

# Java Spring Boot

Unidad 5. Descubrimiento de servicios con Eureka

# Índice

## **Unidad 5: Descubrimiento de servicios con Eureka**

Introducción

Objetivos

Mapa conceptual

Servidor Eureka

Implementación de un servidor Eureka

Registro de un servicio en un servidor Eureka

Acceso a un microservicio a través de Eureka

Recuerda

## **Ejercicio práctico y solución**

## **Autoevaluación**

# Introducción

Uno de los problemas que surgen con la interacción entre microservicios es que el servicio cliente debe conocer la localización del servicio que va a utilizar, de modo que si éste es desplegado en una dirección diferente habrá que modificar el código de la aplicación cliente.

La utilización de los servidores Eureka vienen a paliar esta situación. A través del servicio Eureka, las aplicaciones clientes pueden descubrir microservicios sin necesidad de conocer la ubicación real de los mismos.

Este mecanismo de interacción entre microservicios facilita enormemente el despliegue de microservicios en la nube y la creación de aplicaciones basadas en ellos.

# Objetivos



Comprender la necesidad de acceder a microservicios a través de un servidor de descubrimiento.



Conocer el proceso de implementación de un servidor Eureka.

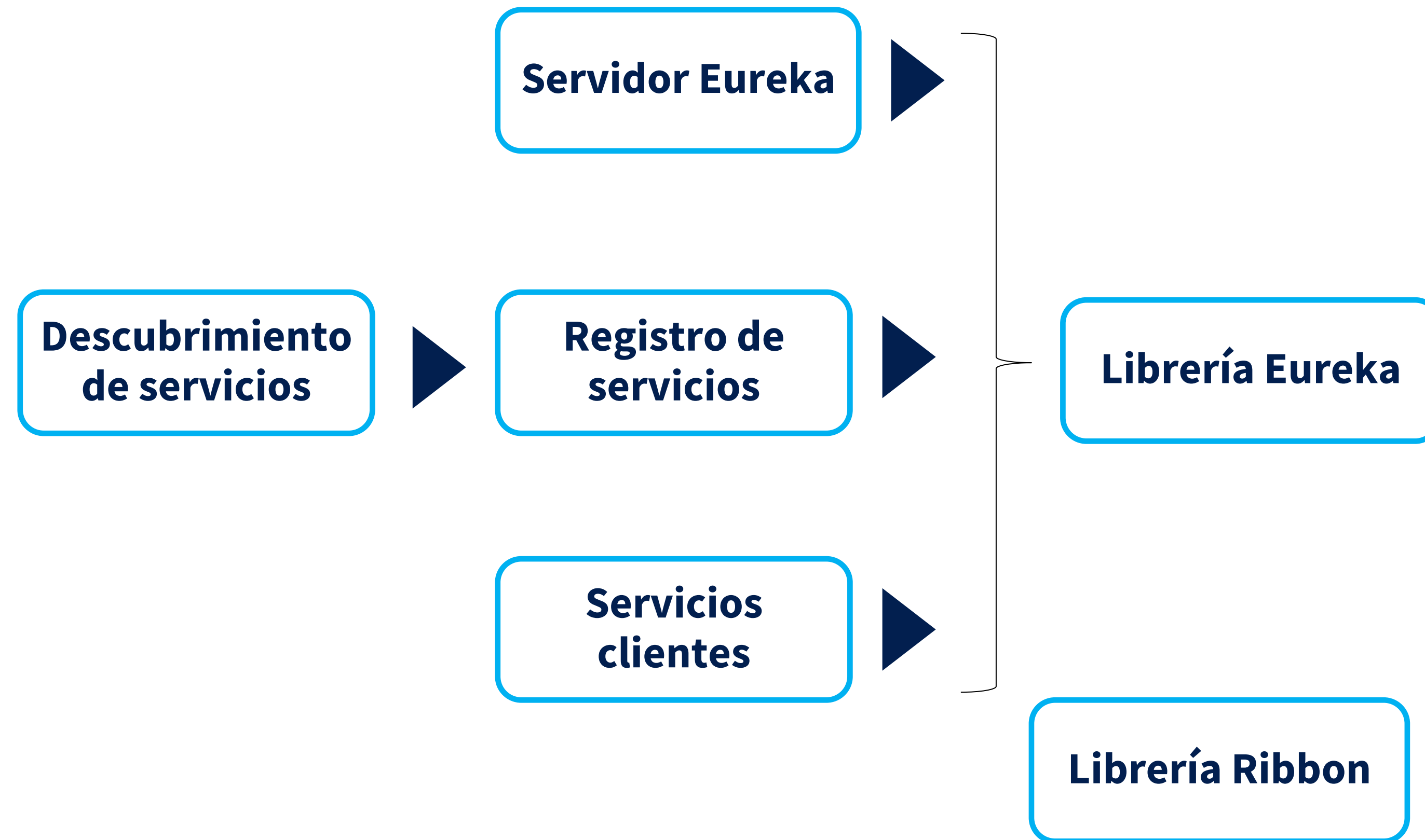


Registrar un microservicio en un servidor Eureka.



Desarrollar aplicaciones que accedan a servicios a través de Eureka.

# Mapa Conceptual





# Servidor Eureka

## Fundamentos

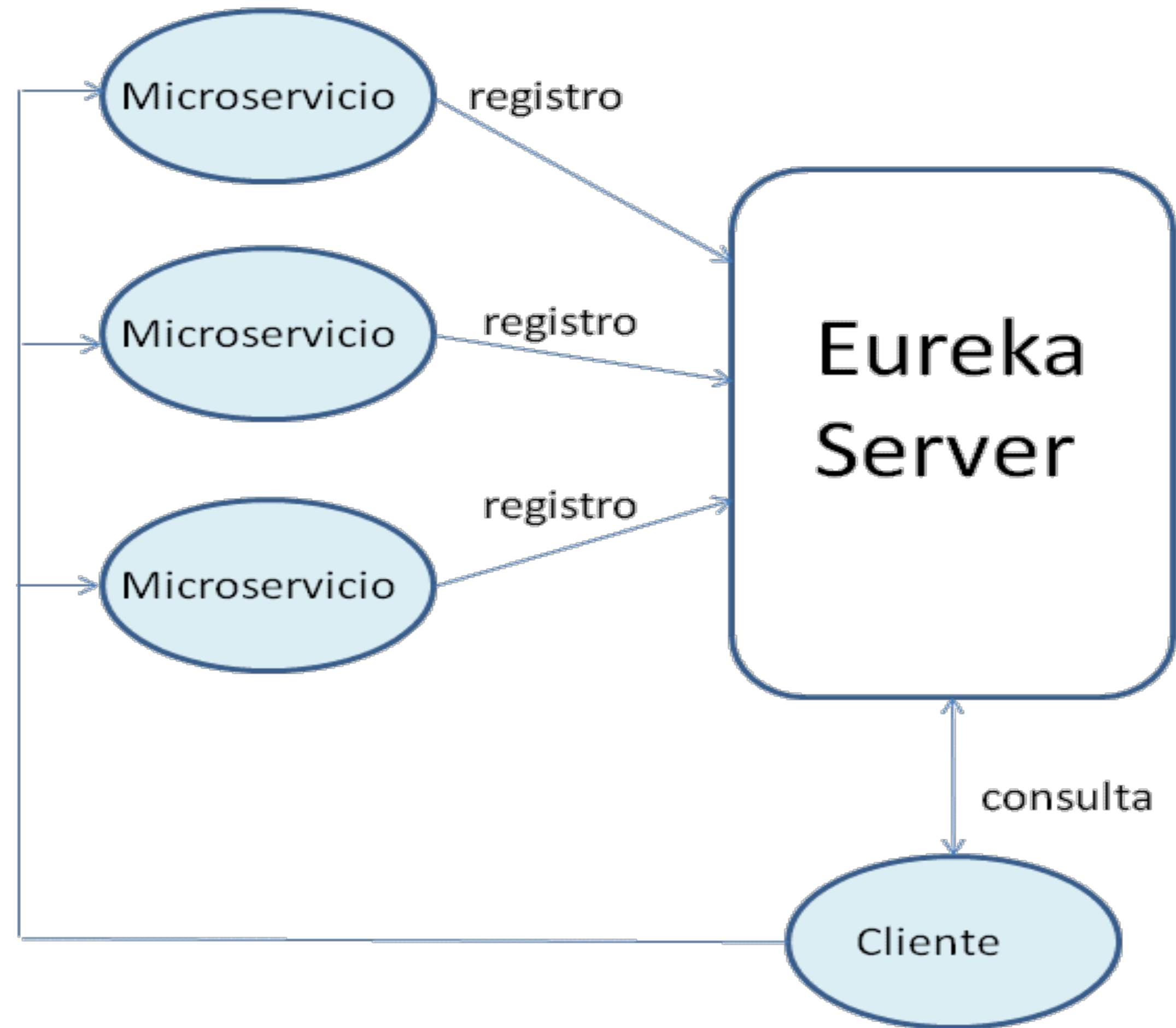
La existencia de múltiples microservicios autónomos e independientes distribuidos por la red, dificulta el descubrimiento y utilización de los mismos por parte de las aplicaciones cliente. Para facilitar esta labor, Spring, dentro de la **suite de componentes Spring Cloud proporciona el servidor Eureka**.

Eureka es un servidor para el registro y localización de microservicios. Su objetivo es **registrar las diferentes instancias de microservicios desplegados por la red**, indicando su localización, estado y otra serie de datos importantes. De esta manera, cuando un cliente (que puede ser otro microservicio) necesite acceder a un microservicio, acudirá al servidor Eureka para obtener la información asociada al mismo y **comunicarse con él, sin necesidad de conocer su localización real**.

A nivel de implementación, el **servidor Eureka no es más que otro microservicio** que nos ofrece información de otros servicios.

# Servidor Eureka

## Fundamentos



# Implementación de un servidor Eureka

## Creación del proyecto

Dado que un **servidor Eureka es realmente un microservicio**, para implementarlo crearemos una aplicación de tipo Spring Boot como las que hemos estado realizando hasta el momento

Durante el proceso de creación del proyecto, en la sección de dependencias, además de incluir *Spring Web*, debemos buscar la dependencia al servidor Eureka, para lo cual introduciremos la palabra Eureka en el buscador y cuando aparezcan los resultados marcaremos **Eureka Server** dentro de la sección Spring Cloud Discovery:

The screenshot shows the Spring Boot dependency management interface. At the top, there are two checkboxes: ☐ Spring Boot Actuator and ☒ Spring Web Starter. Below this, there are two columns: 'Available:' and 'Selected:'. In the 'Available:' column, the search bar contains the text 'eureka'. Below the search bar, there are two expandable sections: 'Pivotal Cloud Foundry' and 'Spring Cloud Discovery'. Under 'Pivotal Cloud Foundry', there is a checkbox for 'Service Registry (PCF)'. Under 'Spring Cloud Discovery', there are two checkboxes: 'Eureka Discovery Client' and 'Eureka Server', with 'Eureka Server' being checked. In the 'Selected:' column, there are two items: 'X Eureka Server' and 'X Spring Web Starter'.



# Implementación de un servidor Eureka

## Creación del proyecto

Una vez finalizado el asistente, nos desplazamos al archivo pom.xml para ver su contenido:

```
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
</properties>
<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>
  <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>
</dependencies>
```

# Implementación de un servidor Eureka

## Creación del proyecto

Además de la sección `<dependencies>`, vemos algunos elementos nuevos que se han añadido al incluir un componente de spring cloud.

Lo primero, en la sección de propiedades, vemos que aparece la versión de Spring Cloud con el valor **Greenwich.SR2**. Además, tenemos una nueva sección `<dependencyManagement>`, en la que quedan registradas las propiedades relativas a Spring Cloud.

# Implementación de un servidor Eureka

## La clase main

Aunque no dispondrá de ningún controlador, al igual que con cualquier otro microservicio, la clase main será la encargada de iniciar el servicio de Eureka. Sin embargo, para que el servicio se inicie como tal, será necesario **añadir la anotación @EnableEurekaServer** en la definición de la clase, que quedaría finalmente así:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
@SpringBootApplication
@EnableEurekaServer
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Al procesar la anotación, Spring aplica una serie de parámetros predefinidos para que el servicio se comporte como un servidor de descubrimiento de servicios.

# Implementación de un servidor Eureka

## Configuración del servidor

A nivel de código, simplemente tenemos que definir la clase main del servicio tal y como hemos visto anteriormente. Sin embargo, es necesario **proporcionar algunos datos de configuración sobre el servidor**, bien a través de application.properties o application.yml. En este ejemplo vamos a ver como proporcionar la configuración con el **segundo método**, por supuesto, ambos archivos pueden convivir y distribuir la información de configuración entre ambos

Para crear un archivo application.yml con eclipse, tendremos que situarnos sobre la carpeta resources y en el menú que aparece al pulsar el botón derecho del ratón elegiremos New ->File. En el cuadro de diálogo indicaremos el nombre del archivo completo con su extensión y pulsaremos finalizar. El contenido que tendremos que incluir en el archivo se muestra en el siguiente listado:

```
spring:
  application:
    name: eureka-server

server:
  port: 8761

eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
server:
  waitTimeInMsWhenSyncEmpty: 0
```

# Implementación de un servidor Eureka

## Configuración del servidor

El primer dato que vemos dentro de la sección *spring* es el nombre del servidor, en nuestro caso "eureka-server". Es importante indicar que **no se admite el carácter de subrayado** como separador dentro del nombre, pero sí el guión.

En la sección *server*, indicamos el puerto por el que va a estar escuchando el servidor Eureka. Habitualmente, es el 8761. A este puerto se conectan tanto los servicios registrados en el servidor, como los clientes

En la sección *eureka* hay dos subsecciones:

- **client**. Contiene información referente al cliente (el propio servicio eureka es un cliente). Una de esas propiedades es *registerWithEureka*, que se establece al valor *false* para que **el servidor no se registre a sí mismo como un servicio**, y *serviceUrl*, que es la **url asignada al servidor Eureka**. Es la dirección con la que los clientes (servicios registrados y aplicaciones clientes de servicios) acceden al mismo.
- **server**. Contiene propiedades adicionales del servidor, como *waitTimeInMsWhenSyncEmpty*, que es el tiempo de espera antes de iniciar el servicio.

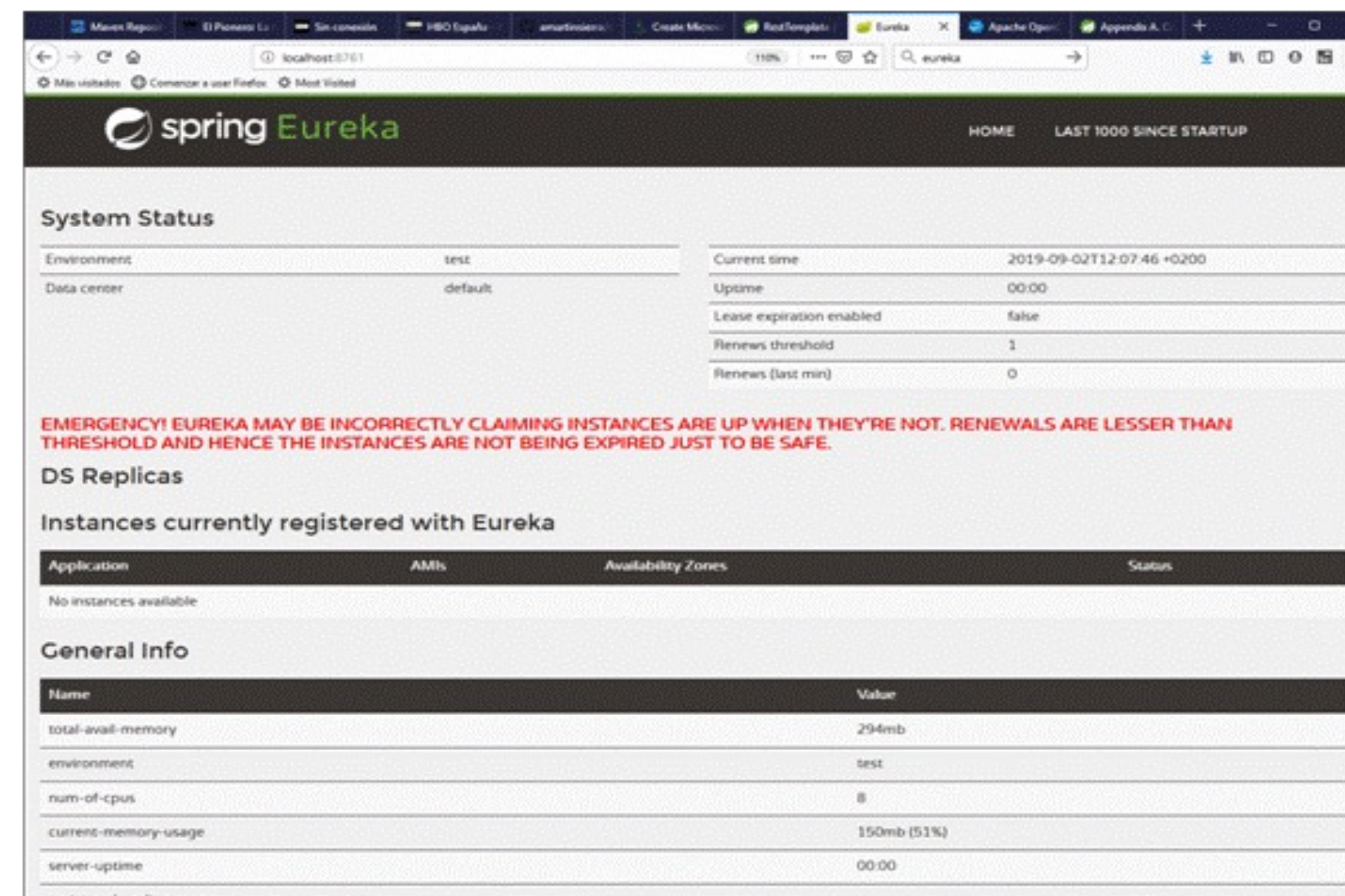


# Implementación de un servidor Eureka

## Inicio del servicio

Y eso es todo, ya está listo nuestro servidor Eureka para ser utilizado. Si lanzamos el servicio ejecutando main, se arrancará el servidor Eureka y estará accesible para cualquier cliente en la url indicada.

De hecho, una vez iniciado, si accedemos a la siguiente dirección desde un navegador, localhost:8761, podremos ver la **información de estado** del servidor:



Se puede ver que la lista de servicios disponibles para ser descubiertos está vacía, ya que aún no se ha registrado ninguno contra el servicio.

# Registro de un servicio en un servidor Eureka

## Dependencias

Una vez que ya tenemos nuestro servidor Eureka funcionando, vamos a ver como **registrar un microservicio en él**. En concreto, vamos a registrar el microservicio de gestión de empleados creado en capítulos anteriores, bien la versión con seguridad aplicada o la no segura.

Para ello, lo primero será agregar las dependencias necesarias al proyecto para poder aplicar Spring Cloud Client:

En primer lugar, en la sección <properties> debemos incluir la versión de Cloud:

```
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
</properties>
```

Seguidamente, en la sección de dependencias, incluiremos la dependencia a Spring Cloud Eureka Client:

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

# Registro de un servicio en un servidor Eureka

## Dependencias

Por último, debemos incorporar la sección de gestión de dependencias, con la configuración de la dependencia de Spring Cloud:

```
<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>
```

Indicar que **todas estas entradas se generan automáticamente si en el momento de la creación del proyecto** con el asistente de eclipse incluimos la dependencia a **Eureka Discovery Client**, que se encuentra dentro de la sección Spring Cloud Discovery.



# Registro de un servicio en un servidor Eureka

## Configuración del servicio

Para conseguir que un microservicio Spring boot quede registrado en un servidor Eureka **no hay que hacer nada a nivel de código**. Tan solo habrá que **definir las configuraciones apropiadas** para que al arrancar el servicio localice el servidor y se registre en él. Si utilizamos en este caso también `application.yml`, así deberá quedar el contenido del mismo:

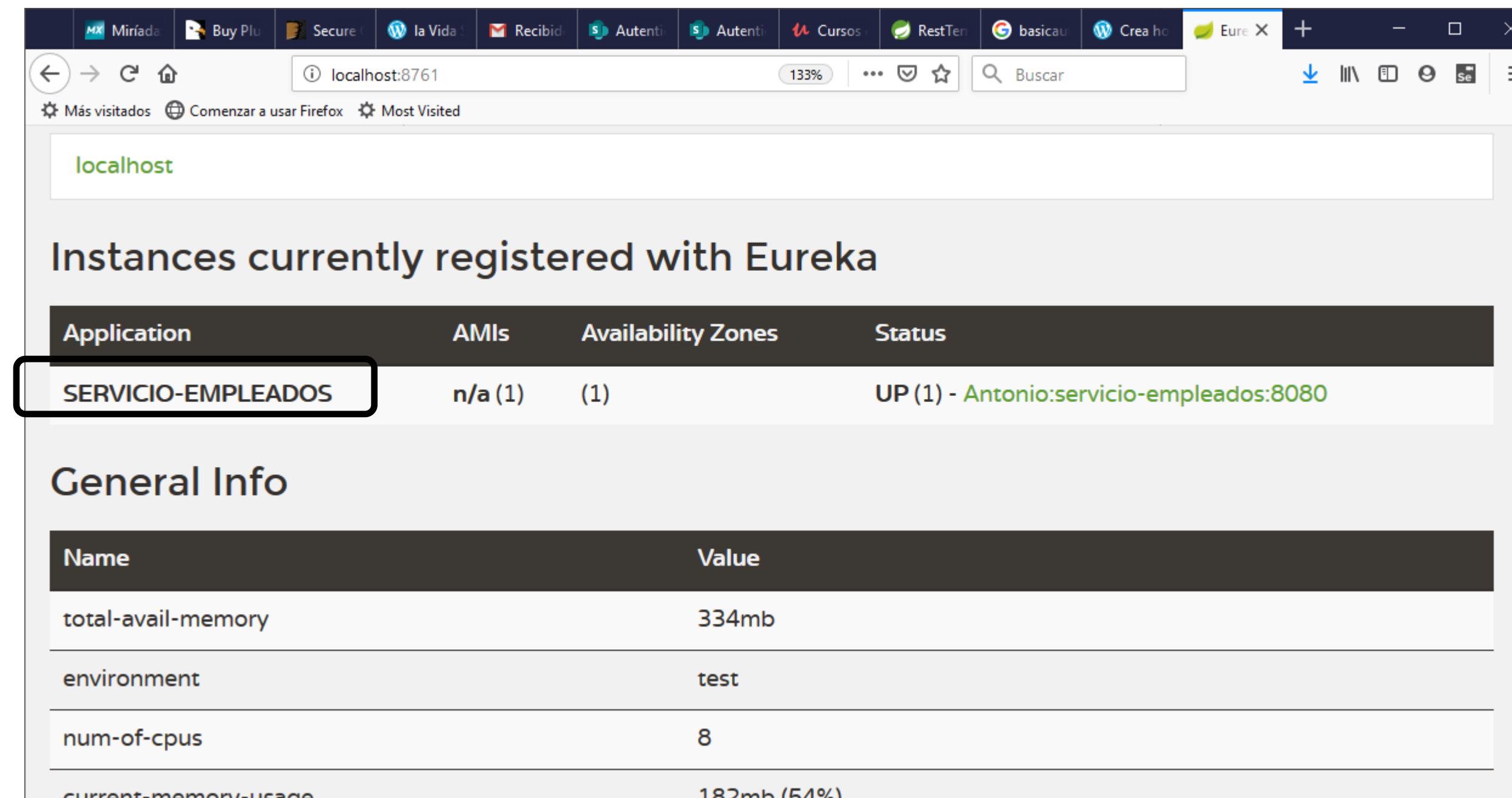
```
spring:
  application:
    name: "servicio-empleados"
server:
  port: 8080
eureka:
  instance:
    leaseRenewalIntervalInSeconds: 5
    leaseExpirationDurationInSeconds: 2
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
```

Entre las propiedades más importantes definidas en este archivo, tenemos a *name* de la sección `spring`, que indica el **nombre con el que será registrado el servicio dentro del servidor**, mientras que *port* de la sección `server` establece el puerto de escucha del microservicio. Para poder registrarse en el servidor, necesita conocer la ubicación de éste, dato que se indica en la sección *serviceUrl*

# Registro de un servicio en un servidor Eureka

## Inicio del servicio y comprobación

Una vez que se inicie el servicio, **se conectará con el servidor de Eureka y quedará registrado dentro del mismo con el nombre asignado**. Si volvemos a refrescar la página de Eureka en la dirección `http://localhost:8761`, podemos comprobar como ahora nos aparece nuestro servicio-empleados en la lista de instancias registradas:



The screenshot shows a web browser window at localhost:8761. The page title is 'localhost'. The main heading is 'Instances currently registered with Eureka'. Below this is a table with the following data:

Application	AMIs	Availability Zones	Status
SERVICIO-EMPLEADOS	n/a (1)	(1)	UP (1) - Antonio:servicio-empleados:8080

Below the table is a section titled 'General Info' with a table of system metrics:

Name	Value
total-avail-memory	334mb
environment	test
num-of-cpus	8
current-memory-usage	182mb (54%)

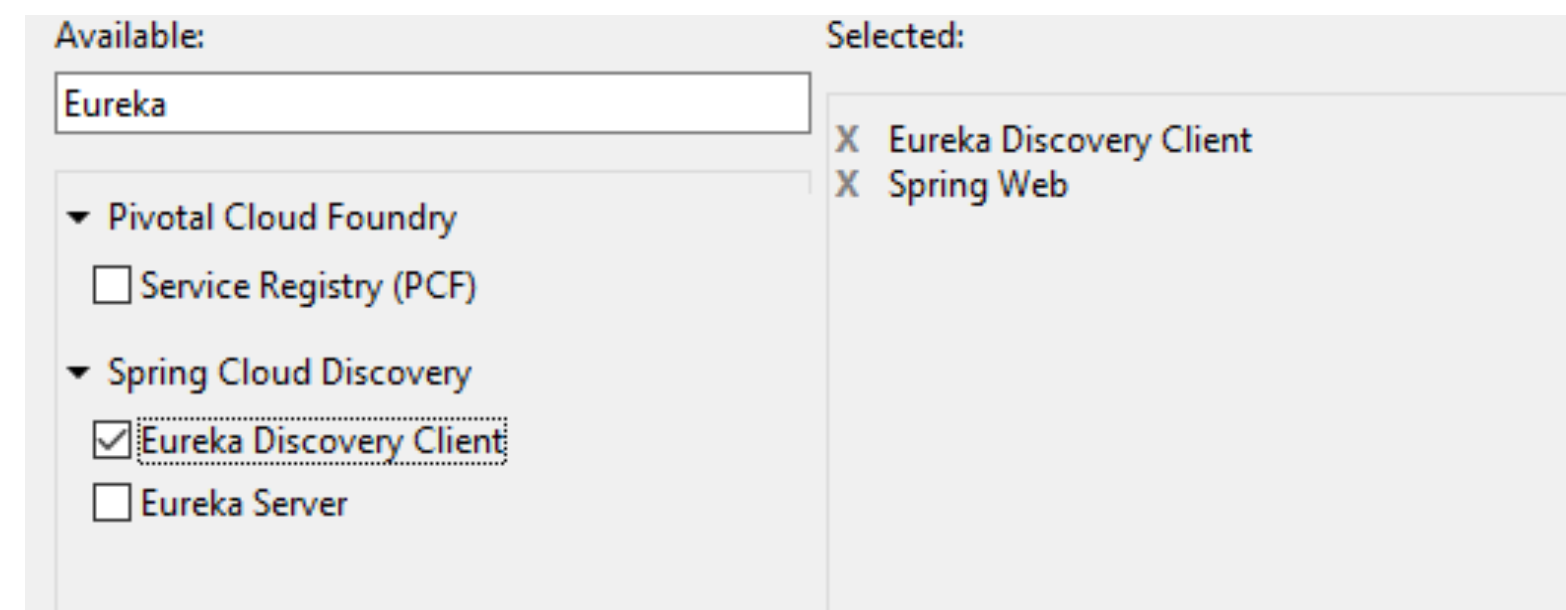


# Acceso a servicios a través de Eureka

## Creación de microservicio cliente

Las aplicaciones clientes de un microservicio registrado en Eureka serán a su vez otros microservicios que se conectarán al servidor y localizarán al servicio que quiere utilizar por su nombre de registro, de esta manera, **el cliente no necesitará conocer los detalles de localización del servicio que quiere utilizar.**

A continuación vamos explicar como crear un microservicio cliente del servicio de empleados, que **acceda a éste a través de Eureka**. Lo primero, como siempre, será crear el proyecto Spring Starter Project y agregar las dependencias, que en esta caso serán, además de Spring Web, la dependencia a Eureka Client:



El servicio que vamos a desarrollar, será igual al cliente creado en el capítulo anterior, pero incluyendo lo necesario para que el acceso se realice a través del servidor Eureka, en vez de directamente.

# Acceso a servicios a través de Eureka

## Implementación del cliente

La implementación del controlador y bean Empleado serán exactamente iguales a las del cliente desarrollado en el capítulo anterior, solo que la **URL de acceso al cliente ya no será la dirección real, sino con la que está registrado en el servidor Eureka**, pues vamos a acceder al servicio a través de dicho servidor. Esta es la implementación del controlador:

```
@RestController
public class ClienteEmpleadosController {
    //dirección de registro en Eureka
    private static final String url="http://servicio-empleados";
    @Autowired
    RestTemplate template;
    @GetMapping(value="procesado/{id}/{nombre}/{salario}",produces=MediaType.APPLICATION_JSON_VALUE)
    public List<Empleado> empleados(@PathVariable("id") int id,
        @PathVariable("nombre") String nombre,
        @PathVariable("salario") double salario){
        //si no existe el empleado lo añade
        if(template.getForObject(url+"/"+id, Empleado.class)==null) {
            Empleado emp=new Empleado(id,nombre,salario);
            template.postForLocation(url, emp);
        }
        //en cualquier caso, devuelve un array con los empleados existentes
        return template.getForObject(url, List.class);
    }
}
```

# Acceso a servicios a través de Eureka

## Implementación del cliente

Para que el acceso se produzca a través de Eureka, es necesario declarar el método encargado de crear el objeto RestTemplate con la **anotación @LoadBalanced**. Sí quedaría ahora la clase main:

```
@ComponentScan(basePackages= {"controllers"})
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
    @LoadBalanced
    @Bean
    public RestTemplate getTemplate() {
        return new RestTemplate();
    }
}
```

Esta anotación lo que hace es **configurar automáticamente Ribbon**, una librería incorporada en el cliente Eureka que hace de **balanceador de carga en el lado cliente** para poder invocar a microservicios a través de Eureka.

# Acceso a servicios a través de Eureka

## Configuración del cliente

Finalmente, nos queda configurar el microservicio cliente para, entre otros datos, indicarle la localización del servidor Eureka. Si utilizamos también en este servicio la configuración por yml, así nos quedaría el archivo application.yml

```
spring:
  application:
    name: "cliente-ciudades"
server:
  port: 8082

eureka:
  instance:
    leaseRenewalIntervalInSeconds: 20
    leaseExpirationDurationInSeconds: 20
  client:
    #No es necesario que la aplicación cliente se registre en Eureka
    registerWithEureka: false
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
```

# Acceso a servicios a través de Eureka

## Configuración del cliente

Además del nombre y puerto del servicio, datos que ya conocemos de servicios anteriores, vemos que en la sección eureka es necesario indicar la **url del servidor Eureka**. Mediante la propiedad *registerWithEureka* al valor *false* indicamos que **no queremos que este microservicio cliente se registre en Eureka**, ya que por defecto, todos los servicios que se ejecuten en la misma máquina del servidor quedarían registrados en él.

Otros datos indicados en la sección Eureka como *leaseRenewalIntervalInSeconds* y *leaseExpirationDurationInSeconds*, se refieren a intervalos de tiempos de comunicación y respuesta entre el servicio y el servidor Eureka.

Una vez iniciados los tres servicios (microservicio de empleados, servidor Eureka y cliente), podemos acceder desde postman al servicio cliente con una petición de tipo GET para comprobar que el funcionamiento es el mismo que en la versión cliente anterior:

[http://localhost:8082/procesado/3000/emp\\_new/20](http://localhost:8082/procesado/3000/emp_new/20)

Con la ventaja de que **éste cliente no está acoplado a la dirección real del servicio** llamado.



# Ejercicio Práctico

En este ejercicio crearemos un nuevo servidor de descubrimiento Eureka, en el que registraremos el servicio de curso creado en los ejercicios anteriores.

Así mismo, modificaremos la configuración del servicio cliente para que acceda al microservicio de cursos a través del servidor Eureka.

# Solución ejercicio

- En el siguiente video tutorial podrás ver la solución al ejercicio propuesto, con la explicación detallada de los componentes desarrollados.

[VER TUTORIAL](#)

# Recuerda

- Un servidor Eureka es un microservicio que permite el registro de otros microservicios a fin de que puedan ser descubiertos desde aplicaciones clientes sin que estas deban conocer la ubicación real de los microservicios a los que quieren acceder.
- Para crear un servidor Eureka crearemos un microservicio Spring Boot con la dependencia Eureka Server y añadiremos la anotación `@EnableEurekaServer` en la definición de la clase main. También se deberá registrar en el archivo de configuración el nombre, puerto y dirección del servidor.
- Para crear un microservicio que se registre en Eureka, basta con añadir la librería Eureka Client a las dependencias y registrar en el archivo de configuración la dirección del servidor, así como el nombre que le queremos asociar y el puerto de escucha.
- Para crear un microservicio cliente, añadiremos la dependencia a la librería Eureka Client y el método que nos devuelve el RestTemplate estará anotado con `@LoadBalanced` para activar la librería ribbon que balancea las peticiones sobre las diferentes instancias del servicio. El objeto RestTemplate obtenido podrá acceder al servicio cliente por su nombre virtual.