Software Architectures for Enterprises

Portfolio Übung 2 - Lars Carpagne

Inhaltsverzeichnis

Microservices

Microservices sind eine Architekturform, bei der Anwendungen in unabhängige, kleine Dienste zerlegt werden.
Jeder Dienst ist eigenständig, fokussiert auf eine spezifische Aufgabe und kommuniziert über klar definierte Schnittstellen, wie REST APIs.
Diese Architektur bietet Vorteile wie Skalierbarkeit, Flexibilität und einfachere Wartung.

Mögliche Tech-Stacks für die Architektur

Backend-Frameworks:

- ASP.NET,
- Node.js,
- Spring Boot,
- Flask/Django,
- Apache Camel.

Datenbanken:

- MySQL,

- PostgreSQL,
- SQLite,
- MongoDB,
- Redis.

Containerisierung:

- Docker,
- Kubernetes.

Frontend-Frameworks:

- React,
- Angular,
- Vue.js

Versionierung + CI/CD + Projektmanagement:

- GitHub (+ GH Projects, GH Actions)
- GitLab

- API Gateway: ASP.NET Core für zentralisierte API-Verwaltung.
- Auth/Validator: ASP.NET Core für Zugriffstoken und Datenvalidierung.
- Wishes-Service: ASP.NET Core für CRUD-Operationen.
- Datenbank: MySQL zur Speicherung von Wünschen.
- Containerisierung: Docker zur Bereitstellung und Skalierung.

Docker-Komposition

Unsere Dienste werden mithilfe von Docker bereitgestellt und über ein gemeinsames Netzwerk verbunden. Hierfür wird eine Docker Compose.yaml genutzt um alle benötigten Services in einem Netzwerk zu simulieren

- API Gateway: Empfang und Routing von Requests.
- Validator-Service: Validierung von Anfragen.
- Wishes-Service: CRUD für Wünsche.
- MySQL-DB: Persistente Speicherung.
- Apache-Camel Scan-Service: Hochladen von Bildern als Wunsch

Datenstrukturen

```
ValidationRequest:
{
   AccessToken: string,
   Wish: {
       Id: int,
       Description: string,
       FileName: string
       Status: string
   }
}

ValidationResponse:
{
   IsValid: bool,
   Message: string,
   ValidatedWish:
    {
        Id: int,
        Description: string,
        FileName: string
        Status: string
    }
}

WishesRequest:
{
    Action: string,
    Wish: {
      Id: int,
      Description: string,
      FileName: string
      Status: string
   }
}

WishesResponse:
{
     Success: bool,
     Message: string,
     Data: [
        {
```

```
            Id: int,
            Description: string,
            Status: string
        }
    ]
}
```

MySQL Struktur:

```
CREATE TABLE Wish (
    Id INTEGER AUTO_INCREMENT,
    Description VARCHAR(500) NOT NULL,
    FileName VARCHAR(100),
    Status ENUM('Formulated', 'InProgress', 'Delivering', 'UnderTree')

    PRIMARY KEY(Id)
).
```

Analyse: Datenverkehr und Skalierung

Wir erwarten 8 Milliarden Wünsche. Es wird angenommen, dass alle Wünsche innerhalb von 90 Tagen um Weihnachten erwartet werden.
Dadurch ergeben sich
92.222.222 API-Aufrufe pro Tag
3.842.593 API-Aufrufe pro Stunde
64.043 API-Aufrufe pro Minute
1067 API-Aufrufe pro Sekunde

Datenvolumen: ~2 bis 4 TB bei Speicherung aller Wünsche.
Die CPU und RAM-Auslastung erlauben durch horizontale Skalierung die Handhabung dieser Last.

Eigene Hardwareanalyse

Hardware:
- CPU: AMD Ryzen 7 3800X (8 Cores, 3.90 GHz).
- RAM: 32 GB DDR4.
- Speicher: 2.5 TB HDD.

Berechnung:
- Jeder Wunsch benötigt ca. 1.2 KB.
- Maximale Speicherung: ~2 Milliarden Wünsche.

DDoS-Schwelle:
- API-Gateway kann bei 10.000 Requests/sec überlasten.
- Skalierbarkeit durch Load-Balancer erforderlich!

Lösung: Verteilte Anwendung

- 1 * API-Gateway als Load-Balancer
- N * Verarbeitungs-Endpunkte, davon hat jeder
- 1 * Validierungs-Service
- 1 * Wunsch-Service