Eureka Server

Microservices

Beschreibung

«Eureka Server» ist eine Softwareplattform und Service-Discovery-Server, die für die Verwaltung und das Durchsuchen von Daten und Informationen in Unternehmens-Applikationen entwickelt wurde. «Eureka» ursprünglich von Netflix entwickelt und jetzt Teil des Spring Cloud-Frameworks ist.

Inhaltsverzeichnis

1 Eureka Server

1.1 Einleitung

Eureka hilft bei der Verwaltung und Lokalisierung von Microservices in einer verteilten Anwendung. Es ermöglicht den Microservices, sich automatisch zu registrieren und Informationen über ihre Verfügbarkeit bereitzustellen, sodass andere Dienste sie leicht finden können.

1.2 Einsatzbereiche und Beispiele

Die folgenden Beispiele zeigen Einsatzbereiche für den Eureka Dienst:

Service-Registrierung und Discovery:

- Eureka kann verwendet werden, um die Microservices einer Applikation, wie Authentifizierungsservice, Produktverwaltungsservice, usw., zu registrieren. Jeder Service meldet sich dabei beim Eureka-Server an. Der Eureka-Server führt Buch darüber, welche Dienste verfügbar sind.

Service-to-Service-Kommunikation:

- Wenn Microservices untereinander kommunizieren müssen, kann der Eureka-Server verwendet werden, um die Adressen der benötigten Dienste herauszufinden. Statt feste IP-Adressen oder URLs zu verwenden, können die Microservices dynamisch die Dienste über den registrierten Namen finden und ansprechen.

Lastenausgleich:

- Eureka kann auch dazu verwendet werden, Lastenausgleichsfunktionen zu unterstützen. Wenn mehrere Instanzen desselben Dienstes bereitstehen, kann Eureka den Lastenausgleich auf diese Instanzen verteilen.

Hohe Verfügbarkeit:

- Durch die Verwendung von Eureka-Servern in verschiedenen Regionen oder Zonen kann eine hohe Verfügbarkeit für die Service-Discovery-Funktionalität gewährleistet werden.

2 Eine einfache Microservice Architektur aufbauen

2.1 Einleitung

In den folgenden Abschnitten werden REST-API-Services in einer Microservice Architektur mit Spring-Boot aufgebaut. Die Anwendung und die Services sind exemplarisch und dienen der Illustrierung vom Umgang mit dem «Eureka-Server».

Um zwei Spring Boot REST-API-Services mit Eureka zu verbinden, benötigen Sie Spring Cloud, dass eine einfache Integration mit Eureka ermöglicht.

2.2 Einrichten der Spring Boot-Anwendungen

Erstellen Sie zwei separate Spring Boot-Projekte für die beiden Services. Sie können hierfür Spring Initializr innerhalb von Intellij verwenden.

```
Spring-Boot:  version: 3.1.2
Java-Version: SDK 17 / Language-Level 17 (corretto)
GroupId:      ch.bbw
ArtifactId:   eureka-server-app und service-01
Depencencies: Spring-Web, Spring-Dev-Tools, Lombok
```

2.3 Konfigurieren der Eureka-Server- und -Client-Einstellungen

Im Projekt Eureka-Server-App konfigurieren Sie zusätzlich den Eureka-Server und im anderen den Eureka-Client. Überprüfen Sie ob allenfalls neuer Versionen verfügbar sind.

Eureka-Server-App

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    <version>4.0.2</version>
</dependency>
```

Service-01

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    <version>4.0.2</version>
</dependency>
```

2.4 Services konfigurieren

Die Services müssen als Client und als Server konfiguriert werden, wie auch einzeln auf separaten Ports laufen, damit diese auf einem «Computer» nebeneinander ausgeführt werden können. Die Einstellungen werden in der `application.properties` Datei gemacht (wahlweise auch als yml).

```
# Einstellungen für den Eureka Server
spring.application.name=eureka-server-app
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.renewalPercentThreshold=0.85
eureka.instance.lease-renewal-interval-in-seconds=10
eureka.instance.lease-expiration-duration-in-seconds=30
```

Die Einstellungen für den Client sehen dann wie folgt aus:

```
# Eureka Client
spring.application.name=service-01
server.port=8081

eureka.client.service-url.default-zone=http://localhost:8761/eureka/
eureka.instance.lease-renewal-interval-in-seconds=5
```

Dabei ist es wichtig, die Service Namen zu setzen, sowie unterschiedliche Ports zu verwenden.

2.5 Aktivieren der Eureka-Registrierung

Fügen Sie im Eureka-Server-Projekt die folgende Annotation zur Hauptklasse hinzu, um die Eureka-Client-Registrierung zu aktivieren. Die Annotation startet die Kommunikation zwischen Eureka-Server und Client im Hintergrund und ermöglicht so das Auffinden bzw. das Registrieren der Services.

```
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

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

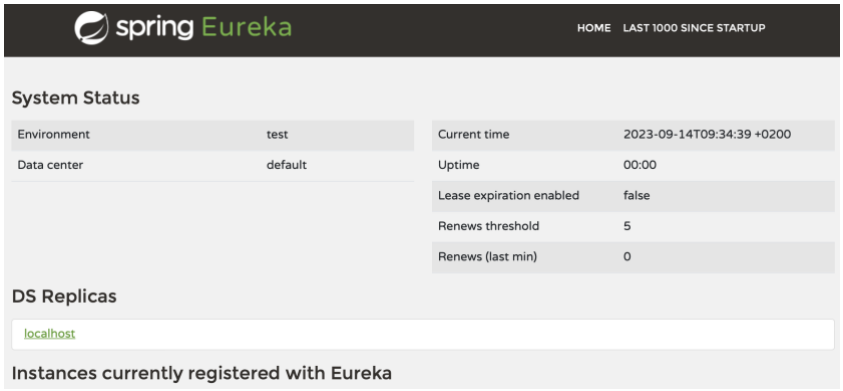
Beim Client braucht es keine Registrierung, da der Client automatisch «aktiviert» wird, wenn die entsprechende Dependency in der `pom.xml` Datei aufgenommen und in der `application.properties` Datei wird.

2.6 Aufstarten der Anwendung

Starten Sie zuerst den Eureka-Server und dann etwas verzögert den Eureka-Client (Service-01). Der Eureka-Client sollte sich automatisch beim Eureka-Server registrieren. Hierfür muss der Server bereits laufen.

2.7 Überprüfung der Registrierung

Öffnen Sie einen Webbrowser und gehen Sie zur URL `http://localhost:8761`. Auf dem Eureka-Dashboard sollten der registrierten Dienstnamen des Eureka-Clients zu sehen sein. Somit sollte der Service-01 (Boot REST-API-Services) erfolgreich mit dem Eureka-Server verbunden sein



3 Aufgaben

3.1 Aufgabe 1 – Grundaufbau der Anwendung

Implementieren Sie die beiden Projekte gemäss der obigen Anleitung und überprüfen Sie deren Funktionalität entsprechend.

Implementieren Sie im Service-01 Projekt auch die folgende API mit der Datenklasse:

MainController

```
@RestController
public class MainController {

    @GetMapping("api")
    public MyData getData() {
        return new MyData("Hello World from Service 01");
    }
}
```

MyData

```
@Data
@AllArgsConstructor
public class MyData {
    private String name;
}
```

→ Starten Sie den Service-01 und rufen Sie die API über die entsprechende URL auf.

→ Wechseln Sie zurück auf das Eureka-Dashboard und überprüfen Sie, ob der neue Service auf dem Eureka Server registriert wurde.

3.2 Aufgabe 2 – Dritten Service implementieren

Implementieren Sie einen dritten Service mit dem Namen Service-02.

Passen Sie die folgenden Punkte entsprechend an:

- Name des Projektes: `Service-02`
- Name der Applikation: `Service02Application`
- Meldung im API-Endpunkt: "`Hello World from` **`Service 02`**"
- Passen Sie in der application.properties Datei den Namen des Service an.
- Passen Sie in der application.properties Datei den Port an. Nehmen Sie den **Port 8082**.

→ Starten Sie nun die Service-02 App und kontrollieren Sie die Registrierung im Eureka Server.

→ Rufen Sie die API und den Endpunkt auf und kontrollieren Sie die Daten.

→ Wechseln Sie zurück auf das Eureka-Dashboard und überprüfen Sie, ob auch der neue Service auf dem Eureka Server registriert wurde.