

Microservicios – Arquitectura y Desarrollo

LAB Microservicio tolerante a fallos con Netflix Hystrix

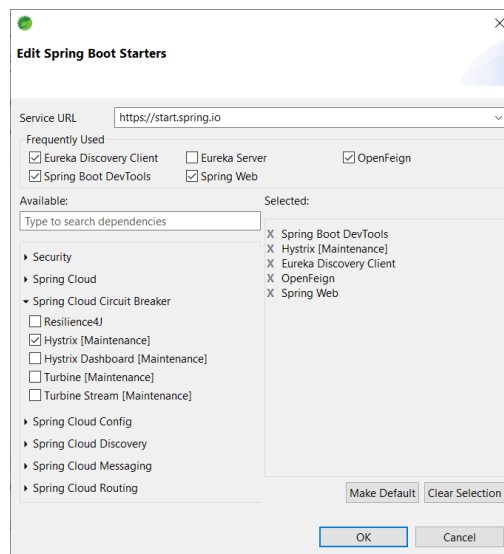
Objetivos

- Mostrar al participante el procedimiento para el desarrollo de Microservicios tolerantes a fallos con Netflix Hystrix.

Procedimiento

- En el proyecto **springboot-servicio-items**, agrega la dependencia Hystrix.

Sobre el proyecto Clic Derecho > Spring > Edit Starters.

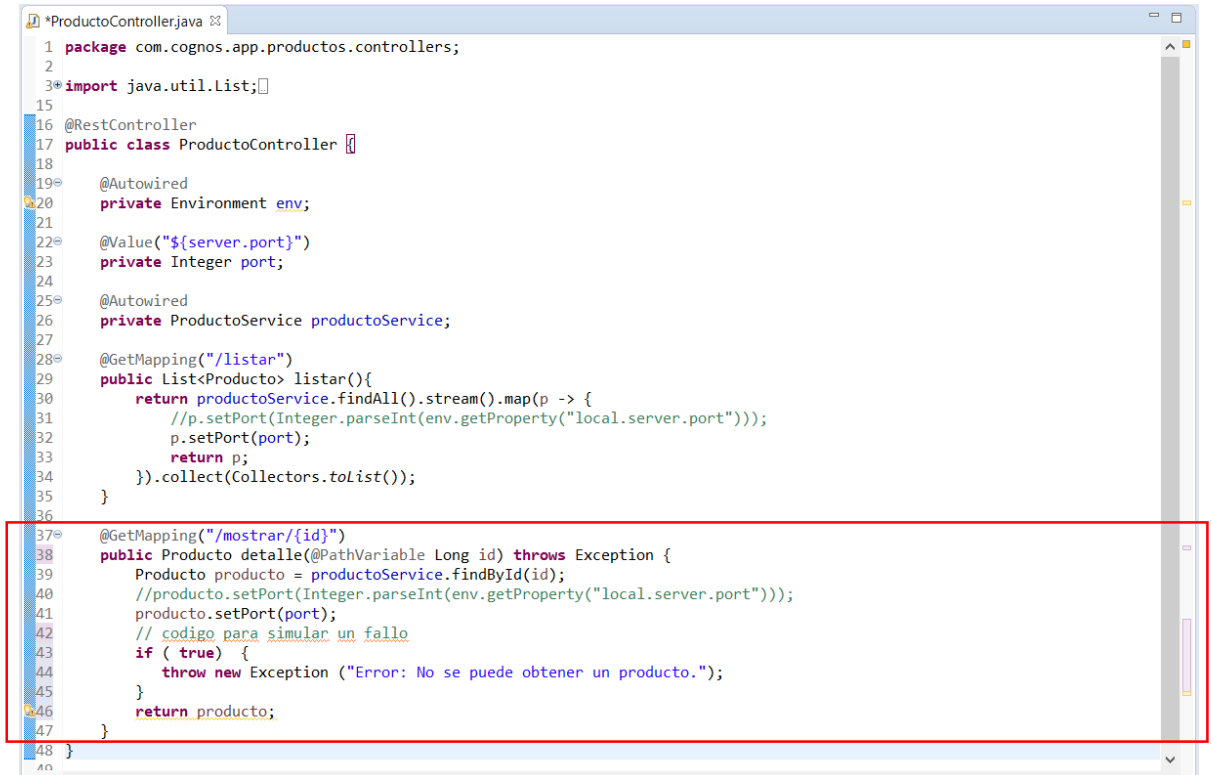


- Habilita el uso del patrón circuit breaker en el proyecto. Agrega la anotación **@EnableCircuitBreaker** a la clase **SpringbootServicioItemApplication**.

```
SpringbootServicioItemsApplication.java
1 package com.cognos.app.items;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @EnableCircuitBreaker
7 @EnableEurekaClient
8 @EnableFeignClients
9 @SpringBootApplication
10 public class SpringbootServicioItemsApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(SpringbootServicioItemsApplication.class, args);
14     }
15 }
16
17
18
19
20
21
```

3. Modifica la clase `ProductoController.java` del servicio `SERVICIO-PRODUCTOS` para simular un error. Agrega al método `detalle()` el siguiente código.

```
if ( true) {  
    throw new Exception ("Error: No se puede obtener un producto.");  
}
```



```
*ProductoController.java  
1 package com.cognos.app.productos.controllers;  
2  
3 import java.util.List;  
4  
5  
6 @RestController  
7 public class ProductoController {  
8  
9     @Autowired  
10    private Environment env;  
11  
12    @Value("${server.port}")  
13    private Integer port;  
14  
15    @Autowired  
16    private ProductoService productoService;  
17  
18    @GetMapping("/listar")  
19    public List<Producto> listar(){  
20        return productoService.findAll().stream().map(p -> {  
21            //p.setPort(Integer.parseInt(env.getProperty("local.server.port")));  
22            p.setPort(port);  
23            return p;  
24        }).collect(Collectors.toList());  
25    }  
26  
27    @GetMapping("/mostrar/{id}")  
28    public Producto detalle(@PathVariable Long id) throws Exception {  
29        Producto producto = productoService.findById(id);  
30        //producto.setPort(Integer.parseInt(env.getProperty("local.server.port")));  
31        producto.setPort(port);  
32        // codigo para simular un fallo  
33        if ( true) {  
34            throw new Exception ("Error: No se puede obtener un producto.");  
35        }  
36        return producto;  
37    }  
38 }  
39  
40
```

4. En el proyecto `springboot-servicio-items`, edita la clase `ItemController.java`, agrega la anotación `@HystrixCommand(fallbackMethod = "metodoAlternativo")` al método `detalle()` e implementa el método `metodoAlternativo()`.

```
package com.cognos.app.items.controllers;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.cognos.app.items.model.Item;  
import com.cognos.app.items.model.Producto;
```

```

import com.cognos.app.items.model.service.ItemService;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController
public class ItemController {
    @Autowired
    @Qualifier("serviceRestTemplate")
    private ItemService itemService;

    @GetMapping("/listar")
    public List<Item> listar(){
        return itemService.findAll();
    }

    @HystrixCommand(fallbackMethod = "metodoAlternativo")
    @GetMapping("/mostrar/{id}/cantidad/{cantidad}")
    public Item detalle(@PathVariable Long id,@PathVariable Integer cantidad) {
        return itemService.findById(id, cantidad);
    }

    public Item metodoAlternativo(Long id, Integer cantidad) {
        Item item = new Item();
        Producto producto = new Producto();
        producto.setId(id);
        producto.setNombre("(Promocion) USB 32 GB");
        producto.setPrecio(30.0);
        item.setProducto(producto);
        item.setCantidad(cantidad);
        return item;
    }
}

```

5. Inicia los servicios y prueba la implementación con Postman. En primer lugar, inicia el servidor Eureka y luego los otros servicios.


```

@RestController
public class ProductoController {

    @Autowired
    private Environment env;

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

    @Autowired
    private ProductoService productoService;

    @GetMapping("/listar")
    public List<Producto> listar(){
        return productoService.findAll().stream().map(p -> {

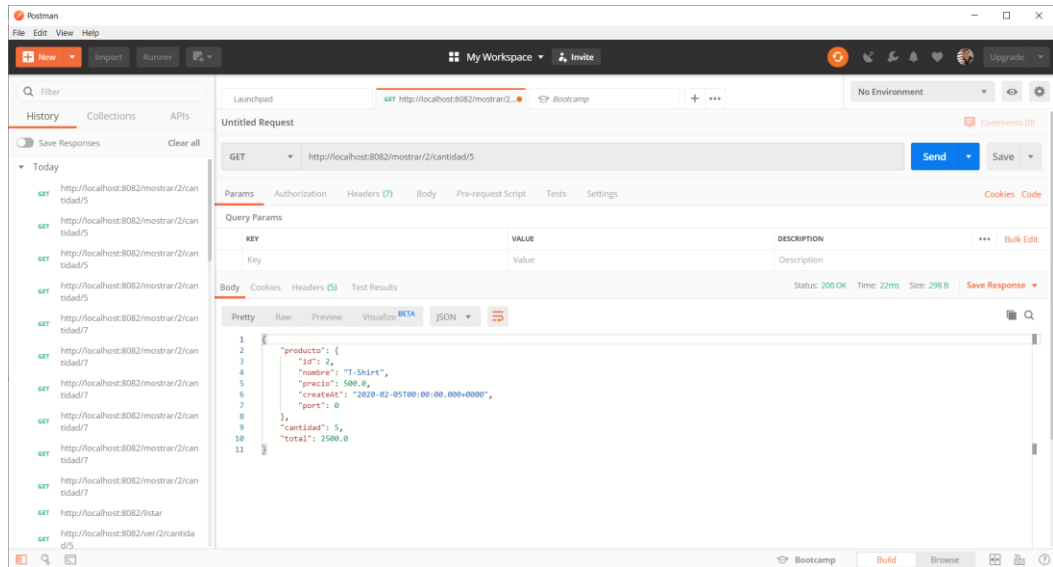
            //p.setPort(Integer.parseInt(env.getProperty("local.server.port")));
            p.setPort(port);
            return p;
        }).collect(Collectors.toList());
    }

    @GetMapping("/mostrar/{id}")
    public Producto detalle(@PathVariable Long id) {
        Producto producto = productoService.findById(id);
        //producto.setPort(Integer.parseInt(env.getProperty("local.server.port")));
        producto.setPort(port);
        // codigo para simular un fallo
        //if ( true) {
        //    throw new Exception ("Error: No se puede obtener un producto.");
        // }
        return producto;
    }
}

```

Guarda los cambios.

8. Verifica con Postman que el servicio **SERVICIO-PRODUCTOS** funciona correctamente.



- Realiza una segunda prueba, detén el servicio **SERVICIO-PRODUCTOS**. Verifica que el servicio **SERVICIO-ITEMS** está manejando los errores, consume el servicio **SERVICIO-ITEMS** con Postman y verifica que usa el método alternativo para retornar el producto en oferta.

