Monolithic vs Microservice Architecture

author: Kacper Bohaczyk

date: 6-Jun-2024

Fragen

- Was versteht man unter Microservices?
    - Microservices sind eine Softwarearchitektur, bei der eine Anwendung in kleine, unabhängige Dienste unterteilt wird, die jeweils eine spezifische Geschäftslogik implementieren. Jeder Microservice ist eigenständig, kann unabhängig entwickelt, bereitgestellt und skaliert werden und kommuniziert mit anderen Microservices über klar definierte Schnittstellen, oft HTTP/REST APIs oder Messaging-Queues.
- Stellen Sie anhand eines Beispiels den Einsatz von Microservices dar.
    - **Benutzerverwaltung**: Verwaltung von Benutzerkonten, Authentifizierung und Autorisierung.

      Produktkatalog: Bereitstellung von Produktinformationen, Suche und Filterung.

      Bestellverwaltung: Verarbeitung von Bestellungen, Verwaltung des Bestellstatus.
- Nennen Sie jeweils 3 Vor- und Nachteile einer monolithischen Architektur? (jeweils mit Begründung)
    - **Vorteile:**

      1. **Einfachheit**: Ein monolithisches System ist oft einfacher zu entwickeln und zu deployen, da alle Komponenten zusammen in einer Anwendung verpackt sind. Dies vereinfacht die Verwaltung und den Betrieb.
      2. **Performance**: Direkte Funktionsaufrufe innerhalb einer Anwendung sind schneller als API-basierte Kommunikation zwischen Diensten.
      3. **Testing**: Einfacher zu testen, da alle Teile der Anwendung in einer Umgebung existieren.

      Nachteile:

      1. **Skalierung**: Skalierung muss für die gesamte Anwendung erfolgen, auch wenn nur ein Teil der Anwendung mehr Ressourcen benötigt.
      2. **Komplexität bei der Weiterentwicklung**: Änderungen in einer Komponente können leicht unerwünschte Nebeneffekte auf andere Komponenten haben.
      3. **Langsameres Deployment**: Jede Änderung erfordert ein vollständiges Deployment, was zu längeren Entwicklungszyklen führt.
- Nennen Sie jeweils 3 Vor- und Nachteile von Microservices? (jeweils mit Begründung)
    - **Unabhängige Entwicklung und Deployment**: Teams können verschiedene Dienste parallel entwickeln und bereitstellen, was die Entwicklungszeit verkürzt.
    - **Technologievielfalt**: Verschiedene Dienste können in verschiedenen Technologien und Programmiersprachen entwickelt werden, die am besten für den jeweiligen Anwendungsfall geeignet sind.

- **Bessere Wartbarkeit**: Kleinere, fokussierte Codebasen erleichtern die Wartung und Weiterentwicklung.

Nachteile:

1. **Komplexität**: Verwaltung und Orchestrierung vieler kleiner Dienste ist komplexer als ein einzelner Monolith.
2. **Netzwerk-Latenz und Ausfälle**: Kommunikation zwischen Diensten über das Netzwerk kann zu Latenzen und möglichen Ausfällen führen.
3. **Testing und Debugging**: Das Testen und Debuggen von verteilten Systemen ist komplizierter als bei monolithischen Anwendungen.

- Was ist ein Service Mesh?
  - Ein Service Mesh ist eine Infrastruktur-Layer für ein Microservices-Netzwerk, das die Kommunikation zwischen Diensten handhabt. Es kümmert sich um die Zustellung von Anfragen, Netzwerksicherheit, Service-Discovery, Load-Balancing und Überwachung. Service Meshes bieten Features wie Traffic-Management, Observability und Sicherheitsfeatures wie TLS-Verschlüsselung.
- Stellen Sie 3 Service Mesh Produkte gegenüber?
  - **Istio**:
    - **Vorteile**: Umfassende Features, gute Integration mit Kubernetes, starke Community-Unterstützung.
    - **Nachteile**: Hohe Komplexität, kann hohe Latenz verursachen.

    Linkerd:
    - **Vorteile**: Einfache Installation und Bedienung, geringe Latenz, speziell für Kubernetes entwickelt.
    - **Nachteile**: Weniger Features als Istio, eingeschränkte Unterstützung für komplexe Szenarien.

    Consul Connect:
    - **Vorteile**: Integriert mit HashiCorp Consul für Service-Discovery und Konfigurationsmanagement, flexible Bereitstellung.
    - **Nachteile**: Komplexere Konfiguration, benötigt zusätzliche Komponenten für vollständige Funktionalität.
- Recherchieren Sie, wie eine Microservice-Architektur mit Java aufgebaut werden kann?
  - **Spring Boot**: Ermöglicht schnelles Erstellen von eigenständigen, produktionsreifen Anwendungen mit eingebauten Web-Servern.

    Spring Cloud: Erweiterungen für die Integration von Microservices-Architekturen mit Features wie Konfigurationsmanagement, Service-Discovery und Load-Balancing.

    Docker: Containerisieren der Microservices für einfaches Deployment.

    Kubernetes: Orchestrieren und Skalieren von Container-Deployments.
- Recherchieren Sie, wie eine Microservice-Architektur mit Python aufgebaut werden kann?
  - **Flask/Django**: Frameworks für die Entwicklung von Microservices.

    Celery: Verwaltung asynchroner Aufgaben und Job-Queues.

    Docker: Containerisieren der Microservices für einfaches Deployment.

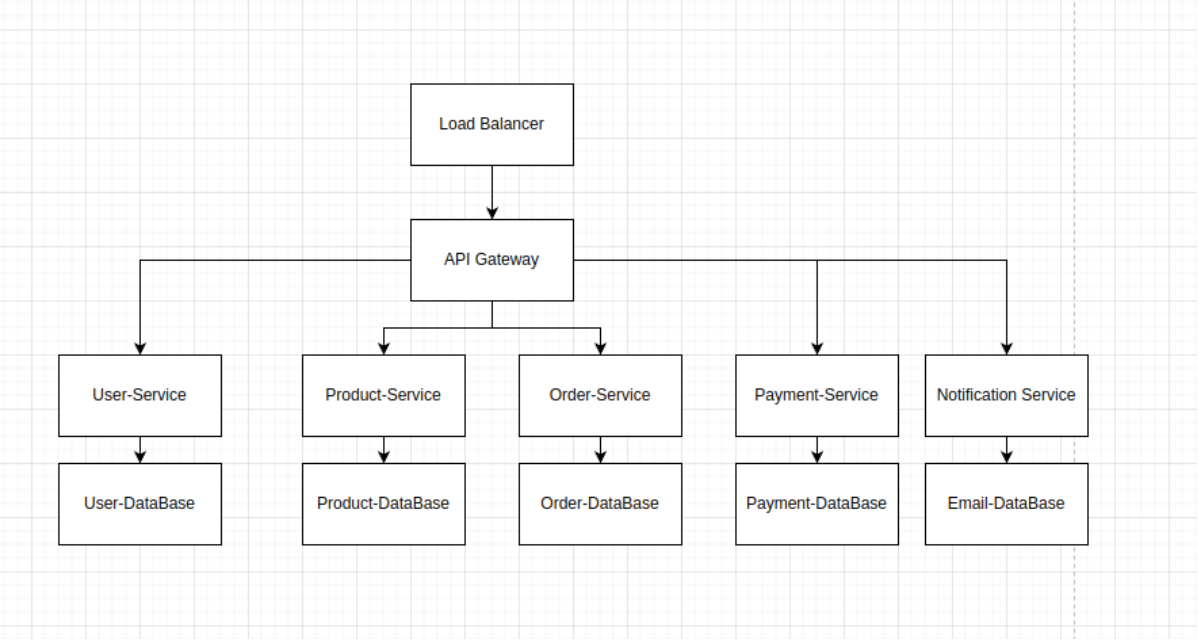
    Kubernetes: Orchestrieren und Skalieren von Container-Deployments.

Ein Grosshändler von Sportprodukten möchte seine Produkte via Online-Portal anbieten. Er bittet Sie ein Konzept zum Aufbau und Entwicklung eines eCommerce-Systems zu erstellen. Der Grosshändler hat in Europa mehrere Zweigstellen und will alle in diesem System einbinden.

Erstellen Sie ein Konzept inklusive Archtiktur-Entwurf (Skizze) für das neue System.

Konzept

Das neue eCommerce-System wird als Plattform dienen, auf der Kunden in ganz Europa Sportprodukte kaufen können. Es soll eine skalierbare, flexible und zuverlässige Architektur bereitgestellt werden, die zukünftige Erweiterungen ermöglicht und eine hohe Verfügbarkeit bietet. Aufgrund der Anforderungen und des Umfangs des Projekts wird eine Microservice-Architektur empfohlen.



Würden Sie in diesem Fall auf eine Microservice-Architektur zurückgreifen? Begründen Sie Ihre Antwort.

Eine Microservice-Architektur ist perfekt für, wegen den oben genannten Beispielen

Skalierbarkeit: Jeder Dienst kann unabhängig skaliert werden, um die Lasten an unterschiedlichen Standorten zu bewältigen.

Flexibilität: Verschiedene Technologien können für verschiedene Dienste verwendet werden, um die besten Lösungen für spezifische Anforderungen zu nutzen.

Wartbarkeit: Kleinere, eigenständige Dienste erleichtern die Wartung und Weiterentwicklung der Anwendung.

Resilienz: Ein Ausfall eines einzelnen Dienstes hat nicht zwingend Auswirkungen auf das gesamte System, was die Zuverlässigkeit erhöht.

Welche Technologien werden Sie einsetzen?

Backend

- **Spring Boot**: Für die Entwicklung der Microservices.
- **Hibernate**: Für die Datenpersistenz.
- **Kafka**: Für die asynchrone Kommunikation zwischen Microservices.
- **MySQL/PostgreSQL**: Für die relationale Datenbank.

Frontend

- **React.js**: Für eine dynamische und reaktionsfähige Benutzeroberfläche.
- **Redux**: Für das State-Management.

API Gateway

- **Kong**: Als API Gateway für das Routing und die Sicherung der APIs.

Authentifizierung und Autorisierung

- **OAuth 2.0**: Für die sichere Benutzeranmeldung und Zugriffskontrolle.

Payments

- **Stripe/PayPal**: Für die Integration verschiedener Zahlungsoptionen.

E-Mail

- **SendGrid/Mailgun**: Für den Versand von Rechnungen und Benachrichtigungen per E-Mail.

Service Discovery und Configuration

- **Eureka**: Für die Service-Discovery.
- **Spring Cloud Config**: Für die zentrale Konfiguration der Microservices.

Quellen

https://www.atlassian.com/microservices/microservices-architecture#:~:text=A%20microservices%20architecture%20provides%20a,technologies%20like%20Docker%20and%20Kubernetes.

https://spring.io/projects/spring-boot

https://microprofile.io/

https://quarkus.io/

https://www.djangoproject.com/

https://istio.io/

https://linkerd.io/