Möglichkeiten zur Kontrolle und Steuerung der Kommunikation von Microservices

**Bachelorarbeit**

im Studiengang  
Softwaretechnik und Medieninformatik

vorgelegt von

**Gerrit Wildermuth**Matr.-Nr.: 74734

am 1. Januar 2027   
an der Hochschule Esslingen

Erstprüfer/in:   
Zweitprüfer/in:

# Kurzfassung

# Abstract

**Keywords:**

# Inhaltsverzeichnis

Kurzfassung 2

Abstract 2

Inhaltsverzeichnis 3

Abbildungsverzeichnis 4

Tabellenverzeichnis 4

Codeverzeichnis 4

Abkürzungsverzeichnis 5

1 Überblick 6

2 Ziele 7

3 Stand der Technik 8

4 Zusammenfassung und Ausblick 9

Glossar 10

Literaturverzeichnis 11

Ehrenwörtliche Erklärung 12

Stichwortverzeichnis 13

# Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Tabellenverzeichnis

Tabelle 1: Unterschiede zwischen Abbildungen und Tabellen 27

Tabelle 2: Beispiele für Überschriftebenen 36

Tabelle 3: Aufstellung der wichtigsten Formatvorlagen der Dokumentvorlage 48

# Codeverzeichnis

[Codeteil 1: Beispiel für einen Codeauszug 34](#_Toc419798065)

# Abkürzungsverzeichnis

ALP Arbeits-, Lern- und Präsentationstechniken

HBI Hochschule für Bibliotheks- und Informationswesen

HdM Hochschule der Medien

HSE Hochschule Esslingen

# Überblick/Einleitung

Microservices, sind grob gesehen Teilmodule Vollständer Softwaresysteme und können sowohl physisch, sprachlich und strukturell getrennt sein. Durch diese Trennung, entsteht eine große Menge, als auch unterschiedliche Arten der Kommunikation. Diese Kommunikation bietet viele Möglichkeiten als auch Tücken. Es muss hierbei, wie so oft, darauf geachtet werden diese möglichst zu erkennen und entsprechend zu behandeln.

Die größte Gefahr hierbei ist, dass das System nicht stabiler, sondern womöglich unübersichtlich wird und dem entsprechend schwieriger zu handhaben. Hier wird im Speziellen darauf geschaut, wie wir das System von Microservices, in möglichst jeder Hinsicht robust gestalten können und welche unterschiedlichen Möglichkeiten wir hierbei verwenden können.

Robust bedeutet in diesem Falle, dass bei einem Fehler keine Kettenreaktion stattfinden darf, dass wenn erwünscht Nachrichten immer ankommen müssen, dass wenn ein Service überlastet ist, dieser sich wieder erholen darf, dass Services eigenständig upgedatet werden können, dass auf Leistungsanfragen automatisch reagiert wird oder diese keinen Einsturz ergeben, dass bei Fehlern auf Fallbacks zurückgegriffen wird, wenn dies Business technisch möglich ist, dass kontinuierliche „Healthchecks“(Lebst du noch?) durch-geführt werden, dass Auslastung über die einzelnen gleichen Microservices aufgeteilt wird und dass nachvollzogen werden kann, wie der Traffic verläuft.

Ein weiterer Punkt der Kommunikation, der hierbei untersucht wird, beinhaltet den Kommunikationsfluss welcher mit wachsender Größe sich immer komplizierter gestalten kann.

Bei einer Vielzahl an Microservices, wird es auch Wichtig, dass die einzelnen Services sich finden können bzw. sich an einem zentralen Punkt Anmelden und sichtbar machen können.

Um eine möglichst geschickte Verwaltung von Updates zu Gewehrleisten, sollte man auf keinen Fall, auf einen zentralen Konfigurationspunkt verzichten. Hierdurch können Updates auf Vielerlei Services gleichzeitig geändert werden, ohne irgendein durcheinander zu verursachen.

Dabei werden sowohl Drittanbieter Software, als auch mögliche Software Patterns und Techniken untersucht und in entsprechende Kategorien und Softwareschichten eingeteilt.

Es soll dabei aber nicht um die Organisation oder die Sicherheit dieser Services gehen.

# Microservices

## Monolith oder Microservices

### Monolith

#### Einführung

Ein Monolith ist im Deutschen ein einheitlicher Stein. In unserem Fall widerspiegelt es ein zusammenhängendes Softwaresystem wieder, welches es seit dem Beginn des Internets gibt und alle drei Komponenten in sich Vereinigt. Diese Komponenten sind die UI, die Datenbank und einen Server. Der Code mag in unterschiedliche Teile aufgeteilt sein aber das System wird immer als ganzes, einheitliches Paket veröffentlicht. Bei jeder Änderung muss somit auch das ganze System neu veröffentlicht werden.

#### Vorteile der Monolithen

* Der größte Vorteil eines Monolithen ist, die nahe Verknüpfung der einzelnen Bestandteile und somit kann auf weitläufige Kommunikation verzichtet werden und einiges an Fehlern und Arbeitsaufwand vermieden werden. Es ist dadurch möglich schneller zu Ergebnissen zu kommen, was vor allem bei eingespielten Entwicklerteam offensichtlich wird.
* Durch die zusammenhängende Architektur, gestaltet es sich einfacher, das System auf einem Server zum Laufen zu bringen. Dies ist dadurch möglich, dass keine weiteren Verbindungen, Ports oder ähnliches festgelegt werden müssen da alles auf einem und demselben System läuft.
* Datenbank ist an einem Punkt wodurch Konsistenz deutlich einfach bzw. überhaupt erreicht werden kann.
* Es ist deutlich einfacher, eine Datenbank konsistent zu halten, da diese eng mit dem Rest des Systems gekoppelt ist.

#### Nachteile der Monolithen

* Je größer die Anwendung wird, also umso mehr Zeilen Code sie enthält und zusätzliche Features implementiert werden umso wertvoller, werden Entwickler die sich schon mit dem ganzen System auskennen. Allerdings wird es genauso auch immer schwieriger für neue Entwickler einzusteigen und eigene Beiträge hin beizusteuern.
* Ein weiteres Problem von zu groß werdenden zusammenhängenden Systemen ist, das jeder Änderung das gesamte System beeinflussen kann. Dies hat zur Folge, dass der Testaufwand bei Änderungen um ein vielfaches größer sein kann. Resultierend kann es schwieriger werden neue Technologien in ein bestehendes System zu integrieren.
* Jede Änderung des Systems, egal ob klein oder groß, hat zur Ursache, dass das gesamte System neu bereitgestellt werden muss. Dies spielt keine große Rolle, solange die Anwendung noch klein ist doch je größer sie wird, umso länger dauert dieser Prozess.
* Durch eine starke Verzahnung der Teilbereiche können Projekte mit monolithischer Struktur, nicht so leicht viel neues Personal aufnehmen.

### Microservices

#### Einführung

Verschiedene Bereiche der Anwendung werden aufgeteilt und ausgelagert in eigene Services. Jeder Service kommuniziert über Schnittstellen mit anderen Services bzw. bietet diese an. Jeder Service ist eine eigenständige leichte, also schnell startende und wenig Platz brauchende, Anwendung welche einst ein Teilmodul eines Monolithen war.

Bei einem Monolithischen System ist es üblich das es eine zentrale Datenbank gibt wo jeglicher Datenverkehr gebündelt wird. Microservices nehmen einen anderen Ansatz und verwalten jeweils eigene Datenbanken. Demnach findet die Kommunikation über APIs statt und nicht über eine geteilte Datenbank.

Dadurch, dass Services eigenständige Datenbanken haben, können diese deutlich stärker spezialisiert werden, was erlaubt verschieden Datenbank Systeme in einem System zu verwenden.

Eine Vielzahl an Microservices, wird generell als verteiltes System bezeichnet. Hierbei wird nochmal in Frontend und Backend Services unterschieden, wobei Ersteres dem Nutzer sichtbar ist und das Zweite nicht, sondern nur intern verwendet werden.

#### Vorteile der Microservices

* Die Anwendung in kleinere handlichere Teilbereiche zu zerteilen hilft dabei den Code leichter zu verstehen und sich besser auf das Wesentliche zu konzentrieren, zu Entwicklern und zu Pflegen.
* Je größer die Anwendung wird, umso eher können ganze Teams auf einzelne Services angesetzt werden und diese Entwickeln, anstatt sich um die gesamte Anwendung kümmern zu müssen.
* Vorausgesetzt die Services sind nicht eng miteinander verzahnt, können Entwicklerteams freier Entscheiden was für Technologien, Frameworks oder sprachen sie benutzten. Vorausgesetzt dies ist vom Business Standpunkt auch möglich.
* Die jeweiligen Bereitstellungen, können durch eine Trennung, vom gesamten System, in kleinere Teil Bereitstellungen verändert werden, was es einfacher macht Änderungen durchzuführen. Tests, bleiben ebenso auf kleinere Service Systeme begrenzt, wodurch sie schneller Veröffentlicht werden können.
* Skalieren der Services wird auch einfacher und Effizienter. Im Gegensatz zum Monolithischen System, wo jeweils das volle System hoch skaliert wird, werden im Micoservice Kontext, nur diejenigen Skaliert welche wirklich benötigt werden, ohne unnötige Ressourcen zu verschwenden.
* Durch die Starke unabhängige Modularität, können Updates je nach Service vollständig unbemerkt durchgeführt werden und es ist nicht mehr notwendig das Gesamte System Abzuschalten. Durch diesen beschleunigten Prozess können Entwickler auch schneller auf Änderungen oder Fehler reagieren.
* Wenn Fehler in Services keine Kettenreaktion herbeiführen, wird eine höhere Robustheit erreicht und das gesamte System kann stabiler laufen.
* Services können, durch die Sprachliche Trennung, einfach in anderen Projekten wiederverwendet werden.

#### Nachteile der Microservices

* Es kann schwieriger sein, Fehler in einem verteilten System zu finden als in einem Lokalen zusammenhängenden System. Dies tritt verstärkt auf, wenn die Services von unterschiedlichen Teams entwickelt werden.
* Da jeder Service seine eigene Datenbank besitzen kann, kann dies zu Datensatz Duplikationen führen.
* Die Unterteilung von Services kann zu weit zu gehen und der Überschuss kann die Nützlichkeit überschreiten.
* Die Notwendigkeit, der Verwendung von Docker oder Kubernetes um das verteilte System zu steuern bzw. Bereitzustellen, kann unerwünscht sein.
* Mit vielen Abhängigkeiten zwischen den Services, kann es herausfordernd werden, ein Ende zu Ende Test durchzuführen.
* Kommunikation zwischen Services, kann sehr teuer werden, wenn sie nicht richtig umgesetzt wird und den entsprechenden Anforderungen entspricht.
* Datenbank Anfragen, können dazu führen das zusätzliche Datenbanken angesprochen werden müssen, welche zu anderen Services gehören. Dies kann zu Problemen führen(wenn dies z.B. mehrfach kaskadierend ist).

## Verteilte Systeme

Was sind Verteilte Systeme? (Klein)

Network

Kommunikation

Unterschiedliche Arten der Kommunikation

Sync

Async

WebSocket

Message Queue

RPC

Hohes Fehler potential (Nicht falls es Fehlschlägt, sondern wenn es Fehlschlägt)

1. Nachricht kommt nicht an Kommunikationsbruch (The network is reliable)
2. Große Last (Bandwith is infinite)
3. Updates
4. Service Abhängigkeiten führen zu kaskadierenden Fehlern
5. Wiederholungen von Anfragen
6. Schnell und separiert Fehlen/ Schnell neustarten

### Die 8 Irrtümer von Verteilten Systemen

# Stand der Technik

# Zusammenfassung und Ausblick

# Glossar

# Literaturverzeichnis

# Ehrenwörtliche Erklärung

|  |  |  |  |
| --- | --- | --- | --- |
| Name: |  | Vorname: |  |
| Matrikel-Nr.: |  | Studiengang: |  |

Hiermit versichere ich, <Vorname, Name>, dass ich die vorliegende <Bachelor- oder Masterarbeit> mit dem Titel <Titel der Abschlussarbeit> selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebene Literatur und Hilfsmittel verwendet habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken ent­nommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ort, Datum Unterschrift

# Stichwortverzeichnis

Abbildung 25