1. Introducción

Las arquitecturas basadas en microservicios permiten dividir una aplicación en múltiples servicios independientes, cada uno con una responsabilidad clara y específica. Sin embargo, para que el sistema funcione de manera coherente, estos microservicios deben comunicarse entre sí. Existen dos tipos principales de comunicación entre microservicios:

- Comunicación **sincrónica**, donde un servicio espera una respuesta inmediata del otro.
- Comunicación **asincrónica**, donde los servicios intercambian mensajes sin necesidad de esperar una respuesta inmediata.

En este capítulo, exploraremos ambos tipos de comunicación, y mostraremos cómo implementarlos usando Spring Boot. Usaremos como ejemplo dos microservicios: **producto-service** y **pedido-service**.

Next

2. Comunicación sincrónica entre microservicios

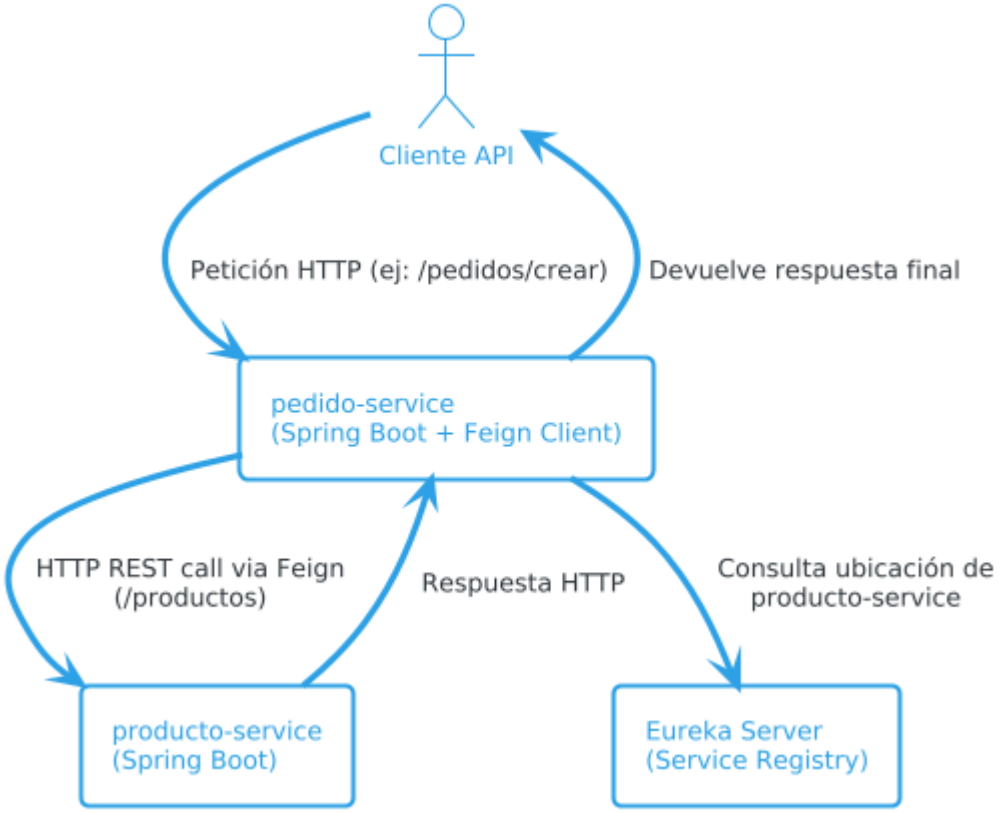
En arquitecturas basadas en microservicios, la comunicación eficiente entre servicios es clave. Una forma común de lograrlo es a través de **HTTP REST** de manera **sincrónica**, donde un servicio realiza una solicitud y espera la respuesta antes de continuar.

Para facilitar esta comunicación y reducir el acoplamiento entre servicios, Spring Cloud ofrece herramientas como:

- **OpenFeign**: Cliente HTTP declarativo.
- **Eureka**: Service Discovery para registrar y localizar microservicios automáticamente.

En este capítulo, veremos cómo integrar ambos para construir microservicios robustos y flexibles.

3. Arquitectura general



- `producto-service` y `pedido-service` **se registran en Eureka**.
- `pedido-service` usa **OpenFeign** para consultar a Eureka y encontrar la dirección de `producto-service` automáticamente.

Componentes del sistema:

1. **Eureka Server**
2. **producto-service**
3. **pedido-service**

Back

Next

4. Configuración del Eureka Server

Dependencias Maven

Creamos un proyecto Spring Boot separado para el Eureka Server con:

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

Clase principal

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

application.yml

```
server:
  port: 8761

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

Aquí configuramos Eureka Server para que **NO se registre a sí mismo** (solo actúe como servidor).

Comunicación entre microservicios: Sincrónica y Asincrónica con Spring Boot

5. Configuración de producto-service

Dependencias Maven

Agregamos:

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

application.yml

```
server:
  port: 8081

spring:
  application:
    name: producto-service

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
    register-with-eureka: true
    fetch-registry: true
    lease-renewal-interval-in-seconds: 5   # Heartbeat cada 30 segundos (valor recomenda
    lease-expiration-duration-in-seconds: 90 # Tiempo para considerar DOWN si no recibe
    renewal-percent-threshold: 0.85  # Default es 0.85, puedes bajarlo un poco, por ejem
    enable-self-preservation: true   # Esto ya viene en true, pero confírmalo
```

Controlador REST

```
@RestController
@RequestMapping("/productos")
public class ProductoController {

    private static final List<Producto> PRODUCTOS = List.of(
            new Producto(1L, "Laptop", 1200.00),
            new Producto(2L, "Smartphone", 800.00),
            new Producto(3L, "Laptop ASUS", 1200.00),
            new Producto(4L, "Smartphone Samsung", 800.00)
    );

    @GetMapping("/{id}")
    public Producto obtenerProducto(@PathVariable Long id) {
        return PRODUCTOS.stream()
                .filter(p -> p.id().equals(id))
                .findFirst()
                .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND));
    }

    @GetMapping("/productos")
    public List<Producto> listarProductos() {
        return List.of(
                new Producto(1L, "Laptop", 1500.0),
                new Producto(2L, "Mouse", 25.0),
                new Producto(3L, "Laptop ASUS", 1200.00),
                new Producto(4L, "Smartphone Samsung", 800.00)
        );
    }
}
```

Record Producto

```
public record Producto(Long id, String nombre, Double precio) {}
```

Back

Next

6. Configuración de pedido-service

Dependencias Maven

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

application.yml

```yaml
server:
  port: 8082

spring:
  application:
    name: pedido-service

producto-service:
  url: http://localhost:8081

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
    lease-renewal-interval-in-seconds: 10
    lease-expiration-duration-in-seconds: 30

  instance:
    prefer-ip-address: true
```

Habilitar Feign y Eureka Client

```java
@SpringBootApplication
@EnableFeignClients
@EnableDiscoveryClient
public class PedidoServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(PedidoServiceApplication.class, args);
    }
}
```

Feign Client Sin URL

```java
@FeignClient(name = "producto-service")
public interface ProductoClient {

    @GetMapping("/productos")
    List<ProductoDTO> obtenerProductos();
}
```

Nota: **NO necesitas poner URL fija. Eureka resuelve la dirección.**

Controlador

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

    private final ProductoClient productoClient;

        @Autowired
    public PedidoController(ProductoClient productoClient) {
        this.productoClient = productoClient;
    }

    @GetMapping("/crear")
    public ResponseEntity<?> crearPedido() {
        List<ProductoDTO> productos = productoClient.obtenerProductos();
        return ResponseEntity.ok("Pedido creado con productos: " + productos);
    }
}
```

El objeto de transferencia ProductoDTO

```java
@Getter
@Setter
public class ProductoDTO {
    private Long id;
    private String nombre;
    private Double precio;

    @Override
    public String toString() {
        return "ProductoDTO{" +
                "id=" + id +
                ", nombre='" + nombre + '\'' +
                ", precio=" + precio +
                '}';
    }
}
```

Entidades de pedido

Aunque no se usan, se agregan aqui como referencia (muy basica).

El record PedidoRequest

```java
public record PedidoRequest(Long productoId, int cantidad) {}
```

El record Pedido

```java
public record Pedido(String id, ProductoDTO producto, int cantidad) {}
```

Back Next

7. Prueba de funcionamiento

Pasos:

1. Levantar **Eureka Server** en `localhost:8761`.
2. Levantar **producto-service** en puerto `8081`.
3. Levantar **pedido-service** en puerto `8082`.
4. Verificar en la consola Eureka (`http://localhost:8761`), ambos microservicios deben aparecer como **UP**.
5. Hacer GET en:

```
http://localhost:8082/pedidos/crear
```

Deberías obtener:

```
Pedido creado con productos: [ProductoDTO{id=1, nombre='Laptop', precio=1500.0
```

8. Aclaraciones finales

Beneficios de Eureka

- **Ya no necesitas configurar la URL manualmente en pedido-service.**
- Si despliegas en otro entorno, Eureka sabe la IP/puerto actual.
- Si tienes múltiples instancias de producto-service, Eureka hace load balancing (con Spring Cloud LoadBalancer o Ribbon si lo configuras).

¿Qué hace OpenFeign en este caso?

- OpenFeign consulta a Eureka para saber la dirección de `producto-service`.
- Abstrae el HTTP REST, serializa/deserializa objetos y maneja la conexión por ti.