Creating a Service Registry

In a microservices architecture, services need to communicate with each other. However, tracking the exact location (IP address and port) of each service becomes challenging, especially when:

- Services are deployed across multiple machines
- Multiple instances of the same service are running
- Services are dynamically scaled up or down

This is where a Service Registry plays a crucial role in enabling microservices to find and communicate with each other.

👉 Netflix Eureka Server

Eureka Server is a service registry developed by Netflix and part of the Spring Cloud ecosystem. It provides a robust solution for service discovery in microservices architecture.

- ➢ Microservices register themselves with information about their location
- ➢ Microservices query to discover the location of other services
- ➢ Service health is monitored through regular heartbeat mechanisms

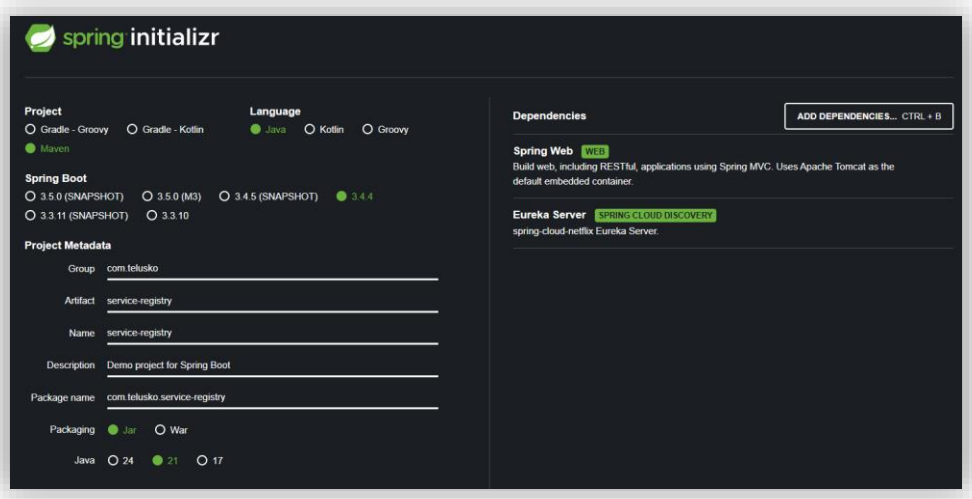
This approach eliminates the need for hardcoded service URLs and enables dynamic service discovery, making the microservices environment more resilient and flexible.

👉 Setting Up Eureka Server

➢ Creating the Project

Let's start by creating a dedicated project for our Eureka Server:

1. Go to Spring Initializer (https://start.spring.io/)
2. Add these dependencies
   - Spring Web
   - Eureka Server



➢ Configuring the Application

After downloading and opening the project in IDE, we need to make two important changes:

1. Enable Eureka Server in the Main Class

Add the @EnableEurekaServer annotation to the main application class:

```java
package com.telusko.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);
    }
}
```

The @EnableEurekaServer annotation activates the Eureka Server functionality, transforming this Spring Boot application into a service registry. This annotation configures the application to receive registration requests from microservices and respond to discovery queries.

2. Configure Application Properties

Add these properties to your application.properties file:

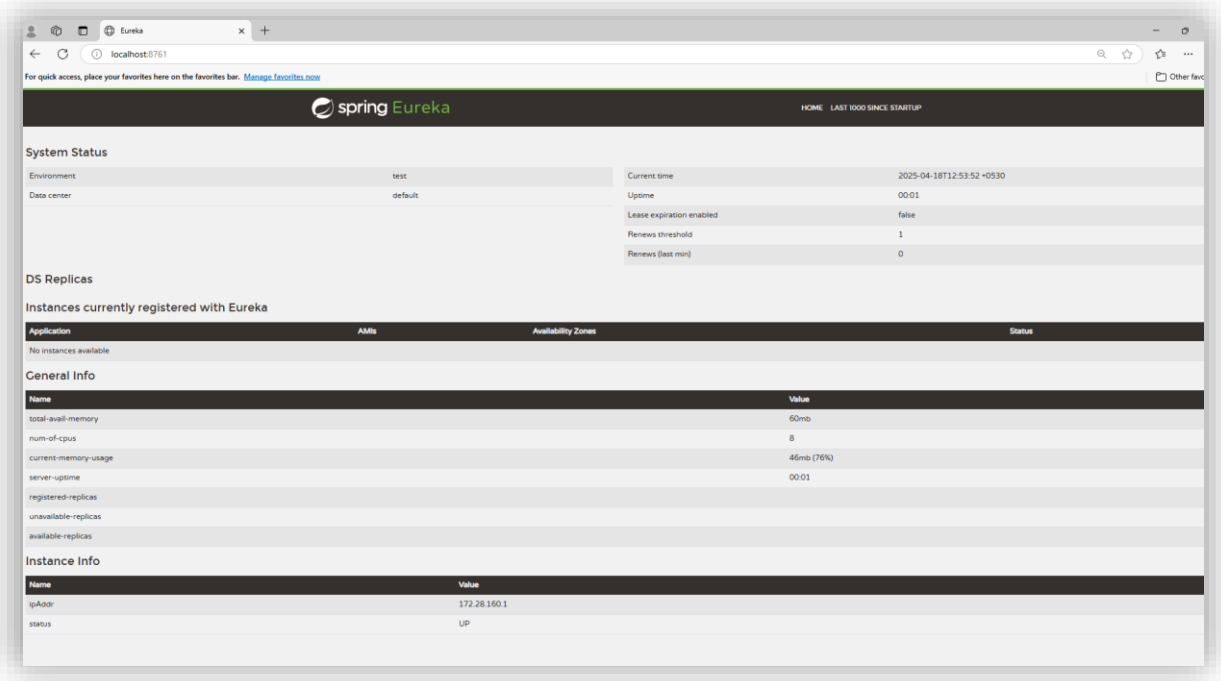
```properties
spring.application.name=service-registry
server.port=8761

eureka.instance.hostname=localhost
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

- **spring.application.name**: Identifies the application in the Spring ecosystem
- **server.port**: Sets the port to 8761, which is the standard port for Eureka Server
- **eureka.instance.hostname**: Specifies the hostname for this Eureka instance

- **eureka.client.fetch-registry**=false: Prevents the server from trying to fetch registry information (since it is the registry)
- **eureka.client.register-with-eureka**=false: Prevents the server from trying to register itself with itself

- When we run the application, Eureka Server starts on port 8761 and provides a dashboard where we can see registered service.



- The Eureka dashboard runs on http://localhost:8761 and provides a visual interface to monitor registered services. For local development, any services running on the same machine can automatically discover and register with this Eureka instance, making it easy to test microservices communication locally.

👉 Registering Services with Eureka

To connect our Question Service to the Eureka Server:

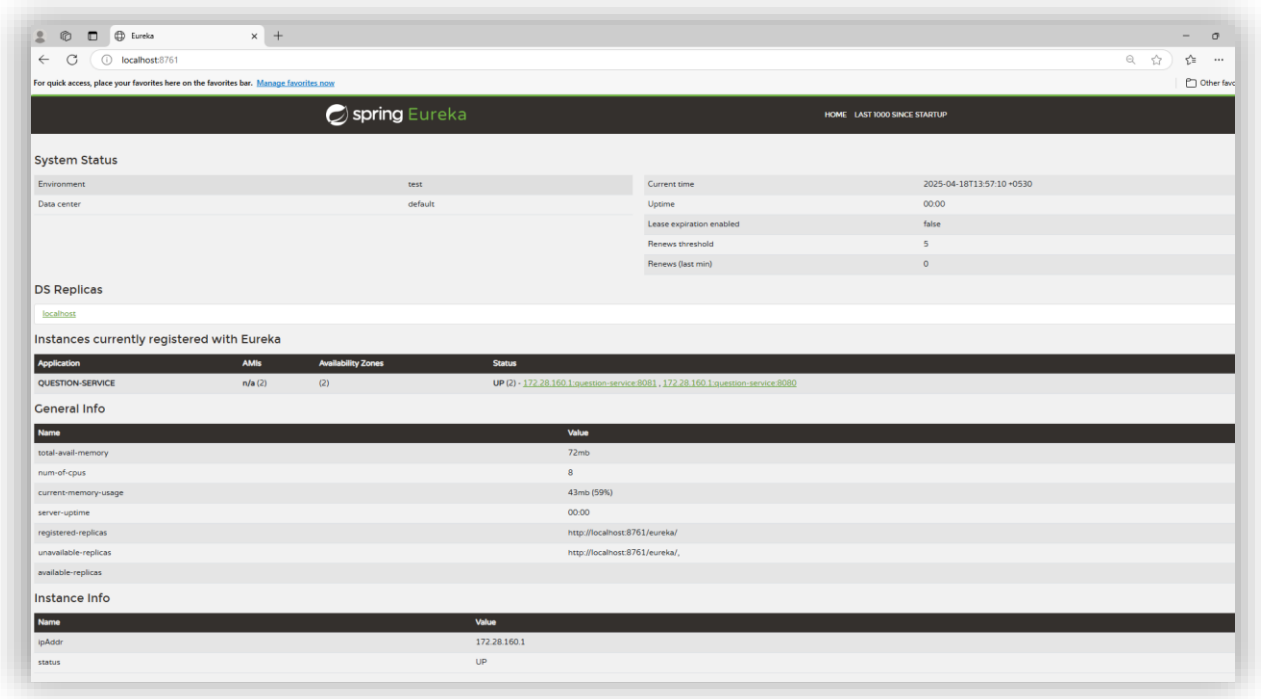
1. Uncomment the Eureka Client dependency in the pom.xml file
2. Add the service name to the application.properties file:

```
spring.application.name=question-service
```

3. Restart the application

- Once restarted, the Question Service will automatically register itself with the Eureka Server, and we'll see it appear in the Eureka dashboard.

- If we run multiple instances of our Question Service (on different ports), each instance will register separately, allowing Eureka to facilitate load balancing between them.



- With service registry in place, our microservices can now discover each other by name rather than relying on hardcoded URLs. This enables true service-to-service communication in a flexible, resilient way that supports the dynamic nature of microservices architectures.