

Respuesta Prueba Técnica Java Backend Developer

Lorenzo Bárzaga Meriño

Para la realización de la Prueba Técnica se crearon 5 microservicios, usando maven, java 8 y base de datos en memoria H2

- **NamingService**
- **GatewayService**
- **CloudConfigServer**
- **WasteManagerAddressService**
- **WasteManagerService**

Microservicio NamingService: es un microservicio que funciona como un servidor y es capaz de gestionar el registro y búsqueda de microservicios que están en nuestro desarrollo por medio de un nombre, al mismo tiempo cada microservicio está enviando información del estado del mismo, si está funcionando o está bajo o detenido, este microservicio ofrece un dashboard donde se puede consultar esta información, para hacer uso de esta característica del proyecto spring cloud utilizamos la dependencia maven “spring-cloud-starter-netflix-eureka-server” como se indica a continuación

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


Además se hace necesario las siguientes configuraciones en el archivo application.properties como se muestra en la imagen

```
#Eureka Server
server.port=8761
#Evitar que el mismo microservicio se intente registrar como cliente de Eureka
eureka.client.register-with-eureka=false
#Especificamos a los clientes que no se guarden en su cache local las direcciones de los diferentes instancias
eureka.client.fetch-registry=false
```

Debe anotarse la clase principal del proyecto con @EnableEurekaServer para el correcto funcionamiento

```
@SpringBootApplication
@EnableEurekaServer
public class NamingService {
```

Consola de EurekaServer a través del microservicio NamingService



HOME

LAST 1000 SINCE STARTUP

System Status

Environment	N/A	Current time	2024-04-16T19:31:14 -0400
Data center	N/A	Uptime	00:02
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	3

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - gateway-service:f7532c04-5ca6-4584-9b21-0f7b7bb1fc52
WASTE-MANAGER	n/a (1)	(1)	UP (1) - waste-manager:177bf9e0-3922-4047-a23e-ca733c93431b
WASTE-MANAGER-ADDRESS	n/a (1)	(1)	UP (1) - waste-manager-address:3577bf56-d5ab-496f-a4dc-9c5376ee7df5

Microservicio GatewayService: es un microservicio que funciona como puerta de enlace, interceptando las llamadas al api y enrutandolas hacia el microservicio WasteManagerService, así como encargarse del balanceo de carga, para proveer esta característica se usó la dependencia maven “spring-cloud-starter-gateway” del proyecto spring cloud, como se indica en la imagen a continuación.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
  <version>3.0.0</version>
</dependency>
```

Así como además de las configuraciones que se indican en el archivo application.properties

```
#Gateway#
#Es similar a usar la anotacion @EnableDiscoveryServer
spring.cloud.gateway.discovery.locator.enabled=true
#Desabilita la estrategia predeterminada de balanceo de carga
spring.cloud.loadbalancer.ribbon.enabled=false
#Configurar el ruteo dinamico
spring.cloud.gateway.routes[0].id= waste-manager
spring.cloud.gateway.routes[0].uri=lb://WASTE-MANAGER
spring.cloud.gateway.routes[0].predicates[0]=Path=/wasteManager/**
```

Para que este esté microservicio pueda participar en la arquitectura que se utiliza en este desarrollo se hace necesario el uso de la dependencia “spring-cloud-starter-netflix-eureka-client” del proyecto spring cloud para que se convierta en cliente del **servidor eureka** que se crea por medio del microservicio “NamingService”

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>3.0.5</version>
</dependency>
```

Además se hace uso de las configuraciones en el archivo application.properties como muestra la imagen

```
#Application name
spring.application.name=gateway-service
#Eureka Client
eureka.instance.instance-id=${spring.application.name}:${random.uuid}
```

Para el correcto funcionamiento del proyecto se debe anotar la clase principal con **@EnableEurekaClient**

```
@SpringBootApplication
@EnableEurekaClient
public class GatewayService {
```

Microservicio CloudConfigServer: es un microservicio que funciona como un servidor de recursos, con una configuración centralizada, nos da la facilidad de cargar cualquier cambio realizado en las properties de nuestros microservicios sin la necesidad de detener y recompilarlos nuevamente, para proveer esta característica primeramente debemos tener las configuraciones de los archivos application.properties en un repositorio remoto de donde se servirá el ConfigServer, planteado esto tenemos un repositorio github en esta dirección <https://github.com/lbarzagam/service-configuration>, nos interesa los archivos [waste-manager-dev.properties](#) y [waste-manager-address-dev.properties](#), que son los que configuramos para esta implementación y corresponden con los microservicios que tenemos para acceso a datos, para poder hacer uso del ConfigServer debemos tener configurado en nuestro pom.xml las dependencias de maven “spring-boot-starter-actuator” y “spring-cloud-config-server”, la primera para poder refrescar alguna configuración que se suba al repositorio sin necesidad de parar la ejecución del microservicio y la segunda para crear nuestro ConfigServer

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

Así mismo se crearon las siguientes configuraciones en el archivo application.properties

```
spring.application.name=cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri=https://github.com/lbarzagam/service-configuration
#Clonar al iniciar
spring.cloud.config.server.git.clone-on-start=true
```

Para el correcto funcionamiento del proyecto se debe anotar la clase principal con @EnableConfigServer

```
@SpringBootApplication
@EnableConfigServer
public class CloudConfigServerApplication {
```

MicroServicio WasteManagerAddressService: es un microservicio el cual se basa en crear un CRUD de la entidad **WasteManagerAddressEntity** y que sea accedido desde el microservicio **WasteManagerService** haciendo uso de los endpoint descritos en el controlador WasteManagerAdressRestController, este microservicio utiliza una base de datos H2, además debe tomar la configuración centralizada que la brinda el microservicio **CloudConfigServer**, a su vez debe registrarse como cliente del servidor Eureka el cual se crea a partir del microservicio **NamingService**, para proveer estas características se hace necesario el uso de las dependencias de maven “spring-boot-starter-data-jpa” para implementar el repositorio en tiempo de ejecución, “spring-boot-devtools” conjunto de herramientas que ayudan a los desarrolladores hacia la creación de aplicaciones Spring y “H2” para utilizar las bases de datos H2

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>

```

Se hace necesario la configuración en el archivo application.properties

```

# Enable show Sql
spring.jpa.show-sql = true

# Enabling H2 Console
spring.datasource.url= jdbc:h2:mem:pruebatecnicadb
spring.datasource.driverClassName= org.h2.Driver
spring.datasource.username= sa
spring.datasource.password= password
spring.jpa.database-platform= org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.h2.console.enabled= true
# Enable Web Access
spring.h2.console.settings.web-allow-others=true

```

Además se hace necesario agregar las siguientes dependencias “spring-cloud-starter-netflix-eureka-client” para hacer el microservicio cliente del servidor eureka y pueda ser descubierto y registrado por el microservicio **NamingService**

```

<!-- Para que el microservicio se registre en eureka server -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>3.0.5</version>
</dependency>

```

“spring-cloud-config-client” para hacer que el microservicio obtenga la configuración centralizada que brinda el microservicio **CloudConfigServer**

```

<!-- Para convertir el microservicio en cliente de config server -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-client</artifactId>
  <version>3.0.6</version>
</dependency>

```

“spring-cloud-starter-bootstrap” para cargar la configuración en el ConfigServer cuando el microservicio inicie

```

<!-- Para hacer un fetch en el config server cuando el microservicio inicie -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bootstrap</artifactId>
  <version>3.0.0</version>
</dependency>

```

Se hace necesario la creación del archivo bootstrap.properties y las configuraciones siguientes

```
#Application name
spring.application.name=waste-manager-address
#URI donde esta corriendo el servidor de Config Server
spring.cloud.config.uri=http://localhost:8888
#Cargar el perfil dev
spring.cloud.config.profile=dev

#Nombre random de la instancia del microservicio
eureka.instance.instance-id=${spring.application.name}:${random.uuid}
```

Para el correcto funcionamiento del proyecto se debe anotar la clase principal con @EnableEurekaClient

```
@SpringBootApplication
@EnableEurekaClient
public class WasteManagerAddressServiceApplication {
```

Microservicio WasteManagerService: este microservicio se encargará de recibir las peticiones propias y las del microservicio **WasteManagerAddressService** haciendo uso de la creación de una interface cliente “WasteManagerAddressClient” y anotándola con @FeignClient, se le debe proporcionar el nombre (“waste-manager-address”) con el cual se registra el microservicio en EurekaServer, se debe utilizar la dependencia de spring “spring-cloud-starter-openfeign”, hay múltiples opciones para llamadas api dentro de microservicios, la primera es RestTemplate o un WebClient pero no cuentan con balanceo de carga, la mejor opción es usar feignClient, que ya viene con un mecanismo de balanceo de carga.

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
  <version>3.0.6</version>
</dependency>
```

Se debe hacer uso de la anotación @FeignClient

```
@FeignClient(name = "waste-manager-address")
public interface WasteManagerAddressClient {
```

Este microservicio utiliza una base de datos H2, además debe tomar la configuración centralizada que la brinda el microservicio **CloudConfigServer**, a su vez debe registrarse como cliente del servidor Eureka el cual se crea a partir del microservicio **NamingService**, para proveer estas características se hace necesario el uso de las dependencias de maven “spring-boot-starter-data-jpa” para implementar el repositorio en tiempo de ejecución, “spring-boot-devtools” conjunto de herramientas que ayudan a los desarrolladores hacia la creación de aplicaciones Spring y “H2” para utilizar las bases de datos H2

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>

```

Se hace necesario la configuración en el archivo application.properties

```

# Enable show Sql
spring.jpa.show-sql = true

# Enabling H2 Console
spring.datasource.url= jdbc:h2:mem:pruebatecnicadb
spring.datasource.driverClassName= org.h2.Driver
spring.datasource.username= sa
spring.datasource.password= password
spring.jpa.database-platform= org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.h2.console.enabled= true
# Enable Web Access
spring.h2.console.settings.web-allow-others=true

```

Además se hace necesario agregar las siguientes dependencias “spring-cloud-starter-netflix-eureka-client” para hacer el microservicio cliente del servidor eureka y pueda ser descubierto y registrado por el microservicio **NamingService**

```

<!-- Para que el microservicio se registre en eureka server -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>3.0.5</version>
</dependency>

```

“spring-cloud-config-client” para hacer que el microservicio obtenga la configuración centralizada que brinda el microservicio **CloudConfigServer**

```

<!-- Para convertir el microservicio en cliente de config server-->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-client</artifactId>
  <version>3.0.6</version>
</dependency>

```

“spring-cloud-starter-bootstrap” para cargar la configuración en el ConfigServer cuando el microservicio inicie

```

<!-- Para hacer un fetch en el config server cuando el microservicio inicie -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bootstrap</artifactId>
  <version>3.0.0</version>
</dependency>

```

Se hace necesario la creación del archivo bootstrap.properties y las configuraciones siguientes

```
#Application name
spring.application.name=waste-manager

#Eureka Client
eureka.instance.instance-id=${spring.application.name}:${random.uuid}

#URI como cliente de Config Server
spring.cloud.config.uri=http://localhost:8888
#Cargar el perfil dev
spring.cloud.config.profile=dev
```

Para el correcto funcionamiento del proyecto se debe anotar la clase principal con @EnableEurekaClient y con @EnableFeignClients para hacer llamadas al otro microservicio

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class WasteManagerServiceApplication {
```

Para poder hacer uso de esta arquitectura de microservicios hay que clonar el repositorio de GitHub en esta dirección https://github.com/lbarzagam/prueba_tecnica , el orden para la ejecución de los microservicios es el siguiente:

- **NamingService**
- **GatewayService**
- **CloudConfigServer**
- **WasteManagerAddressService**
- **WasteManagerService**