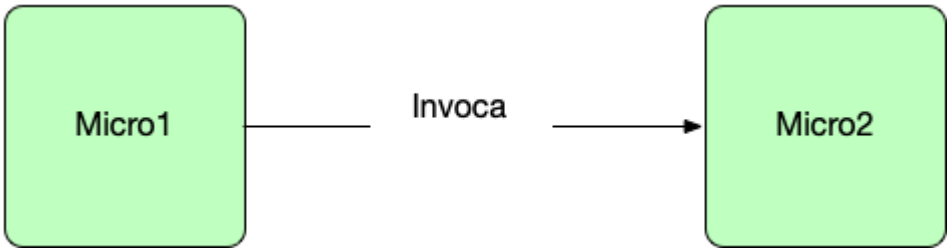
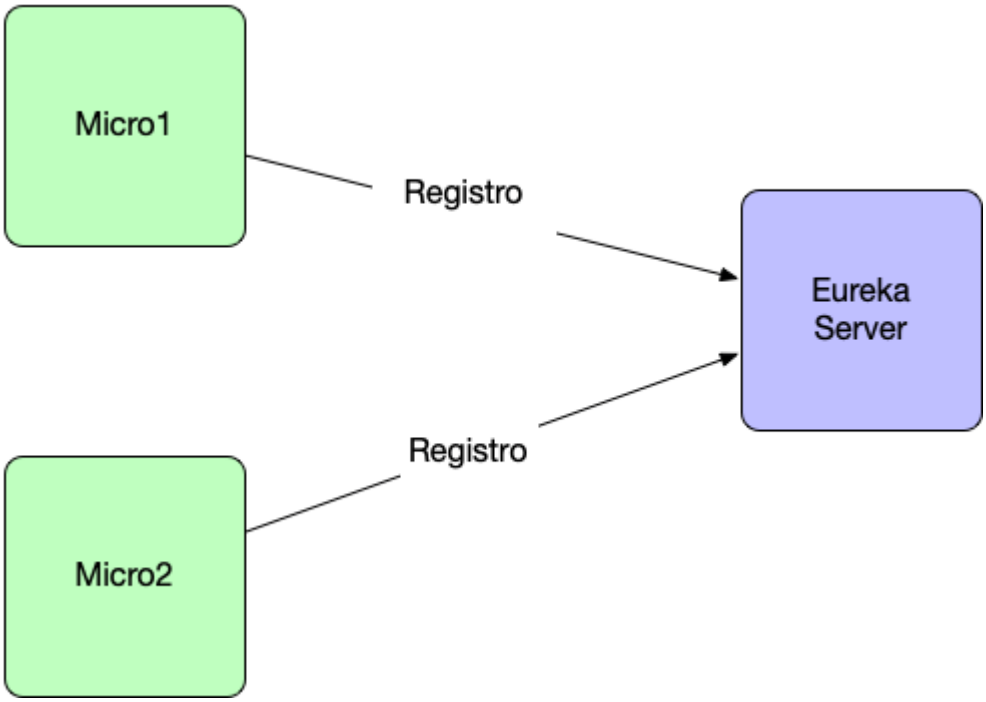
El concepto de Spring Cloud Eureka Server y sus capacidades de Discovery es una de las características principales de Spring Cloud a la hora de gestionar el registro y búsqueda de MicroServicios. Cuando tenemos varios MicroServicios es muy común querer invocar de unos a otros .



El gran problema le tenemos con que nombre tiene la maquina / el contenedor o el pod en el que se esta ejecutando . Para simplificar esta problemática Spring Cloud soporta Eureka Server que no es ni más ni menos que un servidor que se encarga de levantarse y admite el registro de MicroServicios (aplicaciones de Spring Boot) en el para su posterior consulta.



Spring Cloud Eureka Server

Vamos a configurar un proyecto de Srpring Boot para que sirva de servidor de Eureka . Para

ello las operaciones que tenemos que realizar no son muy complicadas. Ya que basta con crear un proyecto que incluye la dependencia de eureka server.

```
<dependencies>
		<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-server</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-
test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
	<dependencyManagement>
		<dependencies>
			<dependency>
<groupId>org.springframework.cloud</groupId>
				<artifactId>spring-cloud-
dependencies</artifactId>
				<version>${spring-
cloud.version}</version>
				<type>pom</type>
```

```
                                <scope>import</scope>
                        </dependency>
                </dependencies>
        </dependencyManagement>

        <build>
                <plugins>
                        <plugin>
<groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>netflix-candidates</id>
                        <name>Netflix Candidates</name>
<url>https://artifactory-oss.prod.netflix.net/artifactory/maven-oss-ca
ndidates</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </repository>
        </repositories>
```

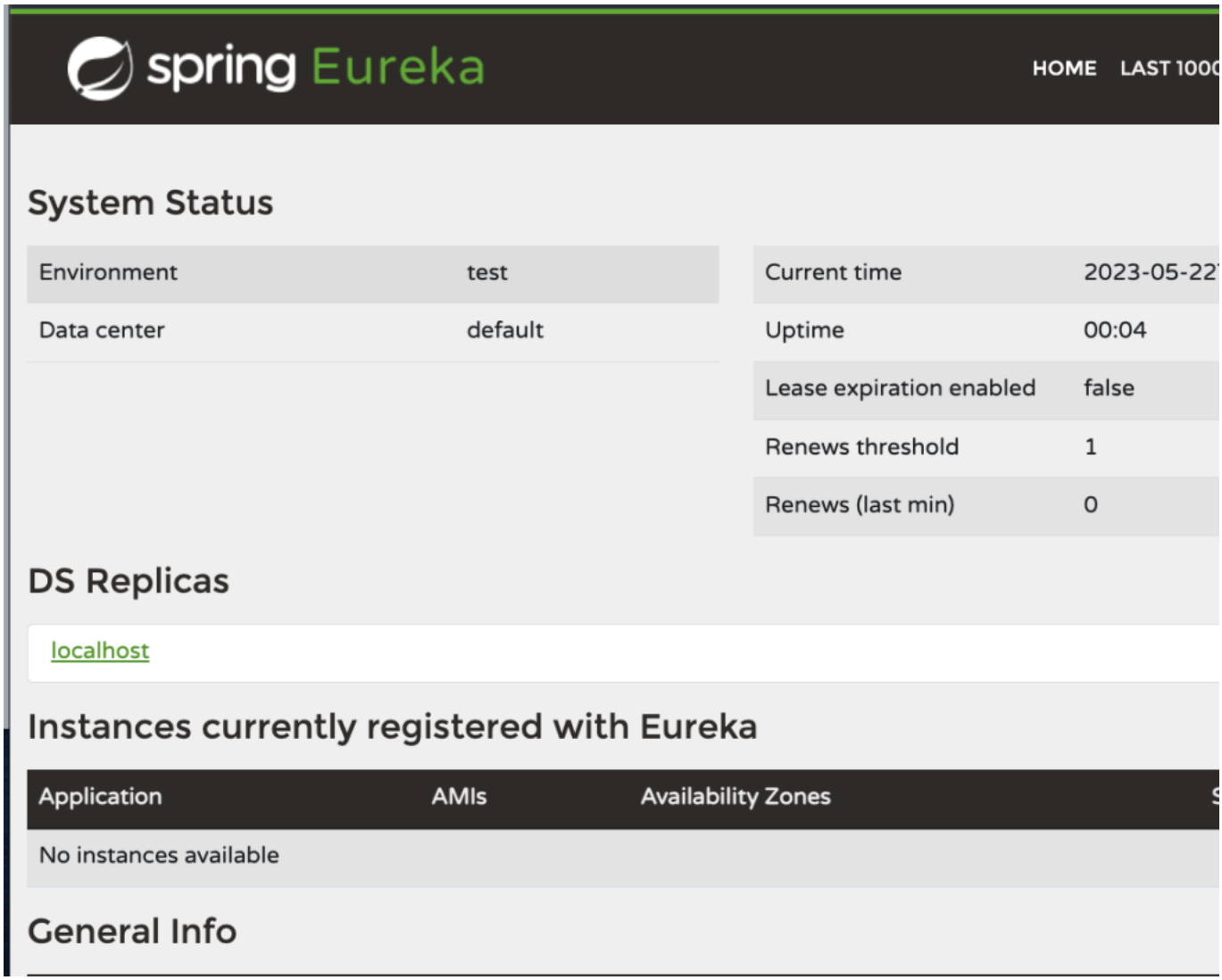
Una vez que tenemos el proyecto descargado es momento de modificar el application.properties para configurar los parámetros por omisión de Eureka Server. Siempre se necesita un puerto y evitar que el propio servidor de eureka intente registrarse el mismo. Con esto suele ser a día de hoy suficiente ya que al tener el starter de eureka lo demás lo configura Spring.

no se

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Spring Cloud Eureka Starting

Con esto configurado es momento de arrancar el proyecto de Spring Boot y Eureka como servidor se dará de alta de forma automática.



Registro MicroServicios

Es momento ahora de construir un sencillo MicroServicios que nos devuelva un mensaje de Hola e intentar que se registre en Eureka para su posterior búsqueda. Este caso será suficiente con añadir el Starter de cliente de Eureka.

```
<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-
web</artifactId>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework.cloud/spring-cl
oud-starter-netflix-eureka-client -->
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-
eureka-client</artifactId>

		</dependency>


		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-
test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
	<dependencyManagement>
```

```
		<dependencies>
			<dependency>
<groupId>org.springframework.cloud</groupId>
				<artifactId>spring-cloud-
dependencies</artifactId>
				<version>${spring-
cloud.version}</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
	</dependencyManagement>

	<build>
		<plugins>
			<plugin>
<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-
plugin</artifactId>
			</plugin>
		</plugins>
	</build>
	<repositories>
		<repository>
			<id>netflix-candidates</id>
			<name>Netflix Candidates</name>
<url>https://artifactory-oss.prod.netflix.net/artifactory/maven-oss-ca
ndidates</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
```

```
            </repository>
        </repositories>
```

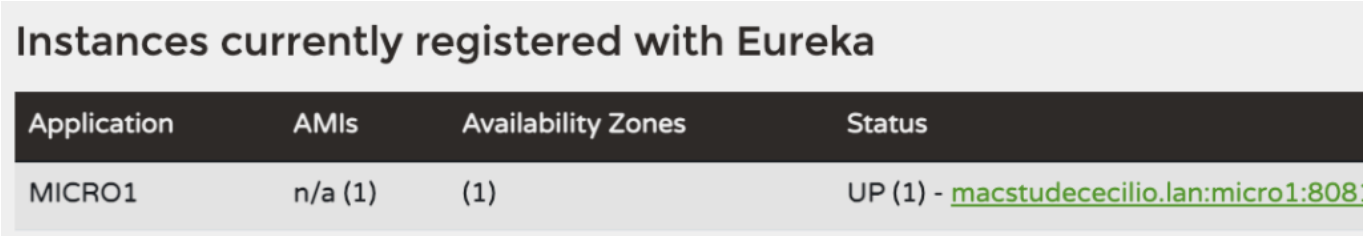
Realizada esta operación el siguiente paso es ver que se necesita en el applicación.properties para que la aplicación funcione de forma correcta sobre Eureka Server y se registre.

```
server.port=8081
spring.application.name=micro1
eureka.client.register-with-eureka= true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

En este caso le asignamos el puerto 8081 y le damos un nombre de registro , junto con dos lineas adicionales que registran en el servidor Eureka que estamos utilizando. Eso sí antes de levantar el MicroServicio vamos a implementar una funcionalidad básica que pueda ser invocada.

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaController {

    @GetMapping("/hola")
    public String hola() {

        return "hola";
    }

}
```

Es momento de arrancar el servicio y ver como el servidor de Eureka lo registra con el nombre que tenemos escrito en el application.properties de Spring Boot.



Invocando otro Servicio con Eureka

Una vez tenemos registrado el primer servicio podemos construir un segundo servicio que llame al primero . Dispondrá de la misma estructura a nivel de POM pero realizara operaciones un poco diferentes ya que el no tendrá tanta funcionalidad sino que delegará esa funcionalidad en el MicroServicio ya registrado. Vamos a ver su application.properties , evidentemente al estar en un entorno local tendrá otro puerto.

```
server.port=8082
spring.application.name=micro2
eureka.client.register-with-eureka= true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

Una vez tenemos el fichero de properties . El siguiente paso es Registrar un RestTemplate de una forma un poco más peculiar para que nos asegure que invoca al servidor Eureka de Registro. Como vemos usa la anotación @LoadBalanced que permite que nos apoyemos en el servicio de registro

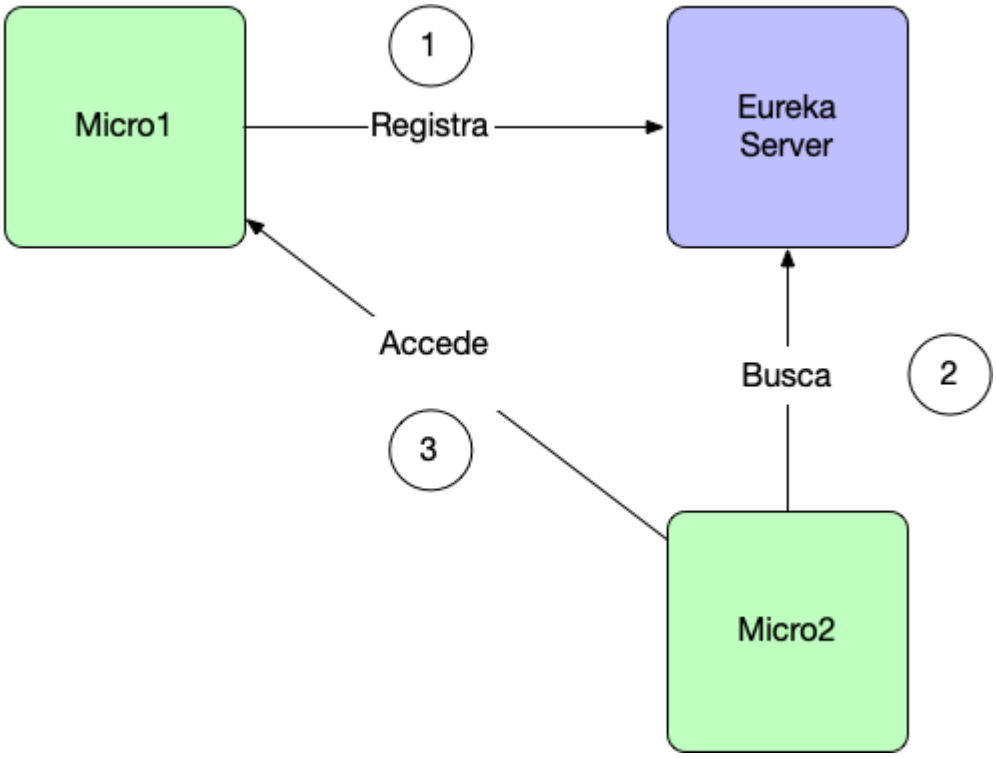
```
@SpringBootApplication
public class Micro2Application {

	public static void main(String[] args) {
		SpringApplication.run(Micro2Application.class, args);
	}

	@LoadBalanced
```

```
    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }

}
```

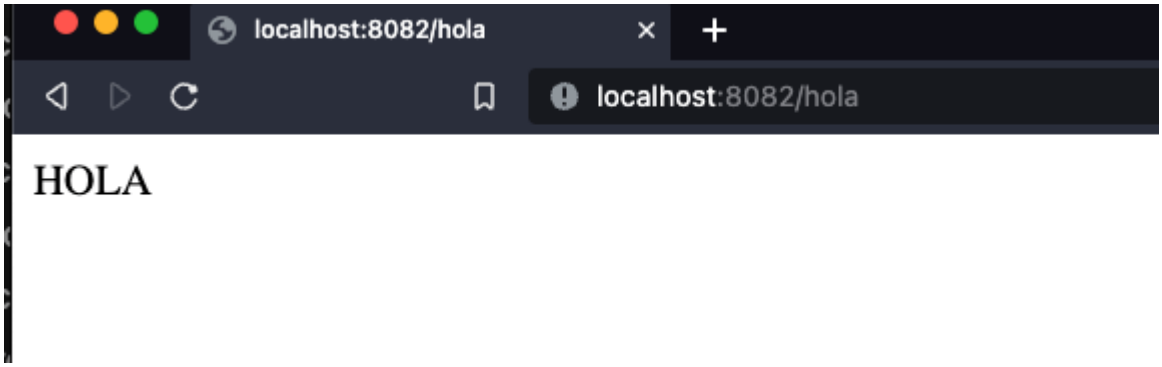
Por último necesitamos el servicio REST que hace de cliente en este microServicio 2 y que invoca al 1

```
package com.arquitecturajava.micro1.micro2;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class HolaController2 {
    @Autowired
    RestTemplate plantilla;

    @GetMapping("/hola")
    public String holaCliente() {
        return  plantilla.getForObject("http://micro1/hola",
String.class).toUpperCase();
    }
}
```

Como se puede observar el servicio 2 no invoca a la url del uno sino que delega en el nombre de registro del servidor Eureka para acceder .



Si ejecutamos el código será funcional:



- ¿Qué es Spring WebFlux?
- Spring REST Client con RestTemplates
- Java EE 6 uso de filtros dinámicos
- Curso Java EE desde Cero
- ¿Qué es un Microservicio?