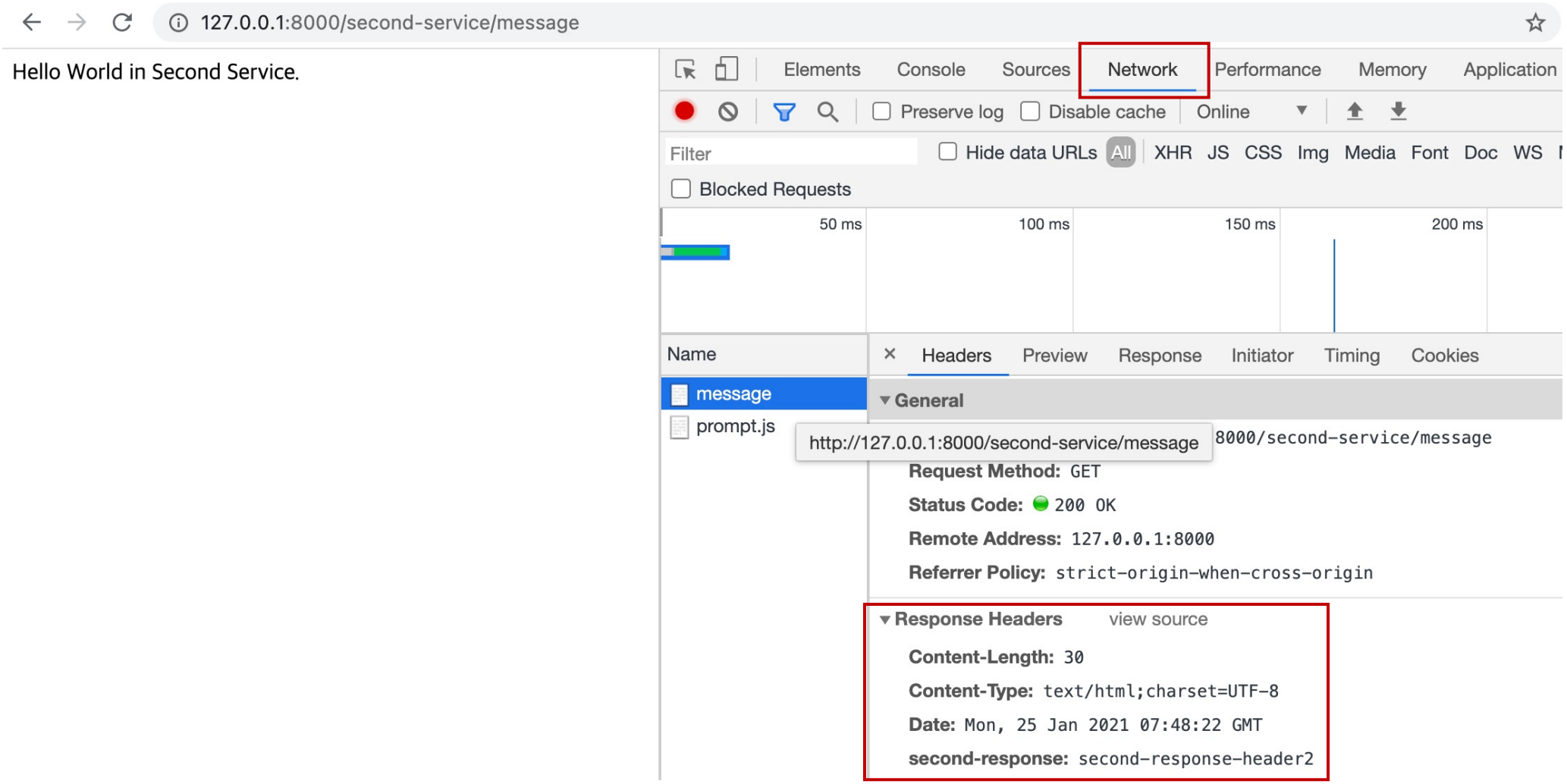
Spring Cloud Gateway – Filter

- 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

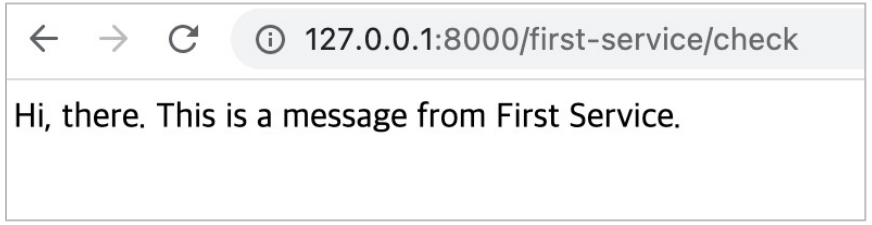
```yaml
      routes:
        - id: first-service
          uri: http://localhost:8081/
          predicates:
            - Path=/first-service/**
          filters:
#           - AddRequestHeader=first-request, first-request-header2
#           - AddResponseHeader=first-response, first-response-header2
            - 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 – LoggingFilter.java

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

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

            log.info("Logging filter baseMessage: " + config.getBaseMessage());
            if (config.isPreLogger()) {
                log.info("Logging PRE filter: request uri -> {}", request.getURI());
            }
            return chain.filter(exchange).then(Mono.fromRunnable(()->{
                if (config.isPostLogger()) {
                    log.info("Logging fPOST filter: response code -> {}", response.getStatusCode());
                }
            }));
        };
    }

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

Spring Cloud Gateway – Custom Filter (Logging)

- Step8) Logging Filter – application.yml

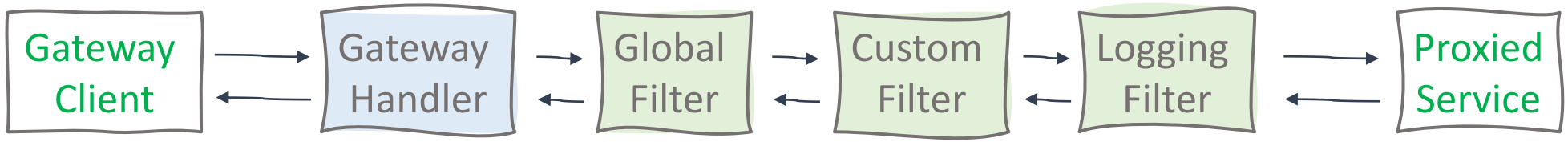
```yaml
routes:
  - id: first-service
    uri: http://localhost:8081/
    predicates:
      - Path=/first-service/**
    filters:
      - CustomFilter
  - id: second-service
    uri: http://localhost:8082/
    predicates:
      - Path=/second-service/**
    filters:
      - name: CustomFilter
      - name: LoggingFilter
        args:
          baseMessage: Hi, there.
          preLogger: true
          postLogger: true
```

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

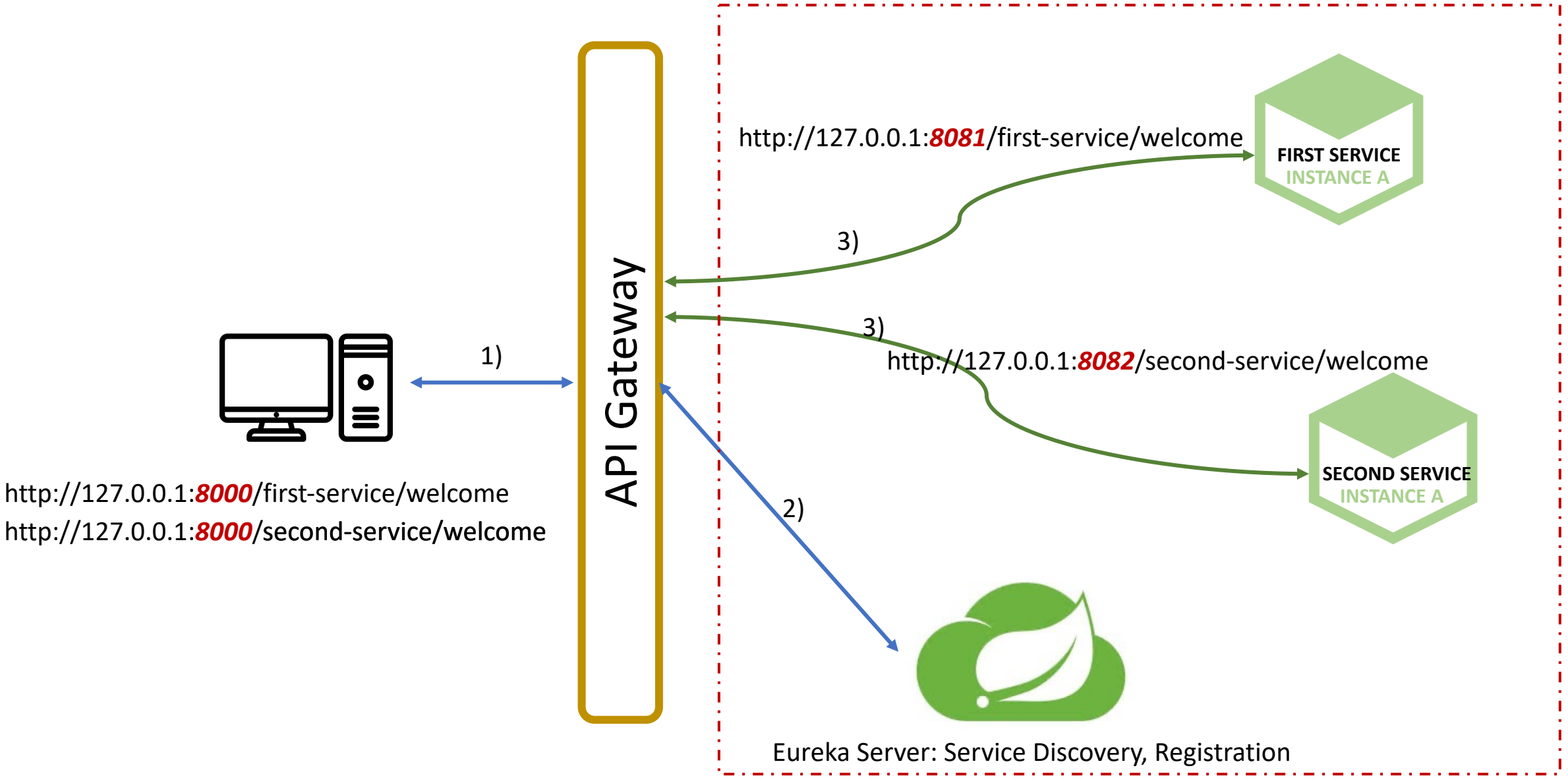
Spring Cloud Gateway – Custom Filter (Logging)

- 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>

<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

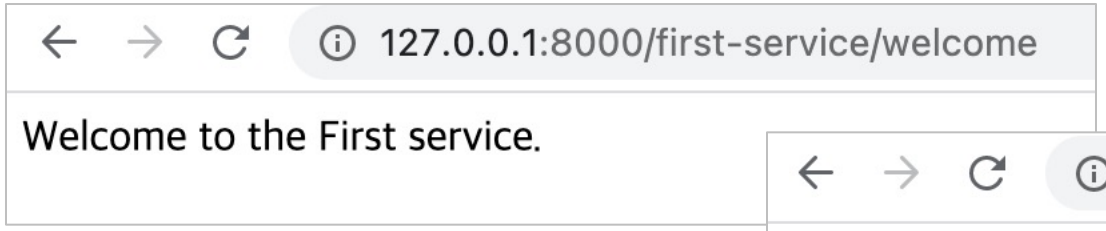
```yaml
cloud:
  gateway:
    default-filters: <1 item>
    routes:
      - id: first-service
        uri: lb://MY-FIRST-SERVICE
        predicates:
          - Path=/first-service/**
        filters: <1 item>
      - id: second-service
        uri: lb://MY-SECOND-SERVICE
        predicates:
       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

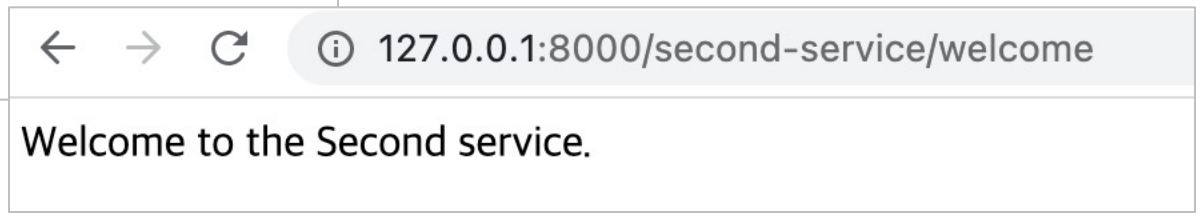
Spring Cloud Gateway – Eureka 연동

- 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

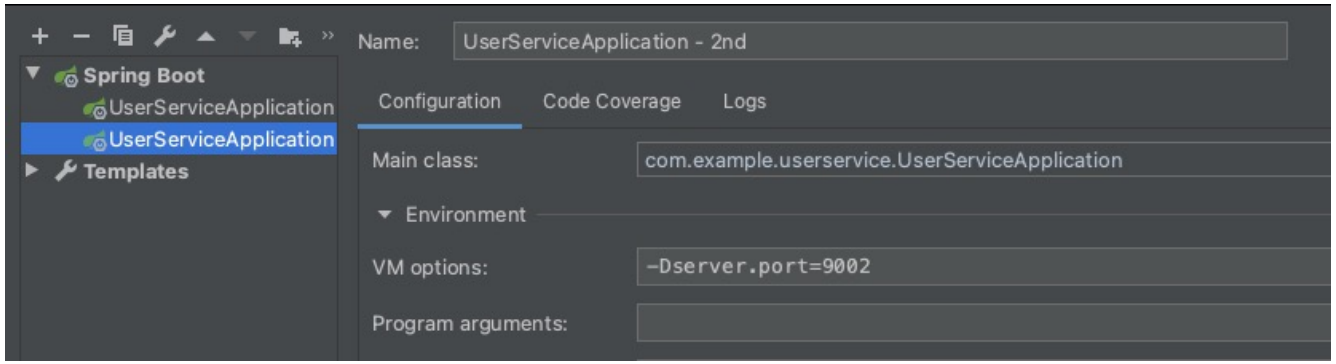




Spring Cloud Gateway – Load Balancer

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

1) VM Options → -Dserver.port=[다른포트]



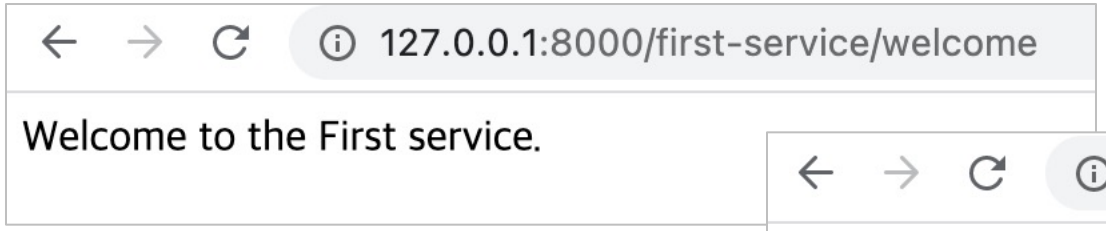
2) $ mvn spring-boot:run -Dspring-boot.run.jvmArguments='-Dserver.port=9003'

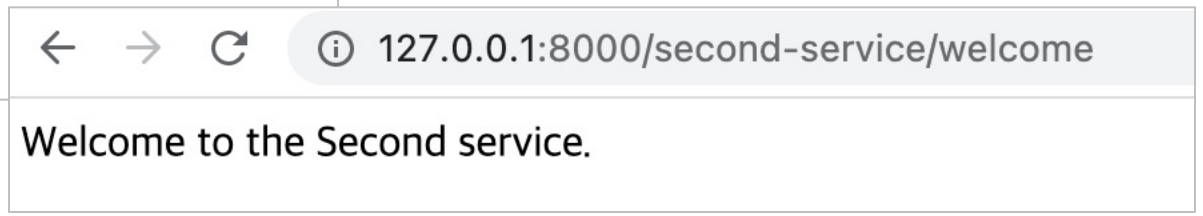
3) $ mvn clean compile package
$ java -jar -Dserver.port=9004 ./target/user-service-0.0.1-SNAPSHOT.jar

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.