Assignment

To incorporate the relevant microservice design patterns (Service Registry/Discovery, Client-side Load Balancing, and API Gateway) into a previous microservice-based implementation using Spring Cloud, we can modify the architecture and implement the following components:

**1. Service Registry/Discovery**

**Service Registry** is a crucial component that allows services to register themselves so that other services can discover them. **Spring Cloud Netflix Eureka** is commonly used for this purpose.

**Steps to implement Service Registry with Spring Cloud Eureka:**

1. **Add dependencies** in pom.xml for both the Eureka server and client.

**Eureka Server**:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

**Eureka Client** (for each microservice):

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

1. **Create Eureka Server**:

In main Spring Boot application class,to enable Eureka Server:

@SpringBootApplication

@EnableEurekaServer

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

Add necessary configuration (application.yml )

spring:

application:

name: EduEurekaServer

server:

port: 8761

eureka:

client:

registerWithEureka: false

fetchRegistry: false

server:

waitTimeInMsWhenSyncEmpty: 0

enableSelfPreservation: false

1. **Configure Eureka Client for Microservices**:

In each of r microservices, need to register them as Eureka clients.

Enable Eureka client in the Spring Boot application class:

@SpringBootApplication

@EnableEurekaClient

public class EeduexcellenceStudentMsApplication{

public static void main(String[] args) {

SpringApplication.run(EeduexcellenceStudentMsApplication.class, args);

}

}

Configure the client to connect to the Eureka Server in application.propertis:

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

1. **Start Eureka Server** and **Microservices**. The microservices will register themselves with Eureka, and other services can discover them.

**3. API Gateway**

An **API Gateway** acts as a reverse proxy to route client requests to the appropriate microservice and handles cross-cutting concerns such as authentication, logging, and rate limiting. **Spring Cloud Gateway** can be used for this purpose.

**Steps to implement API Gateway with Spring Cloud Gateway:**

1. **Add dependencies** for Spring Cloud Gateway:
2. <dependency>
3. <groupId>org.springframework.cloud</groupId>
4. <artifactId>spring-cloud-starter-gateway</artifactId>
5. </dependency>

**Complete Architecture Example:**

1. **Eureka Server**: Manages the discovery of microservices.
2. **Microservices**: Multiple services like service-a, service-b, etc., are registered with Eureka and can communicate with each other via service discovery and client-side load balancing.
3. **API Gateway**: Routes requests from clients to the appropriate microservice with load balancing, authentication, etc.

**Final Summary:**

With these changes, 've introduced the following Spring Cloud patterns into r microservice architecture:

* **Service Registry/Discovery** via **Spring Cloud Eureka**
* **Client-side Load Balancing** via **Spring Cloud Ribbon** (or **Spring Cloud LoadBalancer**)
* **API Gateway** via **Spring Cloud Gateway**

These design patterns help scale, manage, and balance microservices efficiently while providing a unified access point for clients.