Projektdefinition: Microservices-basierte Shop-Anwendung

Inhaltsverzeichnis

1. Einleitung

In der modernen Softwareentwicklung wird zunehmend die **Microservices-Architektur** genutzt, um Anwendungen skalierbar, wartbar und flexibel zu gestalten. Microservices erlauben es, unterschiedliche Funktionalitäten einer Anwendung in separate, unabhängige Dienste zu unterteilen, die unabhängig voneinander entwickelt, skaliert und gewartet werden können. In diesem Projekt wird eine Shop-Anwendung entwickelt, bei der diese Architektur in Form von verschiedenen Microservices umgesetzt wird. Jeder Microservice übernimmt eine spezifische Aufgabe im E-Commerce-Ökosystem.

Die Shop-Anwendung nutzt bewährte Technologien, um eine effiziente und robuste Lösung zu gewährleisten. Dazu gehören:

- **Spring Boot**: Für die Entwicklung der Microservices.
- **Docker**: Zur Containerisierung der Microservices, um die Bereitstellung und Skalierbarkeit zu erleichtern.
- **Kafka**: Für die asynchrone Kommunikation und eventbasierte Integration zwischen den Microservices.
- **Eureka**: Als Service Discovery-Tool, um die Kommunikation und das Auffinden der Microservices zu ermöglichen.
- **Resilience4J**: Für Fehlertoleranz und die Absicherung gegen Systemausfälle durch den Einsatz von Circuit Breakern.

2. Architekturübersicht

2.1 Frontend

Das Frontend der Anwendung wird mit **React** entwickelt. Es dient als Benutzeroberfläche für den Shop, über die Kunden Produkte durchsuchen und Bestellungen aufgeben können. Das Frontend kommuniziert über **REST-APIs** mit den Backend-Microservices, um produktbezogene Daten abzurufen, den Warenkorb zu verwalten und Bestellungen zu erstellen.

2.2 Backend (Microservices)

2.2.1 Warenkorb-ServiceFrontend

Der **Warenkorb-Service** verwaltet alle Produkte, die ein Kunde in seinem Warenkorb abgelegt hat. Der Service berechnet den Gesamtpreis der Artikel, ermöglicht das Hinzufügen und Entfernen von Produkten und speichert den Warenkorb temporär, bis der Kunde den Bestellprozess abschließt. Dieser Service ist eine zentrale Komponente im gesamten Einkaufsprozess.

2.2.2 Produktkatalog-Service

Der **Produktkatalog-Service** stellt alle relevanten Informationen zu den Produkten bereit. Dazu gehören Produktnamen, Beschreibungen, Preise, Verfügbarkeit und eventuell auch Bilder. Diese Daten werden aus einer **NoSQL-Datenbank** (z. B. MongoDB) abgerufen, um eine schnelle und skalierbare Suche zu ermöglichen.

2.2.3 Bestellservice

Der **Bestellservice** übernimmt die Erstellung und Verwaltung der Bestellungen. Nachdem ein Kunde den Warenkorb überprüft und bestätigt hat, erzeugt der Service eine Bestellung und speichert diese in der Datenbank. Zudem leitet er die Bestellung an den **Versandservice** weiter, der den weiteren Prozess übernimmt.

2.2.4 Lagerverwaltung-Service

Der **Lagerverwaltungs-Service** kümmert sich um die Verwaltung der Lagerbestände und stellt sicher, dass keine Produkte überverkauft werden. Nach einer Bestellung wird der Lagerbestand automatisch angepasst. Zudem stellt der Service Bestandsinformationen für den Produktkatalog bereit.

2.2.5 Versand- und Lieferdienst

Der **Versand- und Lieferdienst** berechnet die Versandkosten, legt Lieferzeiten fest und sorgt dafür, dass die Bestellung an den Kunden ausgeliefert wird. Dabei werden externe Logistik-APIs genutzt, um Tracking-Informationen bereitzustellen und den Status der Lieferung zu verfolgen.

2.3 Infrastruktur-Komponenten

API-Gateway
Vermittelt den Traffic zwischen Frontend und den Microservices, übernimmt Routing, Authentifizierung und Lastverteilung.

Eureka Service Discovery
Ermöglicht das dynamische Auffinden von Microservices und reduziert die Abhängigkeit von festen IP-Adressen.

Kafka Messaging
Ermöglicht eine asynchrone Kommunikation zwischen den Services, z.B. für Bestellstatus-Updates.

Circuit Breaker (Resilience4J)
Schützt das System vor Kaskadenfehlern und verhindert, dass fehlerhafte Services das gesamte System beeinträchtigen.

Um die Kommunikation zwischen den Microservices und der Skalierbarkeit der Anwendung zu gewährleisten, werden mehrere Infrastrukturkomponenten eingesetzt:

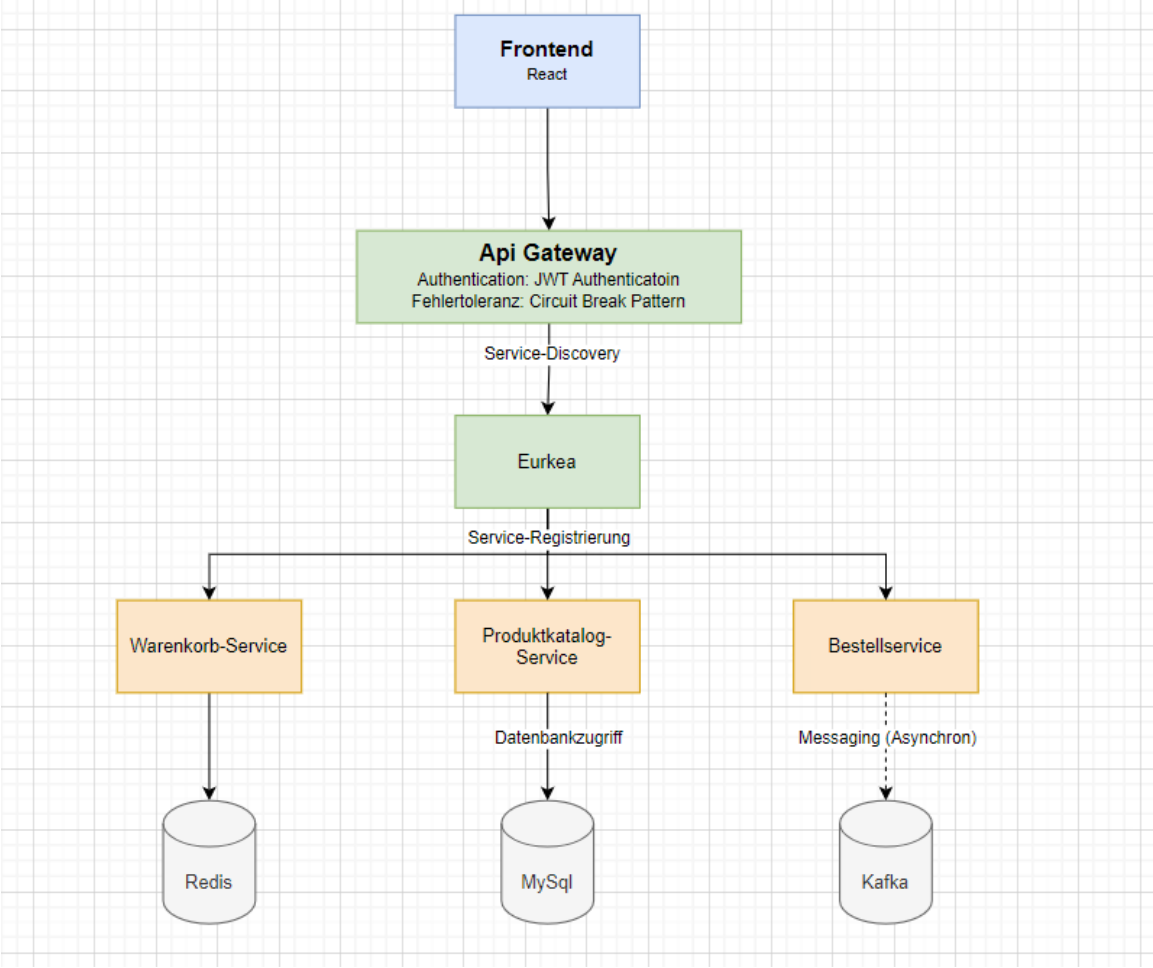
- **API-Gateway**: Das **API-Gateway** vermittelt den Datenverkehr zwischen dem Frontend und den Microservices. Es übernimmt Aufgaben wie Routing, Authentifizierung und Lastverteilung.
- **Eureka Service Discovery**: **Eureka** ermöglicht das dynamische Auffinden und die Registrierung der Microservices innerhalb des Systems, was die Notwendigkeit reduziert, feste IP-Adressen zu verwenden und somit die Flexibilität und Resilienz der Anwendung erhöht.
- **Kafka Messaging**: **Kafka** wird verwendet, um eine asynchrone Kommunikation zwischen den Microservices zu ermöglichen. Zum Beispiel können Bestellstatus-Updates oder Lagerbestandsänderungen als Events an Kafka gesendet werden, wodurch andere Services reagieren können.
- **Circuit Breaker (Resilience4J)**: **Resilience4J** bietet eine Fehlertoleranz, indem es als **Circuit Breaker** agiert. Sollte ein Service ausfallen oder nicht erreichbar sein, verhindert der Circuit Breaker, dass dieser Ausfall das gesamte System beeinflusst.

3. Git-Repository

Ein Git-Repository wird erstellt, in dem alle relevanten Services abgelegt werden.

Repository-Link: https://github.com/ColinMarti1/WebShop.git

4. Architekturskizze



5. Workflow

Der Workflow der Anwendung beschreibt den Prozess vom Besuch des Frontends bis zum Abschluss der Bestellung und Lieferung:

1. **Nutzer besucht das Frontend**: Der Kunde öffnet die Webanwendung, die eine Produktübersicht vom Produktkatalog-Service erhält
2. **Produkte werden in den Warenkorb gelegt**: Der Kunde fügt Produkte über den Warenkorb-Service dem Warenkorb hinzu.
3. **Bestellung wird erstellt**: Nach Bestätigung des Warenkorbs wird der Bestellservice aufgerufen, um die Bestellung zu erstellen.
4. **Lagerbestand wird überprüft und aktualisiert**: Der Lagerverwaltungs-Service überprüft die Bestände und passt sie an.
5. **Versandprozess wird eingeleitet**: Der Versand- und Lieferdienst erhält die Bestellung und berechnet Versandkosten sowie Lieferzeiten.
6. **Asynchrone Nachrichtenübermittlung über Kafka**: Eine Kafka-Nachricht informiert den Bestellservice über den Fortschritt der Lieferung.
7. **Bestellstatus im Frontend**: Der Kunde kann den aktuellen Status seiner Bestellung jederzeit im Frontend einsehen.

6. Reflexion

Während der Umsetzung wurde deutlich, dass eine Microservices-Architektur viele Vorteile bietet, insbesondere in Bezug auf Skalierbarkeit, Wartbarkeit und Flexibilität. Herausforderungen ergaben sich jedoch in folgenden Bereichen:

- Komplexität der Infrastruktur: Das Aufsetzen und Konfigurieren von Kafka, Eureka und dem API-Gateway erforderte eine gründliche Einarbeitung.
- Asynchrone Kommunikation: Die Implementierung von Kafka für die eventbasierte Kommunikation brachte Herausforderungen bei der Fehlerbehandlung mit sich.
- Datenkonsistenz: Die Aufteilung der Datenbankstrukturen auf mehrere Services machte ein durchdachtes Transaktionsmanagement notwendig.

7. Dokumentation & Abgabe

Die Dokumentation wird kontinuierlich in einer Markdown-Datei gepflegt. Die finale Abgabe umfasst:

- Projektdefinition
- Architekturskizze
- Architekturüberlegungen
- Beschreibung der Microservices
- Reflexion