

**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](#_Toc79234486)

[II. Abstract III](#_Toc79234487)

[III. Aufgabenstellung IV](#_Toc79234488)

[IV. Abbildungs und Tabellenverzeichniss IV](#_Toc79234489)

[1 Einleitung 1](#_Toc79234490)

[1.1 Problemstellung 1](#_Toc79234491)

[1.2 Ziele 1](#_Toc79234492)

[2 Grundlagen 1](#_Toc79234493)

[2.1 Microservices 1](#_Toc79234494)

[2.1.1 Eigenschaften 1](#_Toc79234495)

[2.1.2 Vorteile 1](#_Toc79234496)

[2.1.3 Nachteile 1](#_Toc79234497)

[2.2 Spring / ASP.Net 1](#_Toc79234498)

[2.3 Docker 2](#_Toc79234499)

[2.3.1 Allgemein 2](#_Toc79234500)

[2.4 API Datenaustausch 3](#_Toc79234501)

[2.5 API Gateway 3](#_Toc79234502)

[2.6 BUS 3](#_Toc79234503)

[2.7 Load balancer 3](#_Toc79234504)

[2.8 Buildpipeline 3](#_Toc79234505)

[2.9 Token 3](#_Toc79234506)

[3 Anforderungen 3](#_Toc79234507)

[3.1 Funktionale Anforderungen 3](#_Toc79234508)

[3.2 Qualitätskriterien 3](#_Toc79234509)

[4 Architektur 3](#_Toc79234510)

[4.1 Lösungsstrategie 3](#_Toc79234511)

[4.1.1 Docker 3](#_Toc79234512)

[4.2 Bausteinsicht Ebene1 3](#_Toc79234513)

[4.3 Bausteinsicht Ebene2 3](#_Toc79234514)

[4.4 Verteilungssicht 3](#_Toc79234515)

[4.5 Laufzeitsicht 3](#_Toc79234516)

[4.6 Querschnittliche Konzepte 4](#_Toc79234517)

[4.6.1 Domainmodell 4](#_Toc79234518)

[4.6.2 Testverfahren 4](#_Toc79234519)

[4.6.3 Sicherheit 4](#_Toc79234520)

[5 Implementierung 4](#_Toc79234521)

[6 Auswertung 4](#_Toc79234522)

[6.1 Ergebnis 4](#_Toc79234523)

[6.2 Ausblicke 4](#_Toc79234524)

[7 Zusammenfassung 4](#_Toc79234525)

[V. Literaturverzeichnis IV](#_Toc79234526)

[VI. Anhang V](#_Toc79234527)

[VII. Selstständigkeitserklärung VI](#_Toc79234528)

1. Abbildungs und Tabellenverzeichniss

# Einleitung

## Problemstellung

Die Methoden zur Softwareentwicklung sind im Ständigen Wandel. Über Jahre hinweg wurden Softwaresystem als Monolithen deployt. Weil diese Anwendungen über die Jahre immer größer wurden, entstanden für die Entwickler Organisatorische Probleme.[[1]](#footnote-1) Extrem große Systeme sind nicht leicht zu verstehen und zu verwalten. Durch die enge Kopplung der Komponenten bei einem Monolithen 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 stellt sich in Bezug auf die Skalierbarkeit. 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.[[2]](#footnote-2)  
  
Gegenüber diesen Nachteilen schafft eine Microservice Architektur Abhilfe.  
Seitdem gibt es einen regelrechten Hype um dieses Architekturmuster.

## Ziele

# 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.[[3]](#footnote-3)

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.[[4]](#footnote-4) 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.[[5]](#footnote-5)

-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.[[6]](#footnote-6)  
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

## Spring / ASP.Net

## 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.[[7]](#footnote-7) Container bieten einen universellen Paketierungsansatz, bei dem alle Anwendungsabhängigkeiten in einem Container gebündelt werden. Die Container werden von der Docker Engine ausgeführt.[[8]](#footnote-8)  
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.[[9]](#footnote-9)  
Docker läuft auf Linux- (CentOS, Debian, Fedora, Oracle Linux, RHEL, Suse und Ubuntu) und Windows Server – Betriebssystemen. [[10]](#footnote-10)

## API Datenaustausch

## API Gateway

## BUS

## Load balancer

## Buildpipeline

## Token

# Anforderungen

## Funktionale Anforderungen

## Qualitätskriterien

# Architektur

## Lösungsstrategie

### 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.[[11]](#footnote-11)

## Bausteinsicht Ebene1

## Bausteinsicht Ebene2

## Verteilungssicht

## Laufzeitsicht

## Querschnittliche Konzepte

### Domainmodell

### Testverfahren

### Sicherheit

**OAuth2**

# Implementierung

# 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.

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

**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/.

**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.

**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. 62-63.

1. Anhang
2. Selstständigkeitserklärung

1. (Alzve, 2021) [↑](#footnote-ref-1)
2. (Gnatyk , 2018) [↑](#footnote-ref-2)
3. (Mohapatra, et al., 2019) [↑](#footnote-ref-3)
4. (Wolff, 2018) [↑](#footnote-ref-4)
5. (Wolff, 2018) [↑](#footnote-ref-5)
6. (Wolff, 2018) [↑](#footnote-ref-6)
7. (Öggl, et al., 2019) [↑](#footnote-ref-7)
8. (Docker, 2021) [↑](#footnote-ref-8)
9. (Öggl, et al., 2019) [↑](#footnote-ref-9)
10. (Docker, 2021) [↑](#footnote-ref-10)
11. (Wolff, 2018) [↑](#footnote-ref-11)