Prof. Bernardo Copstein

Prof. Júlio Machado

Roteiro 2: Name Server e Balanceamento de Carga

Introdução

No roteiro 1 vimos como subir dois micros serviços e fazer os dois trocarem informações entre si. O resultado é o que pode ser visto na figura 1.

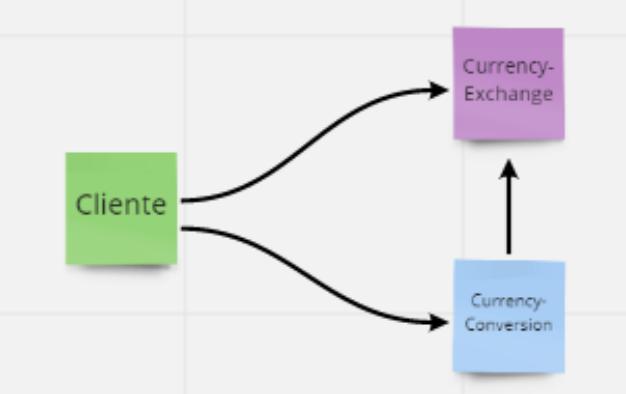


Figura 1 – sistema desenvolvido no roteiro 1

Embora a solução funcione perfeitamente, o micro serviço de conversão de moeda necessita saber em que porta o micro serviço de câmbio foi instalado. Isso pode se tornar um problema na medida que a complexidade do sistema aumenta e, principalmente, se quisermos subir novas instâncias do micro serviço de câmbio caso uma única instância fique sobrecarregada.

Para facilitar a questão da localização dos micros serviços e do balanceamento de carga vamos introduzir em nosso sistema um "name server" (servidor de nomes). A arquitetura do sistema ficará então como pode ser visto na figura 2.

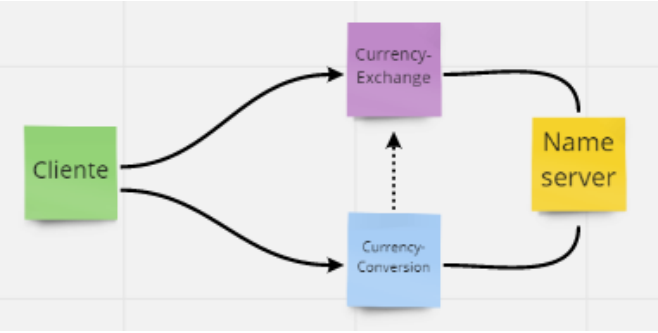


Figura 2 – arquitetura do sistema com o "name server"

Os micros serviços se registram no "name server". Quando o micro serviço de conversão deseja "conversar" com o micro serviço de câmbio ele o faz com o auxílio do "name server" que localiza a instância do micro serviço de câmbio que estiver menos "carregada" no momento.

A construção dessa arquitetura irá ocorrer em 4 passos.

Passo 1: criando o “name server”

Neste projeto iremos utilizar o “Eureka Server” desenvolvido pela Netflix e fornecido dentro do contexto do Spring. A tecnologia de implementação será o Java através do *framework* Spring. Adicionalmente o sistema de *build* será realizado através do Maven e, portanto, o primeiro passo será a configuração das dependências.

Para facilitar o processo, será utilizado o sistema “Spring Initializr” via web no endereço https://start.spring.io/. A figura 3 traz as configurações necessárias, já a figura 4 apresenta o arquivo de dependências correspondente. Clique no botão “GENERATE” e faça o download do projeto em um arquivo “zip” para um diretório local.

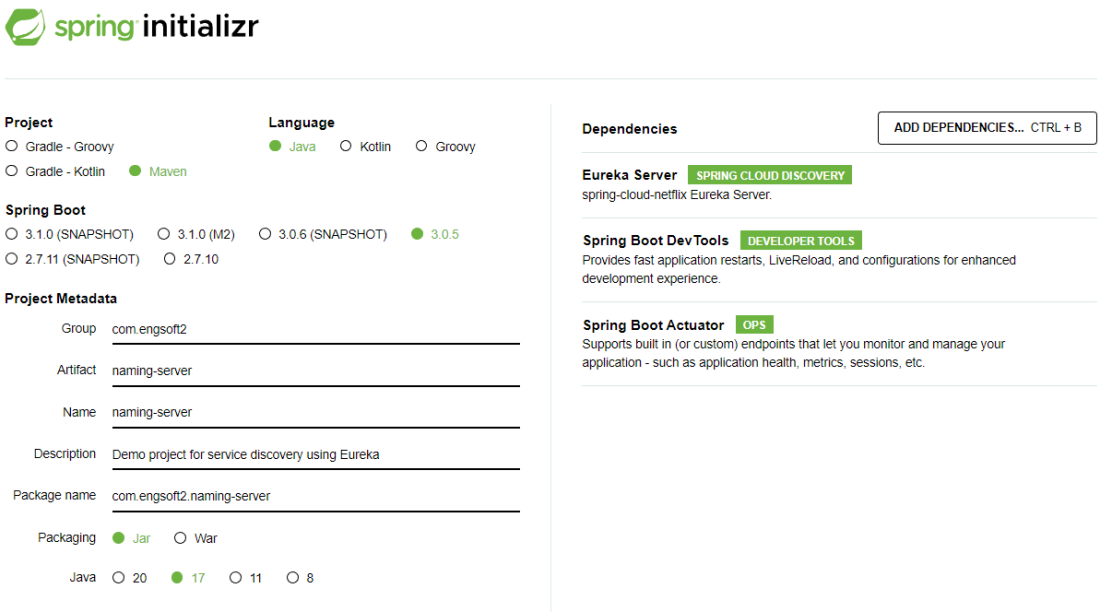


Figura 3 – configuração do Spring Initializr

```
<?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>3.0.5</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.engsoft2</groupId>
        <artifactId>naming-server</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>naming-server</name>
        <description>Demo project for service discovery using Eureka</description>
        <properties>
                <java.version>17</java.version>
                <spring-cloud.version>2022.0.2</spring-cloud.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</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-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</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-candidates</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</repository>
	</repositories>

</project>
```

Figura 4 - arquivo POM do “name server”.

O código da aplicação é muito simples. O aspecto mais importante é a anotação “@EnableEurekaServer”. Em relação as propriedades, além das que definem o nome do serviço e a porta – presentes em todos os micros serviços vistos até agora – destacam-se aquelas que evitam que o “Eureka” tente registrar a si mesmo. O código encontra-se na figura 5. As propriedades na figura 6.

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

@EnableEurekaServer
@SpringBootApplication
public class NamingServerApplication {

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

}
```

Figura 5 – código do “name server”

Além do código propriamente dito, necessitamos configurar algumas propriedades no arquivo “application.properties” no diretório “resources” como pode ser visto na figura 6.

```
spring.application.name=naming-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Figura 6 - arquivo de propriedades

Para subir o “Eureka Server” use o comando de sempre: “mvn spring-boot:run”. Em seguida consulte http://localhost:8761/. Deve aparecer a tela do servidor. Note que neste momento nenhum micro serviço aparece como registrado (ver figura 7). O próximo passo detalha como isso deve ocorrer.

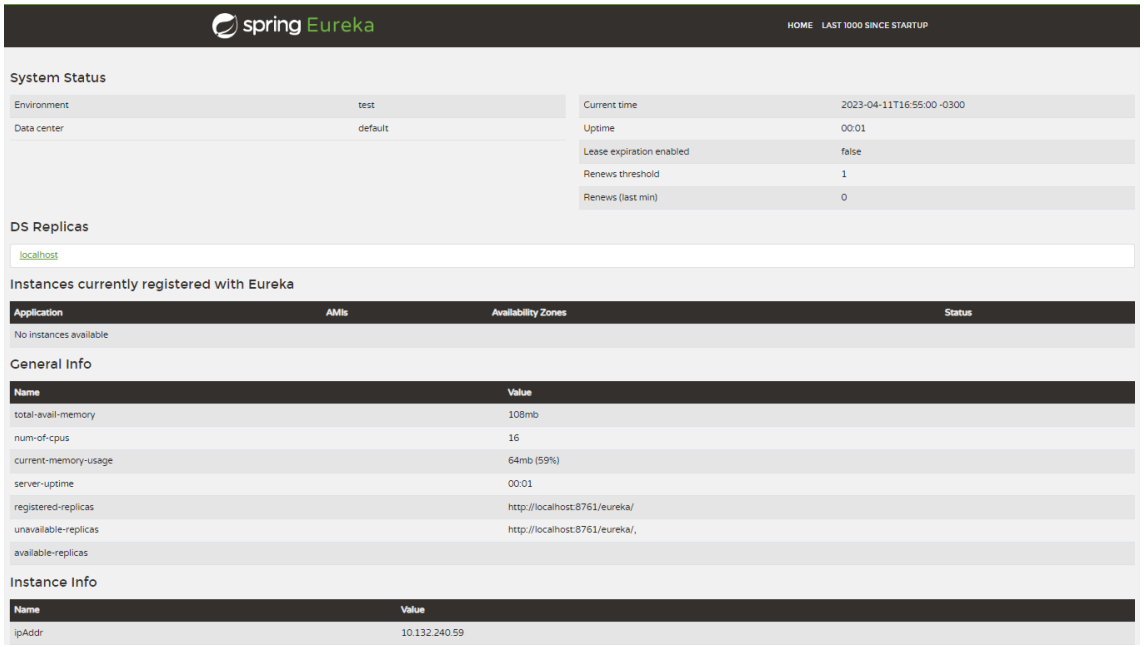


Figura 7 – status do “name server”

Passo 2: registrando os micros serviços no “name server”

Tanto o micro serviço de câmbio como o de conversão desenvolvidos no roteiro 1 devem ter suas dependências atualizadas para incluir o cliente para o sistema de descoberta de serviços. A figura 8 e a figura 9 apresentam a atualização que deve ser realizada a cada um dos arquivos “pom.xml”. A dependência acrescenta as classes necessárias para que a aplicação seja considerada um “cliente” do “Eureka” e consiga se registrar nele. Aplique a atualização após a tag “</description>” e antes da tag “</project>”.

```
	<properties>
		<java.version>17</java.version>
		<spring-cloud.version>2022.0.2</spring-cloud.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
		</dependency>
		<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-
client</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>com.h2database</groupId>
			<artifactId>h2</artifactId>
			<scope>runtime</scope>
		</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-candidates</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</repository>
	</repositories>
```

Figura 8 – atualização para o cliente Eureka no micro serviço “currency-exchange”

```
<properties>
    <java.version>17</java.version>
    <spring-cloud.version>2022.0.2</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

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

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </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-
candidates</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>
```

Figura 9 – atualização para o cliente Eureka no micro serviço “currency-conversion”

Também é necessário atualizar o arquivo de propriedades com o valor indicado na figura 10. A propriedade indica onde o “Eureka” está localizado de maneira que o cliente consiga se registrar.

```
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

Figura 10 – propriedades dos clientes

Passo 3: comunicando através do “name server”

Do ponto de vista do “cliente” o uso do “name server” não faz diferença. O “name server” tem dois papéis importantes: simplificar a localização de um micro serviço por outro e controlar o balanceamento de carga. Neste passo vamos ver como funciona a localização de um micro serviço por outro.

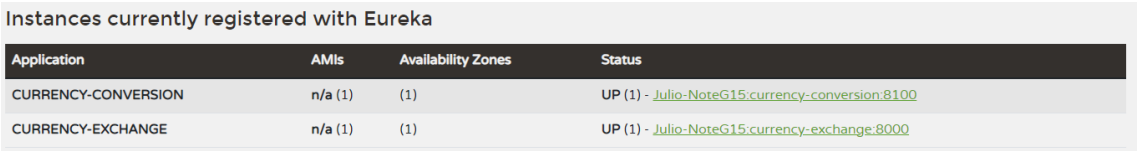
Da forma como fizemos no roteiro 1, o micro serviço de conversão precisa conhecer a porta onde o micro serviço de câmbio está escutando. A porta é indicada na anotação que antecede a interface “proxy”:

```
@FeignClient(name="currency-exchange", url="localhost:8000")
```

A partir do momento que trabalhamos com o “name server”, não é mais necessário indicar a porta, deixando esta tarefa para o “name server” alterando a anotação como segue:

```
@FeignClient(name="currency-exchange")
```

Uma vez feitas estas alterações, suba todos os micros serviços na ordem: “name server”, câmbio e conversão (crie uma janela para disparar cada um como feito no roteiro 1). Acesse a URL do “name server” e observe que tanto o micro serviço de câmbio como o de conversão estão registrados (veja figura 11).


Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION	n/a (1)	(1)	UP (1) - Julio-NoteG15:currency-conversion:8100
CURRENCY-EXCHANGE	n/a (1)	(1)	UP (1) - Julio-NoteG15:currency-exchange:8000

Figura 11 – registro dos micros serviços no “name server”

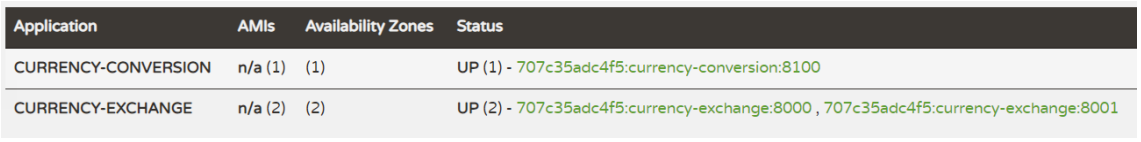
Teste o uso dos dois “endpoints” com as seguintes requisições HTTP:

http://localhost:8000/currency-exchange/from/USD/to/INR

http://localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10

Passo 4: explorando balanceamento de carga

Para podermos ver o balanceamento de carga em ação teremos de criar uma instância adicional do serviço de câmbio. Para tanto duplique a pasta do serviço de conversão, altere a propriedade que define a porta de execução, por exemplo para 8001, e dispare o novo serviço como feito antes. Verifique se as duas instancias estão registradas no “name server” como mostra a figura 12.



Application	AMIs	Availability Zones	Status
CURRENCY-CONVERSION	n/a (1)	(1)	UP (1) - 707c35adc4f5:currency-conversion:8100
CURRENCY-EXCHANGE	n/a (2)	(2)	UP (2) - 707c35adc4f5:currency-exchange:8000 , 707c35adc4f5:currency-exchange:8001

Figura 12 – Registro de duas instâncias do serviço de câmbio

Uma vez que as duas instâncias estão ativadas acione o serviço de conversão usando "localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10". Se você acionar este serviço repetidas vezes verá que o parâmetro "environment" da resposta uma vez indica a porta 8000 e na seguinte a porta 8001, mostrando que o serviço de balanceamento de carga está funcionando de acordo (veja a figura 13).

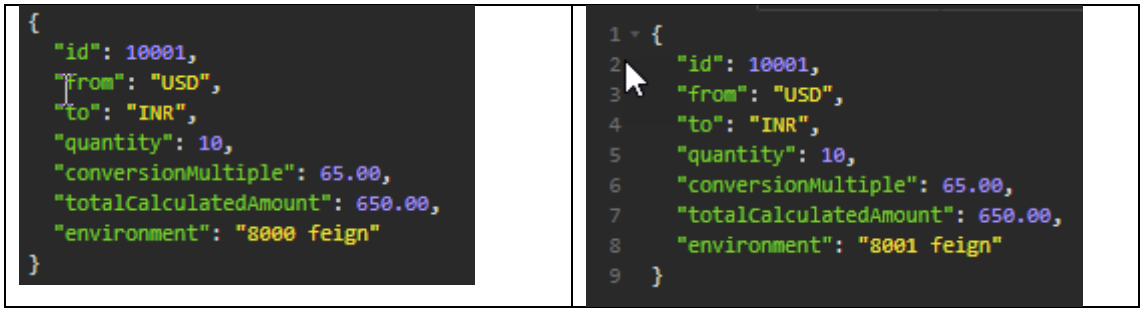


Figura 13 – alternância dos micros serviços em função do balanceamento de carga

Exercício

Verifique se o serviço de coleta de informações criado no exercício anterior está perfeitamente integrado ao "name server". Teste com mais de uma instancia do mesmo.