FACHHOCHSCHULE ERFURT UNIVERSITY OF APPLIED SCIENCES

Bachelorarbeit

In der Angewandten Informatik

Technologische Ansätze zur Umsetzung einer Microservice-Architektur

Prototypische Implementierung einer Anwendung zur Verwaltung der IT-Kontaktmesse an der Fachhochschule Erfurt

Benjamin Swarovsky

Abgabedatum: 01.11.2021

Prof. Dr. Steffen Avemarg

Dipl.-Inf. Steffen Späthe

I. Kurzfassung

Die Arbeit behandelt das Thema…

Dem Leser soll ein Verständnis für die Problematik … erlangen

Kapitel … wird … erläutert

Anschließend wird …

Am Ende werden die Ergebnisse ausgewertet…

II. Abstract

III. Aufgabenstellung

Inhaltsverzeichnis

IV. Abbildungs und Tabellenverzeichniss

1 Einleitung

1.1 Problemstellung

Über Jahre hinweg wurden Softwaresystem als Monolithen deployt. Aufgrund ihrer eng gekoppelten Komponenten bilden solche Systeme eine Untrennbare Einheit.[1] Weil diese Anwendungen mit der Zeit immer größer wurden, entstanden für die Entwickler Organisatorische Probleme.[2] Extrem große Systeme sind nicht leicht zu verstehen und zu verwalten. Durch die enge Kopplung der Komponenten ist es schwierig Änderungen in der Anwendung einzuspielen. Weil sich Codeänderungen auf das gesamte System auswirken, müssen diese gründlich koordiniert werden. Diese Problemstellung kann dazu führen, dass bei der Einführung einer neuen Technologie die gesamte Anwendung komplett neu geschrieben werden muss. Ein weiteres Problem lässt sich gegenüber der Skalierbarkeit erkennen. Bei einem Monolithen kann nur das komplette System skaliert werden. Die Skalierung von einzelnen Komponenten ist nicht möglich, weil einzelne Teilbereiche nicht unabhängig voneinander interagieren können.[3]

Gegenüber diesen Nachteilen schafft eine Microservice Architektur Abhilfe. Seit einigen Jahren erlebt dieses Architekturmuster einen Regelrechten Hype. Beispiele für die erfolgreiche Umsetzung einer Microservice-Architektur liefern große Firmen wie z.B. Amazon, Netflix und Zalando. Laut Eberhard Wolf bringt der Hype einen großen Nachteil mit sich. Die Architektur wird oft ausgewählt, weil sie gerade in Mode ist. Microservices sind eines von vielen Architekturmustern, welches je nach Anwendungsfall mehr oder weniger für ein System geeignet ist. Über die Umsetzung werden sich dann in vielen Fällen zu wenig Gedanken gemacht. Dabei kann die Umsetzung einer solchen Architektur als sehr anspruchsvoll angesehen werden. Es gilt Herausforderungen zu überwinden wie zum Beispiel die Modellierung der Datenmodelle mit jeweils einer eigenen Datenbank pro Microservice, Autorisierung und Authentifizierung in verteilten Systemen, sowie Fehlerbehandlung und Monitoring über mehrere Services hinweg.[4]

1.2 Ziel

Diese Wissenschaftliche Arbeit behandelt Problemstellungen bei der technischen Umsetzung einer Microservice-Architektur. Es soll dabei dargelegt werden, wie sich die Herangehensweisen bei der Erstellung einer Microservice Architektur von denen einer Monolithischen Architektur unterscheiden. Dabei werden unter anderem populäre Frameworks, Bibliotheken und Entwurfsmuster vorgestellt, welche in einer Microservice-Architektur häufig zum Einsatz kommen und es wird dargelegt welchen nutzen diese bringen.

Mit dem Thema wird sich anhand eines Praktischen Beispiels auseinandergesetzt. Dieses wird umgesetzt anhand eines Entwurfes für ein System zur Verwaltung der IT-Kontaktmesse an der Fachhochschule Erfurt. Die Messe findet jährlich auf dem Gelände der Hochschule statt. Unternehmen aus der Region stellen sich gegenüber den Studierenden an Messeständen vor und Präsentieren sich anhand eigener Vorträge. Das System soll den Firmen unter anderem die Möglichkeit bieten sich für die Messe zu

[1] (Fink, 2012)
[2] (Alzve, 2021)
[3] (Gnatyk , 2018)
[4] (Wolff, 2017)

registrieren, sich zu informieren und einen eigenen Messeauftritt zu Organisieren. Im Weiteren verlauf dieser Arbeit wird das System anhand einer Microservice-Architektur entworfen. Dabei wird auf die Hindernisse eingegangen, welche bei der Entwicklung der Anwendung aufgrund der Architekturentscheidung entstehen. Im Anschluss wird ein Prototyp implementiert. Dieser soll zeigen mit welchen Mitteln die genannten Problemstellungen in der Praxis gelöst werden können. Am Ende wird ausgewertet ob die Konzipierten Lösungsansätze für die Anwendung geeignet sind.

2 Grundlagen

2.1 Monolithen

2.2 Microservices

Microservices stellen einen Software-Architekturansatz dar. Dieser entstand in den frühen 1980er Jahren mit den von der Firma Sun Microsystems entwickelten Remote Procedure Calls, welche als eine der ersten Technologien zur Umsetzung von verteilten Systemen entwickelt wurden. Die ersten Praktische Einsätze von Microservices wurden von James Lewis und Martin Fowler im Jahr 2014 Beschrieben.[5]

Im Gegensatz zum Architekturansatz des Deployment-Monolithen, bei dem das System nur als Ganzes deployt werden kann, gelten Microservices laut Eberhard Wolff als unabhängig deploybare Module. Die Größe der einzelnen Services hängt vom jeweiligen Anwendungsfall ab.[6] Ein Service sollte klein genug gehalten werden, um von einem einzelnen Entwicklerteam entwickelt zu werden. Bei zu kleinen Services steigt die Anzahl der Services im gesamten System. Verteilte Aufrufe anderer Systeme über das Netzwerk sind Zeitaufwändiger als Aufrufe im selben Prozess. Um einer Erhöhung der Verzögerungszeit entgegenzuwirken, sollten die Services nach Möglichkeit nicht zu klein gehalten werden.[7]

-Conteniuous Delivery

2.2.1 Vorteile

Eine Microservice-Architektur ist weniger anfällig für das ungewollte Einbauen von Abhängigkeiten zwischen einzelnen Komponenten. Dieser Vorteil entsteht aufgrund der hohen Modularität von Microservices, und der Schwierigkeit die Grenzen der einzelnen Microservices zu überschreiten. Dem Zerfall der Architektur kann dadurch entgegengewirkt werden.[8]

Microservices können leicht ersetzt werden, weil es einfacher ist ein kleines Programm auszutauschen als ein komplexes System

TODO

Aufgrund der Aufteilung von Fachlichkeiten bei einer Microservices-Architektur, ist die Logik für Entwickler einfacher zu verstehen. Entwickler müssen nicht die Funktionalitäten der gesamten Anwendung verstehen, sondern nur die, des für sie zugewiesenen Microservice.

2.2.2 Nachteile

[5] (Mohapatra, et al., 2019)
[6] (Wolff, 2018)
[7] (Wolff, 2018)
[8] (Wolff, 2018)

2.3 Frameworks

Laut Stephan Augsten schafft ein Framework einen Ordnungsrahmen durch Basisbausteine für den Entwickler. Die Basisbausteine unterstützen in Form von Entwurfsmustern. Dadurch bildet sich ein Programmiergerüst mit dem Entwicklungszeit eingespart und damit Entwicklungskosten reduziert werden können.[9]

Microservice Frameworks

Laut Thomas Bayer erleichtern Microservice Frameworks die Implementierung von Features, welche ein Microservice in der Regel anbieten sollte. Beispiele dafür sind Sicherheit, Monitoring, Service Discovery und Konfigurierbarkeit. Eine Basis für Frameworks wie zum Beispiel Spring Boot für die Java Plattform bietet die Dependency Injection welche die flexible Verbindung von Komponenten ermöglicht. Die Frameworks bieten für die Service zu Service Kommunikation Rest API Clients an.

Vor der Festlegung für den Einsatz eines Microservice Frameworks sollten Vor und Nachteile abgewogen werden. Folgende Vor und Nachteile sollten beachtet werden.

Vorteile:

- Weniger Code pro Service
- Ein Microservice ist schneller für die Cloud bereit
- Kürzere Entwicklungszeit
- Infrastrukturcode muss nicht selbst entwickelt werden

Nachteile:

- Entwickler benötigen Einarbeitungszeit zum Verständnis der Framework Konzepte
- Entwickler geben Kontrolle ab und wissen unter umständen nicht welche Funktionen das Framework im Hintergrund ausführt
- Der Einsatz von einer Vielzahl von Framework Bibliotheken kann zu Versionsproblemen mit umständlicher Fehlersuche führen [10]

Deployment

Skalierbarkeit

[9] (Augsten, 2020)
[10] (Bayer, 2019)

2.4 Kommunikation zwischen Microservices

Die Kommunikation zwischen den Microservices kann synchron oder asynchron erfolgen. Die Entscheidung hängt vom jeweiligen Anwendungsfall ab. In einer Microservice Architektur können beide Kommunikationsmöglichkeiten zum Einsatz kommen.

https://www.hosteurope.de/blog/microservices-grundlagen-und-technologien-von-verteilter-architektur/

2.4.1 Synchrone Kommunikation

Bei einer Synchronen Kommunikation wird eine Antwort geschickt und anschließend auf eine Antwort gewartet. Es handelt sich dabei um eine relativ simple Herangehensweise. Die Ausführung erfolgt in der Regel per Rest-Schnittstelle über HTTP.

URI Uniform Resource Identifier (URI)

Mit einem URI lassen sich abstrakte oder physische Ressourcen wie zum Beispiel Webseiten, Sender oder Empfänger von E-Mails ansprechen. Einer Anwendung wird dadurch eine eindeutige Identifikation für die Abfrage der Ressourcen ermöglicht. Die Syntax einer URI besteht laut IONOS höchstens aus folgenden Komponenten.

- Scheme (gibt Auskunft über das verwendete Protokoll wie zum Beispiel http)
- Authority (kennzeichnet die Domäne)
- Path (zeigt den genauen Pfad zur Resource)
- Query (bietet die Möglichkeit für Abfragen)
- Fragment (kennzeichnet einen Teilaspekt einer Resource)

Ein URI muss mindestens aus den Komponenten Scheme und Path bestehen. Eine URI mit allen Komponenten wird wie folgt aufgebaut.

scheme://authorithy path ? query # fragment

Ein Beispiel URI wäre http://google.de

https://www.ionos.de/digitalguide/websites/web-entwicklung/uniform-resource-identifier/

Hypertext Transfer Protocol (HTTP)

HTTP ist ein Protokoll welches die Kommunikation in einem IP-Netzwerk (zum Beispiel zwischen Web-Server und Web-Browser). Im Falle einer Webanwendung fungiert der Webserver als HTTP-Server und der Client als Browser. Der Client sendet einen Request an den Port des Servers (in der Regel Port 80) und erhält von diesem eine Response-Nachricht. Die Adressierung der Resourcen erfolgt per URI. Folgende Aktionen können per HTTP auf Resourcen umgesetzt werden

- GET (Ruft Resourcen auf)
- POST (Erstellt eine neue Instanz einer Resource)
- PUT (ändert eine Resource)
- DELETE (löscht die Instanz einer Resource)

Representational State Transfer (Rest)

Rest bildet eine Softwarearchitektur, welche den Datenaustausch in einem Client-Server Softwaresystem ermöglicht. Jede Ressource erhält eine eindeutige Adresse. Im Anschluss können die Grundaktionen Auslesen (GET), Erstellen (POST), Ändern (PUT), und Löschen (DELETE) ausgeführt werden. Es können alle beschreibenden Parameter zwischen Client und Server ausgetauscht werden und der Aufbau einer Sitzung ist dabei nicht notwendig. Dieses Verhalten wird als zustandslos bezeichnet. Für Rest gibt es kein festgelegtes Übertragungsprotokoll. In der Praxis kommt in der Regel HTTP zum Einsatz.

Rest API

2.4.2 Asynchrone Kommunikation

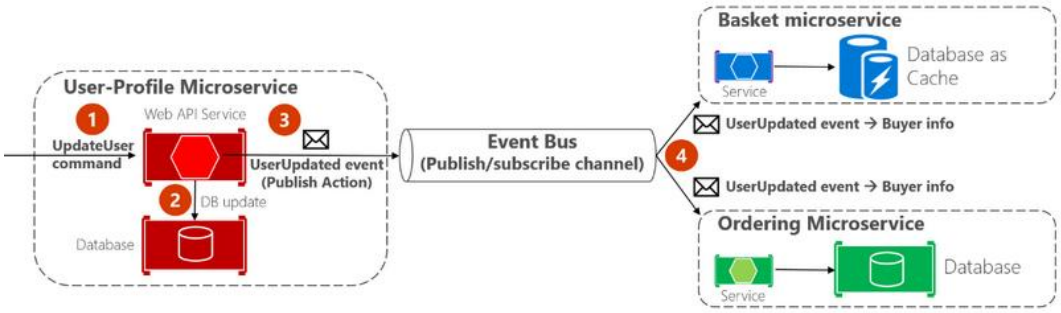
Asynchrone Kommunikation wird eingesetzt, wenn Änderungen mehrere Datenmodelle und zugehörige Microservices betreffen. Dabei müssen verschiedene Modelle wie zum Beispiel Firma, Vortrag und Newsletter angepasst werden. Diese Anpassung wird über ereignisgesteuerte Kommunikation realisiert. Microservices können dabei einen Ereignisbus abonnieren, um dadurch Benachrichtigungen zu empfangen. Die Benachrichtigungen werden versendet, wenn ein Microservice Ereignisse auf einem Ereignisbus veröffentlicht. Ein Protokoll welches eine zuverlässige Kommunikation gewährleistet ist das Advanced Message Queuing Protocol. Dieses Arbeitet auf der Anwendungsschicht und orientiert sich an dem Transmission Control Protokol (TCP).

AMQP realisiert eine schnelle und Robuste Datenübertragung, weil es keinen Leerlauf produziert. Dieser Vorteil wird durch den Einsatz einer Nachrichtenwarteschlange erzielt. Sender und Empfänger agieren asynchron. Der Sender muss nicht zwingend auf die Bestätigung der Nachricht des Empfängers warten, ohne weiterzuarbeiten. Hat der Empfänger freie Kapazitäten wird die Nachricht aus der Warteschlange geholt, verarbeitet und bestätigt. Die Kompatibilität zwischen verschiedenen Systemen wird gewährleistet, weil laut TechTarget AMQP als ein binäres Nachrichtensystem mit strikten Verhalten für die Nachrichten gestreamt wird.

Für den Nachrichtenaustausch können folgende verschiedene Betriebsmodi gewählt werden:

- Exactly-once (Einmalige garantierte Auslieferung)
- At-least-once (Dopplungen beim Nachrichtenaustausch möglich, garantierte Auslieferung)
- At-most-once (Die Nachricht wird einmalig gesendet und kann verloren gehen)

Abbildung .. zeigt eine Eventgetriebene Kommunikation zwischen 3 Microservices über einen Ereignisbus



https://docs.microsoft.com/de-de/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication

https://whatis.techtarget.com/de/definition/Advanced-Message-Queuing-Protocol-AMQP

https://www.ionos.de/digitalguide/websites/web-entwicklung/advanced-message-queuing-protocol-amqp/

Cloud

JSON Webtoken

3 Anforderungsanalyse

3.1 Aufgabenstellung

Für die Fachhochschule Erfurt soll ein System zur Verwaltung der jährlichen IT-Kontaktmesse an der Fachhochschule Erfurt umgesetzt werden. Das System wird in Form einer Webseite realisiert, welche die folgenden Inhalte abdeckt:

Firmenverwaltung

Firmen können folgende Features des Systems nutzen:

- Registrierung
- Newsletter Abonnieren (Firmen erhalten dadurch regelmäßig Informationen über die Messe per E-Mail.)
- Anmeldung / Abmeldung für die Teilnahme als Messeausteller
- Verwaltung von Vorträgen (Themen, Präsentatoren, Zeitslots, benötigtes Equipment)
- Informationen für Besucher bereitstellen (Firmenbeschreibung, Link zur Webseite, Ansprechpartner)
- Nachrichtenaustausch mit der Fachhochschule

Informationsbereitstellung für Besucher

Das System soll einen öffentlichen Zugang für Besucher in Form einer Webseite bieten. Diese sollen sich über folgende Themen informieren können:

- Allgemeine Messedaten (Veranstaltungsort, Zeitpunkt und Ansprechpartner)
- Informationen über ausstellende Firmen (Firmenbeschreibung, Lage des Messestandes, Link zur Webseite, Ansprechpartner)
- Information zu Vorträgen (Thema, Firma, Zeitpunkt, Gebäudenummer und Raumnummer)

Administration

Für Mitarbeiter der Fachhochschule wird eine Oberfläche zur Verwaltung des Systems geboten. Folgende Verwaltungsmöglichkeiten werden gegeben:

- Allgemeine Messedaten (Veranstaltungsort, Zeitpunkt und Ansprechpartner)
- An / Abmeldung der Messeausteller (bestätigen)
- Anlegen von Zeitslots für Vorträge (Zeitslot für Zugehörigen Vortragsraum)
- Bearbeitung und Versendung des Newsletters (Sendung per Broadcast an Abonnenten)
- Nachrichtenaustausch mit einzelnen Firmen
- Löschen Registrierter Firmenaccounts
- …

3.2 Qualitätsziele

Laut Dr. Peter Hruschka und Dr. Gernot Starke beeinflusst die Ernennung der für die Stakehholder wichtigsten Qualitätsziele, die Softwarearchitektur maßgebend.[11] Daher werden im folgenden Abschnitt die wichtigsten Qualitätsziele aufgelistet.

Bedienbarkeit (Einfache Benutzung / Erlernbarkeit, Zugänglichkeit)

Das System verfügt mit seinen Benutzergruppen (FH-Erfurt Mitarbeiter, Firmenvertreter, Studierenden) über eine Vielzahl von verschiedenen Benutzern. Eine schlechte Bedienbarkeit könnte untere anderem dazu führen, dass weniger Studierende an der Messe teilnehmen. Zum Beispiel könnte die Organisation der Messe aufgrund einer falschen Bedienung aus dem Ruder laufen. Darüber hinaus könnten Studierende aufgrund einer schlechten Bedienbarkeit nicht die benötigten Informationen für eine Teilnahme an der Messe erhalten. Damit die genannten Beispiele vermieden werden wird die Bedienbarkeit als sehr wichtig betrachtet. Zusätzlich wird Wert darauf gelegt, dass auch Benutzer mit Einschränkungen das System verwenden können

Zuverlässigkeit (Verfügbarkeit)

Das System soll im Besonderen, während des Zeitraumes vor der Messe eine hohe Verfügbarkeit sicherstellen. Lange Ausfallzeiten würden während dieses Zeitraumes den Informationsaustausch zwischen Fachhochschule, Firmen und Studierenden blockieren. Die Akteure würden unter Umständen keine Informationen über Änderungen des Ablaufes der Messe erhalten oder diese Informationen zu spät erhalten. Deshalb wird ein hoher Wert auf die Verfügbarkeit des Systems gelegt.

Wartbarkeit (Modularität)

Die Fachhochschule Erfurt wird voraussichtlich noch über viele Jahre (Jahrzehnte) die Fachrichtung Angewandte Informatik und die dazugehörige IT-Kontaktmesse anbieten. Das System wird im Laufe der Zeit stark anwachsen. Das System müsste im schlimmsten Fall bei weitreichenden Änderungen komplett neu geschrieben werden, falls zwischen den einzelnen Modulen zu starken Abhängigkeiten bestehen. Um die Wartbarkeit auf Dauer sicherzustellen wir dieses Qualitätsziel mit hoher Priorität eingestuft.

Integrität

Sollten zwischen der Fachhochschule Erfurt, den Teilnehmenden Firmen und Studierenden Missverständnisse entstehen könnte im schlimmsten Fall die Gesamte Messe scheitern. Beispielsweise würden die Messeausteller an einer Messe ohne Besucher Teilnehmen, wenn Besucher nicht über eine Änderung des Messetermins informiert werden.

[11] (Hruschka, et al., 2017)

3.3 Stakeholder

Im folgenden Abschnitt werden die wichtigsten Akteure ermittelt, welche mit dem Projekt in Verbindung stehen. Es werden Erwartungen und Einflüsse der Akteure erfasst, um Probleme rechtzeitig zu lösen. Weiterhin soll dadurch ein Schieflaufen des Projekts vermieden werden. In der Tabelle … werden alle Stakeholder für das Projekt abgebildet.

Rolle	Beschreibung	Erwartungshaltung
FH-Sekretariat	Anlaufstelle für Studierende und StudiumsbewerberInnen Verwalten die Studierenden und beantworten deren Fragen	Möchte eine leicht zu bedienende Administrationsoberfläche zur verwaltung der IT-Kontaktmesse
Dozenten	Halten Lehrveranstaltungen	
Studenten	Studieren an der FH-Erfurt Angewandte Informatik	Möchten sich möglichst unkompliziert mit wenigen Klicks über Ort und Zeitpunkt der Messe, Teilnehmende Firmen und deren Jobangebote informieren. Möchten anhand der Messe Jobangebote von den ausstellenden Firmen erhalten.
Studiumsinteressenten	Sind an einem Studium der Angewandten Informatik an der Fachhochschule Erfurt interessiert	Möchten sich möglichst unkompliziert mit wenigen Klicks über Ort und Zeitpunkt der Messe, Teilnehmende Firmen und deren Jobangebote informieren. Erwarten eine gut Organisierte Messe, um sich zu vergewissern, dass ein Studium an der Fachhochschule Erfurt die richtige Entscheidung ist.
Firmenvertreter	Nehmen als Ansprechpartner Ihres Unternehmens an den Firmeneigenen Messeständen an der Messe Teil.	Möchten sich möglichst unkompliziert mit wenigen Klicks Ihre Messeteilnahme verwalten und Studenten / Studiumsinteresssenten Informationen bereitstellen. Möchten. Möchten frühestmöglich über den Organisatorischen Ablauf der Messe informiert werden. Dabei sollen keine Unklarheiten entstehen.
Präsentatoren	Halten Vorträge für die Besucher der Messe	Möchten sich möglichst unkompliziert mit wenigen Klicks Ihre Präsentationen

		verwalten und Studenten / Studiumsinteresssenten Informationen über die Vorträge bereitstellen. . Möchten frühestmöglich über den Organisatorischen Ablauf der Messe informiert werden. Dabei sollen keine Unklarheiten entstehen.

4 Konzepte

4.1 Domain Driven Design

Domain Driven Design ist eine Sammlung von Zusammenhängenden Entwurfsmustern, welche im gleichnamigen Buch Domain-Driven Design von Eric Evans beschrieben werden. Laut Eberhard Wolff hilft Domain Driven Design dabei Microservices zu verstehen, weil es dabei um die Strukturierung größerer Systeme nach Fachlichkeit geht. Es wird anhand von Strategic Design beschrieben wie komplexe Systeme aufgebaut werden können und Domänenmodelle miteinander interagieren. Bounded Context stellt dabei einen zentralen Punkt des Strategic Designs dar. Bounded Context beschreibt den gültigen Einsatzbereich für ein Domänenmodell und stellt einen in sich geschlossenen Fachbereich dar. Zum Beispiel steht ein Artikel für die Versandabteilung eines Onlineshops in einem anderen Kontext als für die Buchhaltung. Die Versandabteilung betrachtet unter anderem die Maße des Artikels. Für die Buchhaltung sind zum Beispiel Preise und Steuersätze von Bedeutung.[12]
Laut Arne Limburg und Lars Röwekamp sollte ein Microservice einen Bounded Context abbilden. Dieser führt bei einer Einteilung nach Domänenobjekt selten zum Ziel. Eine bessere Lösung bietet die Einteilung nach Anwendungsfällen.[13]

Für die Anwendung IT-Kom Verwaltung lassen sich folgende Bounded Contexts darstellen:

Für die Beschreibung der Interaktionen und Abhängigkeiten unter den Bounded Contexts können laut Michael Plöd folgende Domain Driven Design Entwurfsmuster genutzt werden:

- Shared Kernel
- Customer / Supplier
- Anticorruption Layer
- Separate Ways
- Conformist
- Published Language
- Open / Host Service

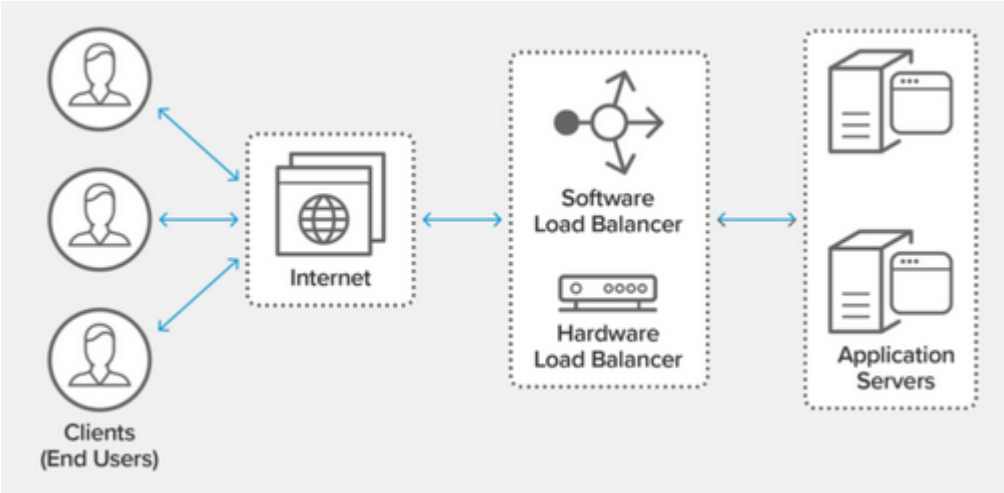
[12] (Wolff, 2018)
[13] (Röwekamp, et al., 2016)

Für die Interaktion zwischen den Bounded Contexts sorgt im besten Fall ein Eventsystem. [14]

Im Idealfall sollte ein Microservice nur aus einem Bounded Context bestehen. Dadurch wird das Ziel erreicht, das ein Team an einem Microservice unabhängig arbeiten kann. Unter bestimmten Situationen kann es jedoch vorkommen das ein Microservice aus mehreren Bounded Contexts besteht. In einem solchen Fall kann unter anderem das Shared Kernel Entwurfsmuster aus dem Domain Driven Design zum Einsatz kommen. [15]

4.2 Load Balancer

Ein Load-Balancer welcher eine funktionelle Einheit aus Hard und Software darstellt, setzt die Lastverteilung in einem Netzwerk um. Ziel ist es Workloads auf Rechenressourcen wie zum Beispiel Servern gleichmäßig zu verteilen um dadurch die Zuverlässigkeit, Effizienz und Kapazität des Netzwerkes zu optimieren. Load Balancing kann physisch oder virtuell umgesetzt werden. Der Load-Balancer ermittelt in Echtzeit welche Rechenressource die entsprechende Clientanforderung erfüllen kann. Dabei soll eine Netzwerküberlastung vermieden werden. Es gibt mehrere Methoden, um Loadbalancing umzusetzen. Eine davon ist der Round Robin Algorithmus, welcher in sequenzieller Reihenfolge eine Liste der verfügbaren Rechenressourcen durchläuft. Weitere Möglichkeiten bieten unter anderem der Hashbasierte Ansatz, der Least-time-Algorithmus und die Least-Connection-Methode. Eine Veranschaulichung des Load Balancings wird mit Abbildung …… dargestellt.[16]



Quelle https://www.nginx.com/resources/glossary/load-balancing/

[14] (Plöd, 2016)
[15] (Wolff, 2018)
[16] (ComputerWeekly, 2020)

4.3 Service Discovery

Laut Alexander S. Gillis Identifiziert die Service Discovery Geräte und Dienste, welche sich gegenseitig ohne weiteres in einem komplexen verteilten Netzwerk nicht finden. Dabei werden Instanziierten Diensten dynamische Netzwerkstandorte zugewiesen. Dadurch wird der Konfigurationsaufwand beim Erstellen einer Microservices Struktur vermindert. Die Umsetzung erfolgt auf Basis eines gemeinsamen Netzwerkprotokolls wie zum Beispiel Domain Name System Service Discovery oder Dynamic Host Configuration Protocol. [17] Beispiele für Service Discovery sind Netflix – Eureka und Consul.

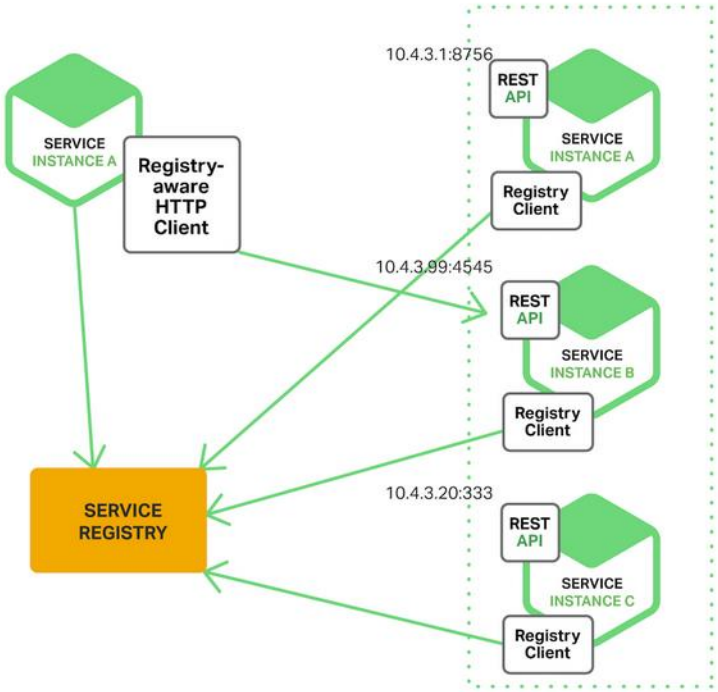
Service Discovery wir als Software realisiert, bei der sich alle Service-Instanzen eines Systems registrieren. Bei Anfragen wird diese Liste mit allen verfügbaren Zielen abgerufen. Service Discovery ermöglicht es Clients Service Adressen über den Namen aufzulösen, anstatt die tatsächliche Adresse mit Hostnamen und Port zu verwenden.[18] Für die Auflösung gibt es die beiden folgenden Möglichkeiten.

https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

Clientseitig Erkennung

Beim Clientseitigen Erkennungsmuster ist der Client für die Erkennung der der Netzwerkstandorte der Dienstinstanzen verantwortlich und stellt Anforderungen an den Load Balancer. Der Client fragt eine Datenbank mit verfügbaren Dienstinstanzen ab welche als Service Registry bezeichnet wird. Netzwerk Instanzen werden beim Starten an der Service Registry angemeldet und beim Herunterfahren wieder abgemeldet. Die Registrierung wird in regelmäßigen Abständen automatisch abgefragt.
Die Client Side Discovery ist relativ unkompliziert. Sie hat jedoch den Nachteil, dass der Client jedem Service zur Seite gestellt werden muss, wodurch die Services aufgebläht werden.

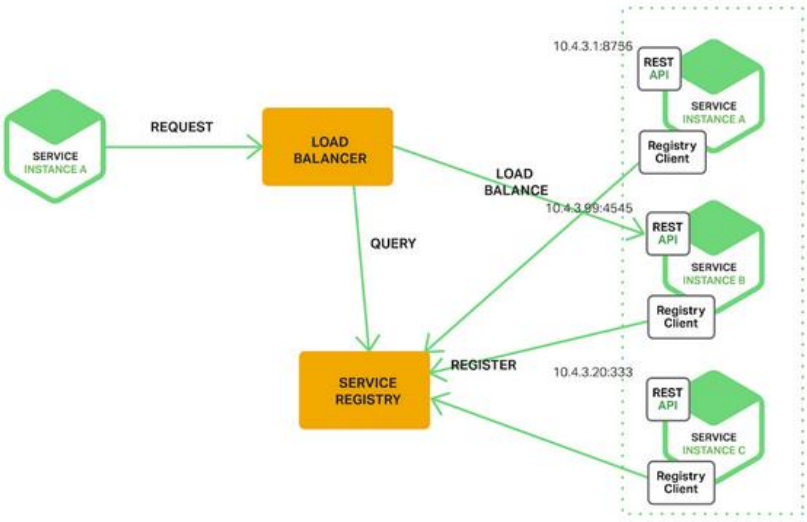
[17] (Gillis, 2021)
[18] (Bayer, 2019)



Beispiele: Netflix Eureka

Serverseitige Erkennung

Bei der Serverseitigen Erkennung nimmt ein Load Balancer die Anfragen des Clients entgegen und leitet diese an den Entsprechenden Dienst weiter. Der Load Balancer fragt dazu die Service Registry ab. Serverside Discovery hat den Nachteil, dass der Router bei einem Ausfall das ganze System lahmlegen kann. Sie bietet den Vorteil, dass Abfragen für Clients vereinfacht werden.



4.4 API Gateway

Beim Einsatz einer (Komplexen) Mikroservices Struktur kann es sich als problematisch erweisen wenn der Client direkt mit den einzelnen Microservices kommuniziert. Dabei können folgende Nachteile entstehen

- Sicherheitsprobleme: Für eine direkte Kommunikation müssen alle Microservices für den Client offengelegt werden. Dadurch wird eine große Angriffsfläche angeboten.
- Enge Kopplung: direkte Verweise zwischen Client und Microservices führen zu einer engen Kopplung. Dadurch verschlechtert sich die Wartbarkeit des Systems
- Hohe Latenz. Der direkte Aufruf mehrere Dienste kann zu mehreren Netzwerkroundtrips zwischen Client und Server führen. Dadurch entsteht eine hohe Verzögerungszeit.

Funktionen:

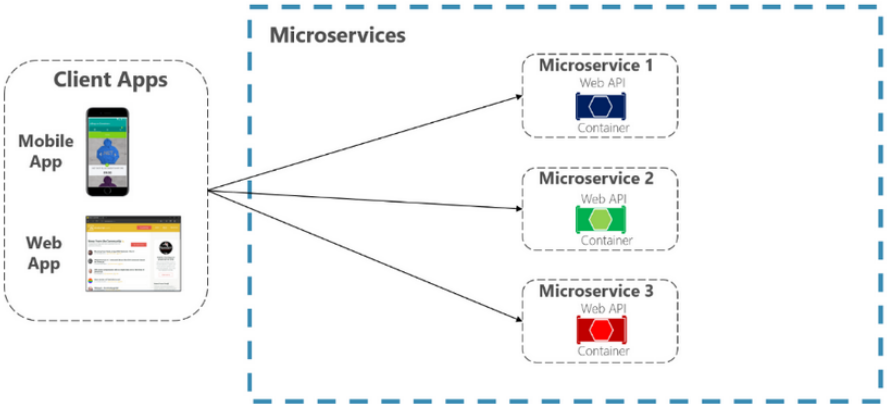
Ein API-Gateway gleicht bezüglich seiner Funktionalitäten dem Fassadenmuster. Es bildet einen Kontaktpunkt für ein- und ausgehen Netzwerkverkehr. Es stellt dazu ein vereinheitlichtes Interface bereit, welches mit dem Client interagiert. Ein API-Gateway stellt die Funktionalität eines Reverseproxy bereit. Dementsprechend werden Gruppen interner Microservices unter einer einzigen URL für den Client bereitgestellt. Eine einzelne Clientanfrage kann mehrere Microservices Aggregieren. Dadurch wird der Datenaustausch zwischen Back-End-API und Client reduziert.

https://docs.microsoft.com/de-de/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern

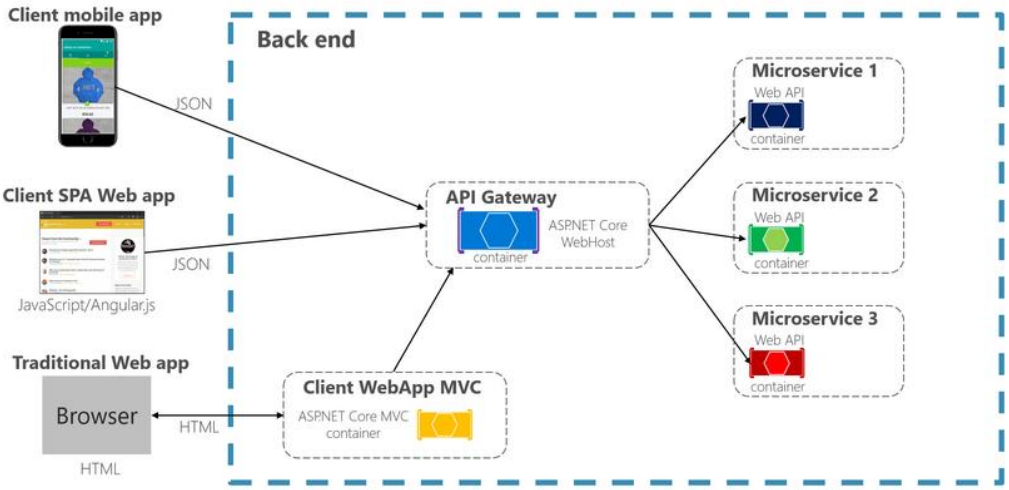
Zur Gewährleistung der Systemsicherheit bietet ein API-Gateway die Möglichkeit der Autorisierung und Authentifizierung. Zusätzlich bietet es die Möglichkeit ein Zentrales Logging für Requests durchzuführen.

https://www.sidion.de/lernen/sidion-labor/blog/spring-cloud-gateway.html

Ohne Gateway



Mit Gateway



4.5 Autorisierung und Authentifizierung

Laut Mattias te Wierik bedarf es bei einer Microsevice Architektur eine andere Herangehensweise zur Umsetzung der Autorisierung und Authentifizierung als wie bei einer Monolithischen Architektur. Bei Monolithen wurden häufig Sitzung eingesetzt, welche im Arbeitsspeicher gespeichert wurden. Unter Verteilten Anwendungen können Sitzungen nicht gemeinsam genutzt werden. Es ist auch nicht sinnvoll für jeden Service eine eigene Sitzung zu erstellen, weil sich der Benutzer dann zum Beispiel vor der Benutzung jedes Services Einloggen müsste, um eine Sitzung zu speichern. Der Einsatz eines Identity Servers bietet unter Verwendung des OAuth2 Protokolls eine Lösung dieser Problemstellung.[19]

OAuth2

Laut Mitchell Anicas definiert OAuth2 folgende Rollen:

- Client – Die Anwendung, welche auf das Benutzerkonto zugreifen möchte
- Ressourcen Besitzer – Autorisiert der Anwendung den Zugriff auf seine Ressource
- Ressourcen Server – Hostet die geschützten Benutzerkonten
- Autorisierungsserver (Identity Server) – stellt nach einer erfolgreichen Identitätsprüfung ein Zugriffstoken an die Anwendung aus

1. Der Benutzer erhält von der Anwendung eine Anfrage auf die Zugriffsberechtigung auf Dienstressourcen
2. Nach der Autorisierung der Anfrage durch den Benutzer wird der Anwendung eine Autorisierung erteilt.
3. Der Autorisierungsserver erhält eine Anforderung auf ein Zugriffstoken von der Anwendung. Diese legt dabei die Autorisierung ihrer eigenen Identität und Autorisierungserteilung vor.
4. Nachdem Gültigkeit der Autorisierungserteilung und authentifizierte Anwendundungsidentität nachgewiesen werden kann, sendet der

[19] (te Wierik, 2020)

Autorisierungsserver ein Zugriffstoken an die Anwendung. Die Autorisierung ist abgeschlossen
5. Die Anwendung fordert die Ressource vom Ressourcenserver an. Die Authentifizierung erfolgt über das Zugriffstoken.
6. Bei der Gültigkeit des Tokens erhält die Anwendung vom Ressourcenserver die Ressource.[20]

OpenId Connect

OpenId Connect baut auf dem OAuth 2.0 Framework auf. Es bietet gegenüber OAuth2 mit nur einer Anmeldung die Möglichkeit, die Anwendung über mehrere Anwendungen hinweg zu verwenden, was als Single Sign-On bezeichnet wird. Mit signle Sign-On ist es möglich, sich von sozialen Netzwerkdiensten wie zum Beispiel Facebook,Twitter oder Xing anzumelden.

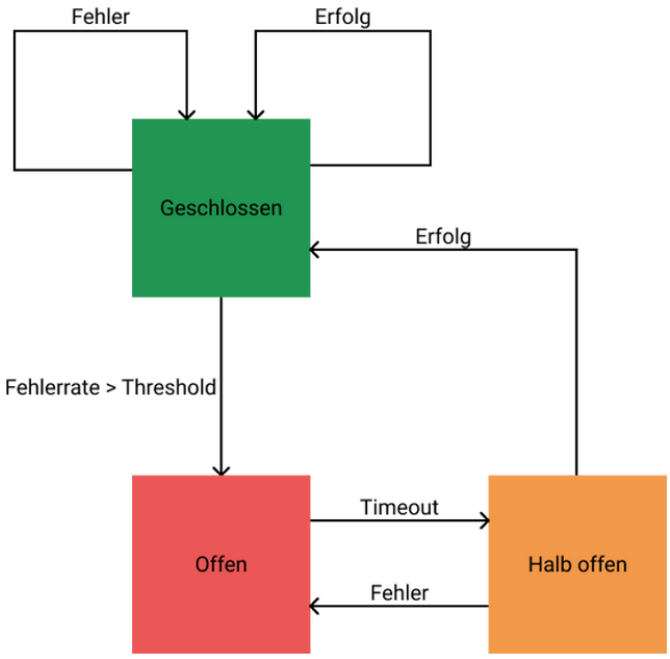
https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc

4.6 Circuit Breaker Pattern

In einer Microservice-Architektur sind in der Regel mehrere Services voneinander abhängig. Fällt einer dieser Services aus, dann warten abhängige Microservices nach Anfragen an diesen Service auf dessen Antwort. Während der Wartezeit können weitere Anfragen in das System eintreffen. Dieses Verhalten kann dazu führen, dass sich die Anfragen anstauen. Im schlimmsten Fall kann dadurch das gesamte System lahmgelegt werden. Das Circuit Breaker Pattern dient zur Vermeidung einer solchen Problemstellung. Die Funktionalität kann, mit der einer Sicherung in einem elektrischen Stromkreis verglichen werden. Fließt in einem Stromkreis ein zu hoher elektrischer Strom dann unterbricht die Sicherung den Stromfluss bevor weitere Komponenten des Systems Schaden nehmen können.

Bei dem Circuit Breaker Pattern werden Anfragen mit einem Fehler beantwortet, wenn eine Instanz eine bestimmte Fehlerrate überschreitet. Der Circuit Breaker ist in diesem Fall offen. Der Ausgefallene Service soll dadurch die Möglichkeit erhalten, in einen fehlerfreien Zustand zurückzukehren. Tritt kein Fehler mehr auf, dann werden alle Anfragen wieder weitergeleitet und der Service ist verfügbar. Der Circuit Breaker ist in diesem Fall geschlossen. Einen Übergang zwischen den Zuständen offen und geschlossen bietet der Zustand halb offen, welcher nach einer gewissen Zeit im Zustand offen aktiviert wird. In diesem Zustand werden Anfragen teilweise weitergeleitet. Nach erfolgreichen Anfragen wird der Circuit Breaker wieder geschlossen. Sollten bei Anfragen weiterhin Fehlerauftreten ändert sich der Zustand wieder auf offen. Dieser Prozess wiederholt sich in regelmäßigen Abständen, bis Anfragen wieder erfolgreich weitergeleitet werden. Der Ablauf des Circuit Breaker Pattern wird mit Abbildung … dargestellt.

[20] (Anicas, 2014)



4.7 Distributed Tracing

In einer Microservice-Architektur können jederzeit neue Instanzen eines Service erstellt oder gelöscht werden. Dadurch lassen sich Logs nicht zuverlässig nachverfolgen. Anfragen eines Clients können über eine Vielzahl von Microservices hinweg ausgeführt werden. Die Dauer für die einzelnen Aufrufe unter den Services lässt sich nicht ohne weiteres feststellen. Engpässe lassen sich dementsprechend kaum nachvollziehen. Bei einer Entwicklung mit einzelnen Entwicklerteams pro Microservice kann es sich als aufwendig erweisen herauszufinden welches Entwicklerteam für einen Fehlerfall verantwortlich ist.

Distributed Tracing ist ein Verfahren welches zum Profilieren, Überwachen und Debuggen von verteilten Anwendungen eingesetzt wird. Funktionen, welche die App verlangsamen oder einen Ausfall verursachen können anhand dieses Verfahrens ermittelt werden. Weiterhin kann das Distributet Tracing zum Optimieren von Code eingesetzt werden. Das Verfahren nutzt ein Konzept, bei dem Daten zwischen den Aufrufen unter den Microservices mitgeliefert werden. Diese werden als Span bezeichnet. Ein Span wird mit einer eindeutigen ID versehen. Verschachtelte Aufrufe unter den Microservices erhalten zusätzlich eine Parent Span ID, welche an die betreffenden Microservices übergeben wird. Zu Identifikation der übergeordneten Operationen generiert der erste Aufruf eine Root Trace ID. Der Fluss der Microservice Aufrufe wird also über Trace, Span und Parent Span ID beschrieben.

In einem Span werden folgende Informationen gespeichert:

- Zeitstempel für den Abschluss eines Zwischenschrittes
- Tags zur Klassifizierung per Schlüssel- / Wertpaar
- Textbasierte Anmerkungen zum Beispiel über Fehlermeldungen oder Informationen über die Beeinflussung des Flusses

In einer Web-UI werden die zeitlichen Abläufe der Aufrufe dargestellt. Diese können dann anhand der Informationen wie zum Beispiel Begin und Dauer der einzelnen Spans ausgewertet werden. Dadurch entsteht die Möglichkeit, die Anwendung als Gesamtsystem zu überwachen.

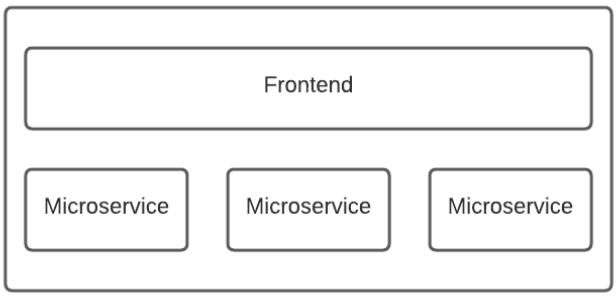
Open Tracing API

4.8 User Interface

4.8.1 Frontend Monolith

Laut Eberhard Wolff sollte der Einsatz eines Frontend Monolithen bei einer Microservice Architektur stets hinterfragt werden, weil das Userinterface bei vielen fachlichen Änderungen verschiedener Microservices angepasst werden muss. Dadurch kann ein Frontend Monolith zu einem änderungsschwerpunkt werden. Es kann jedoch Gründe geben, die für eine Umsetzung eines Monolithen im Frontend sprechen. Unter folgenden Voraussetzungen ist ein Monolithisches Frontend die richtige Wahl:

- Single Page Apps
  Diese Anwendungen bieten die Möglichkeit für eine Modularisierung. Laut Eberhard Wolff führen sie allerdings mit der Zeit zu einem Monolithischen Frontend.
- Native mobile Anwendungen
  Diese Anwendungen können nur als Ganzes deployt werden.
- Frontend Entwicklerteams
- Beim Einsatz eines Teams welches speziell für die Frontendentwicklung spezialisiert ist bietet sich die Umsetzung eines Monolithen an. Dadurch kann das Team in gewohnter Arbeitsumgebung agieren.[21] Abbildung .. zeigt eine Microservice Architektur mit einem Monolithischen Frontend.

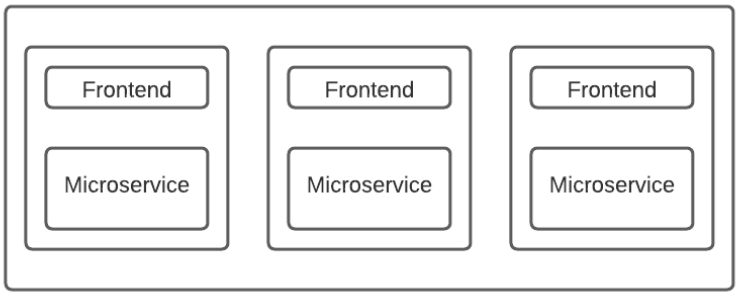


4.8.2 Modularisiertes Frontend

Bei einem modularisierten Frontend erhält jeder Microservice sein eigenes Frontend. Dadurch wird zu einer entkoppelten Entwicklung beigetragen, wenn die Microservices fachlich unabhängig gehalten werden. Weiterhin entstehen bei Modularisierte Frontends keine Einschränkungen bei Technologie Entscheidungen. Ein Modularisiertes Frontend droht nicht zu einem Änderungsschwerpunkt zu werden.

[21] (Wolff, 2018)

Die getrennt deployten Frontends müssen integriert werden damit aus ihnen ein Gesamtsystem entsteht. Diese Integration wird durch eine Verlinkung unter den einzelnen Systemen umgesetzt. Eine weitere Möglichkeit bieten Redirects, welche die Frontendintegration mit einer Datenübertragung Kombinieren. Ein Beispiel für einen Redirect wäre zum Beispiel eine Umleitung bei einem Onlineeinkauf zu einem Onlinezahlungsdienstleister. Der Kunde loggt sich ein beim Dienstleister, tätigt die Zahlung und wird anschließend wieder zum Onlineshop zurückgeleitet.



Microservice Praxisbuch seite 98-103

4.9 Container / Deployment

Deployment

Die Bereitstellung und Verteilung von Software wird als Deployment bezeichnet. Dazu werden die Software Bestandteile in Paket aufgeteilt. Die Pakete stellen eine Dateisammlung dar, welche durch die Anwendung automatisch Konfiguriert und installiert werden. Zur Erstellung und Bereitstellung der Pakete ist eine Paketierungssoftware erforderlich. Die Bereitstellung und Verteilung der Pakete über das Netzwerk wird von einem Paketierungsserver übernommen. Zur Speicherung der Konfigurationsdaten bedarf es weiterhin einer Konfigurations-Datenbank. Ein Konfigurationsserver stellt Dienste und Infrastruktur bereit mit denen die Konfigurationsdaten für Computer und Server zugänglich werden. Damit die Konfiguration gepflegt werden kann kommt eine Konfigurationssoftware zum Einsatz welche das Speichern von Paket-, Rechner- und Programmverzeichnissen ermöglicht. Deployment erfolgt manuell oder automatisch. Ein manuelles Deployment bietet sich für kleinere Anwendungsbereiche an und kann von einem Administrator oder einem geschulten Mitarbeiter durchgeführt werden. Bei größeren Organisationen und Anwendungsbereichen bietet sich eine automatische Bereitstellung und Verteilung über Deployment-Skripte an.

Deployment umfasst in der Regel folgende Teilschritte:

1. Orchestrierung
   Die Software wird ausgewählt und Zusammengestellt (entsprechend des Bedarfs in Abhängigkeit der zu nutzenden Endgeräte).
2. Download der Software
3. Paketierung
   Die zu installierende Software wird mitsamt den zugehörigen Anweisungen und Konfigurationen zusammengestellt
4. Testphase
   Die Pakete werden einem ausführlichen Anwendungstest unterzogen

5. Verteilung auf die Endgeräte (zum Beispiel PC oder Server) der Anwender
6. Installation der Pakete auf den Endgeräten

https://www.dev-insider.de/was-ist-deployment-a-1025926/

Container

Damit die Vorteile von Microservices ausgenutzt werden können, müssen diese laut Eberhard Wolff mindestens getrennte Prozesse sein. Es soll vermieden werden, dass ein Absturz eines Microservices zum Absturz weiterer Microservices führt. Das System soll dadurch robust bleiben. Für die Gewährleistung der Skalierbarkeit eines Systems ist diese Trennung allerdings nicht ausreichend. Laufen mehrere Prozesse auf einen Server dann steht nur eine begrenzte Menge an Hardwarekapazität zur Verfügung. Kompatibilitätsprobleme mehrerer Bibliotheken auf nur einem Betriebssystem führen zu weiteren Hindernissen.

Eine Möglichkeit zur Lösung dieser Probleme bieten Virtuelle Maschinen. Diese ermöglichen es auf einem Rechner mehrere Betriebssysteme wie zum Beispiel Windows und Linux zur gleichen Zeit laufen zu lassen. Die Aufteilung der Microservices auf Virtuellen Maschinen beanspruchen allerdings viel Speicher, weil dementsprechend jeder Microservice die Instanz eines Betriebssystems besitzt. Eine effizientere Lösung bietet der Einsatz von Containern. Diese bieten einen universellen Paketierungsansatz, bei dem alle Anwendungsabhängigkeiten in einem Container gebündelt werden. Container bringen gegenüber Virtuellen Maschinen mehrere Vorteile mit sich. Beim Einsatz von Containern wird kein ganzes Betriebssystem installiert, sondern nur alle zum ausführen notwendigen Pakete einer Anwendung Dadurch bleibt ein Microservice so leichtgewichtig wie ein Prozess.[22] Dementsprechend wird die Erzeugung des Overheads vermieden welcher zum Beispiel beim Ausführen von Softwarekomponenten wie Webserver, Programmiersprachen oder Datenbanken bei einer Virtuellen Machine entsteht. Darüber hinaus lässt sich ein Conteiner auch deutlich schneller starten als eine Virtuelle Machine, bei der das Kompletten Betriebssystem hochgefahren werden muss. Dadurch lassen sich Container schneller aufsetzen als Virtuelle Maschinen und Entwicklern werden neue Möglichkeiten im Deployment geboten.[23] Container können zum Beispiel im System für einen Lastenausgleich sorgen, indem diese je nach Systemlast Instanzen eines Services zu- oder abschalten. Dieser Vorgang lässt sich automatisieren, wodurch der IT-Betrieb auf lange Sicht entlastet wird.

https://www.innoq.com/de/articles/2015/11/docker-perfekte-verpackung-fuer-micro-services/

[22] (Wolff, 2018)
[23] (Öggl, et al., 2019)

5 Architekturentwurf

5.1 Lösungsstrategie

Allgemeine Architektur

Die Anwendung IT-Kom-Verwaltung wird für eine Messe entwickelt, welche auf unbestimmte Zeit jährlich an der Fachhochschule Erfurt stattfindet. Das System wird über Jahre hinweg ausgebaut. Die einzelnen Komponenten müssen deshalb aus langer Sicht wartbar bleiben. Dieser Zustand wird erreicht, wenn Abhängigkeiten unter den Komponenten minimal gehalten werden. Bei einem Monolithischen System wäre die Wartung im Laufe der Zeit immer schwieriger zu handhaben. Daher wird eine Microservice-Architektur welche aus einzelnen möglichst unabhängigen System besteht umgesetzt. Neue Features wie zum Beispiel ein Chat zwischen Hochschule und einzelnen Firmen könnte relativ schnell als neuer Microservice hinzugefügt werden. Diesem kann ein eigenes Entwicklerteam zugewiesen werden, wodurch sich die Erweiterung unabhängig vom Rest der Anwendung entwickeln lässt. Durch den Einsatz von Containern können die Microservices relativ schnell und einfach deployt werden, weil die Container auf jeder Umgebung mit der entsprechenden Container-Engine funktionieren. Diese können dadurch auf den verschiedensten Maschinen ausgeführt werden. Weiterhin wird die Möglichkeit geboten einen automatisierten Lastenausgleich umzusetzen, was zur Verbesserung der Verfügbarkeit beiträgt und die Administration entlastet.

Frontend

Damit die Bedienbarkeit der Benutzeroberfläche für die Zugehörigen Services optimiert werden kann, wird das Frontend Modular entwickelt. Das bedeutet jeder Microservice wird zusammen mit einem eigenen Frontend deployt. Dadurch wird auch die Wartbarkeit auf Dauer verbessert, weil es kaum Abhängigkeiten unter den Einzelnen Frontends gibt. Dadurch können die Microservices nach den Standarts von self contained systems entwickelt werden. Für das Frontend wird eine Serverseitige Template-Engine eigesetzt. Es wird unterschieden zwischen Serverseitigen- und Clientseitigen Rendern. Serverseitiges Rendern bietet den Vorteil das die Seite beim ersten laden schneller zur Verfügung steht.

Weiterhin werden dadurch Vorteile bezüglich der Suchmachinenopimierung geboten. Serverseitiges Rendern ist ein ausgefeiltes Konzept und daher relativ leicht umzusetzen. Der hohe Konfigurationsaufwand beim Einsatz einer serverseitigen Single-Page-Application wird vermieden. Serverseitiges Rendern hat den Nachteil, das es bei weiteren Seitenaufrufen jedes mal die Komplette Seite lädt, während beim Clientseitigen Rendern nur einzelne Komponenten neu geladen werden. Die Benutzeroberfläche für die Anwendung erfordert keine komplexe Benutzeroberfläche. Viele Seitenaufrufe wie zum Beispiel bei einem Bestellprozess eines Onlineshops sind nicht notwendig. Daher wird das Rendering des Frontends Serverseitig umgesetzt.

https://dzone.com/articles/client-side-vs-server-side-rendering-what-to-choos

https://blog.codeinside.eu/2012/06/03/client-side-vs-server-side-html-rendering/

Für die Benutzer teilt sich die Oberfläche in drei Bereiche ein. Diese sind eine Öffentliche Webseite für Besucher, eine Verwaltungsoberfläche für die teilnehmenden Firmen und ein

Administrationsbereich für die Hochschule. Die Bereiche dürfen von Aussehen und Handhabung voneinander abweichen. Nach Möglichkeit sollten sie aber einheitlich gehalten werden. Innerhalb dieser Bereiche muss das Design einheitlich bleiben. Die einzelnen Entwicklerteams müssen daher in enger Absprache stehen und festgelegte Stylekonventionen einhalten. Stylesheets könnten zentral platziert und entwickelt werden. Die Anwendung würden sich in diesem Fall die eigenen Styles herunterladen. Das ermöglicht den Einsatz eines Entwicklerteams für Styles. Es wurde sich gegen diese Herangehensweise entschieden weil auch dadurch im System eine Art Flaschenhals entsteht was der Idee von Microservices wiederspricht.

Backend

Die Einzelnen Backend Komponenten werden in Kapitel 5.3 aufgelistet. Diese Kommunizieren untereinander per HTTP über REST. Die Einzelnen Microsevices werden als Container deployt. ……. Außer dem Besucherservice enthält jeder Microservice eine eigene Datenbank ……..

5.2 Systemkontext (Ebene 0)

Im folgenden Abschnitt wird das Umfeld des Systems beschrieben. Die Nutzer des Systems werden beschrieben, sowie die die Fremdsysteme welche mit dem System interagieren.

Das System wird für folgende Nutzer entworfen

- **Firma (Aussteller)**
  Vertreter von Firmen nutzen das System, um sich für die Teilnahme an der Messe zu registrieren, den Eigenen Messeauftritt zu verwalten und den Newsletter zu abonnieren.

- **FHE-Mitarbeiter**
  Zu den FHE-Mitarbeitern zählen Sekretariat-Mitarbeiter und Dozenten. Diese verwalten die Messedaten und die teilnehmenden Firmen über eine Administrationsoberfläche. Zusätzlich erstellen und versenden sie den Newsletter.

- **Besucher**
  Zu den Besuchern gehören Personen, die sich auf der Öffentlichen Webseite über die Messe informieren wollen. Dazu gehören zum Beispiel Studenten und Studieninteressierte.

Folgende Fremdsysteme interagieren mit dem System:

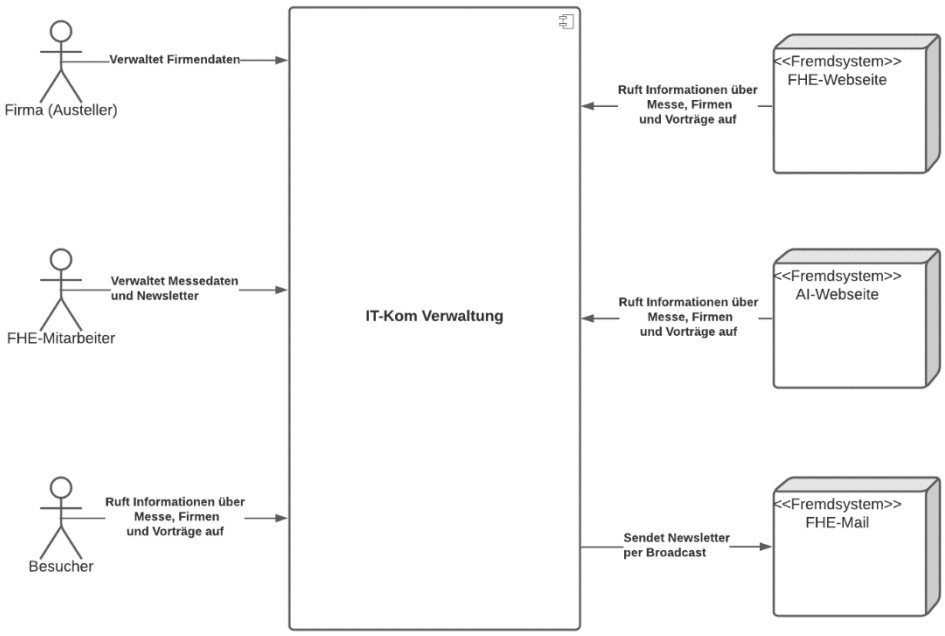
- **FHE-Webseite**
  Die Webseite der Fachhochschule Erfurt kann Daten des Systems aufrufen und bereitstellen.

- **AI-Webseite**
  Die Webseite der Angewandten Informatik kann Daten des Systems aufrufen und bereitstellen.

- **FHE-Mail**

Der Newsletter für die Messe wird über den Mailserver der Fachhochschule Erfurt versendet per E-Mail. Das System sendet die Newsletter per Broadcast an alle Firmen, die den Newsletter abonniert haben.

Der Systemkontext wird auf Abbildung .. dargestellt.



5.3 Bausteinsicht (Ebene 1)

Domain Driven Design

Im folgenden Abschnitt werden die Subsysteme der Anwendung dargestellt. Dazu gehören die einzelnen Microservices und weitere Subsysteme, welche für den reibungslosen Betrieb beitragen. Die Microservices wurden auf Basis der Bounded Contexts (siehe Kapitel 4.1, Abbildung ..) bestimmt. Die Einzelnen Module werden als Container deployt. Folgende Subsysteme gehören zur Anwendung:

- **API-Gateway**
  Leitet Anfragen, welche in das System eintreffen von zentraler Stelle aus an entsprechende Services weiter. Aufrufe unter den Microservices werden ebenfalls über das Gateway weitergeleitet. Der integrierte Loadbalancer erzeugt beim Aufruf eines Microservice mit mehreren Instanzen einen Lastenausgleich.

- **Service Discovery**
  Ermöglicht eine Adressauflösung über die Namen der einzelnen Services.

- **Identity Server**

Der Identity Server setzt das OAuth2 Protocol um und ermöglicht die Umsetzung von Autorisierung und Authentifizierung im System.

- **Circuit Breaker**
  Wird von jedem Microservice implementiert. Dieser blockiert fehlerhafte Anfragen, um eine Überlastung des Systems zu verhindern.

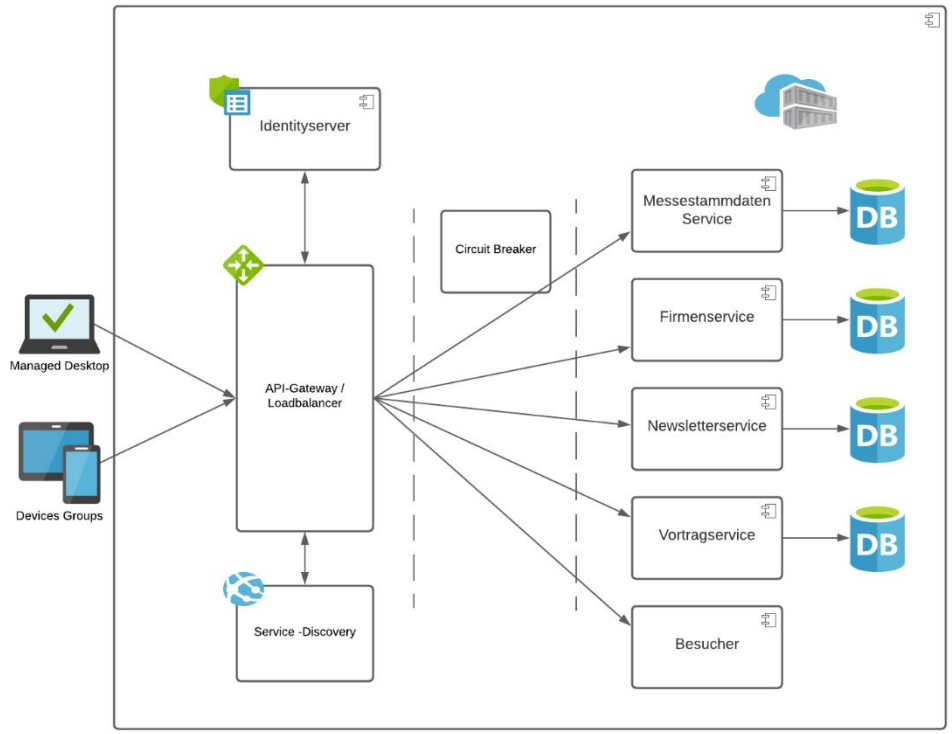
- **Messestammdatenservice**
  Enthält Methoden Datenstrukturen und User Interface zur Verwaltung der Messestammdaten.

- **Firmenservice**
  Enthält Methoden Datenstrukturen und User Interface zur Verwaltung der Firmendaten.

- **Newsletterservice**
  Enthält Methoden Datenstrukturen und User Interface zur Verwaltung des Newsletters.

- **Vortragservice**
  Enthält Methoden, Datenstrukturen und User Interface zur Verwaltung der Vorträge.

- **Besucherservice**
  Enthält Methoden zum Aufrufen der Daten von den Microservices Messestammdatenservice, Firmenservice und Vortragservice.



https://www.graylog.org

https://www.jaegertracing.io

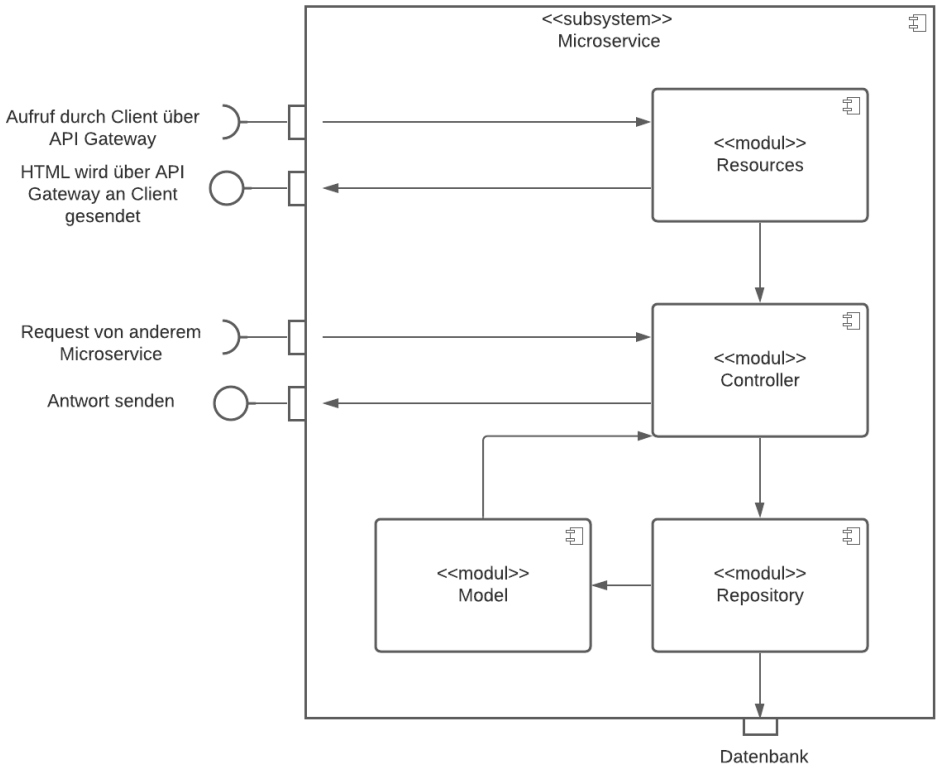
https://medium.com/swlh/tracing-in-spring-boot-with-opentracing-opentelemetry-dd724134ca93

5.4 Bausteinsicht (Ebene2)

Im folgenden Abschnitt werden die Module der Microservices Messestammdatenservice, Firmenservice, Newsletterservice, Vortragservice, und Besucherservice vorgestellt. Diese enthalten folgende Module:

- **Controller**
  Stellt eine Rest-API bereit. Dabei werden Anfragen von der Benutzeroberfläche entgegengenommen und verarbeitet. Zusätzlich kommunizieren die Microservices untereinander über die API.
- **Resources**
  Hier lagern die Frontendelemente wie zum Beispiel HTML-, CSS- und Javascriptdateien. Zusätzlich beinhaltet das Modul die Konfigurationsdateien in denen unter anderem der Applicationsname und Server-Port definiert werden. Diese dienen zum Beispiel für die Umsetzung der Service Discovery.
- **Model**
  Enthält die jeweiligen Datenmodelle eines Microservices. Ein Beispiel dafür ist die Klasse CompanyData im Firmenverwaltungsservice welche zum Erstellen, Lesen und Bearbeiten von Firmendaten verwendet wird.
- **Repository**
  Setzt die Datenbankzugriffe um
- **Datenbank (nicht im Besucherservice)**
  Speichert die Daten des Services. Die Wahl der Datenbank hängt von den Anforderungen an den jeweiligen Service ab. Je nach Anwendungsfall könnte zum Beispiel eine Relationale Datenbank oder eine NoSQl Datenbank (welche einen nicht-Relationalen Ansatz verfolgt) verwendet werden. Für den Prototyp wird jeweils eine H2 Datenbank verwendet. Es handelt sich dabei um eine in Memory Datenbank welche sich besonders leicht einrichten lässt und daher für die Implementierung eines Prototypen geeignet ist. Der Besucherservice enthält keine eigene Datenbank, weil dieser keine Daten anlegt. Er dient lediglich dazu, die Daten anderer Microservices den Besuchern auf der von ihm bereitgestellten Besucherwebseite anzuzeigen.

Die Module werden auf Abbildung .. dargestellt



5.5 Verteilungssicht

5.6 Laufzeitsicht

Querschnittliche Konzepte

Domainmodell

Testverfahren

6 Implementierung

6.1 Spring Framework

Beim Springframework handelt es sich um ein Open Source Java Framework, welches über Aspektorientierte Programmierung und Dependency Injection einen gut wartbaren und leichteren Programmcode ermöglichen soll. Zusätzlich soll damit die Komplexität der Java-Plattform deutlich reduziert werden. [24]

Aspektorientierte Programmierung ermöglicht eine Modulare Gestaltung des Codes. Wichtige Funktionen wie zum Beispiel Logging Fehlerbehandlung und Caching werden dabei zentral verwaltet.[25]

Bei der **Dependency Injection** liefert ein Objekt die Abhängigkeit für ein anderes Objekt. Die Abhängigkeit wird gleichermaßen als Objekt realisiert und kann verwendet werden. Die Weitergabe einer Abhängigkeit an ein Abhängiges Objekt wird laut Jesko Landwehr als Injection bezeichnet. Die Übertragung einer Abhängigkeit eines Clients an einen externen Code welcher als Injector bezeichnet wird.

……..

https://it-talents.de/it-wissen/was-ist-dependency-injection/

Spring Beans

Bei Spring Beans handelt es sich um Objekte die von einem Inversion of Control (loc) Container instanziert und anderweitig verwaltet werden. Bei loC handelt es sich um einen Vorgang, bei dem alle Abhängigkeiten eines Objekts definiert werden ohne diese zu erstellen. Bei einer Spring Bean werden die Abhängigkeiten aus einem loC-Container abgerufen, welcher mit den entsprechenden Konfigurationsmetadaten versehen werden muss.

https://www.baeldung.com/spring-bean

Am häufigsten kommt das Framework zur Programmierung von Webanwendungen zum Einsatz.

Spring Boot

Laut Stefan Waldman setzt Spring Boot auf dem Spring Framework auf. Es bietet anhand von Autokonfigurations-Mechanismen sehr einfach zu entwickelnde Spring Anwendungen.

6.2 Abhängigkeitsverwaltung mit Maven

Die einzelnen Module des Systems werden als Springboot Projekte realisiert. Die Abhängigkeiten von Externen Bibliotheken werden jeweils pro Microservice über das Build-Tool Maven zentral verwaltet. Jede Springboot Version verfügt über eine Liste von Abhängigkeiten, welche für die Version getestet wurden, wodurch eine Kompatibilität gewährleistet wird. Die Versionen der Abhängigkeiten müssen nicht angegeben werden.

[24] (Augsten, 2019)
[25] (Biswanger, 2016)

Die Verwaltung der gewählten Abhängigkeiten wird von Springboot automatisch umgesetzt. Abhängigkeiten, Name des Projekts und Spring Version werden in der Datei pom.xml angegeben. Diese Datei wird von Maven für die Umsetzung des Build-Prozesses vorausgesetzt. https://spring.io/guides/gs/maven/

Beim Build-Prozess werden die einzelnen Quelldateien eines Programms in ein lauffähiges Konstrukt konvertiert. https://www.dev-insider.de/was-ist-ein-build-a-702737/

6.3 Services

6.3.1 Firmenverwaltung

Die Geschäftslogik für die Firmenverwaltung läuft über das gleichnamige Springboot Projekt

6.3.2 Newsletter

Für das Versenden der Newsletter per Broadcast und zum Abonnieren der Newsletter wird das ASP.NET Core Projekt Newsletter angelegt

6.3.3 …..

6.3.4 Frontend mit Thymeleaf

Das Frontend wird in den Einzelnen Microservices mit der Java Template-Engine Thymeleaf realisiert. Dieses lässt sich leicht in eine Spring-Anwendung integrieren und ermöglicht dadurch die schnelle Entwicklung eines Frontends für ein relativ kleines System wie ein Self Contained System. Zur Integrierung von Thymeleaf in die jeweiligen Microservices wurde in der Dartei pom.xml die Abhängigkeit *spring-boot-starter-thymeleaf* hinzugefügt. Thymeleaf beinhaltet eine Komponente, welche View-Namen auf Thymeleaf Templates mappt welche bei einer Spring Anwendung unter *src/main/resources/templates* gesucht werden. Ein solches Template wurde zum Beispiel im Besucherservice unter den Namen *start.html* angelegt. Dieses Stellt die Startseite für die öffentliche Besucherwebseite dar. Abbildung .. zeigt die Datei *start.html*

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Startseite</title>
</head>
<body>

    <h1>Herzlich Willkommen zur IT-Kontaktmesse an der Fachhochschule Erfurt</h1>

</body>

</html>
```

6.3.5 ZUSAMMENSPIEL SERVICES !!!!!!!!

6.4 Eureka Discovery Service

Die Service Discovery wird clientseitig von dem Netflix Tool Eureka umgesetzt.

Zur Einrichtung des Eureka Servers wurde das Spring Boot Projekt ServiceDiscovery erstellt. In der Datei pom.xml wurde die Abhängigkeit spring-cloud-starter-netflix-eureka-server wie in Abbildung ….. hinzugefügt. Die Klasse Main-Klasse wurde mit der Annotation @EnableEurekaServer versehen. Dadurch dient die Springboot-Anwendung als Eureka-Server

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>
         spring-cloud-starter-netflix-eureka-server
   </artifactId>
</dependency>
```

Für die Registrierung der Services muss jeder Microservice als Eureka-Client fungieren. Deshalb wurde jede Main-Klasse der einzelnen Services mit der Annotation @EnableEurekaClient versehen. Zusätzlich wurde die Abhängigkeit spring-cloud-starter-netflix-eureka-client jeweils in der Datei pom.xml der einzelnen Services und des API-Gateways hinzugefügt.

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
   <version>3.0.3</version>
</dependency>
```

Zur Registrierung einer ASP.NET Core Anwendung wird per NuGet-Paket-Manager die Abhängigkeit Steeltoe.Discovery.Eureka installiert. Weiterhin muss in der Datei startup.cs folgender Code hinzugefügt werden.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddDiscoveryClient(Configuration);

                . . .
        }

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            . . .

            app.UseDiscoveryClient();
        }
```

VERWERFEN???

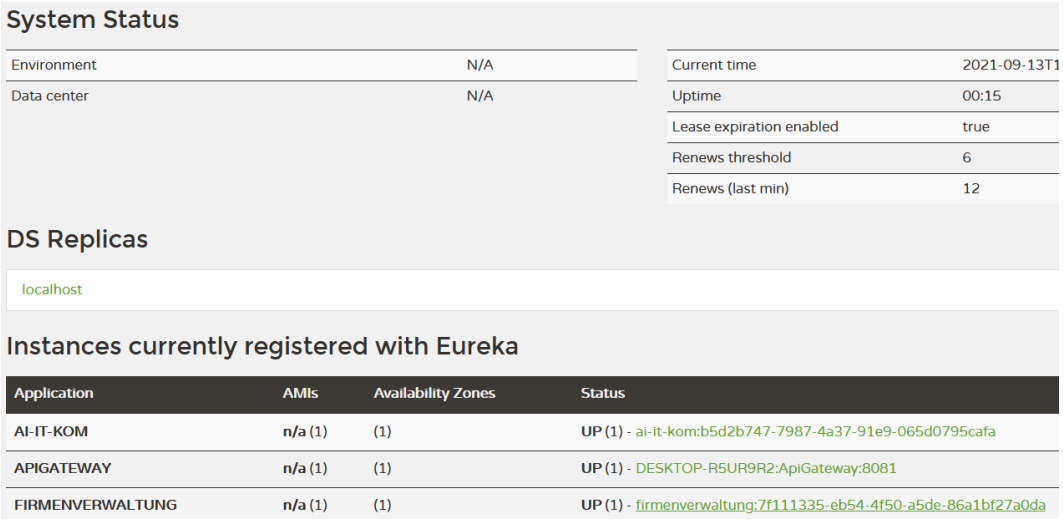
In der Datei application.properties werden zentral die Eigenschaften der jeweiligen Anwendung gespeichert. Für den Eureka Server wurden Server Port und Applikationsname in der Datei wie in Abbildung…. definiert. Zusätzlich wurde festgelegt, dass sich der Server nicht mit sich selbst registrieren kann. https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka

```
server.port=8010

spring.application.name=discoveryservice
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

eureka.client.serviceUrl.defaultZone = http://localhost:8010/eureka
```

Client

```
server.port                                 =                                 ${PORT:0}
spring.application.name                           =                      firmenverwaltung
eureka.instance.instance-id=${spring.application.name}:${random.uuid}
```

Über den festgelegten Port des Eureka-Servers erhält man Zugriff zum Eureka Dashboard. Man erhält von dort aus unter anderem Informationen über alle Registrierten Eureka-Clients. Abbildung … zeigt das Eureka Dashboard mit dem Registrierten Gateway und zwei Microservices.



6.5 Spring Cloud API Gateway

Das API-Gateway wurde anhand des Spring Cloud API-Gateway umgesetzt. Dieses läuft unter dem asynchronen event-driven Framework Netty. Laut Michael Wellner bietet es folgende Features:

- Discovery Client mit Eureka
- Load Balancer mit Ribbon
- Sicherheitskonfigurationsmöglichkeiten mit Spring Security
- Robustheit mit Hystrix
- Limitsetzung für eine bestimmte Anzahl an Requests pro Zeiteinheit (zum Beispiel mit Redis)
- Pfadänderungen

- zeitabhängiges Routing
- Einbindung eigener Filter

Für die IT-Kom Anwendung wurde das Gateway als Spring Boot Projekt ApiGateway umgesetzt. Diesem wurde die Abhängigkeiten spring-cloud-starter-gateway hinzugefügt. In der Datei applications.properties wurde der Port 8081 festgelegt über den Anfragen des Clients zum jeweiligen benötigten Service weitergeleitet werden. Zum Beispiel leitet ein Aufruf der URL localhost://8081/Firmenverwaltung/ einen GET-Request zum Service Firmenverwaltung. Somit muss dem Aufrufer nur noch der Port des Api-Gateways und der Name des Services bekannt sein. Mit dem Integrierten Loadbalancer ist es möglich für Requests eine Lastverteilung auf mehrere Serverinstanzen umzusetzen.

Die verwendung des Loadbalancers wird ermöglicht indem in der Datei application.properties des API-Gateways folgende Notation angegeben wird:

spring.cloud.gateway.routes[*nummer der Route*].uri=lb//*servicename*

Der Port für jede Service Instanz wird automatisch festgelegt, wenn in jedem Service der Port in der Datei applications.properties auf 0 gesetzt wird. Die manuelle Zuweisung von Ports ist dadurch für die Microservices nichtmehr erforderlich.

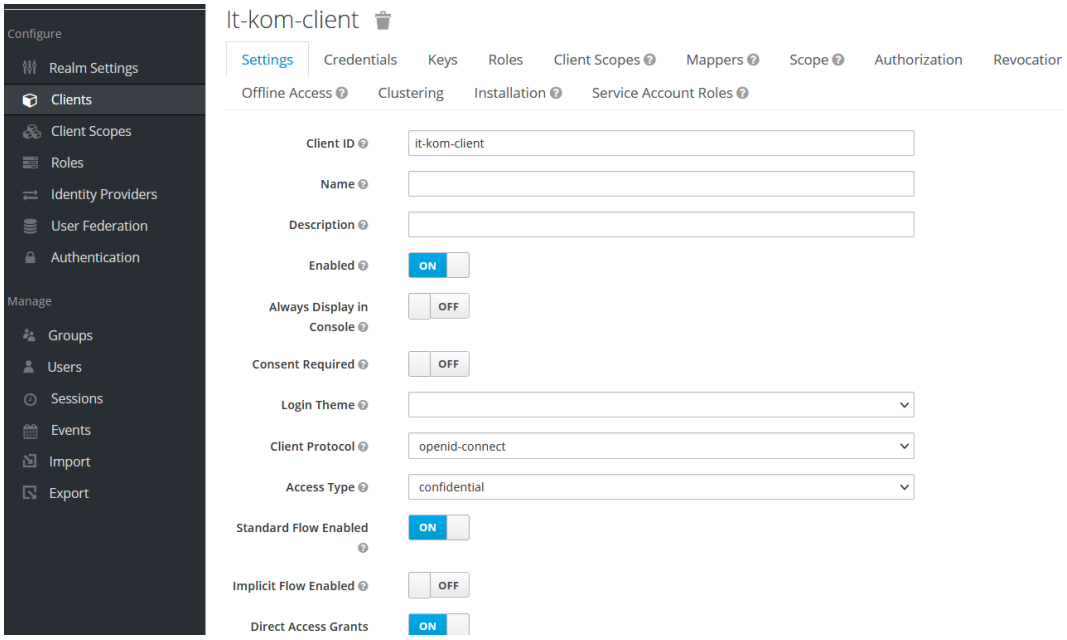
6.6 Keycloak und Spring Security

Keycloac ist ein Opensource Projekt für Identity und Access Management von der Fiirma RedHat. Ziel des Projektes ist es eine sichere Authentifizierung für Anwendungen und Dienste mit möglichst wenig Code sicher zur Verfügung zu stellen. Keycloak stellt einen Authorisierungsserver zur Verfügung, welcher das OpenId Connect Protokoll verwendet. Keyckloak dient als Identity Broker zwischen identity providern und bietet die Möglichkeit auf Accounts von Drittanbietern wie zum Beispiel Facebook oder Youtube zu referenzieren. Dadurch kann dem Benutzer unter anderem das Anlegen von Benutzeraccounts für die Anwendung erspart bleiben. Für die Anwendung der IT-Kom könnte mit dem Einsatz von Keycloak ebenfalls ein Login über Drittanbieter wie zum Beispiel mit der Verwendung des Hochschulaccounts über den Server der Hochschule ermöglicht werden. Damit ein angemeldeter Benutzer nur Ressourcen verwenden kann, für die er Autorisiert ist, wurden für die IT-Kom Anwendung die Rollen ……. definiert. Diese Rollen lassen sich mit geringem Aufwand über die Keycloak-Administrationsoberfläche konfigurieren.

Auf der offiziellen Webseite von Keycloak steht eine Standalon-Server-Distribution frei zum Download zur Verfügung. Diese lässt sich per Konsole über das skript bin\standalone.bat (für Windows) und über bin\standalone.sh (für Linux) starten. https://m.heise.de/developer/artikel/Eine-Identitaet-fuer-alles-mit-Keycloak-3834525.html?seite=all

Anschließend lässt sich im Browser über die URL *http://localhost:8080/auth/admin* die Administrationsoberfläche aufrufen. Für die IT-Kom-Anwendung wurde über diese Oberfläche der Realm it-kom eingerichtet. Unter dem Realm lassen sich Benutzer, Anmeldeinformationen, Rollen und Gruppen anlegen und verwalten. Zusätzlich lassen sich Clients Managen. Bei Clients handelt es sich laut keycloak.org um Entitäten welche von

Keycloak angefordert werden um den Benutzer zu Authentifizieren. Dabei handelt es sich um Dienste oder Anwendungen. Für die IT-Kom Anwendung wurde der Client it-kom-client eingerichtet. Die Abbildung … zeigt die Administrationsoberfläche von Keycloak.



Spring Security

Spring Security ist ein Authentifizierungs- und Zugriffskontroll-Framework für Java Anwendungen. Laut spring.io bietet es unter anderem folgende Features:

- Erweiterbare Unterstützung für Autorisierung und Authentifizierung
- Schutz vor Angriffen wie zum Beispiel Clickjacking, Cross-Side-Request-Forgery und fixation
- Integration mit Spring Web MVC

Zur Integration von Spring Security wurde in der Datei pom.xml der Services und im Api-Gateway die Abhängigkeit spring-boot-starter-security hinzugefügt. Weiterhin wurde jeweils die Klasse SecurityConfig erstellt und mit @EnableWebSecurity gekennzeichnet. Dadurch werden bei jedem Request die Sicherheitskonfigurationen umgesetzt. Zusätzlich wurden Filterketten definiert. Es wurde festgelegt für welche URL ein Login erforderlich ist. Weiterhin können Filter anhand von Rollen gesetzt werden. Die Methoden pathMatchers und permittAll() ermöglicht einen Aufruf der gewählten URL ohne Auhentifizierung. Die Methode authenticated setzt eine Authentifizierung des Benutzers voraus. In Abbildung … wurden alle URLs des Besucher Service welcher unter der URL /ai-it-kom verfügbar ist für alle Benutzer verfügbar gemacht weiterhin kann der Besucherservice über den Firmenverwaltungsservice über die URL /firmenverwaltung/allCompanies alle Firmen ausgeben, die an der Messe teilnehmen. Die URLs /firmenverwaltung und /firmenverwaltung/create sind nur von Firmenaccounts oder Administratoren aufrufbar.

```
public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {
    http.authorizeExchange()
            .pathMatchers("/ai-it-kom/**").permitAll()
            .pathMatchers("/firmenverwaltung/").authenticated()
            .pathMatchers("/firmenverwaltung/create").authenticated()
            .pathMatchers("/firmenverwaltung/allCompanies").permitAll()
```

https://www.keycloak.org/docs/latest/server_admin/

6.7 Synchrone Kommunikation mit Feign Client

Eine Möglichkeit für den Datenaustausch zwischen den Microservices bietet das Spring RestTemplate. Es bietet einen Client zum synchronen Datenaustausch per HTTP. Es stellt dabei Vorlagen für eine Vielzahl möglicher HTTP Anfrageszenarien zur Verfügung. Darüber hinaus bietet es die Möglichkeit Clientseitiges Loadbalancing einzusetzen. Die Verwendung von RestTemplate ist allerdings umständlich weil relativ viel Code benötigt wird, welcher nicht rein intuitiv zu verstehen ist. Eine einfachere Lösung bietet Feign. Es handelt sich dabei um einen HTTP Client welcher von Netflix zur Vereinfachung von HTTP-Clients entwickelt wurde. Der Einsatz von Feign bringt folgende Features mit sich:

- Frei konfigurierbarer Decoder / Encoder zum Beispiel für XML oder JSON
- Logger
- Errorhandler
- Loadbalancer (mittels Ribbon und Eureka)

Feign-Clients bringen jedoch den Nachteil mit sich, das diese nicht mit Binärdateien wie zum Beispiel Datei-Download / -Upload umgehen können. Nur der Einsatz von Textbasierten Schnittstellen wird unterstützt.

Das Folgende Beispiel zeigt den Einsatz von Feign in der IT-Kom Anwendung. Die Ausgabe der Besucherseite wird über den Microservice Besucherservice ermöglicht. Dieser benötigt vom Microservice Firmenverwaltung die für die Messe Angemeldeten Firmendaten. Dazu wird über den Controller guestController ein Request per Feign-Client an den Microservice Firmenverwaltung gesendet. Dieser Ruft die Methode allCompanies auf, welche alle Firmen aus der Datenbank des Firmenveraltungsservice holt und als Liste von CompanieData Objekten zurückgibt.

Zur Verwendung von Feign wird die Abhängigkeit spring-boot-starter-openfeign benötigt, welche in der IT-Kom Anwendung in den entsprechenden Microservices hinzugefügt wurde weiterhin wird in der Main-Methode die Annotation @EnableFeignClients hinzugefügt. Für den Zugriff auf die API-des Firmenverwaltungsservice wurde das Interface FirmenverwaltungServiceClient erstellt. Dieses stellt eine Methode zum Aufruf der API bereit. Abbildung .. zeigt die Deklaration vom Interface.

Das Interface wurde mit der Annotation @FeignClient versehen, wodurch dieses von Spring als Komponente erkannt wird. Die Annotation enthält weiterhin den Namen des Aufzurufenden Services, wodurch eine Adressauflösung per Discovery Service erfolgt. Der Name muss dem Namen entsprechen welcher beim aufzurufenden Service in der Datei applications.properties festgelegt wurde. Die Methode allCompanies im Interface ruft die gleichnamige Methode des Firmenverwaltungsservice auf, welche die Firmendaten aus der Datenbank des Services holt und diese Daten als Liste zurückgibt. Der Pfad der aufzurufenden Methode wird per annotation @GetMappin(„Pfad“) bestimmt.

Das Interface FirmenverwaltungServiceClient und die Methode des aufgerufenen Services werden auf Abbildung .. und .. dargestellt.

https://www.jambit.com/aktuelles/toilet-papers/leichtgewichtige-rest-clients-mit-feign-feign-is-fine/

```
@FeignClient(name="firmenverwaltung")
public interface FirmenverwaltungServiceClient {

  @GetMapping("/allCompanies")
```

```
    public List<CompanyData> allCompanies();

}
```

```
@GetMapping("/allCompanies")
@ResponseBody
public ResponseEntity <List<CompanyData>> allCompanies() {

    List<CompanyData> companyDataList = companyDataRepository.findAll();
    return ResponseEntity.status(HttpStatus.CREATED).body(companyDataList);

}
```

https://login-master.com/keycloak-als-identity-broker/

Zum Aufrufen der Methode des Interfaces wurde dieses im guestController Deklariert und mit der Annotation @Autowired versehen. Das interface wird dadurch von Spring per Dependency-Injection-Funktion automatisch verdrahtet, wodurch die Funktion allCompanies automatisch implementiert wird. Abbildund .. und .. zeigen die Implementierung des Interfaces und den Aufruf der Funktion allCompanies() im guestController.

```
@Autowired
FirmenverwaltungServiceClient firmenverwaltungServiceClient;
```

```
firmenverwaltungServiceClient.allCompanies();
```

6.8 Resilience4J Circuit Breaker

Resilience4J ist eine Fehlertoleranzbibliothek, welche von der populären Bibliothek Netflix Hystrix inspiriert wurde, welche sich mittlerweile im Wartungsmodus befindet. Folgende Funktionalitäten werden von Resilience4J zur Verfügung gestellt:

- Circuit-breaking
- Rate-Limiting (Festlegung der Anzahl gleichzeitiger Anfragen, die ein Client stellen darf)
- Retry (Konfigurierbare Anzahl von Wiederholungen bei fehlgeschlagenen Anfragen, bevor ein Fehler geworfen wird)
- Bulkhead (fehlerhafte Elemente werden in Pools isoliert, so dass funktionsfähige Komponenten weiterhin funktionieren)

Für die IT-Kom Anwendung wurde das Circuit Breaker Pattern über Resilience4J implementiert. Dieses wurde unter anderem im Microservices „Besucher“ angewendet. Zur Integrierung des Circuit-Breakers wurde in der Datei Pom.xml die Abhängigkeit spring-cloud-starter-circuitbreaker-resilience4j hinzugefügt. Das Circuit-Breaker Pattern speichert und aggregiert Aufrufe in einem sliding window. Dieses kann Zählbasiert oder Zeitbasiert implementiert werden. Ein Zählbasiertes sliding window setzt den Zustand des Circuit Breaker Pattern auf geöffnet, wenn eine bestimmte Anzahl an Aufrufen fehlschlägt. Wird die Konfiguration auf Zeitbasierend gesetzt, dann wird das Circuit Breaker nach einer bestimmten Anzahl fehlerhafter Aufrufe in einem bestimmten Zeitraum auf geöffnet. Für Testzwecke wurde das Circuit Breaker Pattern im Rest Controller guestController implementiert. Der Typ des sliding window wurde auf Zählbasiert gesetzt. Für die Größe des sliding indow wurde der Wert 3 festgelegt.

Zusätzlich wurden die Methode slowCallRateThreshold und slowCallDurationThreshold verwendet. SlowCallRateThreshold setzt einen schwellenwert in Abhängigkeit zur Konfigurierten Fenstergröße in Prozent. Überschreitet die Anzahl an Aufrufen mit Zeitüberschreitung den Schwellenwert, dann wird der Zustand des Circuit Breaker Pattern auf Offen gesetzt, nachdem das sliding window befüllt wurde. Im Beispiel wurde dieser Wert auf 50 gesetzt. SlowCallDurationThreshold setzt für jeden Aufruf ein Zeitfenster. Wird dieses Zeitfenster überschritten dann wird dieser Aufruf mit einer Zeitüberschreitung vermerkt. Das Zeitfenster wurde im Beispiel auf maximal eine Sekunde gesetzt. Folgende Abbildung zeigt die Konfiguration und die Erstellung einer Circuit Breaker Instanz mit dem Namen test.

```
CircuitBreakerConfig config = CircuitBreakerConfig.custom()
        .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.COUNT_BASED)
        .slidingWindowSize(3)
        .slowCallRateThreshold(50.0f)
        .slowCallDurationThreshold(Duration.ofSeconds(1))
        .build();

CircuitBreakerRegistry registry = CircuitBreakerRegistry.of(config);
CircuitBreaker circuitBreaker = registry.circuitBreaker("test");
```

Über einen Feign Client wird ein Request an den Microservice Firmenverwaltung gesendet. Dieser erhält als Antwort eine Liste mit allen teilnehmenden Firmen. Welche den Gästen an entsprechender Stelle im User Interface angezeigt werden. Der Aufruf des Feign Client erfolgt über einen Supplier, welcher mit dem Circuit Breaker dekoriert wurde. Siehe Abbildung…

```
Supplier<List<CompanyData>> companyDataSupplier =
        () -> firmenverwaltungServiceClient.allCompanies();
Supplier<List<CompanyData>> decoratedCompanyDataSupplier =
        circuitBreaker.decorateSupplier(companyDataSupplier);
```

https://reflectoring.io/circuitbreaker-with-resilience4j/

https://resilience4j.readme.io/docs/circuitbreake

Ein Test wird auf Abbildung .. gezeigt. Eine for-Schleife wird zehn mal durchlaufen. Bei jedem Durchlauf wird im try-Block versucht über decoratedCompanySupplier.get() die Firmendaten vom Firmenverwaltungsservice zu holen. Der Name der ersten Firma wird zum prüfen der erfolgreichen Datenübertragung bei jedem durchlauf Ausgegeben. Werden die in der Konfiguration festgelegten Grenzwerte überschritten, dann wird eine CallNotPermittetException geworfen und der Code wird im catch-Block weiter ausgeführt. Im falle eines fehlerfreien Ablaufes werden alle Firmendaten an ein Model übergeben. Am Ende wird „firmen“ zurückgegeben, wodurch die Seite firmen.html gerendert wird. Die Firmendaten werden über das Model ausgegeben. Im fehlerfall wird die Seite firmen.html ohne Ausgabe der Firmendaten gerendert. Die Fehlermeldung wird in der Konsole ausgegeben.

```
for (int i = 1; i < 11; i++){

    try {

        companies = decoratedCompanyDataSupplier.get();
        System.out.println(companies.get(0).getCompanyName());

    } catch (CallNotPermittedException e) {

        System.out.println(e.getMessage());
        companies.clear();
```

```
        break;
    }
}
model.addAttribute("companies", companies);
return  "firmen";
```

Die Anfrage des Services Besucher ruft im Firmenverwaltungsservice die Methode allCompanies() (siehe Abbildung ..) auf. Diese gibt eine Liste mit allen verfügbaren Firmendaten zurück. Im Test auf Abbildung .. soll geprüft werden ob der Circuit Breaker entsprechend der Konfiguration auf den Zustand offen gesetzt wird und eine CallNotPermittetException geworfen wird. Dieses Verhalten wird entsprechend der Konfiguration nach drei Aufrufen mit höchstens einer Zeitüberschreitung ausgelöst. Zur Umsetzung wurde in der Methode allCompanies(), Thread.sleep(1100) eingefügt. Dadurch erfolgt nach jedem Aufruf eine Zeitüberschreitung.

Die Konsolenausgabe wird auf Abbildung.. dargestellt. Es wird dreimal der Firmenname des ersten Datensatzes aus der Firmendatenliste ausgegeben. Weil alle drei Aufrufe die Zeit überscheiten wird nach dem dritten Aufruf eine CallNotPermittetException geworfen und der Circuit Breaker geöffnet. Die Fehlermeldung wird in der Konsole ausgegeben. Über einen bestimmten Zeitraum ist keine Anfrage an den Firmenverwaltungsservice mehr möglich. Der Zeitraum für den offenen Zustand kann mit „ .waitDurationInOpenState(Duration.ofSeconds(seconds)) “ festgelegt werden. Nach Ablauf der festgelegten Zeit geht der Circuit Breaker in den Zustand halb offen und nach weiteren Erfolgreichen Aufrufen in den Zustand geschlossen.

```
@GetMapping("/allCompanies")
@ResponseBody
public ResponseEntity <List<CompanyData>> allCompanies() throws
InterruptedException {

    Thread.sleep(1100);

    List<CompanyData> companyDataList = companyDataRepository.findAll();
    return ResponseEntity.status(HttpStatus.CREATED).body(companyDataList);

}
```

```
<tr th:if="${companies.empty}">
    <td colspan="2"> Service ist nicht verfügbar </td>
</tr>
<tr th:each="company : ${companies}">
    <td><span th:text="${company.companyName}"> Title </span></td>
    <td><span th:text="${company.takesPart}"> Author </span></td>
    <td><span th:text="${company.logoPath}"> Author </span></td>

</tr>
```

Erneuern!!!!

```
Firma1
Firma1
Firma1
CircuitBreaker 'test' is OPEN and does not permit further calls
```

https://www.microservice-api-patterns.org/patterns/quality/qualityManagementAndGovernance/RateLimit

https://blog.codecentric.de/en/2019/06/resilience-design-patterns-retry-fallback-timeout-circuit-breaker/

https://docs.microsoft.com/de-de/azure/architecture/patterns/bulkhead

6.9 Docker

Docker ist die Containersoftware der Firma Docker Inc. welche Laut Bernd Öggl und Michael Kofler den Container-Markt als solchen geschaffen haben und aufgrund der schnellen Entwicklung in der Branche das Tempo vorgeben.[26] Die Container werden von der Docker Engine verwaltet.[27] Docker läuft auf Linux- (CentOS, Debian, Fedora, Oracle Linux, RHEL, Suse und Ubuntu) und Windows Server – Betriebssystemen. [28] Die Docker Engine lässt sich über die offizielle Webseite *https://docs.docker.com/get-docker/* herunterladen und wird über die Kommandozeile ausgeführt. Die Docker Desktop Version bringt zusätzlich noch eine Benutzeroberfläche mit sich. Jeder Container wird aus einem Image mit eigenem Dateisystem heraus gestartet welches eine Blaupause für einen Container darstellt. Zum Image gibt es zusätzlich noch eine Beschreibungsdatei welche Konfigurationsanweisungen bereitstellen. Diese Datei wird Dockerfile genannt. Docker-Images können weltweit über Dockerhub ausgetauscht werden, Dockerhub ist ein Repository-Registrierungsdienst welcher es ermöglicht Docker-Images öffentlich oder privat in der Cloud bereitzustellen.

Der Verwendung von Docker-Containern für die IT-Kom Anwendung war unter der Entwicklung an einer lokalen Machine nicht möglich, weil keine IP-Konfiguration für das Netzwerk vorgenommen wurde. Der lokale Rechner auf dem die Anwendung Entwickelt wurde, erhielt die Adresse *localhost*, welche innerhalb eines Docker-Container nicht ohne weiteres Aufgerufen werden kann. Beim Prototypen würde zum Beispiel die Adresse http://localhost:8081/besucher/ zur Öffentlichen Webseite für Besucher führen. Im Livebetrieb würde localhost mit der IP-Adresse der Domain oder dem Domainnamen wie zum Beispiel: http://123.123.123.123:8081/besucher/ oder http://www.ai-it-kom/besucher/ ersetzt werden. Diese IP-Adressen können von Docker-Containern aufgerufen werden. Im Folgenden Abschnitt wird die Erstellung eines Docker-Containers für den Besucherservice erläutert, welcher zum Beispiel im Hochschulrechenzentrum verwendet werden könnte.

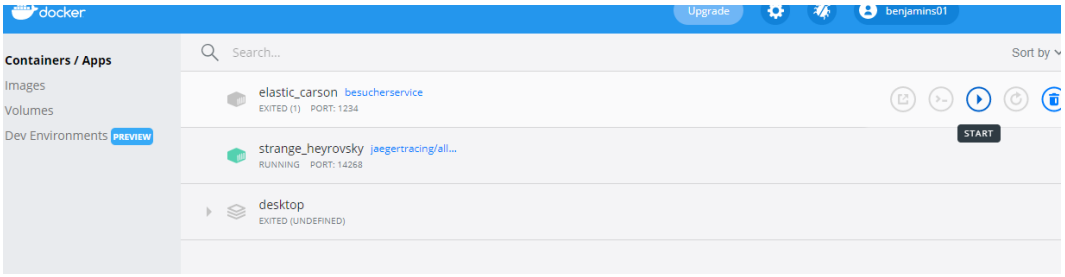
Als erstes wurde eine jar-Datei (*Besucher-0.0.1-SNAPSHOT.jar)* über Maven mit dem Befehl *mvn clean package* erstellt. Für die Erstellung des Docker-Images wurde zunächst eine Datei mit dem Namen *dockerfile* im Besucherservice erstellt. Der Inhalt der Datei wird auf folgender Abbildung dargestellt

```
FROM openjdk:8-jdk-alpine
COPY target/Besucher-0.0.1-SNAPSHOT.jar Besucherservice.jar
ENTRYPOINT ["java","-jar","/Besucherservice.jar"]
```

FROM gibt das Basis-Image an. Es handelt sich hierbei um ein Javafähiges alpine-Linux. COPY kopiert die Jar-Datei in das Image. ENTRYPOINT legt die Ausführbare Datei zum Starten des Containers fest. Das Image wird mit dem Befehl *docker build --tag=besucherservice:latest .* erstellt. Der Container kann mit dem Befehl *docker run -d -p lokaler Port:Container-Port Besucherservice* gestartet werden. Alternativ kann der Container nach der Erstellung des Images über die Docker-Desktop Benutzeroberfläche gestartet werden. Unter der Verwendung eines Dockerhub-Repositories könnte der Container schnell und einfach in der Cloud zum Beispiel für Testzwecke geteilt werden.

[26] (Öggl, et al., 2019)
[27] (Docker, 2021)
[28] (Docker, 2021)

Abbildung .. zeigt die Benotzeroberfäche von Docker-Desktop



https://www.dev-insider.de/was-ist-docker-a-733683/

https://www.innoq.com/de/articles/2015/11/docker-perfekte-verpackung-fuer-micro-services/

http://www.anecon.com/blog/docker-basics-befehle-und-life-hacks/

6.10 Jaeger

Jeaeger ist ein open source Distributet Tracing System von der Firma Uber. Es dient zur Fehlerbehebung und Überwachung in einer Microservice-Anwendung. Es Visualisiert den gesamten Prozessfluss einer Anfrage durch verschiedene Microservices. Folgende Inhalte werden dabei geboten:

- Transaktionsüberwachung
- Ursachenanalyse
- Analyse von Dienstabhängigkeiten
- Latenz- und Performanceoptimierung

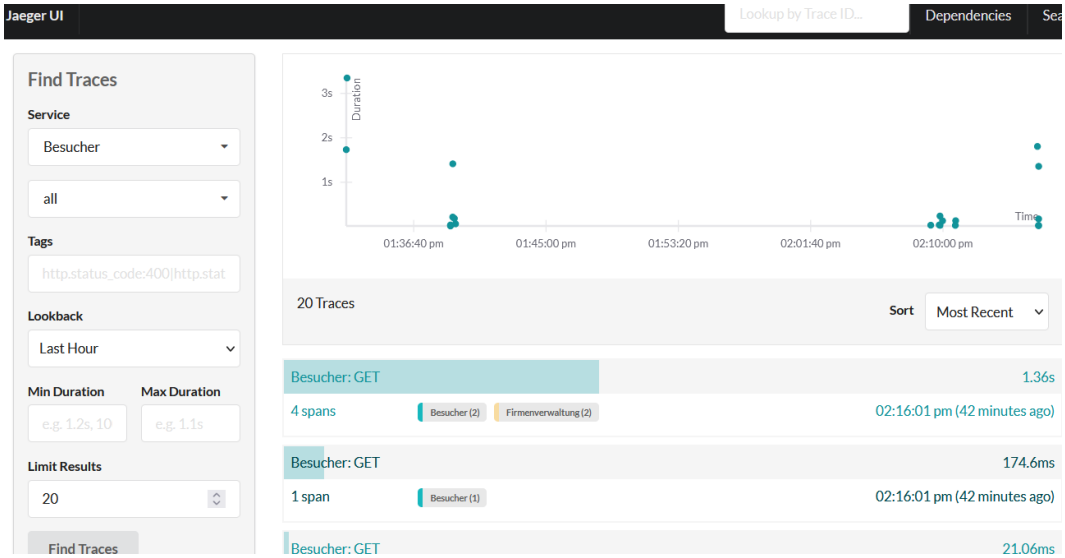
Jeager wurde in der IT-Kom Anwendung implementiert um unter anderem die Wartbarkeit zu Optimieren. Dazu wurden im Api-Gateway und in den jeweiligen Microservices in der Datei Pom.xml die Abhängigkeit opentracing-spring-jaeger-cloud-starter hinzugefügt. Im Api-Gatewway wurde zusätzlich noch die Abhängigkeit opentracing-spring-gateway-cloud-starter hinzugefügt, damit des Spans eins Trace weitergeleitet werden. Ohne diese Abhängigkeit könnten unter der Verwendung des Spring-Cloud-Gateways aufrufe über mehrere Services hinweg nicht zusammen ausgewertet werden, weil die Spans in diesem Fall nicht über das Gateway weitergeleitet werden.

Zur Konfiguration von Jaeger wurde im Gateway und in den einzelnen Services die Klasse JaegerConfig erstellt. Diese wird auf Abbildung .. dargestellt.

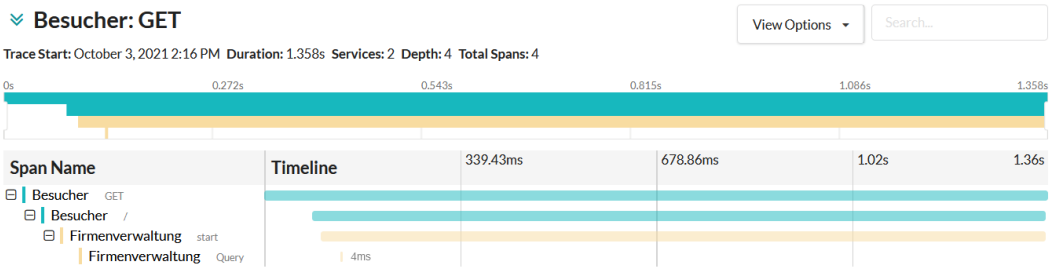
```
@Configuration
public class JaegerConfig {

    @Bean
    public JaegerTracer jaegerTracer() {

        return new io.jaegertracing.Configuration("Besucher")
                .withSampler(new
io.jaegertracing.Configuration.SamplerConfiguration().withType(ConstSampler.TYPE)
                        .withParam(1))
                .withReporter(new
io.jaegertracing.Configuration.ReporterConfiguration().withLogSpans(true))
                .getTracer();
    }
}
```

In der IT-Kom Anwendung wird die All-in-One Lösung von Jaeger verwendet. Diese bietet eine einfache Anwendung für Testzwecke. Sie enthält eine Benutzeroberfläche und eine

integrierte in Memory Speicherkomponente. Starten lässt sich die Anwendung über ein vorgefertigtes Docker-Image welches von DockerHub heruntergeladen wurde. Dazu wurde der Befehl *docker pull jaegertracing/all-in-one* in der Konsole eingegeben. Wird der Container gestartet dann lässt sich im Anschluss die Jaeger-Benutzeroberfläche im Browser über die URL http://localhost:16686/search aufgerufen werden. Auf Abbildung .. wird die Benutzeroberfläche dargestellt.



Zur Veranschaulichung wird die Jaeger-UI per Aufruf der Methode *allCompanies* des Besucher-Service dargestellt. Diese Methode ruft den Firmenverwaltungservice auf, welcher die Firmendaten an den Besucherservice sendet. Die Aufrufe bilden zusammen einen Trace. Die einzelnen Aufrufe bilden die Spans. Der Trace wird zusammen mit den Spans auf der Benutzeroberfläche aufgelistet. Dauer für die einzelnen Spans und für den gesamten Trace kann Per Jaeger UI ausgewertet werden. Abbildung .. zeigt Trace und Spans der Funktion allCompanies des Besucherservice.



TODO Docker Desktop Bild

6.11

7 Auswertung

7.1 Ergebnis

7.2 Ausblicke

8 Zusammenfassung

V. Literaturverzeichnis

Alzve, João. 2021. golem. [Online] 19. Juli 2021. [Zitat vom: 07. August 2021.] https://www.golem.de/news/verteilte-systeme-die-haeufigsten-probleme-mit-microservices-2107-157885.html.

Anicas, Mitchell. 2014. DigitalOcean. [Online] 21. July 2014. [Zitat vom: 19. August 2021.] https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

Augsten, Stephan. 2020. dev-insider. [Online] 03. Juli 2020. [Zitat vom: 19. August 2021.] https://www.dev-insider.de/was-ist-ein-framework-a-938758/.

ComputerWeekly, Redaktion. 2020. ComputerWeekly. [Online] Juli 2020. [Zitat vom: 16. August 2021.] https://www.computerweekly.com/de/definition/Load-Balancing.

Docker. 2021. docker.com. [Online] Docker, Inc., 4. August 2021. [Zitat vom: 4. August 2021.] https://www.docker.com/products/container-runtime.

Fink, Andreas. 2012. Enzyklopädie der wirtschaftsinformatik Online Lexikon. [Online] 31. 10 2012. [Zitat vom: 12. August 2021.] https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Monolithisches-IT-System.

Gillis, Alexander S. 2021. TechTarget. [Online] April 2021. [Zitat vom: 14. August 2021.] https://whatis.techtarget.com/de/definition/Service-Discovery-Diensterkennung.

Gnatyk , Romana . 2018. N-iX. [Online] 03. Oktober 2018. [Zitat vom: 07. August 2021.] https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/.

Hruschka, Peter und Starke, Gernot. 2017. *arc42 template.* Januar 2017.

Mohapatra, Biswa Pujarini, Banerjee, Baishakhi und Aroraa, Gaurav. 2019. Microservices by Example Using .Net Core. Neu-Delhi : BPB Publications, 2019, S. 2.

Öggl, Bernd und Kofler, Michael. 2019. Docker, Das Praxisbuch für Entwickler und DevOps-Teams. Bonn : Rheinwerk Verlag, 2019, Bd. 1. korrigierter Nachdruck, S. 9.

Plöd, Michael. 2016. innoq. [Online] 08. Dezember 2016. [Zitat vom: 18. August 2021.] https://www.innoq.com/de/articles/2016/12/ddd-microservices/.

Röwekamp, Lars und Limburg, Arne. 2016. heise. [Online] 09. Februar 2016. [Zitat vom: 18. August 2021.] https://m.heise.de/developer/artikel/Der-perfekte-Microservice-3091905.html?seite=all&hg=1&hgi=2&hgf=false.

te Wierik, Mattias. 2020. medium. [Online] 11. November 2020. [Zitat vom: 19. August 2021.] https://medium.com/swlh/authentication-and-authorization-in-microservices-how-to-implement-it-5d01ed683d6f.

Wolff, Eberhard. 2018. *Microservices.* Heidelberg : dpunkt.verlag GmbH, 2018, S. 32-33.

—. 2018. *Microservices.* Heidelberg : dpunkt.verlag GmbH, 2018, S. 60.

—. 2018. Das Microservices Praxisbuch. Heidelberg : dpunkt.verlag GmbH, 2018, S. 4.

—. 2018. Das Microservices Praxisbuch. Heidelberg : dpunkt.verlag Gmbh, 2018, S. 96-97.

—. 2018. Das Microservices-Praxisbuch. Heidelberg : dpunkt.verlag GmbH, 2018, S. 62-63.

—. 2017. innoq. [Online] 04. August 2017. [Zitat vom: 08. August 2021.] https://www.innoq.com/de/articles/2017/08/microservices-der-aktuelle-stand/.

—. 2018. Microservices. Heidelberg : dpunkt.verlag GmbH, 2018, S. 44-45.

VI. Anhang

VII. Selstständigkeitserklärung