Spring Boot Microservices - Spring Cloud Netflix Eureka based Service Registry

author: Ramesh Fadatare

MICROSERVICES SPRING BOOT SPRING CLOUD

In this previous couple of tutorials, we have seen:

Spring Boot Microservices Communication Example using RestTemplate

Spring Boot Microservices Communication Example using WebClient

Spring Boot Microservices Communication Example using Spring Cloud Open Feign

In this tutorial, we will learn how to create a Service Registry using **Spring Cloud Netflix Eureka** in the Spring boot microservices project.

Service Registry and Discovery Overview

In the microservices projects, Service Registry and Discovery play an important role because we most likely run multiple instances of services and we need a mechanism to call other services without hardcoding their hostnames or port numbers. In addition to that, in Cloud environments service instances may come up and go down anytime. So we need some automatic service registration and discovery mechanism.

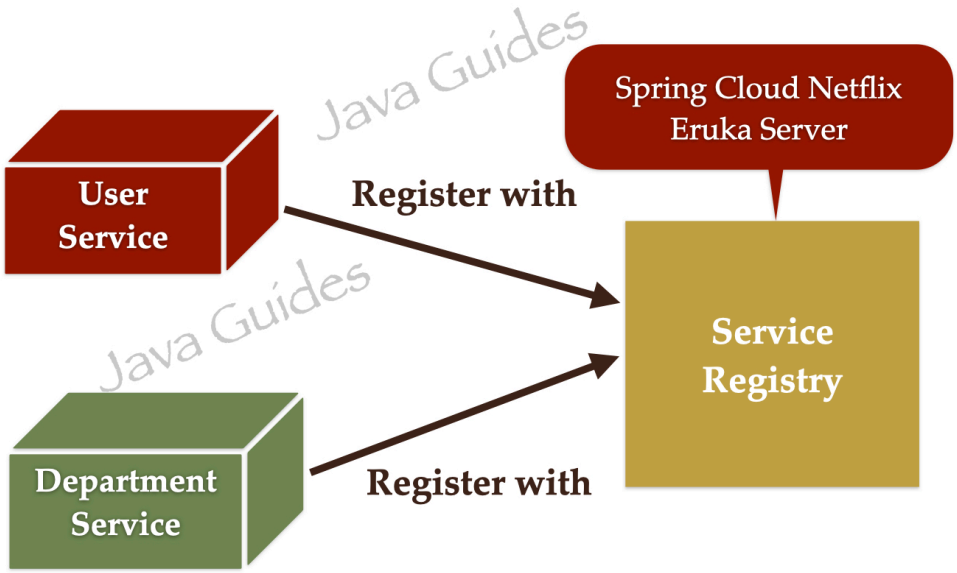
Spring Cloud addresses this problem by providing **Spring Cloud Netflix Eureka** project to create Service Registry and Discovery.

In this tutorial, we will learn how to use **SpringCloud Netflix Eureka** for Service Registry and Discovery.

What we will build?

We can use Netflix Eureka Server to create a Service Registry and make our microservices (`department-service` and `user-service`) as Eureka Clients so that as soon as we start a microservice it will get registered with Eureka Server automatically with a logical Service ID. Then, the other microservices, which are also Eureka Clients, can use Service ID to invoke REST endpoints.

Spring Cloud Netflix Eureka Server



Spring Cloud makes it very easy to create a Service Registry and discover other services using Load Balanced `RestTemplate:`

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

Prerequisites

Refer to the below tutorial to create `department-service` and `user-service` microservices:

Spring Boot Microservices Communication Example using RestTemplate.

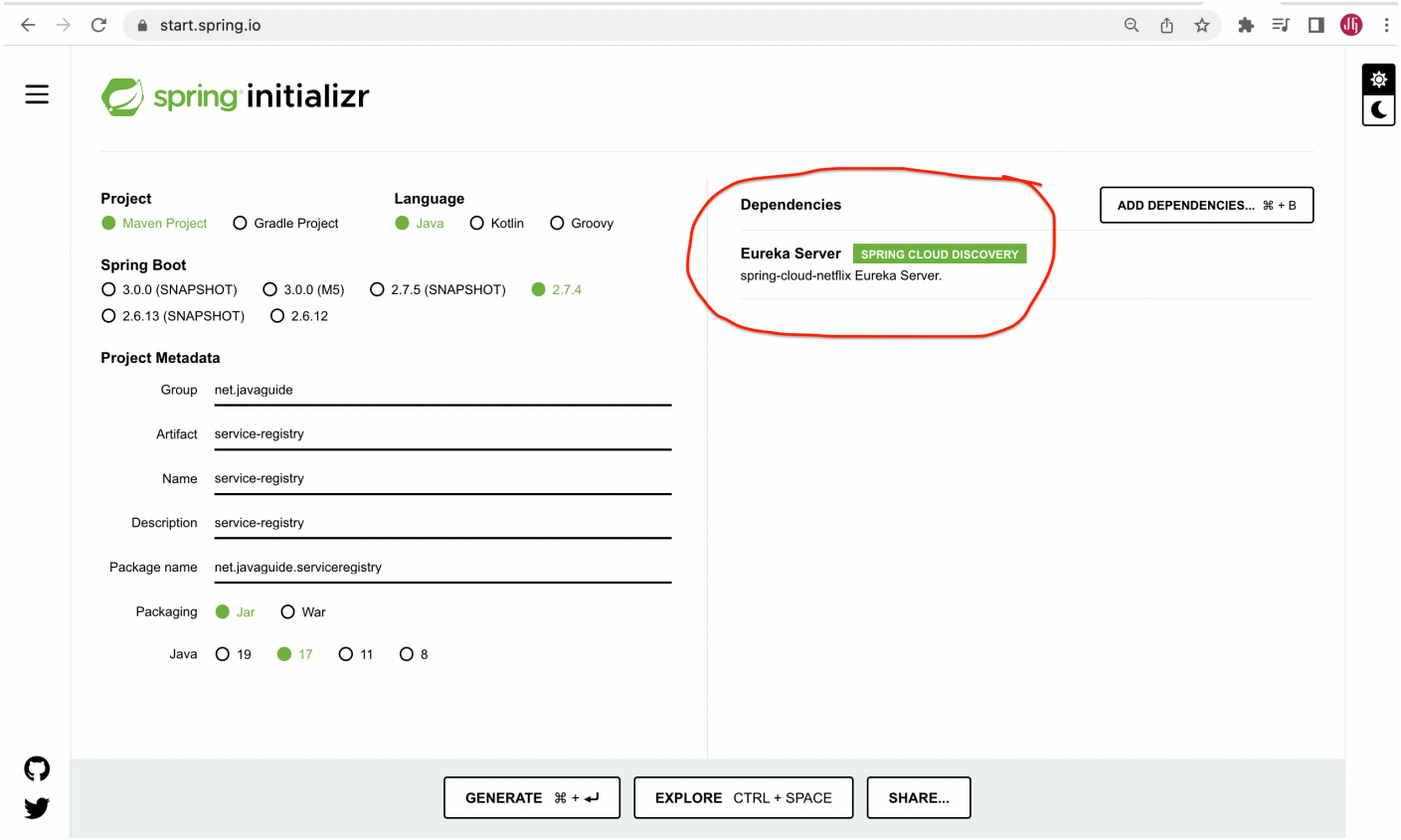
1. Create and Setup Spring boot project in IntelliJ IDEA

Let us create a Service Registry using Netflix Eureka which is nothing but a SpringBoot application with a Eureka Server starter.

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

Let's create a Spring boot project using the **spring initializr**.

Refer to the below screenshot to enter details while creating the spring boot application using the **spring initializr**:



Click on Generate button to download the Spring boot project as a zip file. Unzip the zip file and import the Spring boot project in IntelliJ IDEA.

Here is the `pom.xml` file for your reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.7.4</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>net.javaguide</groupId>
        <artifactId>service-registry</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>service-registry</name>
        <description>service-registry</description>
        <properties>
                <java.version>17</java.version>
                <spring-cloud.version>2021.0.4</spring-cloud.version>
        </properties>
        <dependencies>
                <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>

</project>
```

2. Add @EnableEurekaServer annotation

We need to add `@EnableEurekaServer` annotation to make our SpringBoot application a Eureka Server-based Service Registry.

```
package net.javaguides.serviceregistry;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

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

}
```

3. Disable Eureka Server as Eureka Client

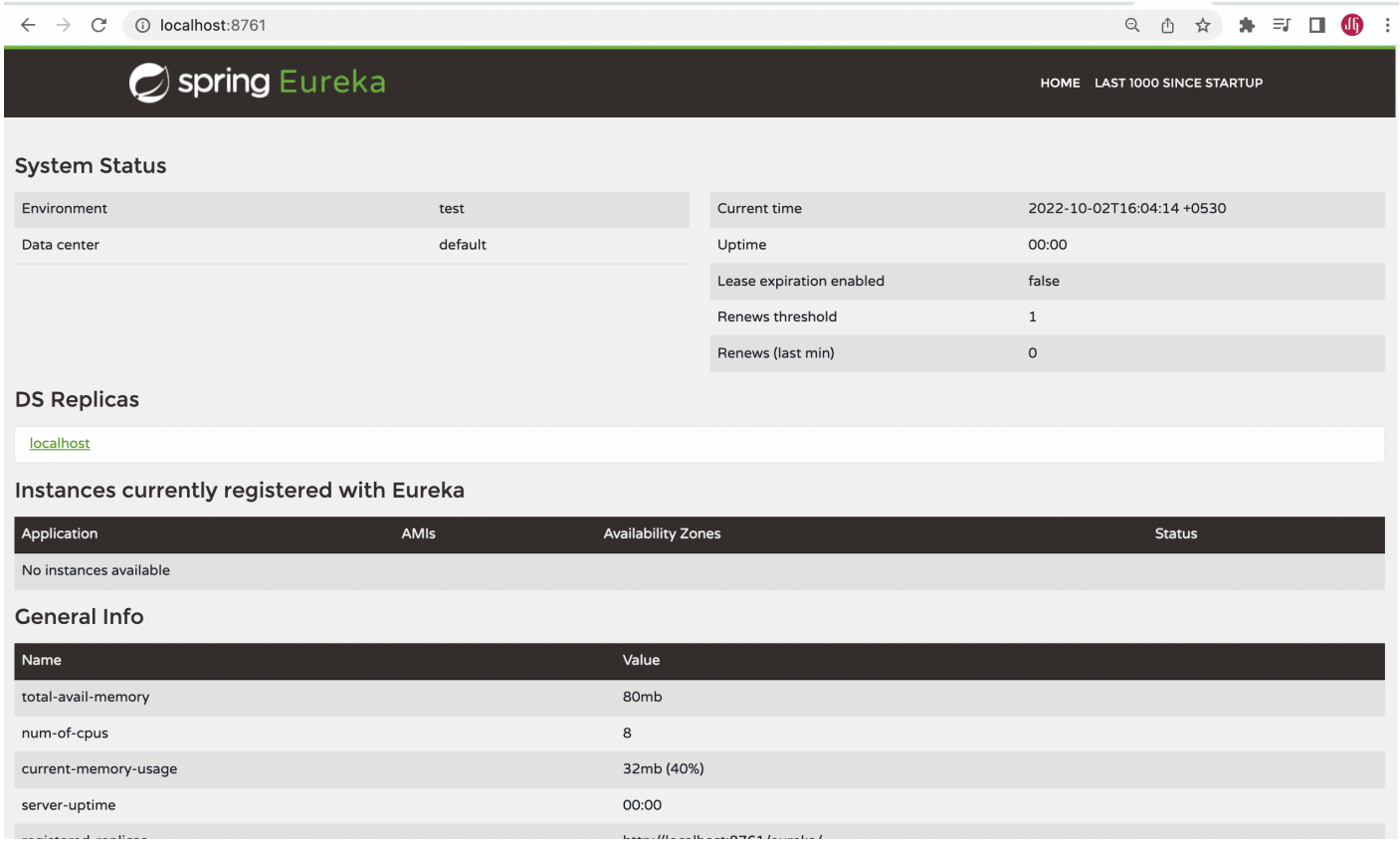
By default, each Eureka Server is also a Eureka client and needs at least one service URL to locate a peer. As we are going to have a single Eureka Server node (Standalone Mode), we are going to disable this client-side behavior by configuring the following properties in the `application.properties` file.

```
spring.application.name=service-registry
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

4. Launch Eureka Server (Demo)

Netflix Eureka Service provides UI where we can see all the details about registered services.

Now run `ServiceRegistryApplication` and access **http://localhost:8761** which will display the UI similar to the below screenshot.



5. Registering Department-Service Microservice as Eureka Client

Refer to this tutorial to create `department-service` and `user-service` microservices: **Spring Boot Microservices Communication Example using RestTemplate**.

Let us make this `department-service` as a Eureka Client and register with the Eureka Server.

Add the Eureka Discovery starter to `department-service` :

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

Also, add the Spring cloud 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>
```

Add version as property:

```
<properties>
        <java.version>17</java.version>
```

```
        <spring-cloud.version>2021.0.4</spring-cloud.version>
</properties
```

With `spring-cloud-starter-netflix-eureka-client` on the classpath, we just need to configure `eureka.client.service-url.defaultZone` property in `application.properties` to automatically register with the Eureka Server.

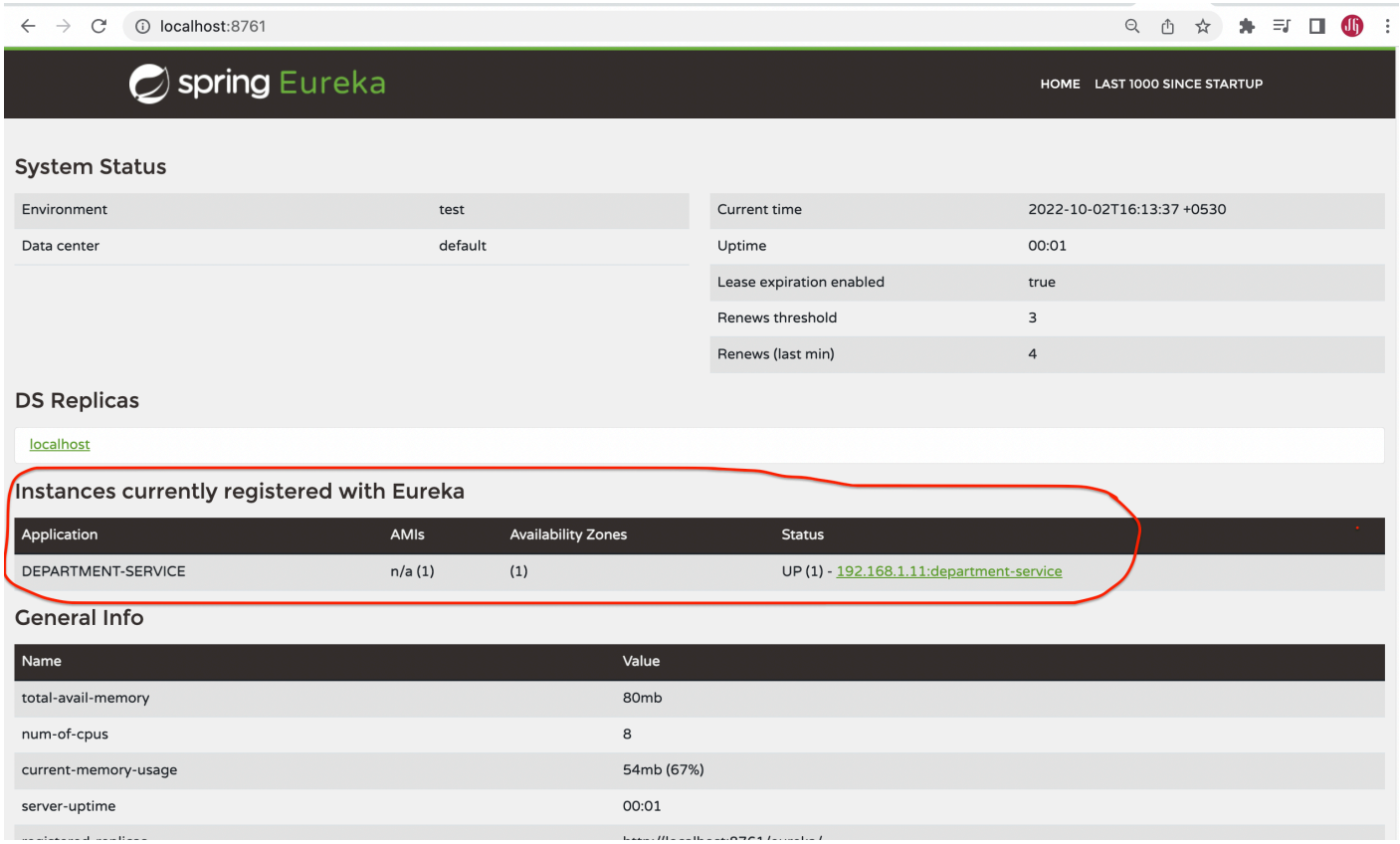
```
spring.application.name=DEPARTMENT-SERVICE
eureka.instance.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
```

When a service is registered with Eureka Server it keeps sending heartbeats for certain intervals. If the Eureka server didn't receive a heartbeat from any service instance it will assume the service instance is down and take it out from the pool.

6. Run department-service Eureka Client (Demo)

With this configuration in place, start `department-service` and visit **http://localhost:8761**.

You should see that `department-service` is registered with SERVICE ID as DEPARTMENT-SERVICE. You can also notice the status as UP(1) which means the services are up and running and one instance of `department-service` is running.



7. Registering User-Service Microservice as Eureka Client

Refer to this tutorial to create `department-service` and `user-service` microservices: **Spring Boot Microservices Communication Example using RestTemplate**.

Let us make this `user-service` as a Eureka Client and register with the Eureka Server.

Add the Eureka Discovery starter to `user-service`:

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

Also, add the Spring cloud 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>
```

Add version as property:

```
<properties>
        <java.version>17</java.version>
        <spring-cloud.version>2021.0.4</spring-cloud.version>
</properties
```

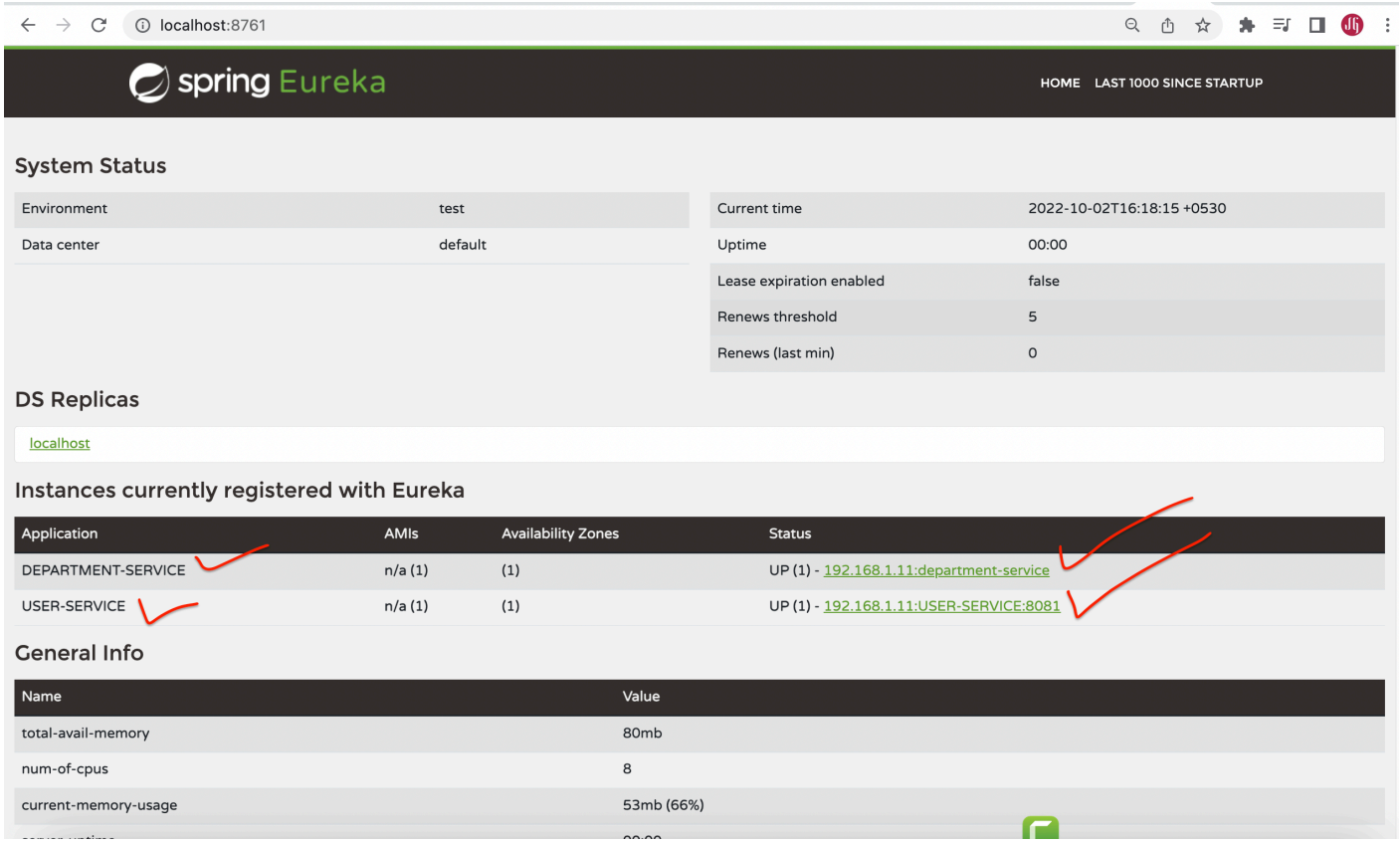
With `spring-cloud-starter-netflix-eureka-client` on the classpath, we just need to configure `eureka.client.service-url.defaultZone` property in `application.properties` to automatically register with the Eureka Server.

```
spring.application.name=USER-SERVICE
eureka.instance.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
```

8. Run user-service Eureka Client (Demo)

With this configuration in place, start user-service and visit **http://localhost:8761**. You should see `user-service` is registered with SERVICE ID as USER-SERVICE.

You can also notice the status as UP(1) which means the services are up and running and one instance of `user-service` is running.



Multiple Instances of Department-Service

Let us start another instance of `department-service` on a different port using the following command.

```
java -jar -Dserver.port=8082 department-service-0.0.1-SNAPSHOT.jar
```

Suppose we want to invoke the `department-service` REST endpoint from the `user-service`. We can use `RestTemplate` to invoke the REST endpoint but there are 2 instances running.

We can register RestTemplate as a Spring bean with `@LoadBalanced` annotation:

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

The `RestTemplate` with `@LoadBalanced` annotation will internally use **Ribbon LoadBalancer** to resolve the ServiceID and invoke the REST endpoint using one of the available servers.

Change the URL in `UserServiceImpl` class:

```
        ResponseEntity<DepartmentDto> responseEntity = restTemplate
                .getForEntity("http://DEPARTMENT-SERVICE/api/departments/" + user.getDepartmentId(),
                DepartmentDto.class);
```

With this kind of automatic Service Registration and Discovery mechanism, we no need to worry about how many instances are running and what are their hostnames and ports, etc

Conclusion

In this tutorial, we learned how to create a Service Registry using **Spring Cloud Netflix Eureka** in the Spring boot microservices project.