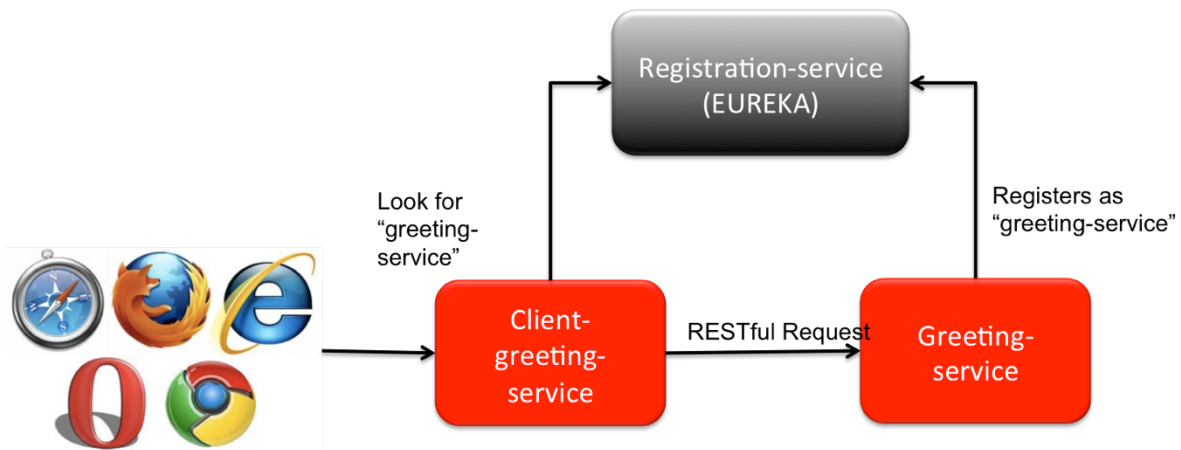


## Laboratorio 7

Voy a intentar explicar, de la manera más simple posible, como construir una sencilla aplicación compuesta por tres microservicios:



El objetivo es implementar un típico *Hola Mundo* con microservicios. Para ello, *greeting-client-service* llamará al método *greeting* del microservicio *greeting-service* usando **RESTful API**, pasándole como parámetro de entrada un nombre (Ej: "*Rob*") y recibiendo como contestación un saludo (Ej: "*Hello, Rob!*"). Para realizar el ejercicio, utilizaremos **Eureka** (*registration-service*), como servidor de registro y descubrimiento de microservicios. *Eureka* está incorporado dentro de **Spring Cloud**.

### Configuración

Para la implementación de los microservicios, utilizaremos **Spring Boot** y **Spring Cloud**.

El laboratorio estará compuesto de tres proyectos (uno por cada microservicio).

- microservices-greeting
- microservices-clientGreeting
- microservices-registration

## New Spring Starter Project



Service URL

Name

☒ Use default location

Location

Type:  Packaging:

Java Version:  Language:

Group

Artifact

Version

Description

Package

### Working sets

☐ Add project to working sets

Working sets:



## New Spring Starter Project Dependencies



Spring Boot Version:

Frequently Used:

- |  |   |  |
|--|---|--|
| <input type="checkbox"/> Cloud Bootstrap   | <input type="checkbox"/> Cloud Security           | <input type="checkbox"/> Config Client |
| <input type="checkbox"/> Eureka Discovery  | <input checked="" type="checkbox"/> Eureka Server | <input type="checkbox"/> Hystrix       |
| <input type="checkbox"/> Hystrix Dashboard | <input checked="" type="checkbox"/> Web           | <input type="checkbox"/> Zuul          |

Available:

Selected:

- ▼ Cloud AWS
  - ☐ AWS Core
  - ☐ AWS JDBC
  - ☐ AWS Messaging
- ▼ Cloud Circuit Breaker
  - ☐ Hystrix
  - ☐ Hystrix Dashboard
  - ☐ Turbine
  - ☐ Turbine Stream
- ▼ Cloud Config
  - ☐ Config Client
  - ☐ Zookeeper Configuration
- ▼ Cloud Contract
  - ☐ Cloud Contract Verifier

- X Eureka Server
- X Web

Make Default

Clear Selection



< Back

Next >

Cancel

Finish

A continuación, muestro la configuración de los ficheros **POM.xml**.

```
<groupId>com.mvega</groupId>
<artifactId>microservice-xxxxxxxxxx</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>microservice- xxxxxxxxxxxx </name>
<description>Lab Microservicios</description>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

<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>
```

## Microservicio Registration

Como comenté antes, para implementar este servicio utilizaremos **Eureka**, un servidor de registro y descubrimiento de microservicios *open source* desarrollado por **Netflix**. Levantar una instancia de este servidor con *Spring Cloud* es muy sencillo. Aquí tienes el código completo:

```
@SpringBootApplication
@EnableEurekaServer
public class MicroserviceRegistrationApplication{

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

La anotación **@SpringBootApplication** indica que se trata de una aplicación *Spring Boot* y **@EnableEurekaServer** habilita el uso *Eureka* como servidor de registro y descubrimiento de microservicios.

Por defecto, cuando inicias una aplicación *spring boot*, se busca un fichero llamado **application.properties** o **application.yml** para acceder a su configuración, el cual deberá estar ubicado en la carpeta **resources** de nuestro proyecto. Su configuración es la siguiente:

```
# Configure this Discovery Server
eureka:
  instance:
    hostname: localhost
  client: # Not a client, don't register with yourself
    registerWithEureka: false
    fetchRegistry: false


server:
  port: 1111 # HTTP (Tomcat) port

# Discovery Server Dashboard uses FreeMarker. Don't want Thymeleaf templates
spring:
  thymeleaf:
    enabled: false # Disable Thymeleaf
```

Observa que:

- En la configuración del fichero **yml** estamos indicando que no se trata de un microservicio cliente y que por tanto no queremos que se registre en Eureka (*registerWithEureka: false*).

- Aunque por defecto Eureka escucha en el puerto **8761**, para este ejemplo hemos indicado que utilice el puerto **1111**. Prueba a ejecutar el servicio, comprueba que el servidor arranca correctamente y accede a <http://localhost:1111> para visualizar el *dashboard* de *Eureka*.
- El *dashboard* de Eureka está implementado usando plantillas *FreeMarket* (los otros dos microservicios que crearemos a continuación usarán *thymeleaf* para construir las vistas HTML).


HOME
LAST 1000 SINCE STARTUP

### System Status

Environment	Current time	2015-11-08T23:03:31 +0100
Data center	Uptime	04:38
	Lease expiration enabled	false
	Renews threshold	0
	Renews (last min)	0

### DS Replicas

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

### General Info

Name	Value
total-avail-memory	297mb
environment	
num-of-cpus	4

## Microservicio Greeting

Este microservicio gestionará peticiones que reciba por HTTP para devolver un saludo (*greeting*) a los nombres de las personas que reciba como entrada. Al arrancar *greeting-service* lo primero que hará será registrarse en *Eureka*.

A continuación, muestro la configuración de los ficheros **POM.xml**.

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

```
@EnableAutoConfiguration
@EnableDiscoveryClient
@SpringBootApplication
public class MicroserviceGreetingApplication{

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

Respecto a las anotaciones utilizadas:

- **@SpringBootApplication** define que es una aplicación Spring Boot
- **@EnableAutoConfiguration** configura la aplicación sin necesidad de otros ficheros de configuración XML, etc.
- **@EnableDiscoveryClient** habilita el servicio de registro y descubrimiento. En este caso, este proceso se auto-registrará en *Eureka* utilizando el nombre de aplicación indicado en el fichero *application.yml* cuya configuración se muestra a continuación:

```
# Spring properties
spring:
  application:
    name: greeting-service # Service registers under this name
  freemarker:
    enabled: false        # Ignore Eureka dashboard FreeMarker templates
  thymeleaf:
    cache: false          # Allow Thymeleaf templates to be reloaded at runtime
    prefix: classpath:/templates/ # Trailing / mandatory
                                # Template location for this application only

# Discovery Server Access
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:1111/eureka/

# HTTP Server
server:
  port: 2222 # HTTP (Tomcat) port
```

En la configuración del fichero *application.yml* observamos que:

- El **nombre de la aplicación** es *greeting-service*, que será utilizado para registrarse en Eureka
- Se configura el **puerto** de escucha del servidor web (**2222**). Esto es necesario ya que vamos a ejecutar varios procesos que usan Tomcat y todos ellos no pueden estar escuchando al mismo tiempo del puerto 8080.
- Se establece la **URL de acceso** al servicio de **Eureka**

Este microservicio utilizará **Spring REST** para ofrecer una interface **RESTful** sobre HTTP y poder así acceder a sus operaciones.

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
```



```

    @RequestMapping("/greeting/{name}")
    public Greeting greeting(@PathVariable("name") String name) {
        return new Greeting(String.format(template, name));
    }
}

```

El método *greeting* devolverá un objeto de tipo *Greeting*, cuya clase se muestra a continuación:

```


public class Greeting implements Serializable {

    private static final long serialVersionUID = 1L;
    private final String content;

    public Greeting(String content) {
        this.content = content;
    }
    public String getContent() {
        return content;
    }
}

```

Por último, ejecuta el microservicio *greeting-service* (método *main* de la clase *MicroserviceGreetingApplication*) y comprueba que al refrescar el *dashboard* de *Eureka*, aparece el servicio registrado. Ten en cuenta que el registro puede durar entre 10-20 segundos en realizarse.


HOME
LAST 1000 SINCE STARTUP

### System Status

Environment	Current time	2015-11-09T00:11:06 +0100
Data center	Uptime	05:45
	Lease expiration enabled	false
	Renews threshold	0
	Renews (last min)	0

### DS Replicas

#### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GREETING-SERVICE	n/a (1)	(1)	UP (1) - MacBookRoberto.local

Comprueba que el microservicio responde correctamente al realizar una petición *RESTful* vía HTTP como la siguiente: <http://localhost:2222/greeting/Rob>



## Microservicio *ClientGreeting*

El objetivo de este microservicio será, a través de Eureka (*registration-service*), localizar la URL de acceso al microservicio *greeting-service*, hacer una petición REST sobre HTTP a uno de sus métodos expuestos en su API, obtener un resultado y presentarlo en pantalla usando *thymeleaf*.

Para consumir un servicio *RESTful*, *Spring* proporcionar la clase ***RestTemplate*** que nos permite enviar peticiones HTTP a un servidor *RESTful* y recuperar los datos en formatos como ***JSON*** y ***XML***. En nuestro ejemplo usaremos *JSON* como formato de intercambio de datos, para lo cual vamos a apoyarnos en ***Jackson***, una librería java para el procesamiento y parseo de datos JSON.

la implementación de este microservicio estará compuesta de tres pasos

### ***Paso 1: Implementación del Service***

Implementaremos el acceso al microservicio *greeting-service* en la siguiente clase:

```
@Service
public class ClientGreetingService {

    @Autowired
    protected RestTemplate restTemplate;
    protected String serviceUrl;

    public ClientGreetingService(String serviceUrl) {
        this.serviceUrl = serviceUrl.startsWith("http") ? serviceUrl: "http://" +
serviceUrl;
    }

    //invoke to greeting-service and return a Greeting object
    public Greeting greeting (String name) {
        Greeting greeting = restTemplate.getForObject(serviceUrl +
"/greeting/{name}", Greeting.class, name);

        return greeting;
    }
}
```

***RestTemplate*** ha sido configurado por *Spring Cloud* para usar un *HttpRequestClient* personalizado que utiliza *Netflix Ribbon* para realizar la búsqueda del microservicio. *Ribbon* además es un **balanceador de carga**, por lo

que si tienes varias instancias disponibles de un microservicio, selecciona una para ti.

La petición REST realizada en la clase *ClientGreetingService* devuelve un mensaje JSON que pasaremos, usando *Jackson*, a un objeto de la clase *Greeting*. A continuación muestro la definición de esta clase:

```
import com.fasterxml.jackson.annotation.JsonRootName;
```

```
@JsonRootName("Greeting")
public class Greeting {
    protected String content;

    protected Greeting() {
        this.content = "Hello!";
    }

    public Greeting(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }
}
```

### ***Paso 2: Implementación de Controller***

A continuación, vamos a crear un **controlador** que va a agrupar un conjunto de acciones a realizar sobre este microservicio. Para ello usaremos las siguientes anotaciones:

- **@Controller**, que registrará el controlador para Spring MVC
- **@RequestMapping**, anotación que se encarga de relacionar un método con una petición http

```
@Controller
public class ClientGreetingController {

    protected ClientGreetingService helloWorldService;

    //constructor
    public ClientGreetingController(ClientGreetingService helloWorldService) {
        this.helloWorldService = helloWorldService;
    }

    @RequestMapping("/greeting")
    public String goHome() {
        return "index";
    }
}
```

```

@RequestMapping("/greeting/{name}")
public String greeting(Model model, @PathVariable("name") String name) {

    Greeting greeting = helloWorldService.greeting(name);

    model.addAttribute("greeting", greeting.getContent());

    return "greeting";
}
}

```

```

@Controller
public class ClientGreetingHomeController {

    @RequestMapping("/")
    public String home() {
        return "index";
    }
}

```

*ClientGreetingController* es un típico **controlador Spring MVC** que devolverá un HTML. Para construir la vista y generar dinámicamente el HTML, la aplicación usará **Thymeleaf**.

### ***Paso 3: Implementación del Microservicio***

Por último creamos el microservicio ***greeting-client-service***, que proporcionará el valor de la variable ***serviceURL*** a *ClientGreetingController*, el cual a su vez se lo pasará a *ClientGreetingService*

```

@SpringBootApplication
@EnableDiscoveryClient
@ComponentScan(useDefaultFilters = false) // Disable component scanner
public class MicroserviceClientGreetingApplication {

    public static final String SERVICE_URL = "http://GREETING-SERVICE";

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

    //A customized RestTemplate that has the ribbon load balancer build in
    @LoadBalanced
    @Bean
    RestTemplate restTemplate() {
        return new RestTemplate();
    }
}

```

```
// The service encapsulates the interaction with the micro-service.
@Bean
public ClientGreetingService helloWorldService() {
    return new ClientGreetingService(SERVICE_URL);
}

//Create the controller, passing it the ClientGreetingService to use.
@Bean
public ClientGreetingController helloWorldController() {
    return new ClientGreetingController(helloWorldService());
}

@Bean
public ClientGreetingHomeController homeController() {
    return new ClientGreetingHomeController();
}
}
```

Ten en cuenta que *Greeting-client-service*, además de ser un microservicio y auto-registrarse en el registro de servicios (@***EnableDiscoveryClient***), utilizará Eureka para localizar el microservicio *greeting-service*.

El contenido del fichero de configuración de este microservicio es el siguiente:

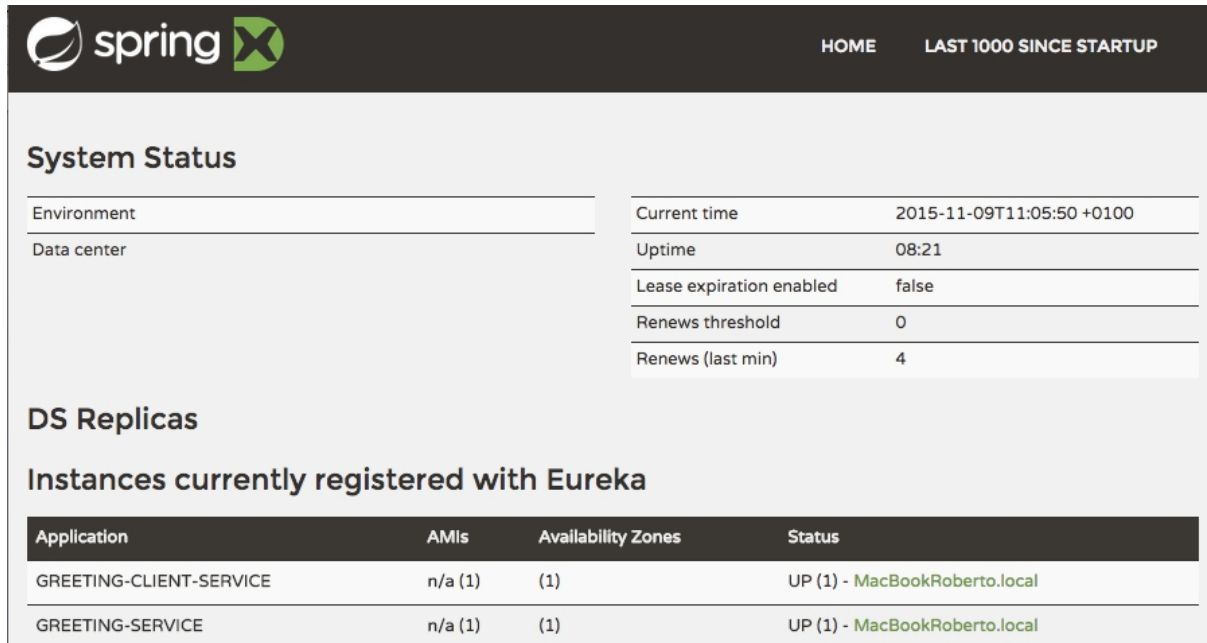
```
# Spring properties
spring:
  application:
    name: greeting-client-service # Identify this application
  freemarker:
    enabled: false # Ignore Eureka dashboard FreeMarker templates
  thymeleaf:
    cache: false # Allow Thymeleaf templates to be reloaded at runtime
    prefix: classpath:/templates/ # Trailing / mandatory
    # Template location for this application onlyy

# Map the error path to error template (for Thymeleaf)
error:
  path=/error

# Discovery Server Access
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:1111/eureka/

# HTTP Server
server:
  port: 3333 # HTTP (Tomcat) port
```

Ya solo queda ejecutar este microservicio y probar que todo funciona correctamente. Comprueba que, al levantar el microservicio, aparece registrado en la consola de *Eureka*



The screenshot shows the Spring Eureka Dashboard. At the top, there's a navigation bar with the Spring logo and 'HOME' and 'LAST 1000 SINCE STARTUP' links. Below this, the 'System Status' section displays a table with system metrics. To the left of this table, there are fields for 'Environment' and 'Data center'. Below the system status, the 'DS Replicas' section is visible. The main part of the dashboard shows 'Instances currently registered with Eureka' in a table with columns for Application, AMIs, Availability Zones, and Status.

Application	AMIs	Availability Zones	Status
GREETING-CLIENT-SERVICE	n/a (1)	(1)	UP (1) - MacBookRoberto.local
GREETING-SERVICE	n/a (1)	(1)	UP (1) - MacBookRoberto.local

El microservicio *client-greeting-service* está levantado y escuchando peticiones en el puerto 3333. Accede a <http://localhost:3333> y comprueba que visualizas la siguiente pantalla:

## Microservices Demo - Greeting-Client Server

### Overview

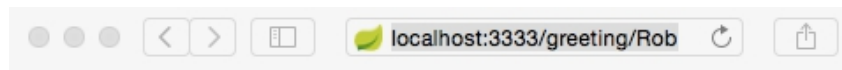
- Demo defines a simple web-application for accessing Greeting microservice.

### The Demo

- Eureka Dashboard: <http://localhost:1111>
- Check applications registered: <http://localhost:1111/eureka/apps/>
- Fetch by name: [/greeting/Rob](#)

Si pinchas en el link de *Fetch by Name*, el microservicio *client-greeting-service* llamará a *Eureka* para localizar el *endpoint* del microservicio *greeting-service* y a continuación realizará una petición REST para llamar al método *greeting* de este servicio pasándole como parámetro un nombre (“*Rob*”).

A continuación, muestro el resultado:



## Microservice Greeting Client

Greeting  
Hello, Rob!

Prueba a cambiar el nombre que se pasa como parámetro y comprueba que el resultado devuelto es distinto.