Spring Boot Microservices API Gateway with Automatic Routes Mapping

author: Ramesh Fadatare

MICROSERVICES SPRING BOOT SPRING CLOUD

In this tutorial, we will create two Spring Boot microservices and set up an API Gateway that automatically maps routes to these microservices using Eureka for service discovery. We'll use the latest Spring Boot version 3.2+ and Spring Cloud 2023.x. This guide is intended for beginners and includes detailed explanations for each step.

Prerequisites

- JDK 17 or later
- Maven or Gradle
- IDE (IntelliJ IDEA, Eclipse, etc.)

Step 1: Set Up the Eureka Server

1.1 Create the Project

Use Spring Initializr to create a new project with the following dependencies:

- Eureka Server

1.2 Configure `application.properties`

Set up the application properties for the Eureka Server.

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

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

Explanation:

- `server.port=8761`: Sets the port for the Eureka Server.
- `spring.application.name=eureka-server`: Names the application.
- `eureka.client.register-with-eureka=false`: Indicates that the Eureka Server itself should not try to register with another Eureka Server.
- `eureka.client.fetch-registry=false`: Indicates that the Eureka Server should not attempt to fetch registry information from another Eureka Server.

1.3 Enable Eureka Server

Add the `@EnableEurekaServer` annotation to the main application class.

```
package com.example.eurekaserver;

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

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

Explanation:

- `@EnableEurekaServer`: Enables the Eureka Server functionality in your Spring Boot application.

Step 2: Set Up `service-a`

2.1 Create the Project

Use Spring Initializr to create a new project with the following dependencies:

- Spring Web
- Eureka Discovery Client

2.2 Configure `application.properties`

Set up the application properties for `service-a`.

```
server.port=8081
spring.application.name=service-a

eureka.client.service-url.default-zone=http://localhost:8761/eureka/
```

Explanation:

- `server.port=8081`: Sets the port for the Product Service.
- `spring.application.name=service-a`: Names the application.
- `eureka.client.service-url.default-zone=http://localhost:8761/eureka/`: Specifies the Eureka Server URL for service registration.

2.3 Enable Eureka Client

Add the `@EnableDiscoveryClient` annotation to the main application class.

```
package com.example.servicea;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class ServiceAApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceAApplication.class, args);
    }
}
```

Explanation:

- `@EnableDiscoveryClient`: Indicates that this application should register with a Eureka Server for service discovery.

2.4 Create a Controller

Create a controller to handle requests.

```
package com.example.servicea;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ServiceAController {

    @GetMapping("/service-a")
    public String getServiceA() {
        return "Response from Service A";
    }
}
```

Explanation:

- `@RestController`: Marks this class as a REST controller.
- `@GetMapping("/service-a")`: Maps GET requests to `/service-a` to this method.

Step 3: Set Up `service-b`

3.1 Create the Project

Use Spring Initializr to create a new project with the following dependencies:

- Spring Web
- Eureka Discovery Client

3.2 Configure `application.properties`

Set up the application properties for `service-b`.

```
server.port=8082
spring.application.name=service-b

eureka.client.service-url.default-zone=http://localhost:8761/eureka/
```

Explanation:

- `server.port=8082`: Sets the port for the Order Service.
- `spring.application.name=service-b`: Names the application.
- `eureka.client.service-url.default-zone=http://localhost:8761/eureka/`: Specifies the Eureka Server URL for service registration.

3.3 Enable Eureka Client

Add the `@EnableDiscoveryClient` annotation to the main application class.

```
package com.example.serviceb;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class ServiceBApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceBApplication.class, args);
    }
}
```

Explanation:

- `@EnableDiscoveryClient`: Indicates that this application should register with a Eureka Server for service discovery.

3.4 Create a Controller

Create a controller to handle requests.

```
package com.example.serviceb;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ServiceBController {

    @GetMapping("/service-b")
    public String getServiceB() {
        return "Response from Service B";
```

```
    }
}
```

Explanation:

- `@RestController`: Marks this class as a REST controller.
- `@GetMapping("/service-b")`: Maps GET requests to `/service-b` to this method.

Step 4: Set Up the API Gateway

4.1 Create the Project

Use Spring Initializr to create a new project with the following dependencies:

- Spring Cloud Gateway
- Eureka Discovery Client

4.2 Configure `application.properties`

Set up the application properties for the API Gateway.

```
server.port=8080
spring.application.name=api-gateway

eureka.client.service-url.default-zone=http://localhost:8761/eureka/
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```

Explanation:

- `server.port=8080`: Sets the port for the API Gateway.
- `spring.application.name=api-gateway`: Names the application.
- `eureka.client.service-url.default-zone=http://localhost:8761/eureka/`: Specifies the Eureka Server URL for service registration.
- `spring.cloud.gateway.discovery.locator.enabled=true`: Enables the Discovery Locator to automatically map routes.
- `spring.cloud.gateway.discovery.locator.lower-case-service-id=true`: Converts service IDs to lowercase.

4.3 Enable Eureka Client and Gateway

Enable Eureka client and gateway functionality by adding the `@EnableDiscoveryClient` annotation in the main application class.

```
package com.example.apigateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
```

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

Explanation:

- `@EnableDiscoveryClient`: Indicates that this application should register with a Eureka Server for service discovery.

Step 5: Run the Microservices

1. **Start the Eureka Server**: Run the `EurekaServerApplication` class.
2. **Start `service-a`**: Run the `ServiceAApplication` class.
3. **Start `service-b`**: Run the `ServiceBApplication` class.
4. **Start the API Gateway**: Run the `ApiGatewayApplication` class.

Step 6: Test the Communication

Open your browser or use a tool like Postman to test the endpoints through the API Gateway:

- **`service-a`**: `http://localhost:8080/service-a/service-a`
- **`service-b`**: `http://localhost:8080/service-b/service-b`

The response from `service-a` and `service-b` should include the respective messages defined in their controllers.

Conclusion

You have successfully set up two Spring Boot microservices and an API Gateway with automatic route mapping using Eureka Server for service discovery. The API Gateway dynamically discovers the services registered with Eureka and routes the requests appropriately.

A quick way to enable Spring Cloud API Gateway to route HTTP requests to a Microservice registered with the discovery service is to enable the Discovery Locator. To enable the automatic mapping of routes in your Spring Cloud API Gateway project, open the application.properties file and add the following configuration properties.

```
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```

- `spring.cloud.gateway.discovery.locator.enabled=true`: Enables the Discovery Locator to automatically map routes.
- `spring.cloud.gateway.discovery.locator.lower-case-service-id=true`: Converts service IDs to lowercase.