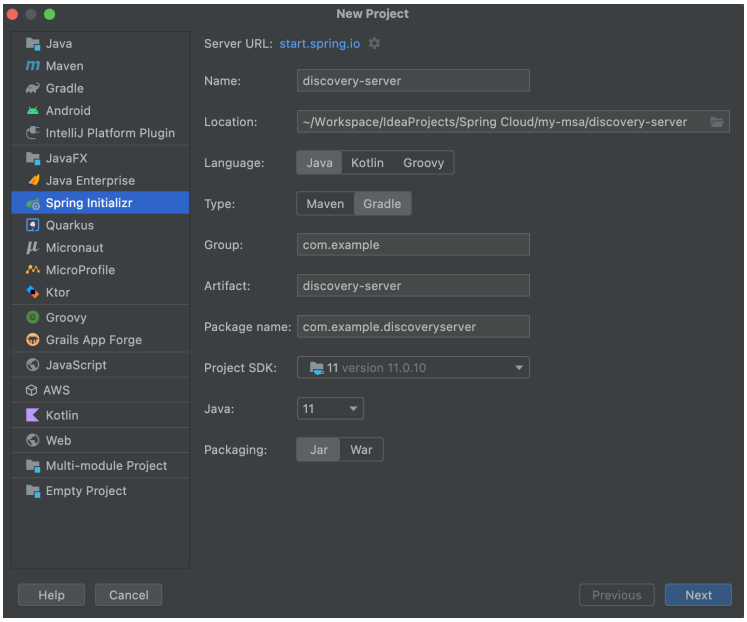
Spring Cloud 환경 구축

1. Eureka

1) Eureka Server

Spring Cloud 와 Spring Boot의 호환 버전 주의

- **spring cloud 2020.0.5 version**
- **spring boot 2.5.10 version 사용**



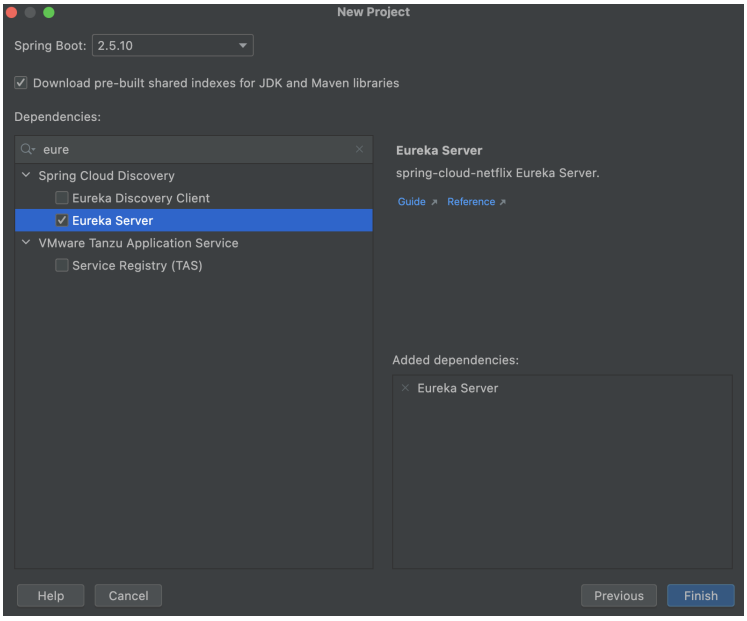


Table 1. Release train Spring Boot compatibility

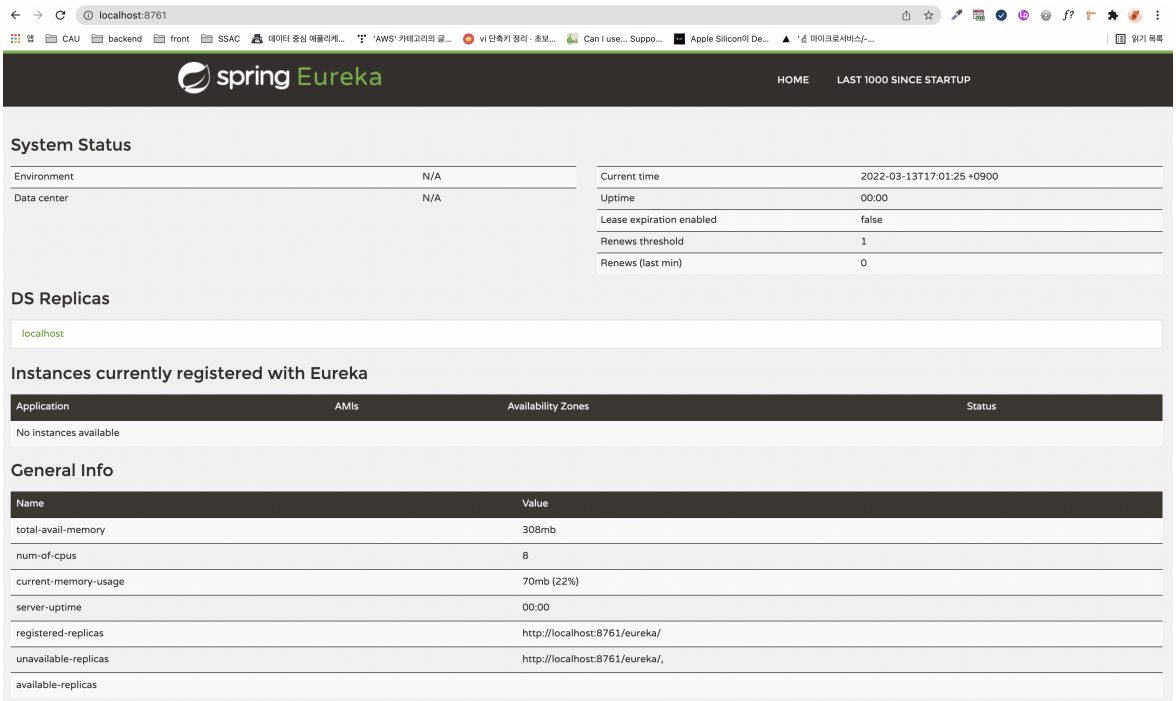
Release Train	Boot Version
2021.0.x aka Jubilee	2.6.x
2020.0.x aka Ilford	2.4.x, 2.5.x (Starting with 2020.0.3)

```
ext {
    set('springCloudVersion', "2020.0.5")
}

dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
    }
}
```

application.yml

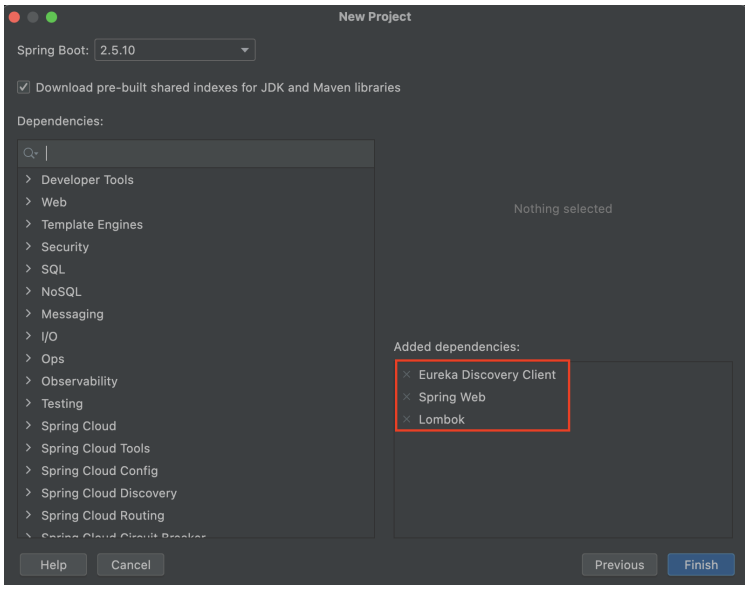
```
server:
  port: 8761  # Eureka server default port

spring:
  application:
    name: discovery-service  # 모든 서비스를 applcation.name으로 식별 !!

eureka:
  client:
    register-with-eureka: false  # client 동작 false
    fetch-registry: false        # 자신을 discovery에 등록하지 않도록 false
  server:
    enableSelfPreservation: true  # 일시적인 네트워크 장애로 인한 서비스 해제 막기 위한 자기 보호 모드
```

```java
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServerApplication {

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

}
```



2) Eureka Client

user-service, order-service



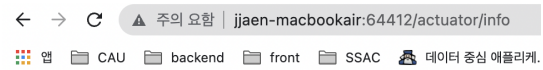
```
ext {
    set('springCloudVersion', "2020.0.5")
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

dependencyManagement {
    imports {
        mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
    }
}
```

application.yml

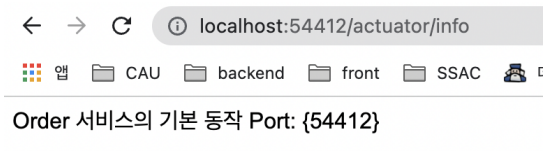
```
# user-service
server:
  port: 64412

spring:
  application:
    name: user-service

eureka:
  instance:
    instance-id: user-microservice-instance

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

---------------------------------------------------------------------

# order-service
server:
  port: 54412

spring:
  application:
    name: order-service

eureka:
  instance:
    instance-id: order-microservice-instance

  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
ORDER-SERVICE	n/a (1)	(1)	UP (1) - order-microservice-instance
USER-SERVICE	n/a (1)	(1)	UP (1) - user-microservice-instance



User 서비스의 기본 동작 Port: {64412}

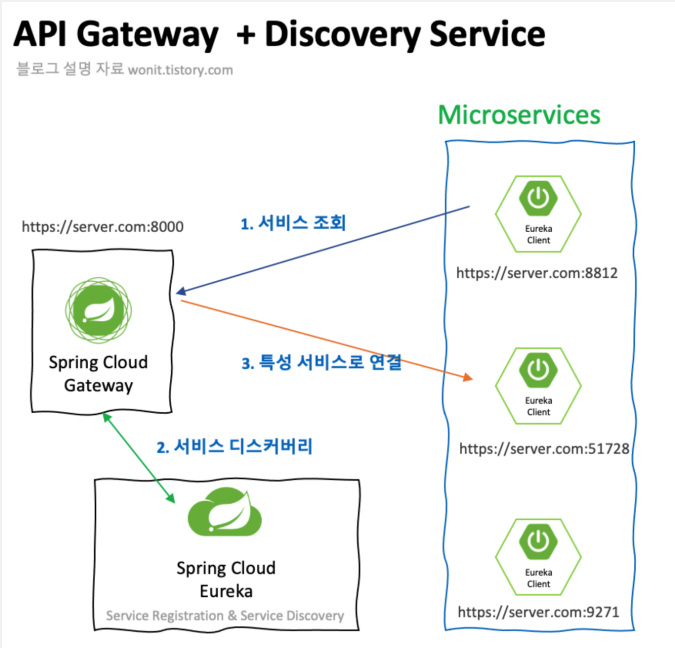


2. API Gateway

- Eureka server만 사용했을 때는, 각각의 ms port 번호를 알고 있어야 함..!
  - `Spring Cloud Gateway` 도입 시, port 신경 X !!

  : Netty server를 내장한 WebFlux 기반 Gateway (비동기)

> Spring Cloud Gateway
> Level up your Java code and explore what Spring can do for you.
> https://spring.io/projects/spring-cloud-gateway

> **Spring Cloud Gateway + Eureka**
>
> 
>
>
> 1. **User 서버는 Order 서버에 보내야할 요청을 Gateway로 전달한다.**
> 2. **Gateway는 Eureka Server로 Order 서버의 정보를 discovery한다.**
> 3. **Gateway가 Order Server로 연결한다.**

Spring Cloud Gateway

Route

: 목적지 URI의 Predicates라는 조건들의 목록 그리고 Filter들을 이용하여 어떤 곳으로 Routing 할 것인지 명시하는 역할

Predicates

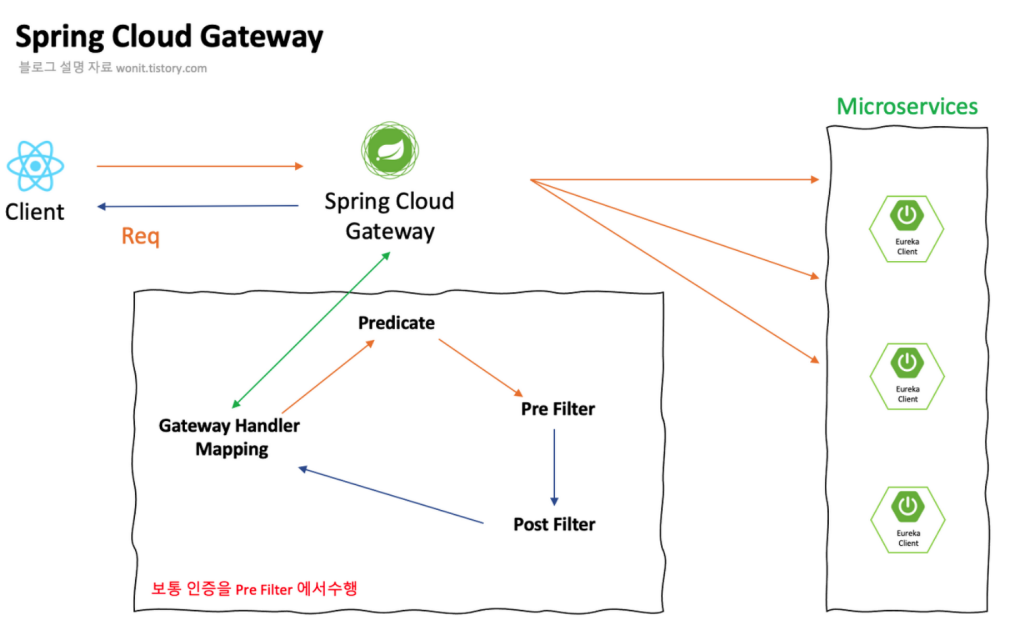
: 조건.

- `predicated: -Path:/user/**`

Filter

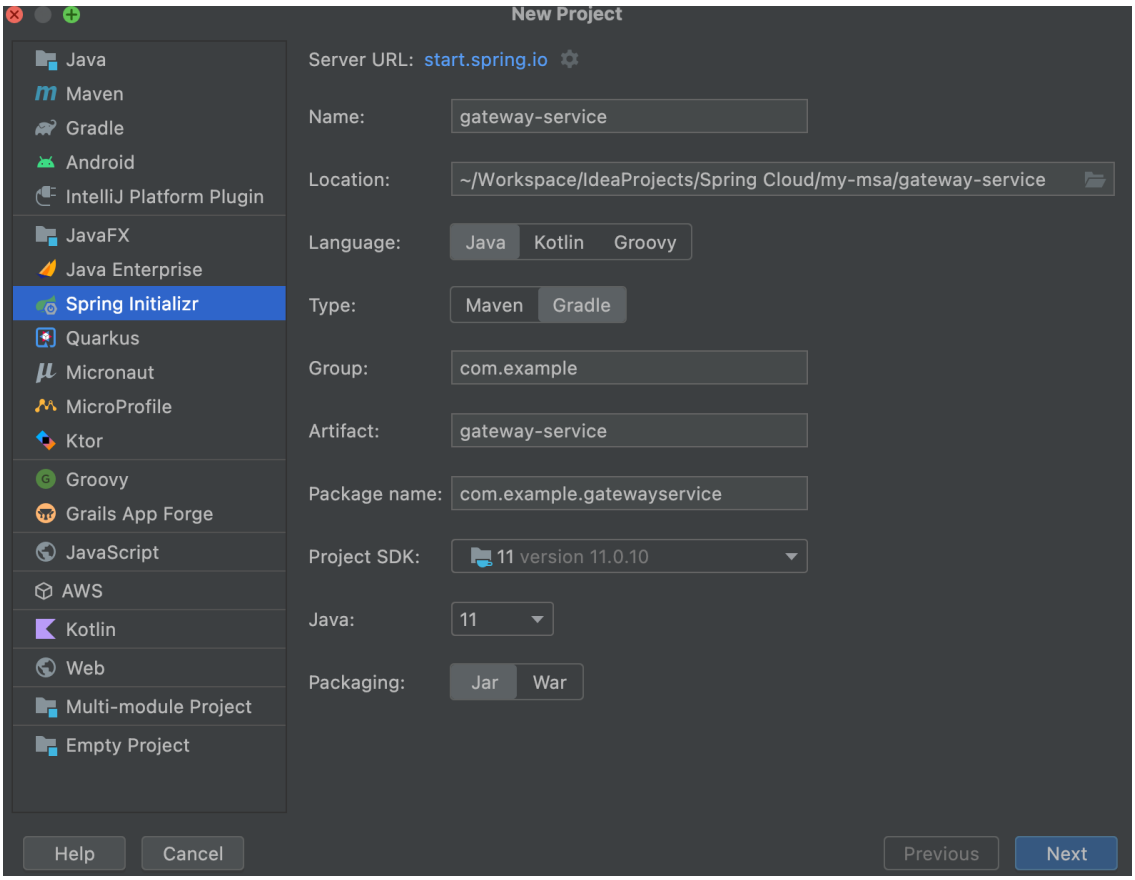
: 들어오는 requests, responses를 Filter를 타게 함으로써 원하는 방식으로 요청을 보내거나 header 조작 가능 + log file 작성 가능

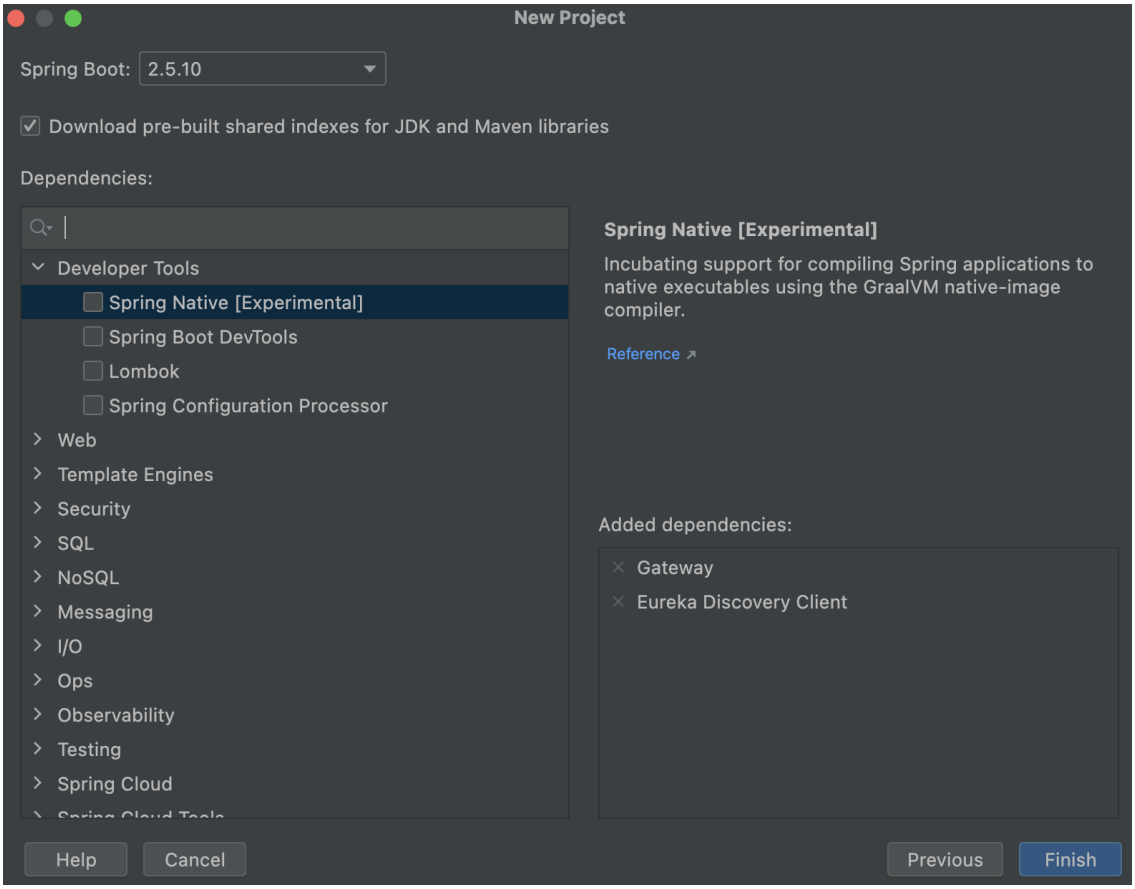
- Gateway Handler Mapping
  - Gateway 가 CLient로 부터 어떤 요청이 왔는지 확인하고 Mapping 하는 작업을 수행한다.
- Predicate
  - Handler Mapping 시에 필요한 Uri 정보나, Path 정보를 확인하는 주체가 된다.
- Filter
  - Handler Mapping이 된 후 들어온 요청에 대한 필터 작업을 수행할 수 있다.
  - 2개의 필터로 크게 나뉘며 사전(Pre Filter)와 사후(Post Filter)로 나눌 수 있다.
    - Pre Filter
      - 특정 작업이 일어나기 전에 지정
    - Post Filter
      - 특정 작업이 끝난 후에 지정



1. **Client 는 Spring Cloud Gateway 에 요청을 보낸다.**

2. **Gateway Handler Mapping 에서 해당 요청에 대한 Route와 Predicates가 일치한다고 판단하면 해당 요청은 Gateway Web handler로 보내진다.**
3. **handler 에서 Filter Chain 을 이용해서 사전 필터 혹은 사후 필터로 나누어 동작한다.**
4. **필터링이 된 후 실제 마이크로서비스에게 전달된다.**





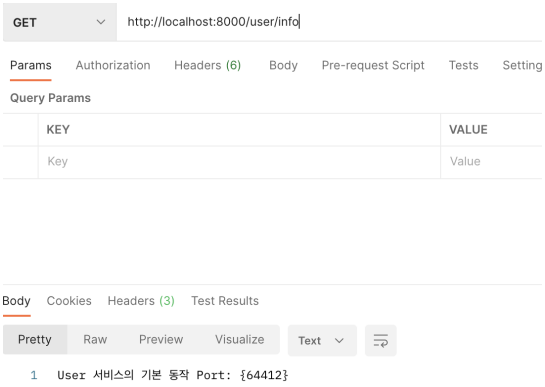
application.yml

```
server:
  port: 8000

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

spring:
  application:
    name: gateway-service
```

Gateway로 service 연결

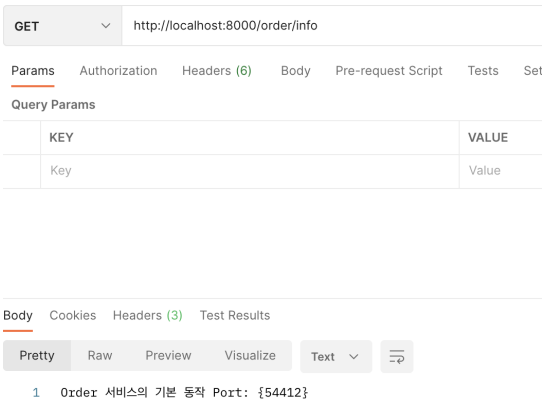
```
# applcation.yml
...
spring:
  application:
    name: gateway-service

  cloud:
    gateway:
      routes:
        - id: user-service
          uri: http://localhost:64412  # forwarding (http://localhost:8000/user -> http://localhost:64412)
          predicates:
            - Path=/user/**  # 해당 gateway server의 /user/**로 들어오는 요청은 user-service로 인식
        - id: order-service
```

```
          uri: http://localhost:54412  # forwarding (http://localhost:8000/order -> http://localhost:54412)
          predicates:
            - Path=/order/**
```



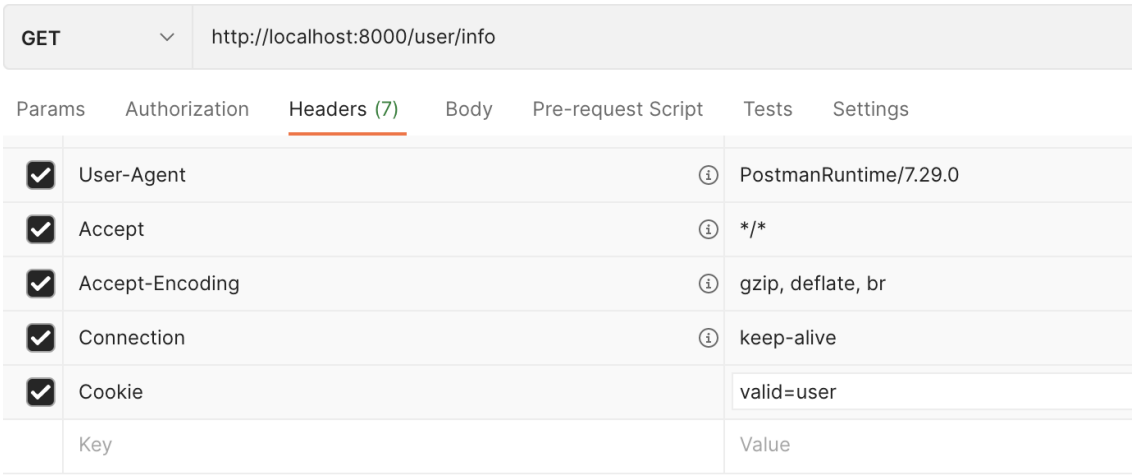
Eureka Server에서 각 인스턴스 정보를 받아 Load Balancing

```
# applcation.yml
...
spring:
  application:
    name: gateway-service

  cloud:
    gateway:
      routes:
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user/**
        - id: order-service
          uri: lb://ORDER-SERVICE
          predicates:
                    - Path=/order/**
```



Built-in Route

: Predicates, Route 조작




```yaml
spring:
  application:
    name: gateway-service

  cloud:
    gateway:
      routes:
        - id: user-service
          uri: lb://USER-SERVICE  # http://localhost:64412  # forwarding (http://localhost:8000/user -> http://localhost:64412)
          predicates:
            - Path=/user/**  # ?? gateway server? /user/**? ???? ??? user-service? ??
          filters:
            - CustomAuthFilter
        - id: order-service
          uri: lb://ORDER-SERVICE  # http://localhost:54412  # forwarding (http://localhost:8000/order -> http://localhost:54412)
          predicates:
            - Path=/order/**
```

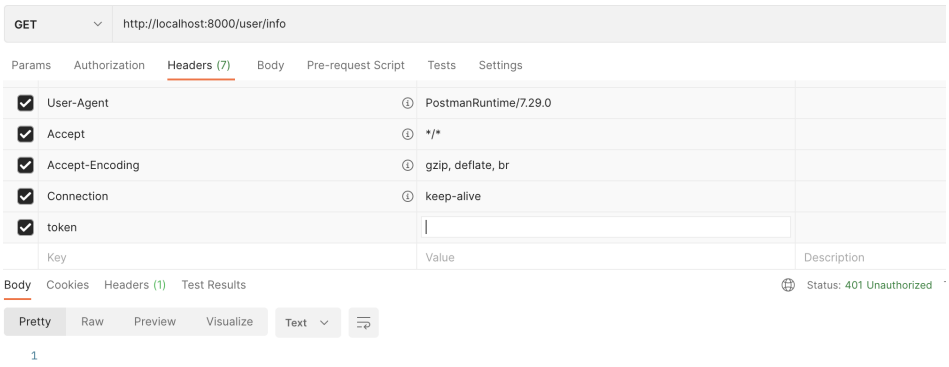
Custom Filter

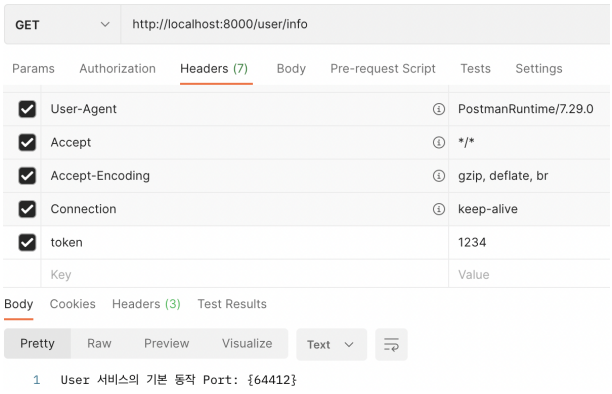
- **Custom Filter**
- **AbstractGatewayFilterFactory 상속과 apply 재정의**
- **검증하기**

CustomFilter.class

- `AbstractGatewayFilterFactory`

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

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

            // Request Header token 이 없는 경우
            if(!request.getHeaders().containsKey("token"))
                return handleUnAuthorized(exchange);  // 401

            List<String> token = request.getHeaders().get("token");
            String tokenString = Objects.requireNonNull(token).get(0);

            // token 검증
            if(!"1234".equals(tokenString)) {
                return handleUnAuthorized(exchange);
            }

            log.info("Pre CustomAuhFilter | request id : " + request.getId());
            // token 확인
            return chain.filter(exchange);
        };
    }

    private Mono<Void> handleUnAuthorized(ServerWebExchange exchange) {
        ServerHttpResponse response = exchange.getResponse();  // PostFilter

        response.setStatusCode(HttpStatus.UNAUTHORIZED);
        log.info("Post handleUnAuthorized | response code -> {}", response.getStatusCode());

        return response.setComplete();
    }

    public static class Config {
        // put the configuration properties
    }
}
```

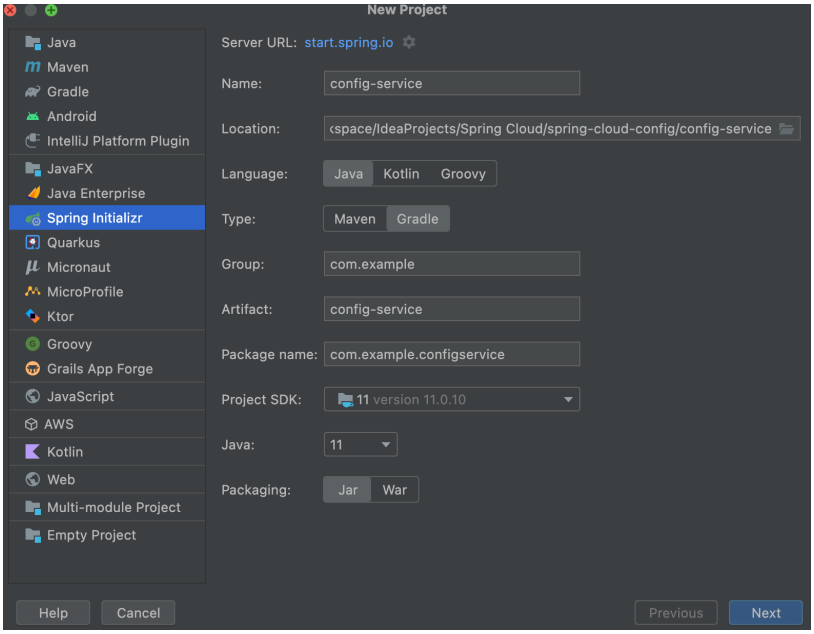


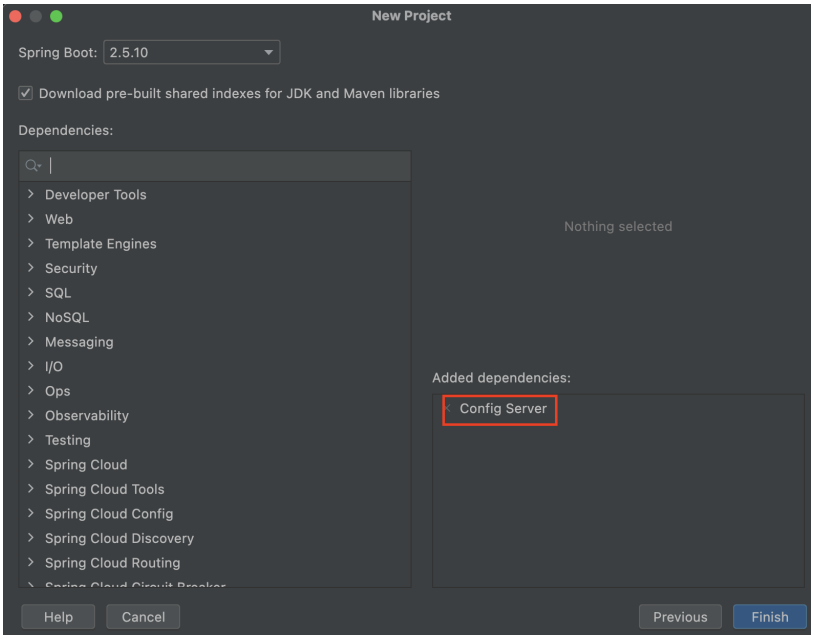


3. Spring Cloud Config

- **Config Server**
  : 설정 정보를 저장하여 관리해주는 주체
- **Config Client**

  : 서버에 저장된 설정 정보를 받아 사용하는 주체






```java
@SpringBootApplication
@EnableConfigServer
public class ConfigServiceApplication {

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

}
```