Hochschule
Kaiserslautern
University of
Applied Sciences

Andreas Eitel
University of Applied Sciences
Kaiserslautern
andreas.eitel@hs-kl.de

SOFTWARE-ARCHITEKTUR:

AUFGABENPAKET 5

06. März 2022

Vorbemerkung

Das Hauptthema dieses Arbeitspakets ist „Verteilte Systeme“. In einer praktischen Übung soll das Thema Microservices weiter vertieft werden. Nachdem in der letzten Übung das Buchlager-Backend in Microservices zerlegt wurde, sollen in diesem Arbeitspaket die fehlenden Managementservices implementiert und alles verknüpft werden. Dabei bedienen wir uns am Netflix und Spring Technologiestack für Microservices. Abschließend wird ein bekanntes Microservice-Pattern implementiert.

Vorbereitende Literatur

Zur Unterstützung und als Startpunkt bei der Implementierung der Management-Services, kann auf die Tutorials [bae20] (Netflix Eureka) und [Kar20] (Zusammenspiel API Gateway mit Eureka mit automatischen Routen) zurückgegriffen werden.

Zur Implementierung des Circuit-Breaker-Patterns verwenden wir Resilience4j. Resilience4j bietet neben verschiedenen Add-ons 6 Kernmodule:

- resilience4j-circuitbreaker
- resilience4j-ratelimiter
- resilience4j-bulkhead
- resilience4j-retry
- resilience4j-cache
- resilience4j-timelimiter

Einen guten Einstieg, wie man die Resilience4j-Annotationen des Circuit-Breaker-Patterns verwendet, findet man in [Inc20].

Neue Spring-Boot-Projekte können Sie recht einfach unter https://start.spring.io/ generieren. In diesem Generator können Sie auch bequem die notwendigen Maven-Abhängigkeiten hinzufügen.

Praktischer Teil

Nutzen Sie eclipse oder IntelliJ zum Bearbeiten der nachfolgenden Aufgaben. Laden Sie Ihre Lösung als ZIP-Datei hoch. Build-Artefakte sollten nicht hochgeladen werden.

1. **Aufsetzen der Management-Services**
   Um eine kleine Microservice-Plattform aufzubauen, brauchen wir noch verschiedene Dienste zum Management unserer Microservices. Dabei implementieren wir nur das Minimum zur Übung.

   (a) (5 Punkte) Implementieren Sie eine Service-Discovery auf Basis von Spring-Boot mit Netflix Eureka [bae20]. Ein Basisprojekt mit entsprechenden Abhängigkeiten können Sie sich hier https://start.spring.io/ erstellen. Prüfen Sie, ob Ihre Service-Discovery läuft (http://localhost:8761/ [Port ist abhängig von Ihrer Konfiguration]). Benennen Sie für diese Übung die einzelnen Services nach den Namen in Abbildung 1.

   (b) (5 Punkte) Implementieren Sie ein API-Gateway auf Basis von Spring-Boot mit Spring Cloud Gateway. Achten Sie darauf, dass die Routen automatisch von der Service-Discovery bezogen werden [Kar20]. Das API-Gateway soll sich bei der Service-Discovery registrieren. Der Port sollte fest auf 8080 eingestellt werden. Ein Basisprojekt mit entsprechenden Abhängigkeiten können Sie sich hier https://start.spring.io/ erstellen. Für die UI (finden Sie im Übungspaket) müssen Sie eine statische Route in der application.yml konfigurieren:

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: ui_route
          uri: http://localhost:9090/ui/
          predicates:
            - Path=/**
```

   (c) (2 Punkte) Konfigurieren Sie die Buchlager-Microservices aus der vorherigen Übung so, dass sie sich bei der Service-Discovery registrieren [bae20]. Starten Sie anschließend die Service-Discovery, danach die Microservices und danach das API-Gateway. Überprüfen Sie, ob die Microservices alle über das API-Gateway erreichbar sind. Sie können einzelne Microservices auch mehrfach starten und den ein- oder anderen abschalten, um zu sehen, wie das API-Gateway darauf reagiert. Starten Sie auch den UI- und Lieferanten-Service. Die UI sollte danach noch so funktionieren wie zuvor.

2. **Circuit Breaker Pattern**
   Wenn ein Warenkorb mit Büchern gekauft wird und die vorhandene Menge an Büchern nicht ausreicht, soll vom jeweiligen Buch eine gewisse Menge (Ihre Wahl) beim Lieferanten nachbestellt werden. Der Lieferant wird durch einen eigenen Microservice repräsentiert, den Sie im Übungspaket finden und in Ihre Plattform einbinden können. Die grobe Darstellung finden Sie in Abbildung 1.

   (a) (5 Punkte) Im Warenkorb-Service wird mit OpenFeign ein REST-Request zur Nachbestellung beim Lieferenten an der passenden Stelle (im CartService) ausgeführt. Die Lieferanten-API finden Sie unter http://localhost:8999/swagger-ui.html. Entsprechende Modellklassen können Sie gerne dem Projekt entnehmen.

(b) (3 Punkte) Sichern Sie den Request an den Lieferanten-Service mit Hilfe des Circuit-Breaker-Patterns ab. Nutzen Sie zur Implementierung die Resilience4j-Bibliothek. In der Fallback-Methode soll eine Error-Lognachricht auf die Konsole ausgegeben werden (simuliert eine Behandlung des Fehlers). Solange es noch keine vollständige UI gibt, können Sie die Funktionen nur mit einem REST-Client wie der Swagger-UI oder Postman testen. Beachten Sie, dass die Signatur der Fallback-Methode als Parameter beispielsweise ein `Throwable`-Objekt verlangt:

```
1    public void errorWhileOrderingBooksFromSupplier(Throwable e) {
2        // todo
3    }
```

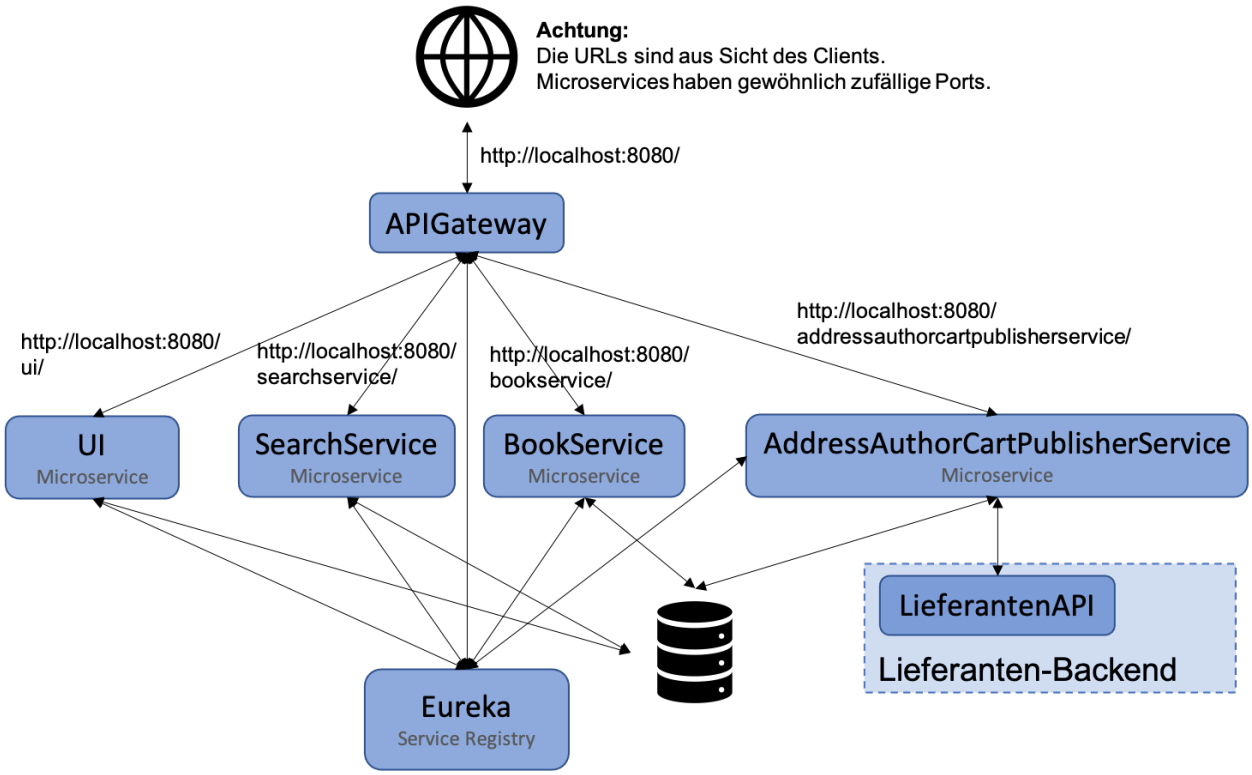

Abbildung 1: Zielausbau nach der Bearbeitung des Übungsblatts: Das Buchlager-Backend als Microservices mit Anbindung der Lieferanten-API (Web-Version).

Literatur

[bae20] BAELDUNG: *Introduction to Spring Cloud Netflix – Eureka*. https://www.baeldung.com/spring-cloud-netflix-eureka. Version: 2020

[Inc20] INC., Knoldus: *Circuit breaker with Resilience4j*. https://medium.com/@knoldus/circuit-breaker-with-resilience4j-23e0b8daba74. Version: 2020

[Kar20] KARGOPOLOV, Sergey: *Spring Cloud API Gateway Automatic Mapping of Routes*. https://www.appsdeveloperblog.com/spring-cloud-api-gateway-automatic-mapping-of-routes/. Version: 2020