Eureka Service Discovery

Table of Contents

1. Introduction to Microservices and Service Discovery

Microservices architecture breaks down applications into smaller, independently deployable services, each responsible for a specific business functionality. In a microservices environment, there are typically several services communicating with each other, sometimes across multiple instances, which can dynamically scale up or down.

Challenge:

With microservices, it becomes difficult to manage the dynamic nature of these services. The services need a way to discover each other without relying on hardcoded IP addresses or hostnames.

Solution:

Service Discovery is the process by which microservices automatically detect and communicate with one another.

2. Importance of Service Discovery

In a distributed system, service instances are ephemeral; they can start and stop frequently depending on traffic demands, failures, or updates. Here's why service discovery is critical:

1. Dynamic Nature of Microservices: With autoscaling and frequent redeployments, service IPs change dynamically, making hardcoding impractical.

2. Fault Tolerance: Service discovery mechanisms help reroute traffic when an instance fails.

3. Load Balancing: Service discovery integrates with load balancers, distributing the traffic evenly across multiple instances of a service.

4. Simplified Configuration: Services don't need to know in advance about other services' locations; they can ask the service registry for the current instances.

Two Types of Service Discovery:

1. Client-side Discovery: The client queries a service registry to get the location of a service. This is common in systems like Netflix Eureka.

2. Server-side Discovery: The client communicates with a load balancer or API gateway, which uses the registry to locate the service.

3. Eureka: Service Registration and Discovery

Netflix Eureka is a service registry developed by Netflix to solve the dynamic service discovery problem in their microservices ecosystem. It allows services to register themselves and discover other services.

Key Components:

1. Eureka Server: The central component where all services register themselves.

2. Eureka Client: Each microservice that registers with Eureka acts as a client. These clients can also query Eureka to discover other services.

3. Service Registry: A dynamic database of services that are available in the system. Eureka Server maintains this registry.

4. Implementing Service Discovery Using Eureka

To implement Eureka, we will configure both the Eureka Server and the Eureka Client.

Step 1: Setting Up the Eureka Server

1. Create a New Spring Boot Application for Eureka Server:

- Add the necessary dependencies for Eureka Server in pom.xml:

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

2. Enable Eureka Server in the main class using @EnableEurekaServer:

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

3. Configure the Eureka Server in application.yml:

```
server:
  port: 8761
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
  instance:
    hostname: localhost
```

4. Run the Eureka Server:

Once you run the server, you can access the Eureka Dashboard at http://localhost:8761. Here, you'll be able to see all registered services.

Step 2: Setting Up the Eureka Client

1. Add Dependencies for Eureka Client in your microservice's pom.xml:

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

2. Enable Eureka Client in the main class by adding @EnableEurekaClient:

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

3. Configure Eureka Client in application.yml:

```
server:
  port: 8081
spring:
  application:
    name: product-service
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

4. Run the Eureka Client:

The client will automatically register itself with Eureka Server. If you go to the Eureka Dashboard, you should see the service listed under 'Instances currently registered with Eureka.'

5. Key Eureka Features

Self-Preservation: Eureka is built to handle service failures gracefully by employing a self-preservation mode. This prevents Eureka from expiring services too aggressively in case of network partitions or other failures.

Instance Discovery: Eureka clients can use either polling or push-based updates to discover service instances.

Health Monitoring: Eureka clients can periodically send heartbeats to inform the server that they are still alive. Eureka removes instances that stop sending heartbeats from its registry.

6. Summary

In a microservices architecture, service discovery is essential to ensure that services can dynamically discover each other. Netflix Eureka provides a reliable and scalable solution for service registration and discovery. By implementing Eureka in your microservices, you can manage dynamic service locations, improve fault tolerance, and simplify inter-service communication.