Microservices & Spring Cloud Lab Report

Amo Samuel

Introduction

In this project, I implemented an API Gateway using Spring Cloud Gateway to route requests to microservices, manage configuration centrally with Spring Cloud Config, and introduce fault tolerance with Spring Cloud Circuit Breaker. The goal was to create a robust and efficient architecture for a distributed system while ensuring security best practices.

1. API Gateway Implementation

Overview

The API Gateway serves as the single entry point for client requests, routing them to the appropriate microservices. I utilized Spring Cloud Gateway, which provides a simple and effective way to manage API routing, load balancing, and service discovery.

Configuration

In my application.yml file, I defined the necessary configurations for the API Gateway

```yaml
server:
  port: 8222
spring:
  cloud:
    gateway:
      discovery:
        locator:
          enabled: true
      routes:
        - id: product-service
          uri: lb:http://PRODUCT-SERVICE
          predicates:
            - Path=/api/v1/products/**
```

Key Elements

- Discovery Locator: Enabled service discovery to dynamically route requests to the PRODUCT-SERVICE.

- Routes: Defined a route with an ID product-service, which uses a load-balanced URI to route requests to the specified service.

2. Centralized Configuration Management

Overview

To centralize configuration management, I utilized Spring Cloud Config. This allows for externalizing configuration properties, making it easier to manage different environments.

Configuration

The application.yml file also included the following settings to import the configuration from the Spring Cloud Config server

```
spring:
  config:
    import: optional:configserver:http://localhost:8888
  application:
    name: gateway-service
```

Key Elements

- Config Server URL: Pointed to the Config Server running on localhost:8888.
- Application Name: Specified the application name as gateway-service for easy identification in the Config Server.

3. Fault Tolerance with Spring Cloud Circuit Breaker

Overview

To ensure fault tolerance, I implemented Spring Cloud Circuit Breaker, which provides resilience in microservice communication. I used Resilience4j as the circuit breaker implementation.

Implementation

In the OrderServiceClient class, I created a Feign client to communicate with the order-service

```java
@FeignClient(name = "order-service", url = "http://localhost:8030")  3 usages
public interface OrderServiceClient {

    Logger logger = LoggerFactory.getLogger(OrderServiceClient.class);  1 usage

    @CircuitBreaker(name = "orderServiceClient", fallbackMethod = "getOrderByIdFallback")
    @GetMapping("/api/v1/orders/{orderId}")
    OrderRequest getOrderById(@PathVariable("orderId") Long orderId);


    default OrderRequest getOrderByIdFallback(Long orderId, Throwable throwable) {  no usages
        logger.error("Circuit breaker triggered for orderId: {}. Error: {}", orderId, throwable.getMessage());
```

Key Elements

- Circuit Breaker Annotation: Annotated the getOrderById method with @CircuitBreaker, specifying a fallback method getOrderByIdFallback.
- Fallback Method: Implemented a fallback method to handle errors gracefully, logging the error and returning a user-friendly message.

4. Security Best Practices

- Authentication and Authorization: I plan to implement OAuth2 for secure access to the API Gateway and microservices. This will allow me to manage user authentication and authorization effectively.
- Rate Limiting: To prevent abuse, I am considering implementing rate limiting on the API Gateway to control the number of requests from clients.

Conclusion

By implementing the API Gateway using Spring Cloud Gateway, managing configurations centrally with Spring Cloud Config, and introducing fault tolerance with Spring Cloud Circuit Breaker, I have laid a solid foundation for a scalable and resilient microservices architecture. This approach not only enhances the maintainability of the system but also ensures that it can handle failures gracefully, providing a better user experience.

As I continue to refine this system, I will focus on enhancing security measures and exploring additional features to further improve the robustness and performance of the architect