

**Bachelorarbeit**

**In der Angewandten Informatik**

**Entwicklung eines Systems zur Verwaltung der ausstellenden Firmen der IT-Kontaktmesse an der Fachhochschule Erfurt…**

Prototypische Implementierung auf Basis einer Microservice-Architektur

**Benjamin Swarovsky**

Abgabedatum: 01.11.2021

**Prof. Dr. Steffen Avemarg**

**Dipl.-Inf. Steffen Späthe**

1. Kurzfassung
2. Abstract
3. Aufgabenstellung

Inhaltsverzeichnis

[I. Kurzfassung II](#_Toc79847991)

[II. Abstract III](#_Toc79847992)

[III. Aufgabenstellung IV](#_Toc79847993)

[IV. Abbildungs und Tabellenverzeichniss IV](#_Toc79847994)

[1 Einleitung 1](#_Toc79847995)

[1.1 Problemstellung 1](#_Toc79847996)

[1.2 Ziel 1](#_Toc79847997)

[2 Grundlagen 3](#_Toc79847998)

[2.1 Microservices 3](#_Toc79847999)

[2.1.1 Eigenschaften 3](#_Toc79848000)

[2.1.2 Vorteile 3](#_Toc79848001)

[2.1.3 Nachteile 3](#_Toc79848002)

[2.2 Spring / ASP.Net 4](#_Toc79848003)

[2.3 Docker 4](#_Toc79848004)

[2.3.1 Allgemein 4](#_Toc79848005)

[2.4 Kubernetes 5](#_Toc79848006)

[2.5 API Datenaustausch 5](#_Toc79848007)

[2.6 Redis ???? 5](#_Toc79848008)

[2.7 Prometheus ????? 5](#_Toc79848009)

[2.8 Consul ??? 5](#_Toc79848010)

[2.9 Istio ??? 5](#_Toc79848011)

[2.10 Microsoft Azure 5](#_Toc79848012)

[2.11 Kafka 5](#_Toc79848013)

[2.12 RabbitMQ 5](#_Toc79848014)

[2.13 API Gateway 6](#_Toc79848015)

[2.14 BUS 7](#_Toc79848016)

[2.15 Load balancer 7](#_Toc79848017)

[2.16 Buildpipeline 7](#_Toc79848018)

[2.17 JSON Webtoken 7](#_Toc79848019)

[3 Anforderungsanalyse 8](#_Toc79848020)

[3.1 Aufgabenstellung 8](#_Toc79848021)

[3.2 Qualitätsziele 9](#_Toc79848022)

[3.3 Stakeholder 10](#_Toc79848023)

[4 Architektur 12](#_Toc79848024)

[4.1 Lösungsstrategie 12](#_Toc79848025)

[4.1.1 Deployment und Management mit Docker 12](#_Toc79848026)

[4.1.2 Microservice Framework VS Containermanager 12](#_Toc79848027)

[4.1.3 Use Interface 12](#_Toc79848028)

[4.2 Systemkontext 13](#_Toc79848029)

[4.3 Bausteinsicht Ebene1 13](#_Toc79848030)

[4.4 Bausteinsicht Ebene2 13](#_Toc79848031)

[4.5 Verteilungssicht 13](#_Toc79848032)

[4.6 Laufzeitsicht 13](#_Toc79848033)

[4.7 Querschnittliche Konzepte 13](#_Toc79848034)

[4.7.1 Domainmodell 13](#_Toc79848035)

[4.7.2 Testverfahren 13](#_Toc79848036)

[4.7.3 Sicherheit 13](#_Toc79848037)

[5 Implementierung 14](#_Toc79848038)

[6 Auswertung 14](#_Toc79848039)

[6.1 Ergebnis 14](#_Toc79848040)

[6.2 Ausblicke 14](#_Toc79848041)

[7 Zusammenfassung 14](#_Toc79848042)

[V. Literaturverzeichnis IV](#_Toc79848043)

[VI. Anhang V](#_Toc79848044)

[VII. Selstständigkeitserklärung VI](#_Toc79848045)

1. Abbildungs und Tabellenverzeichniss

# Einleitung

## Problemstellung

Über Jahre hinweg wurden Softwaresystem als Monolithen deployt. Aufgrund ihrer eng gekoppelten Komponenten bilden solche Systeme eine Untrennbare Einheit.[[1]](#footnote-1) Weil diese Anwendungen mit der Zeit immer größer wurden, entstanden für die Entwickler Organisatorische Probleme.[[2]](#footnote-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]](#footnote-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, Die Performance Verluste aufgrund der Systemverteilung oder das unabhängige Testen einzelner Microservices mit Modularisierten Tests.[[4]](#footnote-4)

## Ziel

Diese Wissenschaftliche Arbeit behandelt die Problemstellung ……………………..

Mit dieser Problemstellung wird sich anhand eines Praktischen Beispiels auseinandergesetzt. Die Umsetzung erfolgt anhand eines Entwurfes für ein Systems zur Verwaltung der teilnehmenden Firmen der IT-Kontaktmesse an der Fachhochschule Erfurt. Diese Messe findet jährlich auf dem Gelände der Fachhochschule Erfurt statt. Unternehmen aus der Region der Hochschule 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 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 Problemstellung………………… eingegangen. Im Anschluss wird ein Prototyp implementiert. Dieser soll zeigen ob die Konzipierten Lösungswege für die praktische Anwendung geeignet sind………..

# Grundlagen

## Microservices

### Eigenschaften

Microservices stellen einen Software-Architekturansatz dar

Microservice-Strukturen 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]](#footnote-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]](#footnote-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]](#footnote-7)

-Conteniuous Delivery

### Vorteile

Eine Microservice-Architektur ist weniger anfällig für das ungewollte Einbauen von Abhängigkeiten zwischen einzelnen Komponenten. Dieses Verhalten entsteht aufgrund der hohen Modularität von Microservices, und der Schwierigkeit die Grenzen der einzelnen Microservices zu überschreiten. Ein Zerfall der Architektur kann dadurch vermieden werden.[[8]](#footnote-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.

### Nachteile

## 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]](#footnote-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]](#footnote-10)

## Docker

### Allgemein

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.[[11]](#footnote-11) Container bieten einen universellen Paketierungsansatz, bei dem alle Anwendungsabhängigkeiten in einem Container gebündelt werden. Die Container werden von der Docker Engine ausgeführt.[[12]](#footnote-12)  
Container bieten gegenüber Virtuellen Maschinen (welche es ermöglichen auf einem Rechner mehrere Betriebssysteme laufen zu lassen) mehrere Vorteile. Beim Einsatz von Containern wird kein ganzes Betriebssystem Installiert. Dadurch wird die Erzeugung des Overheads einer Virtuellen Maschine beim Ausführen von Softwarekomponenten wie zum Beispiel Webserver, Programmiersprachen und Datenbanken vermeiden. Weiterhin lassen sich dadurch Container schneller aufsetzen als Virtuelle Maschinen und Entwicklern werden unter anderem neue Möglichkeiten im Deployment geboten.[[13]](#footnote-13)  
Docker läuft auf Linux- (CentOS, Debian, Fedora, Oracle Linux, RHEL, Suse und Ubuntu) und Windows Server – Betriebssystemen. [[14]](#footnote-14)

## Kubernetes

## API Datenaustausch

## Redis ????

## Prometheus ?????

## Consul ???

## Istio ???

## Microsoft Azure

## Kafka

## RabbitMQ

## API Gateway

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

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.

**Ohne Gateway**



**Mit Gateway**

****

## 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. [[15]](#footnote-15)

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.



Beispiele: Netflix Eureka

**Serverseitige Erkennung**

## BUS

## 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]](#footnote-16)

## 



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

## Buildpipeline

## JSON Webtoken

# Anforderungsanalyse

## 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
* …

## Qualitätsziele

Laut Dr. Peter Hruschka und Dr. Gernot Starke beeinflusst die Ernennung der für die Stakehholder wichtigsten Qualitätsziele, die Softwarearchitektur maßgebend.[[17]](#footnote-17) 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.

## 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. |
|  |  |  |

# Konzepte

## DDD

**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.[[18]](#footnote-18)  
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.[[19]](#footnote-19)

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

Für die Interaktion zwischen den Bounded Contexts sorgt im besten Fall ein Eventsystem. [[20]](#footnote-20)

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. [[21]](#footnote-21)

**Shared Kernel**

**Schnittstellen**

<https://m.heise.de/developer/artikel/Der-perfekte-Microservice-3091905.html>

Laut Lars Röwekamp und Arne Limburg kann ein Service in der Praxis nie vollständig autark sein. Daher ist für die Erfolgreiche Umsetzung einer Microservice-Architektur, das Design der Schnittstellen von entscheidender Bedeutung. Diese sollten einen stabilen Zustand aufweisen. Das Vorgehen bezüglich der Versionierung sollte gut durchdacht werden denn beim Ändern einer Schnittstelle sind immer mindestens zwei Services betroffen.

## 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.[[22]](#footnote-22)

**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 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.[[23]](#footnote-23)

## User Interface

### Frontend Monolith

Laut Eberhard Wolff sollte der Einsatz eines Frontend Monolithen bei einer Microservice Architektur stets hinterfragt 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 Anwendung  
  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 arbeiten.[[24]](#footnote-24)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### Modularisiertes Frontend



## SpringBoot

## Eureka Discovery Server

## Deployment und Management mit Docker

Damit die Vorteile von Microservices (siehe Kapitel 3.1.2) ausgenutzt werden können, müssen diese laut Eberhard Wolff mindestens getrennte Prozesse sein. Dadurch 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. Die Aufteilung der Microservices auf Virtuellen Maschinen beanspruchen allerdings zu viel Speicher, weil dadurch jeder Microservice die Instanz eines Betriebssystems besitzt.   
Eine effizientere Lösung bietet der Einsatz von Docker. Dadurch bleibt ein Microservice so leichtgewichtig wie ein Prozess.[[25]](#footnote-25)

### Microservice Framework VS Containermanager

<https://www.innoq.com/de/articles/2017/06/warum-ein-container-manager-das-bessere-microservice-framework-ist/>

# Architekturentwurf

(Nach dem Stand der Wissenschaft und Technik)

## Lösungsstrategie

### Zusammenspiel API Gateway, Service Discovery, Service Mesh

## Systemkontext



## Bausteinsicht Ebene1

**Domain Driven Design**

**Schneiden Anhand von UseCases**

## Bausteinsicht Ebene2

## Verteilungssicht

## Laufzeitsicht

## Querschnittliche Konzepte

### Domainmodell

### Testverfahren

**OAuth2**

# Implementierung

## Eureka Discovery Service

### Konfiguration

## Services

### Firmenverwaltung

### Newsletter

# Auswertung

## Ergebnis

## Ausblicke

# Zusammenfassung

1. 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.

1. Anhang
2. Selstständigkeitserklärung

1. (Fink, 2012) [↑](#footnote-ref-1)
2. (Alzve, 2021) [↑](#footnote-ref-2)
3. (Gnatyk , 2018) [↑](#footnote-ref-3)
4. (Wolff, 2017) [↑](#footnote-ref-4)
5. (Mohapatra, et al., 2019) [↑](#footnote-ref-5)
6. (Wolff, 2018) [↑](#footnote-ref-6)
7. (Wolff, 2018) [↑](#footnote-ref-7)
8. (Wolff, 2018) [↑](#footnote-ref-8)
9. (Augsten, 2020) [↑](#footnote-ref-9)
10. (Bayer, 2019) [↑](#footnote-ref-10)
11. (Öggl, et al., 2019) [↑](#footnote-ref-11)
12. (Docker, 2021) [↑](#footnote-ref-12)
13. (Öggl, et al., 2019) [↑](#footnote-ref-13)
14. (Docker, 2021) [↑](#footnote-ref-14)
15. (Gillis, 2021) [↑](#footnote-ref-15)
16. (ComputerWeekly, 2020) [↑](#footnote-ref-16)
17. (Hruschka, et al., 2017) [↑](#footnote-ref-17)
18. (Wolff, 2018) [↑](#footnote-ref-18)
19. (Röwekamp, et al., 2016) [↑](#footnote-ref-19)
20. (Plöd, 2016) [↑](#footnote-ref-20)
21. (Wolff, 2018) [↑](#footnote-ref-21)
22. (te Wierik, 2020) [↑](#footnote-ref-22)
23. (Anicas, 2014) [↑](#footnote-ref-23)
24. (Wolff, 2018) [↑](#footnote-ref-24)
25. (Wolff, 2018) [↑](#footnote-ref-25)