

Microservicios – Arquitectura y Desarrollo

LAB Configurando Tiempo de Espera en Microservicios con Netflix Hystrix

Objetivos

- Mostrar al participante el procedimiento para configurar tiempos de espera (**timeout**) en Microservicios con Hystrix y Ribbon.

Procedimiento

1. En el proyecto **springboot-servicio-productos** edita la clase ProductoController, agrega una espera de 3000 ms en el método **detalle()**.

```
package com.cognos.app.productos.controllers;

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.env.Environment;
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.productos.model.entity.Producto;
import com.cognos.app.productos.model.service.ProductoService;

@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.");
    } */
    try {
        Thread.sleep(3000L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return producto;
}
}

```

Nota: El tiempo máximo de espera por una respuesta de un microservicio en Spring Cloud (Hystrix y Ribbon) es de 1 s, el método **detalle()** se detiene por 3 s, esto ocasionara un error en el microservicio cliente.

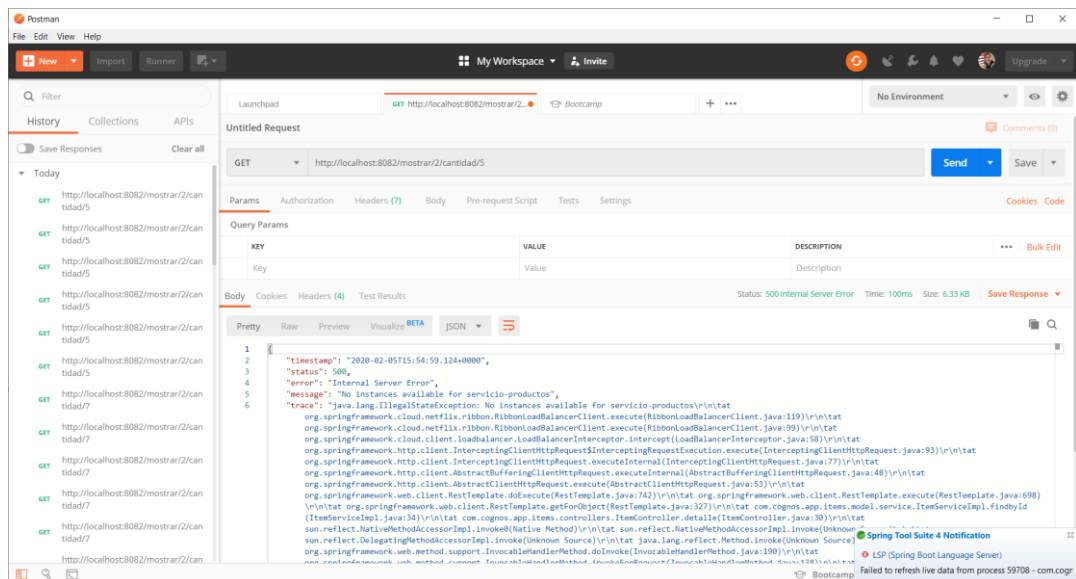
2. Para realizar una prueba, vamos a desactivar la tolerancia a fallos. En el proyecto **springboot-servicio-items**, edita la clase **ItemController** y comenta la línea:
@HystrixCommand(fallbackMethod = "**metodoAlternativo**")

```

1 package com.cognos.app.items.controllers;
2
3 import java.util.List;
4
5 @RestController
6 public class ItemController {
7     @Autowired
8     @Qualifier("serviceRestTemplate")
9     private ItemService itemService;
10
11     @GetMapping("/listar")
12     public List<Item> listar(){
13         return itemService.findAll();
14     }
15
16     // @HystrixCommand(fallbackMethod = "metodoAlternativo")
17     @GetMapping("/mostrar/{id}/cantidad/{cantidad}")
18     public Item detalle(@PathVariable Long id, @PathVariable Integer cantidad) {
19         return itemService.findById(id, cantidad);
20     }
21
22     public Item metodoAlternativo(Long id, Integer cantidad) {
23         Item item = new Item();
24         Producto producto = new Producto();
25         producto.setId(id);
26         producto.setNombre("(Promocion) USB 32 GB");
27         producto.setPrecio(30.0);
28         item.setProducto(producto);
29         item.setCantidad(cantidad);
30         return item;
31     }
32 }

```

3. Inicia los servicios y Prueba con Postman que se genera un error cuando se consume el servicio **SERVICIO-PRODUCTOS** desde **SERVICIO-ITEMS**.



4. En el proyecto **springboot-servicio-items**, edita la clase **ItemController** y des comenta la línea: `//@HystrixCommand(fallbackMethod = "metodoAlternativo")` usando Postman consume **SERVICIO-ITEMS** y verifica que usa el método alternativo.

```

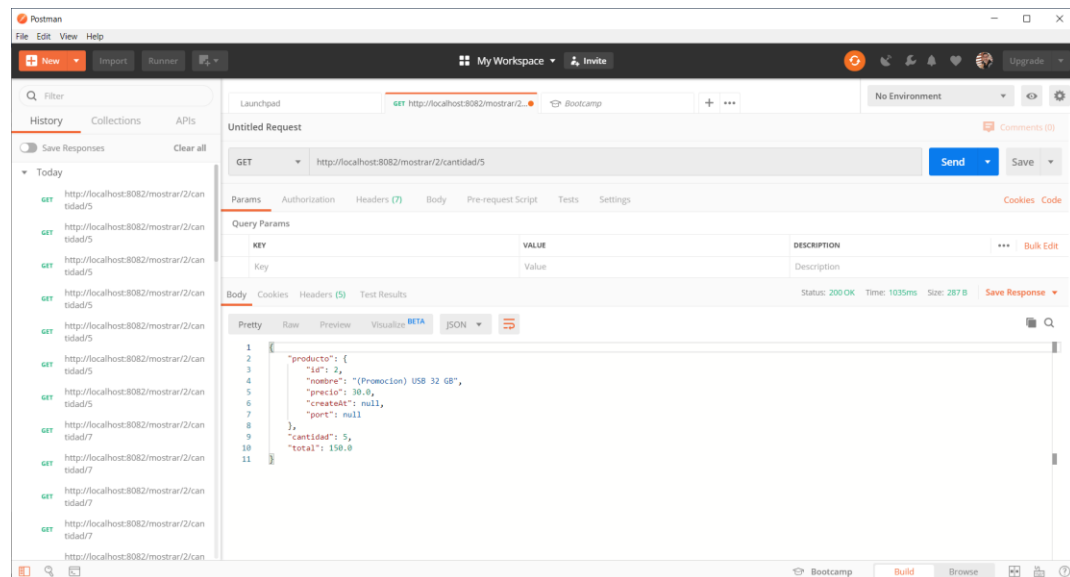
1 package com.cognos.app.items.controllers;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16 @RestController
17 public class ItemController {
18     @Autowired
19     @Qualifier("serviceRestTemplate")
20     private ItemService itemService;
21
22     @GetMapping("/listar")
23     public List<Item> listar(){
24         return itemService.findAll();
25     }
26
27     @HystrixCommand(fallbackMethod = "metodoAlternativo")
28     @GetMapping("/mostrar/{id}/cantidad/{cantidad}")
29     public Item detalle(@PathVariable Long id,@PathVariable Integer cantidad) {
30         return itemService.findById(id, cantidad);
31     }
32
33
34     public Item metodoAlternativo(Long id, Integer cantidad) {
35         Item item = new Item();
36         Producto producto = new Producto();
37         producto.setId(id);
38         producto.setNombre("(Promocion) USB 32 GB");
39         producto.setPrecio(30.0);
40         item.setProducto(producto);
41         item.setCantidad(cantidad);
42         return item;
43     }
44 }
45
46

```

Abre Postman y realiza una petición (request) a:

<http://localhost:8082/mostrar/2/cantidad/5>

Nota: Debes observar que la respuesta es generada por el **metodoAlternativo()**, esto es porque Hystrix está activo.

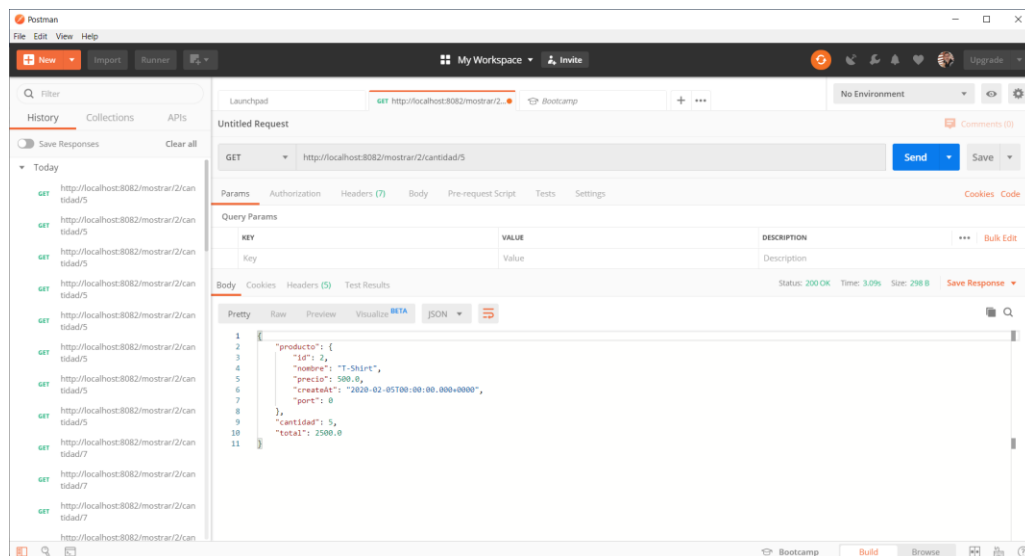


5. Vamos a activar un tiempo de espera en el cliente **SERVICIO-ITEMS**. En el proyecto **springboot-servicio-items** edita el archivo **application.properties** y define el tiempo de espera para Hystrix y Ribbon.

```
application.properties
1 spring.application.name=servicio-items
2 server.port=8082
3
4 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5
6 hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 20000
7 ribbon.ConnectTimeout: 5000
8 ribbon.ReadTimeout: 10000
```

Nota: Recuerda que Hystrix envuelve a Ribbon por lo tanto el tiempo de espera de Hystrix debe ser mayor que el de Ribbon. Esta configuración para Yaml la puedes obtener desde: <https://cloud.spring.io/spring-cloud-static/spring-cloud-netflix/2.2.1.RELEASE/reference/html/#uploading-files-through-zuul>

6. Guarda los cambios y espera a que se actualice el entorno de ejecución y realiza un prueba con Postman, debes que comprobar que a pesar de la espera se recibe la respuesta correcta.



7. En el proyecto springboot-servicio-productos, edita la clase **ProductoController** y comenta el código de la espera.

```
package com.cognos.app.productos.controllers;

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.env.Environment;
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.productos.model.entity.Producto;
```

```

import com.cognos.app.productos.model.service.ProductoService;

@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 -> {

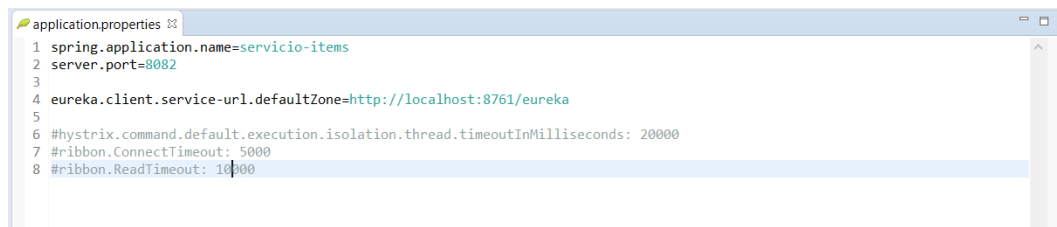
            //p.setPort(Integer.parseInt(env.getProperty("local.server.port")));
            p.setPort(port);
            return p;
        }).collect(Collectors.toList());
    }

    @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.");
        * }
        */

        /*
        * try { Thread.sleep(3000L); } catch (InterruptedException e) {
        * e.printStackTrace(); }
        */
        return producto;
    }
}

```

1. En el proyecto **springboot-servicio-items**, edita el archivo **application.properties** y comenta la definición de espera de Hystrix y Ribbon.



```

application.properties
1 spring.application.name=servicio-items
2 server.port=8082
3
4 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5
6 #hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 20000
7 #ribbon.ConnectTimeout: 5000
8 #ribbon.ReadTimeout: 10000

```

2. Guarda los cambio y verifica usando Postman que todo funcione correctamente.

