# MASTERARBEIT

**VERGLEICH VON OPEN SOURCE FRAMEWORKS IN JAVA UND .NET ZUR ANBINDUNG AN OPEN SOURCE INFRASTRUKTURKOMPONENTEN EINER MICROSERVICE ARCHITEKTUR.**

Welche Aspekte müssen Infrastrukturkomponenten in einer Microservice Architektur abdecken und wie ist die Anbindung
mit Hilfe der Frameworks in Java und .NET im Vergleich.

ausgeführt an der


am Studiengang
Software Engineering Leadership

Von: Simon Bauer
Personenkennzeichen: 1520030002

Winterthur, am 19.04.2018 .....

Unterschrift

# EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

# DANKSAGUNG

Zunächst bedanke ich mich bei Hermann Woock, der mir bei der Themenfindung dieser Masterarbeit geholfen und mir damit die Möglichkeit zur Einarbeitung in diese brandaktuelle Thematik gegeben hat.

Weiterhin möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung der Masterarbeit unterstützt und diese Arbeit möglich gemacht haben.

Ganz besonderer Dank gilt meinem Betreuer Felix Heppner.
Durch seine Unterstützung der Arbeit bis hin zu inhaltlichen Ratschlägen wurde meine Arbeit sehr bereichert.

Des Weiteren bedanke ich mich bei meiner Familie, allen Kollegen und Freunden, die durch Diskussionen und ihre Meinung zur Masterarbeit beigetragen haben.

# KURZFASSUNG

Diese Masterarbeit untersucht, welche Aspekte zum Betreiben einer Microservice Architektur notwendig sind. Die Realisierung kann mit Hilfe von Open Source Infrastrukturkomponenten erfolgen. Es werden Frameworks in Java und .NET zur Anbindung an diese untersucht und anhand eines Kriterienkatalogs verglichen.

# ABSTRACT

This master thesis examines relevant aspects of a microservice architecture.
These aspects can be implemented by open source infrastructure components.
Frameworks in Java and .NET connecting those components are investigated with a catalog of criterias and compared with each other.

# GLEICHHEITSGRUNDSATZ

Aus Gründen der Lesbarkeit wurde in dieser Arbeit darauf verzichtet, geschlechtsspezifische Formulierungen zu verwenden. Jedoch möchte ich ausdrücklich festhalten, dass die bei Personen verwendeten maskulinen Formen für beide Geschlechter zu verstehen sind.

# INHALTSVERZEICHNIS

# 1 EINFÜHRUNG

*„De duobus malis minus*
*est eligendum.“*
*Cicero, DeOfficis 3.3*

Mit diesem Ratschlag des römischen Philosophen Cicero möchte ich diese Masterarbeit beginnen. Wörtlich übersetzt bedeutet dieser so viel wie *„Von zwei Übeln ist das Kleinere zu wählen“,* und bis heute hat dieser Ausspruch nichts an Gültigkeit verloren. Auch bei Software-Architektur-Entscheidungen bleibt dem erfahrenen Architekten oft nichts anderes übrig, als aus den gebotenen Möglichkeiten jene auszuwählen, die weniger gravierende Auswirkungen bzw. das geringere Risiko aufweisen. Oft ist es auch so, dass die Vorteile des Einen, gleichzeitig auch die Nachtteile des Anderen darstellen.

## 1.1 Motivation

In der vorliegenden Arbeit geht es darum, die zwei etablierten Technologien Java und .Net in Bezug auf die Anbindung an eine Open Source Microservice-Infrastruktur zu untersuchen, damit ein Software Architekt eine geeignete Auswahlentscheidung für deren Einsatz treffen kann, sofern sich herausstellt, dass eine dieser beiden (noch) mit „Übeln“ behaftet ist.

Microservice Architekturen erfreuen sich in jüngster Zeit immer größerer Beliebtheit und stellen eine beachtenswerte Alternative zu anderen Architekturansätzen dar.
Sowohl Netflix als auch Amazon haben mit Ihren hochskalierbaren Web-Plattformen den Praxistest für Microservice Architekturen erbracht.

Aus der Bezeichnung Microservice lässt sich bereits entnehmen, dass es sich um „sehr kleine“ Dienste handelt. Diese sollen isoliert voneinander betrieben werden können und dürfen bzw. sollen sogar gemäß ihrer Aufgabe in der entsprechenden Programmiersprache umgesetzt sein. Somit kann eine Applikation Microservices haben, welche in unterschiedlichen Programmiersprachen implementiert sind.

Ein Microservice alleine macht noch keinen Sinn, sondern Microservices laufen immer in einem Verbund (Ecosystem). Damit eine Applikation mit Microservices betrieben werden kann, benötigt man eine Ausführungsumgebung mit der entsprechenden Infrastruktur, sowie die dazugehörigen Dienste zum Betreiben und Überwachen.

Es lassen sich somit folgende Fragen stellen:

- Welche Aspekte der Infrastruktur werden zur Realisierung einer Microservice Infrastruktur benötigt?
- Welche zur Infrastruktur gehörenden Dienste werden für Java bzw. .Net angeboten?
- Wie gut kann man von einem in Java bzw. .Net entwickelten Microservice aus die Aspekte der Infrastruktur nutzen?

## 1.2 Ziel der Arbeit

Ziel der Arbeit ist es zunächst einen Überblick über benötigte Aspekte der Infrastruktur für die Entwicklung von Microservices zu geben.
Weiterhin sollen die zur Verfügung stehenden Software Module (Frameworks) zur Anbindung bzw. Integration in die Infrastruktur genannt und schließlich anhand geeigneter Vergleichskriterien bewertet werden.
Hiermit wird der aktuelle der Stand der Entwicklung beider Technologien aufgezeigt, um die vorangegangen Fragen, sowie schließlich die Forschungsfrage beantworten zu können.

## 1.3 Vorgehensweise und Gliederung der Masterarbeit

Um das in 1.2 gesteckte Ziel zu erreichen, muss in **Kapitel 2** zunächst geklärt werden, was Microservices ausmacht, also welche Merkmale diese besitzen. Des Weiteren werden die zum Verständnis wichtigen Fachbegriffe erläutert.

**Kapitel 3** liefert in kompakter Form das nötige Hintergrundwissen zur Erstellung von Microservices in den Technologien Java und .NET.

**Kapitel 4** erläutert die Untersuchungsmethodik, welche dieser Arbeit zugrunde liegt.

In **Kapitel 5** erfolgt die Beschreibung der verschiedenen Aspekte der Infrastruktur, die man zur Implementierung von Microservices benötigt.

In **Kapitel 6** wird ein Katalog von Vergleichskriterien beschrieben, um eine Bewertung der angebotenen Software Frameworks vornehmen zu können.

Im Praxisteil in **Kapitel 7** werden die für die Aspekte der Infrastruktur angebotenen Infrastrukturkomponenten und die zur Anbindung verfügbaren Frameworks für Java und .Net betrachtet und bewertet.

In **Kapitel 8** findet eine Gesamtauswertung statt.

**Kapitel 9** fasst die Ergebnisse zusammen, beantwortet die Forschungsfrage und gibt einen Ausblick auf weitere Untersuchungsmöglichkeiten.

# 2 GRUNDLAGEN

In diesem Kapitel sollen die elementaren Grundlagen erläutert werden, die zum Verständnis dienen. Es werden zunächst verschiedene Begriffsdefinition vorgestellt und anschließend die Merkmale beschrieben. Danach werden die Gründe für den Einsatz von Microservices verdeutlicht.

## 2.1 Microservices

### 2.1.1 Begriffsdefinition

(Fowler & Lewis, Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr?, 2015) beschreiben in einem Fachartikel über Microservices, dass es bisher noch keine formale Definition für Microservices gibt. Vielmehr sei es *„[…] ein Ansatz für die Entwicklung einer einzigen Anwendung in Form einer Reihe kleiner Services, die jeweils in einem eigenen Prozess laufen und die durch einfache Mechanismen kommunizieren.*"

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 1) definiert Microservices als *„Ansatz zur Modularisierung von Software."* Und erklärt weiter, dass der Begriff nicht fest definiert sei.

(Horsdal Gammelgaard, 2017, S. 3) definiert einen Microservice folgendermaßen:
*„A microservice is a service with one, and only one, very narrowly focused capability that a remote API exposes to the rest of the system."*
Weiterhin stellt er fest, dass Microservices als Begriff selbst, dazu verwendet wird um ein Architekturstil zu beschreiben, bei dem ein Gesamtsystem aus vielen Microservices besteht.

Somit lässt sich feststellen, dass es bei dem Begriff Microservices zum einen um ein Architekturmuster bzw. einen Architekturstil handelt, aber auch um die Services selbst.

Was den Architekturstil anlangt, so schlagen (Fowler & Lewis, Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr?, 2015) vor, diesen mit dem monolithischen Stil zu vergleichen, bei dem eine Anwendung als eine Einheit gebaut sei. Sie erklären, dass beispielsweise eine Enterprise-Anwendung oft in 3 Hauptschichten gebaut sei. Eine für den Kunden sichtbare Benutzungsschnittstelle, eine Datenbank und eine Anwendung auf der Serverseite. Hierbei sei die serverseitige Anwendung ein Monolith, also eine logisch einzeln ausführbare Datei. Im Falle von Änderungen am System müsse eine vollständig neue Version sowohl gebaut und bereitgestellt werden. Im Gegensatz dazu müssen bei einer Microservice-Architektur nur die Services, bei denen es Änderungen gab, neu gebaut und bereitgestellt werden.

Was den Begriff Service anbetrifft, so liefern alle Definitionen unterschiedliche Beschreibungen. Um ein einheitliches Verständnis zu bekommen, was einen Microservice charakterisiert, so betrachtet man am besten deren Merkmale.

### 2.1.2 Merkmale

#### 2.1.2.1 Ein kleines Team kann eine Handvoll Microservices pflegen

Wie die Bezeichnung „Micro“ charakterisiert, sind Microservices sehr klein.
Als Entwickler denkt man hierbei als erstes an eine messbare Größe wie eine Anzahl Codezeilen, oder eine Größe in Byte. Jedoch bezieht sich diese Aussage mehr auf die Anzahl der Personen, welche einen Microservice entwickeln und pflegen.

(Horsdal Gammelgaard, 2017, S. 5) erklärt hierzu, dass Microservices so klein gebaut sein sollen, dass sie von einem kleinen Team gepflegt werden können. Bezüglich der Größe suggeriere „Micro“ zunächst, dass der Service nur aus ein paar wenigen Zeilen Code bestehen könne. In Wahrheit jedoch beziehe sich die Größe auf die Komplexität des gleichen, die benötigt wird, um diesen zu warten. Als Daumenregel hierfür gibt (Horsdal Gammelgaard, 2017, S. 10) an, dass ein kleines Team von ca. 5 Personen in der Lage sein sollte eine Handvoll Microservices zu pflegen. Zu der Pflege gehören nach seiner Meinung folgende Punkte:

- Entwicklung neuer Funktionalitäten
- Zerlegen von Microservices in weitere Services, welche zu groß geworden sind
- Betreiben des Microservice in Produktion
- Überwachen der Microservices
- Fehlerbehebung
- Alles was sonst noch benötigt wird

Um die Teamgröße noch besser zu beschreiben, kann man laut (Gucer & Narain, 2015, S. 2) die sogenannte „Two Pizza Rule“ anwenden: *„The teams that own the microservice should´t be larger than what two pizzas can feed!“*

#### 2.1.2.2 Ein Microservice ist für eine einzige Fähigkeit verantwortlich

(Horsdal Gammelgaard, 2017, S. 6) teilt die Fähigkeiten in 2 Teile auf. Zum einen hat ein Microservice eine einzige Verantwortlichkeit und zum anderen besitzt dieser eine einzige Fähigkeit.

Mit ersterem ist das „*Single Responsibility Principle*“ gemeint, was auch als ein Entwurfsprinzip aus dem traditionellen objektorientierten Entwurf bekannt ist. Dieses besagt, dass es für eine einzelne Klasse nur einen einzigen Grund für eine Änderung geben sollte. In Bezug auf Microservices bedeutet dies nun das gleiche auf Service - Ebene. Also für einen Service sollte es nur einen einzigen Grund für eine Änderung geben.

Mit dem Zweiten, nämlich den Fähigkeiten, unterscheidet (Horsdal Gammelgaard, 2017, S. 6) folgende Typen innerhalb eines Microservice-Systems:

- Eine **Betriebswirtschaftliche Fähigkeit**, welche einen Beitrag zum Zweck des Gesamtsystems beiträgt. Zu diesen gehören z.B. das Verwalten eines Warenkorbs oder die Preisberechnung innerhalb einer e-Commerce Anwendung. Zum Ermitteln der

betriebswirtschaftlichen Fähigkeiten eines Systems bietet sich eine Zerlegung nach den Methoden des Domain-Driven Designs an.

- Eine **technische Fähigkeit**, welche von anderen Microservices benötigt wird. Technische Fähigkeiten sind nicht der Hauptgrund um ein System in Microservices zu zerlegen, können aber immer dann identifiziert werden, wenn mehrere Microservices diese benötigen.

#### 2.1.2.3 Ein Microservice ist einzeln verteilbar

(Horsdal Gammelgaard, 2017, S. 6) schreibt hierzu, dass ein Microservice individuell verteilbar sein solle. Es solle jederzeit möglich sein, ein sich geänderten Microservice direkt in die Produktionsumgebung zu verteilen, ohne dass andere Teile bzw. Microservices davon betroffen sind. Alle anderen Microservices sollen sowohl während der Verteilung des sich geänderten Microservice als auch danach weiterlaufen. Um dies erreichen zu können, müssten die Schnittstellen von Microservices stets abwärtskompatibel gebaut sein, damit die vorhandenen Microservices durch die neue Version nicht beeinträchtigt werden. Außerdem müsse gewährleistet sein, dass das Zusammenwirken der einzelnen Microservices robust genug ist, und dass Microservices trotzdem zuverlässig weiterarbeiten, auch wenn es zu einzelnen Fehlern bei anderen Microservices kommt.

#### 2.1.2.4 Ein Microservice besteht aus einem oder mehreren Prozessen

In (Horsdal Gammelgaard, 2017, S. 7) wird diesbezüglich erklärt, dass Microservices nur in einem oder mehreren Prozessen laufen dürfen und so unabhängig wie möglich von anderen Microservices sein sollen. Im Vergleich hierzu laufen bei einer monolithischen Applikation mehrere Services in einem Prozess. Dieses Merkmal sei eine grundlegende Voraussetzung, dass Microservices einzeln verteilbar sind. Würden mehrere Microservices in einem Prozess laufen, würde bei der Verteilung eines Microservice der Host Prozess gestoppt werden und somit würden alle anderen darin befindlichen Microservices ebenfalls gestoppt werden.

#### 2.1.2.5 Ein Microservice besitzt seinen eigenen Datenspeicher

Laut (Horsdal Gammelgaard, 2017, S. 8) soll ein Microservice für die Daten, die darin persistiert werden müssen, seinen eigenen Datenspeicher besitzen. Um dies zu erreichen müssten die Daten der einzelnen Microservices entsprechend lose gekoppelt sein. Dies biete in der Konsequenz den Vorteil, dass jeder Microservice eine andere Speichertechnologie verwenden könne, die für seine Aufgabe optimal ist. Je nach Aufgabe könne ein Microservice eine relationale SQL Server Datenbank verwenden, während ein anderer eine schemalose NoSql Server Datenbank oder etwa ein zentralen Cache verwendet.

Weiterhin stellt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 36) fest, dass es in Bezug auf die Konsistenz der Daten notwendig sei, Änderungen innerhalb von einer Transaktion durchzuführen und dass diese Transaktion nur einen Microservice umfassen sollte. Andernfalls sei die Größe des Microservice zu klein gewählt.

#### 2.1.2.6 Ein Microservice ist austauschbar.

(Horsdal Gammelgaard, 2017, S. 10) erklärt, dass es möglich sein solle, ein Microservice innerhalb angemessener Zeit vollständig neu zu schreiben. Ein Team solle demnach in der Lage sein, die Implementierung des Microservice innerhalb kürzester Zeit auszutauschen. Dieses Merkmal stelle eine weitere Einschränkung der Größe des Microservice dar, denn nur, wenn ein Microservice klein genug ist, könne er auch kostengünstig durch einen neuen Microservice ersetzt werden.

#### 2.1.2.7 Microservices werden gegenüber anderen Services isoliert

In Kapitel 2.1.2.4 wurde beschrieben, dass Microservices mindestens in einem eigenen Prozess laufen sollen.

(Newmann, 2015) erklärt hierzu, dass Microservices als isolierte Dienste auf einer „Platform as a Serivce“ (PAAS) oder als eigenständiger Betriebssystemprozess betrieben werden sollen. Es sollte nach Möglichkeit vermieden werden mehrere Microservices auf der gleichen Maschine zu betreiben.

#### 2.1.2.8 Microservices sind durch das Design robust

(Newmann, 2015) erklärt, dass sich durch die Isolierung von Microservices das kaskadieren von Fehlern eingrenzen lässt. Dies würde im Besonderen durch das Trennwand- Entwurfsmuster (bulkhead) gelöst. Sofern eine Komponente fehlerhaft ist und dieser Fehler nicht weiter kaskadiert, kann dieser isoliert werden. Im Gegensatz zu monolithischen Systemen, bei der die Applikation als Ganzes stehen bleibt.

#### 2.1.2.9 Microservices sind skalierbar

Während bei einem monolithischen System, die komplette Applikation skaliert werden muss, lassen sich Microservices individuell skalieren. Dies kann sowohl durch vertikale als auch horizontale Skalierung erfolgen.

Aufgrund ihrer Leichtgewichtigkeit können sie jederzeit auf ein anderes System mit mehr Rechenleistung bzw. Speicher übertragen werden. (*Vertikale Skalierung*).

Und das automatische Duplizieren von Microservices bei erhöhter Auslastung kann aufgrund der Leichtgewichtigkeit schnell erfolgen. (*Horizontale Skalierung*).

#### 2.1.2.10 Microservices sind Technologieunabhängig

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 4) erklärt, dass bei der Umsetzung von Microservices Technologiefreiheit herrsche. Dies ermögliche zum einen das Erproben von neuen Technologien und senke auch das Risiko bei deren Einführung.
Weiterhin wird betont, dass diese Technologiefreiheit eine Option sei, die genutzt werden kann. Bei einer monolithischen Architektur, bestünde immer der Zwang, dass von einer Programmbibliothek nur eine einzige Version genutzt werden kann. Bei einer Microservices Architektur bestünde dieser Zwang nicht, da verschiedene Microservices unterschiedliche

Versionen der gleichen Bibliothek nutzen können. Allerdings benötige dies eine gemeinsame technische Basis.

Die Technologiefreiheit spielt insbesondere eine Rolle bei der Auswahl der Programmiersprache für einen Microservice. Diese darf auch für jeden Microservice individuell gewählt werden. Man bezeichnet diese Eigenschaft auch als „polyglott“.

(Fowler S. S., 2016, S. 130) weist bezüglich der Auswahl der Sprache darauf hin, dass es am wichtigsten sei zu betrachten, inwiefern die Umsetzung in der jeweiligen Sprache Einfluss auf die Performanz und Skalierbarkeit hat und weniger darauf wie modern oder beliebt eine Sprache sei. Manche Sprachen eignen sich für bestimmte Zwecke besser, andere weniger gut, da diese nicht für Performanz optimiert seien.

### 2.1.3 Gründe für den Einsatz von Microservices

#### 2.1.3.1 Stetige Auslieferbarkeit (Continous Delivery) wird leichter möglich

Um stetige Auslieferbarkeit zu ermöglichen, müssen neue Software Stände stetig in ein Software Produkt integriert werden. Um die notwendige Software Qualität hierfür zu erreichen, muss die Software eine Deployment-Pipeline durchlaufen und hierbei die notwendigen Tests bestehen. Dieser Prozess läuft meist automatisch mit Hilfe von Build Servern ab und enthält oft folgende Schritte:

- Automatisches Auschecken aus dem Versionsverwaltungssystem (Checkout)
- Automatisches Bauen (Build)
- Bestehen von Tests (Testing)
- Paketieren (Packaging)
- Verteilung auf verschiedene Umgebungen (Deployment)

(Horsdal Gammelgaard, 2017, S. 11) erklärt hierzu, dass Microservices im Gegensatz zu herkömmlichen monolithischen Applikationen folgende Vorteile bieten in Bezug auf Continous Delivery:

- Microservices können schnell entwickelt und geändert werden.
- Microservices können ausgiebig durch automatisierte Tests geprüft werden.
- Microservices können unabhängig entwickelt werden.
- Microservices können effizient betrieben werden.

Der Einsatz von Microservices ohne Continous Delivery würde bei größeren System schnell sehr aufwendig und kostspielig werden.

## 2.2 Infrastruktur

Das folgende Teilkapitel definiert zunächst den für diese Arbeit wichtigen Begriff der „Infrastruktur“ und anschließend in welchen Ausführungsumgebungen Microservices üblicherweise betrieben werden.

### 2.2.1 Begriffsdefinition

Der Begriff Infrastruktur leitet sich von den lateinischen Begriffen „infra“ (dt. „unterhalb“) und „structura“ (dt. „Zusammenfügung“) ab. Im übertragenen Sinn ist damit der Unterbau gemeint, also die Ausführungsumgebung für die Software und im speziellen für Microservices.

### 2.2.2 Ausführungsumgebung für Microservices

Was die Ausführungsumgebung anbetrifft, so stellen sich nun 2 Fragen:

- **Wie** werden Microservices betrieben?
- **Wo** werden Microservices betrieben?

Prinzipiell können Microservices genauso betrieben werden wie jede andere Software auch. Allerdings sollen die oben genannten Merkmale erfüllt werden. Ein Microservice allein stellt nur eine kleine Funktionalität zur Verfügung. Eine vollwertige Applikation besteht aus vielen Microservices, die im Zusammenspiel agieren. Und diese sollen isoliert voneinander ausgeführt werden und robust sein, falls andere Microservices aufgrund einer Neuverteilung nicht verfügbar sind. Außerdem sollen sie skalierbar sein, d.h. bei höherer Last sollen automatisch mehrere von Ihnen bereitgestellt werden.

#### 2.2.2.1 Ein eigener Rechner (Host)

Bisher war es so, dass man eine Applikation, welche im monolithischen Stil umgesetzt war, auf einem einzigen Rechner ausführen konnte. Diese lief dann in einem Prozess und nutzte eventuell mehrere Threads. Microservices könnte man nun genauso betreiben, allerdings sind diese dann nicht wirklich isoliert voneinander und von ihrer Ausführungsumgebung, also bisher dem Betriebssystem. Diese Tatsache ist auch unter dem Begriff „Noisy Neighbor“ Problem bekannt.

(Scholl, Swanson, & Frenandez, 2016, S. 32) haben für das Noisy Neighbor Problem die folgende Definition: *„The noisy neighbor problem describes a situation where one application takes up or blocks much of the resources that are shared amongst other applications on the same infrastructure. This negatively affects the execution of other processes and applications.“*

(Richardson, 2018, S. 62) beschreibt diesen Ansatz auch als „Multiple Service Instance Per Host Pattern“. Der Vorteil liege im einfachen Deployment der Services und der effizienten Nutzung von Resourcen, aber der Nachteil liege neben dem Noisy Neighbor Problem in der fehlenden Möglichkeit den Resourcenverbrauch der Services zu überwachen. Schlimmstenfalls könne eine Service Instanz den kompletten Speicher bzw. CPU für sich selbst beanspruchen.

#### 2.2.2.2 Mehrere (virtualisierte) Rechner

Der nächste Ansatz wäre nun die Microservices auf mehreren Rechnern oder besser noch auf virtuellen Maschinen zu betreiben.

Bei einer virtuellen Maschine wird die Rechnerarchitektur eines real in Hardware existierenden Rechnersystems oder hypothetischen Rechners nachgebildet.

Hiermit ließe sich das Noisy Neigbor Problem lösen, ist aber dennoch für die Ausführung von Microservices nach (Scholl, Swanson, & Frenandez, 2016, S. 31) nicht praktikabel. Zum einen wäre es nicht ökonomisch für einen einzigen Microservice eine einzige virtuelle Maschine zu betreiben, da eine solche viel zu groß und kostspielig hierfür ist und zum anderen wäre diese viel zu träge. Man bedenke, dass man bei einem Neustart eine virtuelle Maschine vollständig neu booten müsste.

(Richardson, 2018, S. 65) nennt diesen Ansatz „Service Instanz per virtual Machine“ und ergänzt, dass man bei diesem Ansatz virtuelle Maschinen in ausgereifte Cloud Infrastrukturen deployen kann, welche diesen unterstützen. Als Nachteil sieht er jedoch den wenig effizienten Resourcenverbrauch. Aufgrund der Trägheit beim Starten einer virtuellen Maschine, lässt sich zu wenig schnell reagieren um automatisch auf eine „stärkere“ Maschine zu skalieren und daher sind die virtuellen Maschinen oft überdimensioniert, was auch zu höheren Kosten führt. Der Ansatz lohnt sich somit insbesondere für Services, welche prinzipiell eine hohe Rechenlast haben. Als Beispiel nennt (Richardson, 2018, S. 65) den Video Streaming Service von Netflix.

#### 2.2.2.3 Container

Es wird daher ein anderer Ansatz notwendig, der auch mit einer neuen Technologie einhergeht, nämlich mit dem Einsatz von Containern. (Richardson, 2018, S. 66) nennt diesen Ansatz auch „Service Instance per Container“. Als Beispiel führt er Docker und Solaris Zones auf.

Laut (Scholl, Swanson, & Frenandez, 2016, S. 30) hat sich in den letzten 2 Jahren Docker zum De Facto Standard etabliert.

(de la Torre, Modernize existing .NET Applications with Azure Cloud and Windows Containers, 2017) schreibt hierzu *„Docker is becoming the defacto Standard in the Container Industry. Docker is supported by the most prominent vendors in the Linux and Windows ecosystems, including Microsoft. […] In the future […] A Docker Container is the standard unit of deployment for any server based application or service.“* Daher wird im Weiteren die Funktionsweise von Container anhand dieser Technologie erläutert.

Einen Docker Container kann man sich gemäß (sdxCentral, 2016) *„[…] als eine andere Form der Virtualisierung vorstellen“.* Virtuelle Maschinen erlauben es ein Stück der Hardware in verschiedene virtuelle Maschinen (also „virtualisiert“) aufzuteilen, damit die Rechenleistung zwischen den einzelnen Nutzern geteilt werden kann und als ein separater Server oder Maschine erscheint. Docker Container virtualisieren das Betriebssystem. Sie teilen dieses in sogenannte virtualisierte Abteilungen auf um darin Container Applikationen zu betreiben.

Man hat sozusagen auf einem Betriebssystem mehrere Ausführungsumgebungen, welche laut (Scholl, Swanson, & Frenandez, 2016, S. 30) ihre eigene isolierte Sicht der Dinge wie Netzwerk Stack, Dateisystem, Prozesse etc. haben. Jeder Container ist in Unkenntnis darüber, dass es andere Container gibt, welche sich mit ihm selbst das Betriebssystem teilen. Folglich könne man sich Container als gekapselte, individuell verteilbare Komponenten auf dem gleichen Betriebssystem-Kernel vorstellen. **Abbildung 1** verdeutlicht diesen Zusammenhang.

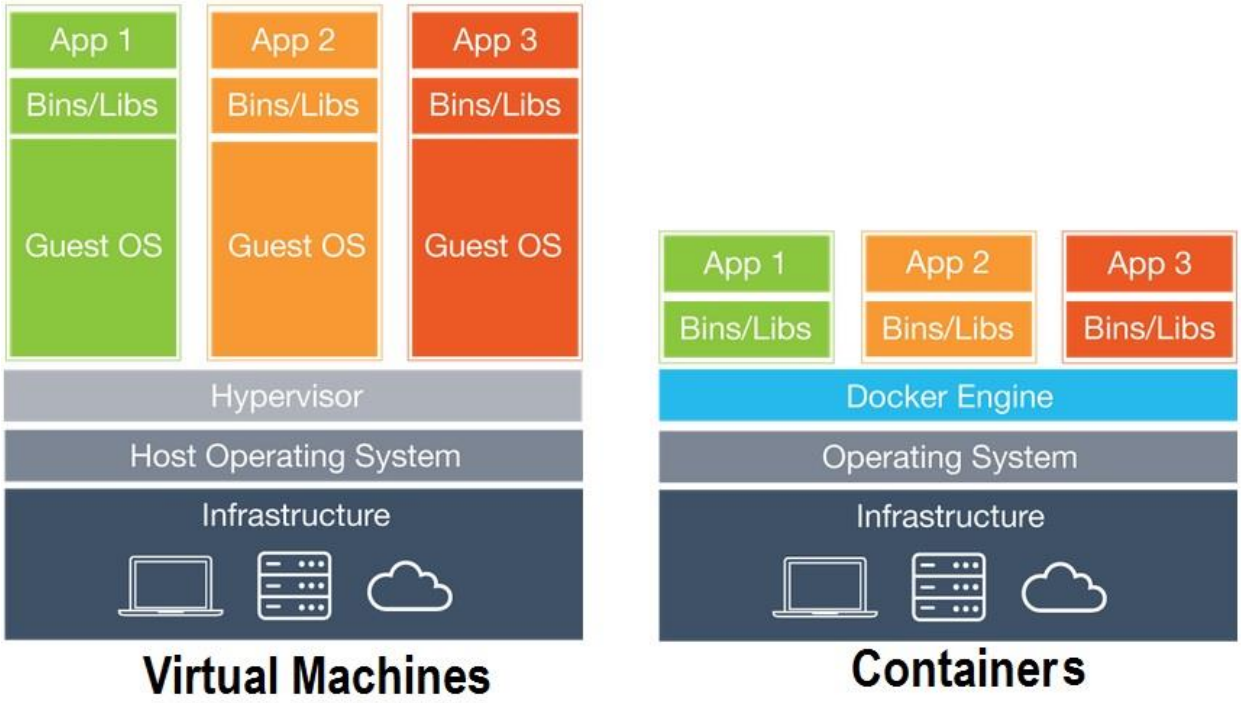



***Abbildung 1*** *Gegenüberstellung Virtuelle Maschine und Container gemäß (TechGlimpse, 2018)*

Auf der linken Seite ist die schematische Aufteilung der Applikationen auf virtuelle Maschinen und auf der rechten Seite die bei Docker Container dargestellt.
Es fällt auf, dass bei Containern das Gast Betriebssystem (Guest OS) wegfällt. Hinzu kommt allerdings zusätzlich die in der Abbildung hellblau dargestellte „Docker Engine“, die für die Orchestrierung der Container verantwortlich ist.

Somit stellen Docker Container die ideale Ausführungsumgebung für Microservices dar, da diese in Containern vollkommen isoliert mit ihrer eigenen Ausführungsumgebung (also Bibliotheken) laufen und ökonomisch in Bezug, sowohl auf die Hardware, als auch die Startup Zeit sind.

Aus Entwicklersicht bieten sie gemäß (Scholl, Swanson, & Frenandez, 2016, S. 30) den Vorteil, dass man seine Applikation mit den dazugehörigen Abhängigkeiten (also Bibliotheken) in einen Container packt und diesen dann auf jeder beliebigen Maschine starten könne, die Container unterstützt. Ebenso seien Container sehr leicht upzudaten bzw. upzugraden und portierbar.

Um schließlich dem Merkmal der Skalierbarkeit noch besser Rechnung zu tragen, bietet es sich an, die Container nicht im unternehmenseigenen Rechenzentrum zu betreiben, welches bezüglich Ressourcen beschränkt ist, sondern in einer Cloud Plattform im Rahmen von Clustern. Beispiele für Cluster Manager sind Docker Swarm, Kubernetes, Marathon oder Mesos. (Richardson, 2018) sieht im Augenblick noch den mangelnden Reifegrad der Technologie im Vergleich zu virtuellen Maschinen als Problem. Insbesondere seien Container weniger sicher als virtuelle Maschinen, da die Container sich den Kernel des Host Betriebssystems teilen.

#### 2.2.2.4 Serverloses Deployment

Bei diesem Ansatz wird Deployment von Services vollständig auf das Vorhandensein eines Servers verzichtet. Es werden nur noch leichtgewichtige Funktionen als Dienst bereitgestellt. Die Cloud Infrastruktur verbirgt sozusagen die Existenz eines Server auf dem diese Funktionen betrieben werden. Beispiele hierfür sind AWS Lamda, Cloud Functions oder Azure Functions. Die Funktionen sind leichtgewichtig und zustandslos. Die Abrechnung erfolgt nach der Anzahl der Aufrufe und der Ausführungsdauer. Letztere ist oft begrenzt beispielsweise auf 300 Sekunden.

# 3 MICROSERVICES IN DEN EINZELNEN TECHNOLOGIEN

Dieses Kapitel gibt eine kurze Einführung in Java und .NET und wie mit diesen Technologien Microservices entwickelt werden können. Es beantwortet die Frage, welche Basis Frameworks zur Entwicklung von Web-Services Anwendung finden. Anhang A - und Anhang B - zeigen dies Anhand eines einfachen Web-Service, welcher einen Endpunkt bereitstellt, der eine http Anfrage mit „Hello World“ beantwortet. Die betrachteten Beispiele sind losgelöst von der Ausführungsumgebung (siehe Kapitel 2.2.2) bzw. dem Microservice Ökosystem.

Zunächst muss klargestellt werden, dass es sich bei Java um eine Programmiersprache handelt, während .NET ein Framework zum Entwickeln von Applikationen von Microsoft ist. Das .NET Framework unterstützt mehrere Sprachen. Die wichtigsten sind C#, Visual Basic.NET und F#.

## 3.1 Java

Mit Java ist zum einen eine objektorientierte Programmiersprache, welche auf der gleichnamigen Softwaretechnik aufbaut und andererseits eine Technologie, mit welcher eine Sammlung von Softwarespezifikationen bezeichnet wird (Java-Platform).

### 3.1.1 Java-Programmiersprache

Die ursprünglich von Sun Microsystems entwickelte Programmiersprache Java existiert seit 1995 und wurde von Anfang an für die Öffentlichkeit freigegeben. Im Jahr 2010 wurde Sun Microsystems von Oracle übernommen und damit auch die Programmiersprache, sowie die damit verbundene Technologie.

### 3.1.2 Java Technologie

Zur Technologie gehören folgende Bestandteile:

- Die Programmiersprache Java
- Das Java Development Kit (enthält grundlegende Teile wie Übersetzer und Bibliotheken)
- Die Java Laufzeitumgebung (eine standardisierte Software-Plattform)

Hauptziel der Technologie ist es, ein Programm auf unterschiedlichen Computersystemen zu betreiben. Ein in Java geschriebenes Programm wird zunächst in Bytecode compiliert und dann von der Laufzeitumgebung (Java Runtime Environment, JRE) beim Starten des Programms in die jeweilige Maschinensprache übersetzt. Die Java-Laufzeitumgebung existiert für weit verbreitete Betriebssysteme wie:

- Microsoft Windows
- Linux
- Solaris
- Mac OS X
- AIX

### 3.1.3 Java Laufzeitumgebung

Die Java Laufzeitumgebung stellt eine Softwareplattform dar, welche es ermöglicht, Programme weitgehend unabhängig vom darunterliegenden Betriebssystem auszuführen. Zum einen definiert sie die Anwendungsprogrammierschnittstellen (APIs) eindeutig und maschinenunabhängig und zum anderen enthält sie die sogenannte Java Virtual Machine (JVM), welche für die Ausführung des Java-Bytecodes verantwortlich ist.
Es stehen folgende Java-Plattformen zur Verfügung:

- Java Platform Card (Zur Ausführung von Java-Card-Applets auf Chipkarten)
- Java Platform, Micro Edition (Zur Ausführung auf Embedded Geräten, Mobiltelefone, PDAs)
- Java Platform, Standard Edition (Zur Ausführung auf PCs)
- Java Platform, Enterprise Edition (Zur transaktionsbasierten Ausführung mehrschichtiger Unternehms- und Web-Anwendungen)

### 3.1.4 Java Versionen

Die Programmiersprache Java hat seit ihrer Entstehung einige Veränderungen und Erweiterungen durchlaufen. Die aktuelle Version trägt die Bezeichnung Java SE 9 und wurde 2017 veröffentlicht. **Tabelle 1** gibt eine Übersicht über die seit der Entstehung verfügbaren Versionen.

| Release | Year |
|---|---|
| JDK Beta | 1994 |
| JDK 1.0 | 1996 |
| JDK 1.1 | 1997 |
| J2SE 1.2 | 1998 |
| J2SE 1.3 | 2000 |
| J2SE 1.4 | 2002 |
| J2SE 5.0 | 2004 |
| Java SE 6 | 2006 |
| Java SE 7 | 2011 |
| Java SE 8 | 2014 |
| Java SE 9 | 2017 |

***Tabelle 1*** *Java Versionen aus (Wikipedia, Java Version History, 2017)*

### 3.1.5 Java Frameworks zur Entwicklung von Microservices

#### 3.1.5.1 Spring

Zur Entwicklung von Java Anwendung im Allgemeinen eignet sich das Spring Framework. Dieses ist gemäß modular aufgebaut und enthält viele wichtige Erweiterungen. Hierzu zählen beispielsweise:

- Spring Boot (für die Entwicklung von Spring-Anwendungen, die per Konvention vor Konfiguration ohne XML-Konfiguration auskommen)

- Spring AMQP (für den Zugriff auf AMQP-basierte Message Oriented Middleware)
- Spring Data (für den Zugriff auf verschiedene relationale und NoSql-Datenbanken)
- Spring MVC (für die Erstellung von Webanwendungen nach dem MVC Entwurfsmuster)
- Spring Web Services (zur Erstellung von Contract-First Webservices)

Hervorzuheben ist beim Spring Framework die Existenz von Inversion of Control Containern. Größte Bedeutung für die Entwicklung von Microservices hat das Spring MVC Framework, welches zur Entwicklung von Web Seiten und Restful Services dient.

#### 3.1.5.2 Spring Cloud

Spring Cloud[1] basiert auf dem bereits genannten Framework Spring Boot und dient zur Entwicklung von Anwendungen in verteilten Systemen. Es enthält beispielsweise Pakete für:

- Konfigurationsmanagement
- Diensterkennung
- Selbstabschaltung
- Routing
- Dienst-Dienst Aufrufe
- Lastverteilung

Die Integration der meisten Features erfolgt nach einem deklarativen Ansatz, indem Klassen Annotationen[2] hinzugefügt werden. Das folgende Beispiel zeigt die Applikationsklasse, welche die gesamte Applikation durch die Annotationen `@SpringBootApplikation` und `@EnableDiscovery` sowohl mit Spring Boot- als auch mit Diensterkennungsfähigkeiten ausstattet.

```
@SpringBootApplikation
@EnableDiscoveryClient
public class Application {
        public static void main (String[] args) {
                SpringApplication.run(Application.class, args);
        }
}
```

***Abbildung 2*** *Klasse Application mit Spring Cloud Integration gemäß (Pivotal Software, 2017)*

### 3.1.6 Spring Tool Suite als Entwicklungsumgebung

Spring Tool Suite[3] (STS) ist eine auf Eclipse basierende Entwicklungsumgebung zum Entwickeln von Java Applikationen für das Spring Framework. STS steht für folgende Betriebssysteme zur Verfügung:

- Windows
- Mac

[1] Siehe http://projects.spring.io/spring-cloud/

[2] In der .NET Welt als Attribute bezeichnet!

[3] Siehe https://spring.io/tools/sts

- Linux

Gemäß (MarketPlace, 2017) existiert STS seit November 2014. Die neueste Version trägt die Nummer 3.9.1. Zu den besonderen Features gehören unter anderem:

- Erzeugen von Spring Boot Projekten innerhalb kürzester Zeit.
- Unterstützung von Spring Java-Config basierten Applikationen.
- Code Completion, Content-Assist, Code Validierung und quick fixes für Spring Applikationen.
- Spring Boot Dashboard, welches als Microservice Entwicklungs-Center dient.
- IDE Integration für Cloud Foundry, einschließlich Debug Möglichkeiten in der Cloud.
- Ultra schnelle Suche während der Code Eingabe in der Arbeitsumgebung.

## 3.2 .Net

### 3.2.1 Das .NET Framework

Das.NET Framework in seiner ersten Version gibt es seit 2000 und wurde von Microsoft entwickelt, um die in die Jahre gekommen Konzepte von Windows durch neue zu ersetzen. Wichtigstes Bedürfnis war hierbei die Plattformunabhängigkeit zu erreichen. Seit Mai 2017 existiert die neueste Version 4.7, welche als eigenständige Installation veröffentlicht wurde.

### 3.2.2 Laufzeitumgebung und Sprachen

Eng verbunden mit dem Framework ist die Laufzeitumgebung für .NET Applikationen, die Common Language Runtime (CLR), welche einen JIT-Compiler für den standardisierten Zwischencode Common Intermediate Language (CIL) enthält.

Durch die Umsetzung des Common-Language-Infrastructure-Standards (CLI) ist es möglich Programme in mehreren Programmiersprachen mit einem Framework zu entwickeln.
Ein in einer beliebigen Programmiersprache umgesetztes Programm wird zunächst in CIL übersetzt, dann in der Laufzeitumgebung vom JIT-Compiler übersetzt und dann in der CLR zur Ausführung gebracht. Als weit verbreitetste Hoch-Sprache hat sich C# etabliert.

### 3.2.3 .NET Framework Versionen

Wie **Abbildung 3** zeigt, hat sich das .NET Framework über die Jahre immer weiterentwickelt und es sind weitere Module hinzugekommen.

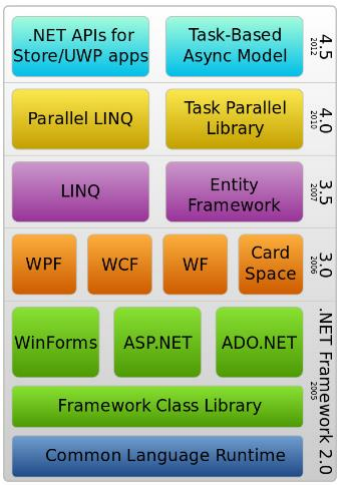



***Abbildung 3*** *Die Microsoft .NET Framework Hierarchie mit Erweiterungen aus (Wikipedia, NetFramework, 2017)*

Die ursprüngliche Version 2.0 enthält die Common Language Runtime und die Framework Class Library, sowie die drei spezifischen Frameworks Winforms, ASP.NET, und ADO.NET. WinForms wurde und wird noch zur Entwicklung von Windows Applikationen verwendet. ASP.NET findet Anwendung zur Entwicklung von Web-Applikationen, während ADO.NET die nötigen Klassen zum Zugriff auf Datenbanken bereitstellt.

Über die Jahre sind auch verschiedene weitere Ableger des .NET Frameworks für verschiedene Zielplattformen entstanden. Diese sind beispielsweise:

- .NET Compact Framework (zur Entwicklung von Handhelds und Mobiltelefone)
- .NET Micro Framework (zur Entwicklung von Embedded-Geräten)
- Silverlight (zur Entwicklung von reichhaltigen Internet Anwendungen)
- XAMARIN (zur Cross Plattform Entwicklung von Mobil Telefonen)

### 3.2.4 .NET Core

Weil das .NET Framework immer größer und schwergewichtiger geworden ist, hat Microsoft 2014 eine Teilmenge des Frameworks unter dem Namen .NET Core veröffentlicht.
Zum einen wurden die im ursprünglichen Framework (Full .NET Framework) Komponenten überarbeitet und zum anderen wurde der zur Verfügung stehende Quellcode auf Github veröffentlicht. Die veröffentlichten Komponenten umfassen unter anderem weitere Werkzeuge für die Softwareentwicklung wie den C# und VB-Compiler, Microsoft Roslyn, sowie den kompletten ASP.NET Webstack. Letzterer wird insbesondere für die Entwicklung von Microservices benötigt.

.NET Core ist sehr schlank, läuft unter Windows, Mac bzw. Linux und ist für die Verarbeitung von Web Arbeitsaufgaben ausgelegt, während das Full .NET Framework ausschließlich unter Windows läuft. Weiterhin erlaubt .NET Core die Beteiligung der .NET Community, welche die Entwicklung von Zusatzpaketen zu .NET Core mit übernommen hat.
Zusatzpakete lassen sich je nach Bedarf zu dem sehr schlanken .NET Core leicht hinzufügen. Für die Entwicklung von Microservices ist es daher von Bedeutung zu wissen, ob für bestimmte Aufgaben Pakete zur Verfügung stehen. Es genügt nicht, sich blindlings auf die vom gewohnten

.NET Full Framework bekannten und vorhandenen Klassen zu verlassen, sondern es bedarf immer einer vorherigen Abklärung, ob entsprechende Pakete für .NET Core vorhanden sind, sofern man dies einsetzen möchte.

### 3.2.5 .NET Framework und Docker

Im Internet stehen prinzipiell zwei verschiedene Docker Images zum hosten von .NET Applikationen zur Verfügung[4]:

- Windows Server Core Base Image
- Windows Nano Server Base Image

#### 3.2.5.1 Windows Server Core Base Image

Das Windows Server Core Base Image enthält das schwergewichtige Full .NET Framework, das man benötigt, wenn man nicht auf das schlankere .NET Core Framework ausweichen kann. Es enthält alle Features des .NET Frameworks, ist dafür aber sehr umfangreich. Es läuft nur unter den neuesten Windows Versionen als Windows (Docker) Container[5].

#### 3.2.5.2 Windows Nano Server Base Image

Das Windows Nano Server Base Image enthält das schlanke .NET Core Framework.
Es sind darin nicht alle Features des .NET Frameworks enthalten, dafür ist es aber sehr performant. Es läuft sowohl unter Linux, als auch unter Windows.

### 3.2.6 Windows Container versus Linux Container

Windows Container existieren seit 2016 im Gegensatz zu den schon wesentlich älteren Linux Containern.[6]

Windows Container benötigen Windows als Hostbetriebssystem.
Linux Container benötigen Linux als Hostbetriebssystem.

Damit Linux Container auch unter Windows zur Verfügung stehen, müssen diese in einer virtuellen Maschine - mit installiertem Linux Betriebssystem - betrieben werden, während Windows Container im Gegensatz hierzu ohne virtuelle Maschine in einem eigenen Prozess direkt unter Windows laufen.

Es ist unbedingt zu beachten, dass Frameworks, welche auf dem Full .NET Framework aufbauen ein Windows Betriebssystem benötigen und daher nur in Windows Containern betrieben werden können, während Frameworks, welche auf .NET Core aufbauen sowohl in

[4] Siehe hierzu https://hub.docker.com/

[5] Im Folgenden nur noch als Windows Container bezeichnet. Siehe auch Kapitel 3.2.6

[6] Weitere Details zu Windows Container siehe (Container, 2016)

Windows als auch Linux Containern betrieben werden können,[7] da .NET Core plattformunabhängig ist.

Dies bedeutet in der Konsequenz, dass auch nur Container-Cluster mit Orchestrierungswerkzeugen verwendet werden können, welche virtuelle Maschinen mit Windows unterstützen.

Gemäß (de la Torre, Modernize existing .NET Applications with Azure Cloud and Windows Containers, 2017, S. 39) ist Service Fabric, seit Mai 2017, das erste Orchestrierungswerkzeug, welches Windows Container unterstützt, während andere Orchestrierungswerkzeuge, wie Kubernetes, DC/OS und Docker Swarm, einen höheren Reifegrad für Linux Container aufweisen.

### 3.2.7 .NET Standard

(Hoffman, 2017, S. 2) erklärt hierzu, dass Entwickler vor der Entwicklung von .NET Core mit dem Konzept von Portierbaren Klassen Bibliotheken (Portable Class Libraries) vertraut waren, mit deren Hilfe sie in der Lage waren, Schnittmengen ihrer Software in Pakete für verschiedene Zielarchitekturen und Plattformen auszulagern (z.B. ein Paket für Windows Phone 8 und ein Paket für ASP.NET).

Dies führte dazu, dass eine Vielfalt von Paketen entstanden, welche nun für unterschiedliche Zielplattformen verfügbar sind. Um diese entstandene Vielfalt an Paketen wieder einzudämmen und eine Vereinheitlichung der Schnittstellen herbeizuführen, wurde der .NET Plattform Standard (bzw. .NET Standard) eingeführt.

Man kann sich daher .NET Standard als eine Sammlung von Schnittstellen vorstellen, die nun entweder vom traditionellen .NET Framework oder von den .NET Core Bibliotheken implementiert wird.
Um nun herauszufinden, welches NuGet Package man für eine bestimmte .NET Version benötigt, genügt es zu wissen, welche Version des .NET Standard unterstützt wird. Falls die Version nicht konform zu einer bestimmten Version des .NET Standards implementiert wurde, ist diese auch nicht kompatibel zu .NET Core.

Die folgende Tabelle zeigt eine Gegenüberstellung der Versionen zwischen .NET Standard, .NET Core und dem Full .NET Framework.

| .NET Standard | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 2.0 |
|---|---|---|---|---|---|---|---|---|
| **.NET Core** | | | | | | | 1.1 | 2.0 |
| **.Full .NET Framework** | | 4.5 | 4.5.1 | 4.6 | 4.6.1 | 4.6.2 | vNext | 4.6.2 |

***Tabelle 2*** *Gegenüberstellung .NET Versionen aus (Hoffman, 2017, S. 3)*

[7] Siehe hierzu (de la Torre, Wagner, & Rousos, 2018, S. 17)

### 3.2.8 ASP.NET Core

ASP.NET Core ist das leichtgewichtige Framework zum Entwickeln von Web-Applikationen und Microservices und besteht laut (Hoffman, 2017, S. 3) aus einer Sammlung von kleinen, modularen Komponenten. Der Quellcode von ASP.NET steht der Entwicklergemeinde offen zur Verfügung. Innerhalb von ASP.NET findet man APIs für Routing, Json-Serialisierung, sowie Komponenten zur Erstellung von Web Applikationen gemäß dem MVC Pattern (Model View Controller).

### 3.2.9 Visual Studio als Entwicklungsumgebung

Visual Studio ist die von Microsoft angebotene integrierte Entwicklungsumgebung für verschiedene Hochsprachen und erschien erstmalig im Jahr 1997. Der ursprüngliche Verwendungszweck war sowohl das Programmieren von Win32/Win64 Programmen als auch Applikationen für das .Net Framework. Über die Jahre sind immer weitere Applikationstypen hinzugekommen wie:

- Windows-Apps
- Dynamische Webseiten bzw. Web-Services für das Internet
- Mobile Apps für Windows Phone, Android, iOS mit Xamarin
- Azure Services

Schwerpunktmäßig werden die Programmiersprachen C# und F# sowie Html, CSS und Json unterstützt.

Neben zahlreichen Code-Bearbeitungsfeatures, enthält Visual Studio einen sehr ausgereiften integrierten Debugger zur Fehlersuche von Programmen.
Die aktuelle Version 2017 ist in der finalen Version am 7. März 2017 erschienen.

Die wichtigste Neuerung[8] in Bezug auf Microservices ist neben der Möglichkeit Applikationen direkt in Container[9] zu deployen und zu debuggen, auch diese direkt aus der Entwicklungsumgebung heraus in Containerverzeichnissen wie Docker Hub bzw. Azure Container Registry zu deployen.

[8] Siehe hierzu https://www.visualstudio.com/de-de/news/releasenotes/vs2017-relnotes

[9] Siehe hierzu https://docs.microsoft.com/en-us/aspnet/core/publishing/visual-studio-tools-for-docker

# 4 UNTERSUCHUNGSMETHODIK

In diesem Kapitel wird grundsätzlich beschrieben, wie verfahren wurde, um die für Microservice Architekturen relevanten Aspekte zu finden.
Um eine Abgrenzung von relevanten und nicht relevanten Aspekten vornehmen zu können wird eine Untersuchungsrelevanz definiert.

## 4.1 Verfahren zum Finden von Aspekten

Um die erwähnten Aspekte zu finden, muss zunächst Recherchearbeit geleistet werden.
Hierzu wird auf die aktuelle Fachliteratur zum Thema Microservices zurückgegriffen.
Diese sind:

- (Hoffman, 2017)
- (Horsdal Gammelgaard, 2017)
- (Newmann, 2015)
- (Richardson, 2018)
- (Vitz, 2018)
- (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016)
- (Fowler S. S., 2016)

Eine weitere Quelle sind Publikationen von Komponentenherstellen (z.B. Nginx), Online Video Tutorials (z.B. Pluralsight, Udemy, edx).

## 4.2 Untersuchungsrelevanz

Gemäß den Merkmalen sollen Microservices technologieunabhängig sein. Das heißt, es sollte keine Rolle spielen, in welcher Technologie ein Microservice umgesetzt wird. In Bezug auf die Infrastruktur müssen Microservices sich jedoch an bestimmte einheitliche Standards halten, damit eine Vielzahl von diesen betrieben werden kann. Genau an dieser Stelle kann es nun sein, dass diese Freiheit eingeschränkt wird, weil die einzelnen Technologien unterschiedliche Möglichkeiten bieten bzw. unterschiedlich ausgereift sind. Es werden daher nur solche Aspekte behandelt, bei denen es erforderlich ist, einen bestimmten Standard einzuhalten, um das Zusammenspiel von Microservices der verschiedenen Technologien zur Anbindung an die Infrastruktur zu ermöglichen. Außerdem sollen die Aspekte die Merkmale unterstützen und nicht im Widerspruch zu ihnen stehen. Am Ende jedes einzelnen Kapitels zu den Aspekten erfolgt eine Begründung, welche Untersuchungsrelevanz diese für die vorliegende Arbeit im Praxisteil besitzen.

# 5 ASPEKTE DER INFRASTRUKTUR

Im folgenden Kapitel werden die gefundenen Aspekte für die Infrastruktur für Microservices erläutert. Jedes Teilkapitel enthält eine kurze Erklärung, welche Merkmale unterstützt werden, warum diese benötigt wird und eine prinzipielle Erklärung, wie die Funktionalität umgesetzt ist.

## 5.1 Kommunikation

Microservices müssen mit einander kommunizieren um eine gemeinsame Logik umzusetzen. Dieses Teilkapitel betrachte die hierfür notwendigen Kommunikationsprotokolle, Kollaborationsstile und Schnittstellen.

### 5.1.1 Kommunikationsprotokoll

Für die Kommunikation zwischen Microservices bietet sich Http bzw. Https als Kommunikationsprotokoll in Verbindung mit REST an.

REST steht für Representational State Transfer.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 179) erklärt hierzu, dass dies *„[…] der Begriff für die grundlegenden Ansätze des WWW“* sei und die folgenden Charakteristiken habe:

- Es gibt eine Vielzahl von Ressourcen, die über URIs identifiziert werden können. URI steht für Uniform Ressourcen Identifier und identifiziert eine Ressource global eindeutig. URLs sind praktisch dasselbe wie URIs.
- Die Ressourcen können über eine feste Menge von Methoden manipuliert werden. Bei HTTP sind das beispielsweise GET für die Abfrage der Ressource, PUT für das Ablegen einer Ressource und DELETE für das Löschen. Die Semantik der Methoden ist fest definiert.
- Für Ressourcen kann es unterschiedliche Repräsentationen geben – beispielsweise als PDF oder HTML. HTTP unterstützt die sogenannte Content Negotiation durch den Accept-Header. So kann der Client festlegen, welche Datenrepräsentationen er verarbeiten kann. Die Content Negotiation ermöglicht es beispielsweise, Ressourcen für Menschen lesbar anzuzeigen und unter derselben URL auch maschinenlesbar auszugeben. Dazu kann der Client über einen Accept-Header mitteilen, dass er für Menschen nur HTML akzeptiert, während der andere nur JSON akzeptiert.
- Beziehungen zwischen Ressourcen können durch Links ausgedrückt werden. Links können auf andere Microservices verweisen, sodass eine Integration der Logik verschiedener Microservices möglich wird.
- Der Server in einem REST-System sollte zustandslos sein. Genau deswegen implementiert HTTP ein zustandsloses Protokoll.

Http mit Rest bietet als Protokoll folgende Vorteile:

- Eine Http Schnittstelle kann sehr einfach mit einem Cache (Web-Cache) erweitert werden.
- Es können die üblichen http-Load-Balancer verwendet werden.
- Es bietet einfache und nützliche Mechanismen für Security mit Hilfe von http-Headern.
- Zum Datenaustausch können statt Html auch Json bzw. Xml verwendet werden.

Http bzw. Https hat sich als Defacto Standard etabliert, während RPC (Remote Procedure Calls) ungeeignet ist, da dieses eine enge Kopplung zwischen Sender und Empfänger benötigt.

Ferner erklärt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 183), dass SOAP ebenfalls ungeeignet sei, da SOAP mit seinen vielen Erweiterungen aus dem WS*-Umfeld zu einem komplexen Protokoll-Stack führen würde. Die Koordination zwischen den Microservices würde dadurch problematisch und es benötige eine Koordinationsschicht. Bei Änderungen dieser wären alle Microservices betroffen, wodurch ein Monolith entstünde. Ferner wiederspräche dies dem Microservice-Gedanken vom unabhängigen Deployment.

Untersuchungsrelevanz:
Kommunikationsprotokolle sind untersuchungsrelevant, da Microservices zur Kommunikation sich an einen gemeinsamen Standard halten müssen, damit das Zusammenspiel funktioniert. Die einzelnen Technologien müssen das gemeinsame Kommunikationsprotokoll unterstützen.

### 5.1.2 Kollaborationsstile

Während Protokolle die technische Umsetzung der Kommunikation beschreiben, geht es bei den Kollaborationsstilen um die Mechanismen bzw. Art der Kommunikation.

(Horsdal Gammelgaard, 2017, S. 79) nennt hierfür Abfragen, Befehle, Ereignisse und Nachrichten.

#### 5.1.2.1 Abfrage (Queries)

Queries werden benötigt, wenn ein Microservice Informationen von einem anderen Microservice benötigt. Die Umsetzung von Queries erfolgt synchron über Http Anfragen (GET).
Mit GET kann der Sender synchron ein Objekt beim Empfänger abrufen.

#### 5.1.2.2 Befehle (Commands)

Befehle werden benötigt, wenn ein Microservice eine Aktion auf einem anderen Microservice ausführen möchte.

Die Umsetzung von Befehlen erfolgt synchron über Http Anfragen (POST bzw. PUT).
Synchron bedeutet in diesem Zusammenhang, dass der Sender in der Ausführung blockiert und auf die Antwort des Empfängers warten muss. Falls der Empfänger nicht antwortet muss der Sender entsprechende Fehlermechanismen[10] implementieren. Mit POST überträgt der Sender

[10] Siehe Kapitel 5.7

ein neues Objekt, welches beim Empfänger erzeugt werden soll. Mit PUT kann der Sender ein vorhandenes Objekt beim Empfänger aktualisieren.

#### 5.1.2.3 Ereignisse (Events)

Die Kommunikation über Events wird benötigt, wenn ein Microservice auf ein Ereignis in einem anderen Microservice reagieren muss. Anders als bei Befehlen und Abfragen erfolgt die Kommunikation asynchron. Asynchron bedeutet in diesem Zusammenhang, dass der Sender in der Ausführung nicht blockiert und aktiv auf die Antwort des Empfängers warten muss.
Man würde an dieser Stelle nun eine Art „Publish-Subscribe Mechanismus“ erwarten. Dies würde jedoch wieder zu einer engen Kopplung führen, was möglichst zu vermeiden ist.

(Horsdal Gammelgaard, 2017, S. 85) beschreibt hier einen anderen Ansatz. Ein Microservice, der Ereignisse zur Verfügung stellen möchte, ruft erstens nicht direkt die Microservices auf, welche sich für das Ereignis interessieren, sondern Microservices, welche an Ereignissen anderer Microservices interessiert sind; fragen diesen nach einem „Polling-Mechanismus“ zyklisch ab. Dies setzt wiederum entsprechende Fehlerbehandlungsmechanismen voraus.[11]
Der Microservice, welcher das Ereignis zur Verfügung stellt, bietet hierzu eine Event Schnittstelle an. Treten Ereignisse auf, werden diese intern beim Microservice, bei dem das Ereignis auftrat sequentiell gespeichert. Es bietet sich der Einsatz von sogenannten Event Stores an. Die an dem Ereignis interessierten Microservices können über Parameter in der Abfrage steuern, welche Ereignisse zurückgeliefert werden sollen.
Im konkreten Fall bedeutet das, dass ein Microservice beispielsweise immer die Ereignisse mit einer bestimmten Nummer oder Zeitstempel abfragt, welche er noch nicht erhalten hat.

#### 5.1.2.4 Messages (Nachrichten)

Eine weitere Art der Kommunikation stellen Nachrichten mit Nachrichtenwarteschlangen (Message Queues) dar.

Hierüber lässt sich die Kommunikation von Sender und Empfänger vollständig entkoppeln. Der Sender schickt seine Nachrichten an eine Nachrichtenwarteschlange, welche dort sequentiell nach dem First In First Out Prinzip verarbeitet werden.
Je nach Implementierung der Nachrichtenwarteschlange kann sich ein Empfänger dort nach dem Publish-Subscribe Mechanismus registrieren bzw. ruft die Nachrichtenwarteschlange zyklisch nach neuen Nachrichten ab. Im Gegensatz zu Ereignissen registriert sich ein Empfänger, also bei der Warteschlange und nicht bei dem Sender der Nachricht.
Für die Warteschlange kann eine externe Technologie herangezogen werden. Damit kann der Sender von der Speicherung der Nachrichten entlastet werden, setzt aber voraus, dass auch alle Empfänger sich mit der Technologie verbinden können.
Nachrichtensysteme bieten weiterhin den Vorteil, dass diese verteilte Transaktionen ermöglichen.

[11] Siehe Kapitel 5.7

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 184) erklärt hierzu zunächst, dass mit dem herkömmlichen Ansatz, bei dem sich Microservices gegenseitig aufrufen, die Garantie von Transaktionen schwierig zu gewährleisten sei, da alle Microservices an der Transaktion teilnehmen müssen und erst Änderungen schreiben dürften, wenn alle Microservices in der Transaktion fehlerfrei durchgelaufen sind. Dies habe zur Folge, dass alle beteiligten Microservices mit dem Abschließen der Änderung (Commit) eventuell längerfristig warten müssen. Insbesondere wenn im Netzwerk Teilnehmer ausfallen. Die verteilte Transaktion würde dann sehr lange offenbleiben oder gar nicht geschlossen werden. Außerdem wäre der Sender der verteilten Transaktion längere Zeit blockiert.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 184) erklärt weiter, dass in einem Messaging System mit Transaktionen anders umgegangen werden könne. Das Schicken und Empfangen von Nachrichten sei Teil einer Transaktion, ähnlich wie z.B. das Schreiben oder Lesen von der Datenbank. Würde beim Verarbeiten der Nachricht ein Fehler auftreten, müssten eben die Nachrichten storniert und die Datenbankänderung rückgängig gemacht werden. Für den Fall des Erfolgs finden alle Aktionen statt. Alle Empfänger können ebenfalls transaktional abgesichert sein. Hiermit würde dann die Bearbeitung einer ausgehenden Nachricht denselben transaktionalen Garantien unterliegen.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 185) behauptet, dass Transaktionen ohne globale Koordination mit Nachrichten umgesetzt werden können, sofern die zugrundeliegende Nachrichten-Technologie das transaktionale Verschicken gewährleistet. Dies setze jedoch voraus, dass die Nachrichten nur gültige verarbeitbare Daten enthalten.

Untersuchungsrelevanz:
Kollaborationsstile sind untersuchungsrelevant, da die Microservices sich an einen gemeinsamen Standard zur Kommunikation mit den angebotenen Diensten wie Event Stores und Message Queues halten müssen. Abfragen und Befehle sind Teil des Rest Protokolls und müssen nicht untersucht werden.
Es wird insbesondere untersucht, welche Frameworks der Technologien Zugriff auf Event Stores und Message Queues bieten.

### 5.1.3 Schnittstellen

Microservices kommunizieren über Schnittstellen.

Hierbei können nach (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 190) interne und externe Schnittstellen unterschieden werden.

#### 5.1.3.1 Interne Schnittstellen

Interne Schnittstellen sind solche, die von Microservices innerhalb der Microservice Applikation verwendet werden. Wird die Schnittstelle nur von Mitgliedern des gleichen Microservice Teams genutzt, ermöglicht dies eine leichtere Änderbarkeit dieser, da die Teammitglieder enger zusammenarbeiten können. Schwieriger wird es jedoch, falls Schnittstellen zu Microservices

anderer Teams geändert werden müssen. Diese Änderungen müssen zum einen abgesprochen werden und können zum anderen nur durch geeignete Tests abgedeckt werden.[12]

Für interne Schnittstellen empfiehlt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 191) mehrere Versionen der gleichen Schnittstelle nur zum Entkoppeln von Deployments zuzulassen um die Komplexität des Microservice-Systems zu reduzieren. Nach dem Deployment einer neuen Schnittstelle dürfen alle abhängigen Microservices noch die alte Schnittstelle weiter nutzen, sollen aber umgehend auf die neue umstellen. Sobald alle umgestellt hätten, müsse die alte Schnittstelle entfernt werden.

#### 5.1.3.2 Externe Schnittstellen

Externe Schnittstellen sind solche, welche das Microservice-System nach außen anbietet bzw. nutzt, welche von Systemen anderer Organisationen der Entwickler genutzt werden können. Falls das Microservice-System eine öffentliche Schnittstelle im Internet anbietet, könnten dies im Extremfall Internet-Nutzer sein.
Änderungen solcher Schnittstellen können in diesem Fall nicht mehr so leicht durch Abstimmung mit den Nutzern umgesetzt werden, da die Anzahl viel zu hoch wäre oder die Nutzer unbekannt sind. Testverfahren eigneten sich aus diesem Grund ebenfalls nicht. Man müsse stattdessen prinzipiell Regeln für diese Schnittstellen definieren und eine Versionierung durchführen. Hierbei spiele die Abwärtskompatibilität eine wesentliche Rolle. Unter Umständen sei es sogar notwendig, verschiedene Versionen der gleichen Schnittstellen anzubieten, damit die Nutzer nicht direkt gezwungen sind Änderungen vorzunehmen.

Für externe Schnittstellen rät (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 191) diese nicht mit internen Schnittstellen zu vermischen, sondern klar zu trennen.

#### 5.1.3.3 Semantische Versionierung (Semantic Versioning)

Für die Versionierung empfiehlt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 191) eine Versionsnummer mit *Semantic Versioning* zu verwenden. Dies bedeutet, dass die Versionsnummer aus verschiedenen Teilen zusammengesetzt sein solle, wie beispielsweise MAJOR Release Nummer, MINOR Release Nummer und PATCH Nummer.
Diese haben typischerweise folgende Bedeutung:

- Eine erhöhte MAJOR Release Nummer zeigt an, dass die neue Version nicht Rückwärtskompatibel ist und alle Nutzer ihre Version anpassen müssen.
- Eine erhöhte MINOR Release Nummer zeigt an, dass die Schnittstelle neue Features anbietet und Rückwärtskompatibel ist. Die Nutzer müssen ihre Version nur anpassen, sofern sie die neuen Features nutzen wollen.
- Eine erhöhte PATCH Nummer zeigt an, dass es Bug Fixes gab. Die Nutzer müssen keine Änderung durchführen, da die Schnittstelle vollständig rückwärtskompatibel ist.

[12] Siehe hierzu Kapitel 5.11.3

#### 5.1.3.4 Postels Gesetz

Das Gesetz von Postel, welches aus als Robustheitsprinzip[13] bekannt ist, besagt laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 192), *„[…] dass Komponenten in dem, was sie tun, streng sein sollen und liberal in dem, was sie von anderen akzeptieren.“*

Dieses Konzept ist nach (Fowler M. , Tolerant Reader, 2011) auch unter der Bezeichnung Tolerant Reader bekannt und gilt sowohl für interne als auch externe Schnittstellen.

#### 5.1.3.5 Routing

Routing wird in Bezug auf Schnittstellen deshalb an dieser Stelle erwähnt, weil es bei der Verwendung einer Schnittstelle, die Http und REST anbietet, die Möglichkeit gibt, die Anfragen an diese zunächst an einen Router umzuleiten, der abhängig von Regeln, die Anfragen an unterschiedliche Versionen der Schnittstelle weiterleitet. Hierzu müssten beispielsweise die aufrufenden Microservices eine Versionsnummer der verwendeten Schnittstelle in Http Headern mit übergeben, die dann in einem vorgeschalteten Router als Kriterium zum Weiterleiten dient. Der Microservice mit der angebotenen Schnittstelle muss sich jedoch bei Änderung der Schnittstelle um das Deployment des Routers bzw. dessen Regeln kümmern.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 191) zieht es sogar in Betracht einen eigenen Adapter vor die angebotene Schnittstelle zu schalten, sofern die externen Aufrufe modifiziert werden müssen.

Untersuchungsrelevanz:
Schnittstellen sind untersuchungsrelevant, da Microservices zum Kommunizieren sich an einen gemeinsamen Standard halten müssen. Interne Schnittstellen sind über das Kommunikationsprotokoll bereits abgedeckt. Externe Schnittstellen sind individuell. Hierbei kann kein gemeinsamer Standard festgelegt werden, da diese von den einzubindenden Diensten abhängig und individuell sind.
Bei Semantic Versioning kann untersucht werden, ob es für die einzelnen Technologien Framework Unterstützung gibt.
Postels Gesetz ist eine Entwurfsentscheidung, die von der Implementierung, aber nicht von der Technologie abhängig ist. Daher kann diesbezüglich nichts untersucht werden.
Bei Routing kann untersucht werden, ob es Frameworks zur Erstellung von Routern in den einzelnen Technologien gibt, welche Möglichkeiten diese bieten und wie gut diese mit Hilfe von Frameworks in den Technologien integriert sind.

## 5.2 Diensterkennung (Service Discovery)

Bei der Diensterkennung geht darum, dass die im System verfügbaren Microservices die Schnittstellen der anderen Microservices kennen, um mit diesen kommunizieren zu können.

[13] Siehe http://tools.ietf.org/html/rfc793#section-2.10

### 5.2.1 Statische Diensterkennung

Bei einer kleinen Anzahl von Microservices wäre es laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 146) denkbar, ohne Diensterkennung auszukommen. In diesem Fall wären die Adressen der anderen Microservices fix im nutzenden Microservices hinterlegt. Die Adresse eines Microservice besteht immer aus der IP-Adresse und dem verwendeten Port. Sobald nun aber die Anzahl der Microservices ansteigt, wird man die Adressen zentral und damit konfigurierbar verwalten wollen.
Hierzu bietet sich zunächst eine zentrale Konfiguration[14] an, welche von allen Microservices gelesen wird.

Laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 143) ist dieser statische Ansatz nicht ausreichend und nennt hierfür folgende Gründe:

- Microservices können kommen und gehen, weil Server ausfallen, Microservices neu ausgerollt werden oder Microservices durch Skalierung[15] mehrfach vorhanden sein können.
- Durch die Diensterkennung sind die aufrufenden Microservices nicht mehr so eng an den aufrufenden Microservice gebunden. Dies ist nützlich für die Skalierung, da ein Client nicht mehr an eine konkrete Instanz des Servers gebunden ist, sondern je nach aktueller Last verschiedene Instanzen des Servers kontaktieren kann.
- Wenn alle Microservices einen gemeinsamen Ansatz für Service Discovery haben, entsteht eine zentrale Registratur aller Microservices. Das kann für einen Architektur-Überblick hilfreich sein oder es können Monitoring-Informationen von allen Systemen abgeholt werden.

### 5.2.2 Dynamische Diensterkennung

Die Diensterkennung muss daher dynamisch sein, d.h. Microservices müssen sich zentral bei einer Registratur eintragen können und damit ihre Verfügbarkeit bekunden.

Laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 143) sind Konfigurationsmechanismen hierfür die falschen Werkzeuge. Wichtig hierbei sei jedoch, dass diese zentrale Registratur eine hohe Verfügbarkeit hat, da bei Ausfall keinerlei Kommunikation mehr zwischen den Microservices stattfinden könne.
Die hohe Verfügbarkeit lässt sich durch Replikation der zentralen Registratur erreichen, wobei im Gegensatz zu Konfigurationssystemen laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 143) ein Trade-off zwischen Konsistenz und Verfügbarkeit klar zu Gunsten der Verfügbarkeit ausfallen müsse.

[14] Siehe Kapitel 5.3

[15] Siehe Kapitel 5.6.1

Eine Technologie zur Umsetzung von Diensterkennung bietet prinzipiell das Domain Name System (DNS) Protokoll an. Hierbei geht es darum, den Hostnamen zu einer IP-Adresse aufzulösen. Der Port, sowie weitere zusätzliche Informationen bezüglich Priorität und Gewicht für einen Server, kann bei DNS über SRV-Records abgebildet werden. Damit erhält man neben der Diensterkennung eine zusätzliche Option, um eine Lastverteilung[16] auf mehrere Server zu erreichen. Für weitere Details bezüglich DNS sei auf (Zytrax, 2016) verwiesen.

Eine weitere Technologie für dynamische Diensterkennung kann über dedizierte Server erfolgen. Dieser speichert für jeden Microservice einen Host und einen Port, unter dem der Service zur Verfügung steht. Die Microservices melden sich dort an bzw. ab und werden je nach Implementierung des Servers sogar auf deren Verfügbarkeit zyklisch abgefragt, womit ebenfalls eine Lastverteilung ermöglicht wird, sofern sich mehrere Instanzen eines Microservice registriert haben.
Um die Kommunikation mit dem Diensterkennungsserver zu reduzieren können die Microservices die Informationen des Servers in einem Cache[17] halten.

(Scholl, Swanson, & Frenandez, 2016, S. 153) ergänzt bzgl. dynamischer Diensterkennung, dass beim Einsatz von Container Clustern das Orchestrierungswerkzeug (Kubernetes, Airbnb, Glieder Labs Registrator) die Microservices automatisch beim Starten von neuen Containern in Knoten (Nodes) erkennen und registrieren.
Zusätzlich gäbe es noch die Möglichkeit einen „Sidecar“ Service in jedem Knoten zu betreiben, welcher sich um die Registrierung der in den Knoten vorhandenen Microservices kümmert.

Untersuchungsrelevanz:
Bei der Diensterkennung müssen sich Microservices an einen gemeinsamen Standard halten, wenn es um den Zugriff auf die Dienstregistrierung geht. Diese kann in unterschiedlichen Technologien umgesetzt sein und muss eine einheitliche Schnittstelle anbieten.
Sie ist daher untersuchungsrelevant. Die statische Diensterkennung wird im Rahmen des Kapitels 5.3 implizit mit untersucht.

## 5.3 Konfiguration

Microservices sollen konfigurierbar sein, d.h. sie benötigen für ihren Betrieb einstellbare Parameter. Hierbei stellen sich 2 Fragen:

- Wo werden diese Parameter verwaltet?
- Wie kommen die Parameter in den Microservice?

Die Parameter werden an einer zentralen Stelle verwaltet und immer in Form eines Schlüssel/Wert Paares gespeichert. Als Speicherort bieten sich daher Key/Value Stores an. Weiterhin gibt es Werkzeuge, die über einen zentralen Server die Parameter aus dem Repository eines Versionsverwaltungssystems auslesen. Die Daten können hierbei in hierarchischer Form gespeichert werden und man hat zusätzlich die Möglichkeit der

[16] Siehe Kapitel 5.6.2

[17] Siehe Kapitel 5.6.3

Versionierung der Parameter.
Die Parameter kommen prinzipiell dadurch hinein, dass die Microservices die Parameter einlesen müssen. Dies kann entweder direkt geschehen, indem auf den zentralen Speicherort bzw. den Konfigurationsserver zugegriffen wird oder durch Einlesen von Umgebungsvariablen, wenn Microservices in Containern betrieben werden. Hierbei besteht die Möglichkeit einem Container Umgebungsvariablen zu injizieren, welche dann beim Start der Microservices eingelesen werden. Allerdings sind diese Parameter dann zur Laufzeit des Microservice nicht mehr änderbar.

Untersuchungsrelevanz:
Für den Zugriff auf die Konfigurationsparameter müssen sich die Microservices an einen gemeinsamen Standard halten. Daher ist dieser Aspekt untersuchungsrelevant.

## 5.4 Persistenz

In Kapitel 2.1.2.5 wurde bezüglich Persistenz gesagt, dass jeder Microservice seine eigene Datenbank zum Abspeichern verwenden darf.

(Serra, 2017) verwendet hierfür auch den Begriff „Polyglot Persistence“. Dieser beschreibt, dass es in Bezug auf Datenspeicherung am besten sei, mehrere Speichertechnologien zu verwenden, abhängig davon, wie die Daten von der individuellen Applikation bzw. von Komponenten einer einzelnen Applikation verwendet werden. Es geht also darum je nach verwendeten Daten und Zweck die passende Speichertechnologie zu verwenden, anstatt sich nur auf eine vertraute Speichertechnologie zu konzentrieren.

Um zu veranschaulichen, welche Datenbankkategorien und Produkte bereits im Jahr 2013 verfügbar waren zeigt die folgende Abbildung.

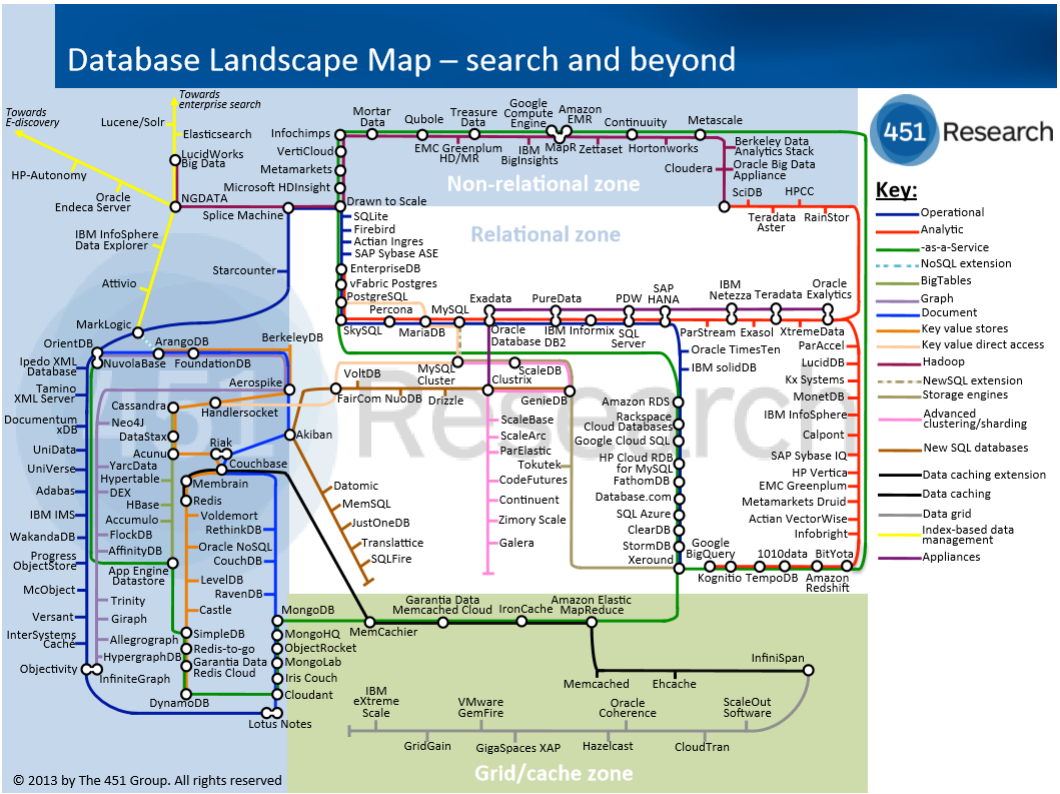


***Abbildung 4*** *Database Landscape Map gemäß (Aslett, 2013)*

Es gilt also entsprechend der zu lösenden Aufgabe eine passende Kategorie aus den oben genannten Kategorien zu finden. Diese sind in der Abbildung farblich hinterlegt in relational (Relational zone), nicht relational (Non-relational zone) und Grid bzw. Caching Datenbanken (Grid/cache zone). Die rechte Seite der Abbildung zeigt eine Legende einer noch feineren Einteilung der verschiedenen Typen.

Untersuchungsrelevanz:
Beim Aspekt Persistenz soll es gemäß den Merkmalen keinen gemeinsamen Standard geben. Jeder Microservice darf seine eigene Speichertechnologie verwenden. Es soll die ausgewählt werden, welche am besten passt. Dies bezieht sich auch auf die eingesetzte Technologie zur Implementierung des Microservice. Daher ist dieser Aspekt **nicht** untersuchungsrelevant.

## 5.5 Konnektivität

Um die Microservices, welche in einem Verbund zusammenarbeiten, mit der Außenwelt zu verbinden, kann ein API Gateway eingesetzt werden.

(Vitz, 2018) bezeichnet das API Gateway auch als Edge Server.

Gemäß (Scholl, Swanson, & Frenandez, 2016, S. 121) wird dieses benutzt für Routing, Lastverteilung von eingehendem Verkehr und darüber hinaus kann dieses Aufgaben bzgl. SSL Terminierung, Authentifizierung, Aufruf Verdichtung (Request Aggregation) und Protokoll Transformation übernehmen.

(Fabrizio Montesi, 2018, S. 6) erklären, dass das API Gateway insbesondere das Problem von unterschiedlichen Clients adressieren und für diese als einziger Zugriffspunkt für viele APIs dienen.

(Richardson, 2018, S. 17) sieht neben den genannten Vorteilen auch Nachteile, da das API Gateway zum einen eine hochverfügbare Komponente sein muss und zum anderen kann dies zu einem Flaschenhals bei der Entwicklung werden, da alle Entwickler das API Gateway für ihre jeweiligen Microservices anpassen müssen.

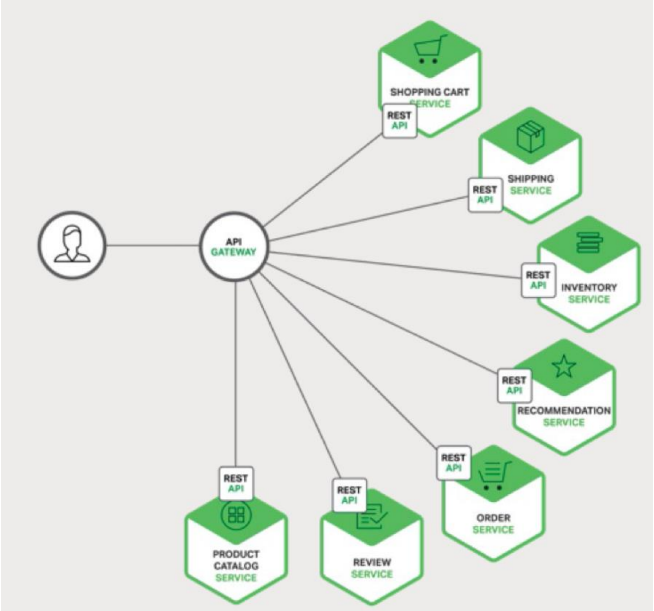



***Abbildung 5*** *API Gateway entnommen von (Richardson, 2018)*

Untersuchungsrelevanz:
Obwohl die Microservices in direkter Verbindung mit dem API Gateway stehen können, benötigen Sie keinen direkten Zugriff auf dieses. Daher ist Konnektivität **nicht** untersuchungsrelevant.

## 5.6 Performanz

Um die Performanz bei einem Microservice-System zu erhöhen, muss man insbesondere die Kategorien Skalierung, Lastverteilung und Caching betrachten.

### 5.6.1 Skalierung

In Kapitel 2.1.2.9 wurde bereits erläutert, was unter Skalierung zu verstehen ist. Microservices bieten insbesondere den Vorteil, dass diese leicht horizontal skaliert werden können. Werden Microservices in Containern betrieben, bedeutet dies, dass einfach ein zusätzlicher Container gestartet werden muss. Das Starten und Stoppen von Containern sollte dynamisch und lastabhängig erfolgen.

Die Microservices in den Containern müssen laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 151) in diesem Fall zustandslos sein, also insbesondere sollen sie keine benutzerspezifischen Zustände speichern. Damit die Skalierung dynamisch erfolgen kann, muss die Last der Microservices ermittelt werden. Dies kann durch Monitoring[18] erfolgen. Welche Möglichkeiten es zur Lastverteilung gibt, wird im nächsten Kapitel beschrieben.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 153) gibt noch zu Bedenken, dass sich durch horizontales Skalieren nicht die Antwortzeiten von einzelnen

[18] Siehe Kapitel 5.9.1

Microservices verkürzen lassen. Dies kann prinzipiell nur durch vertikale Skalierung erfolgen oder durch Caching[19]. Vertikale Skalierung lässt sich einfach dadurch realisieren, dass ein Microservice auf einem schnelleren Rechner (Virtuelle Maschine) verteilt wird.

Untersuchungsrelevanz:
Die Skalierung wird von der Infrastruktur übernommen und ist unabhängig von der Technologie der Microservices und daher **nicht** untersuchungsrelevant.

### 5.6.2 Lastverteilung (Load Balancing)

Bei der Lastverteilung in einem Microservice-System geht es darum, die Anfragen, welche an einen Microservice gehen, auf mehrere Instanzen des gleichen Microservice zu verteilen. Hierbei kommen Nachrichten-Systeme, die proxybasierte Lastverteilung, die Lasterverteilung mit Service Discovery und die clientseitige Lastverteilung zum Einsatz.

#### 5.6.2.1 Nachrichten-Systeme

Für den Fall, dass ein Nachrichten-System eingesetzt wird, lässt sich die Lastverteilung sehr leicht dadurch realisieren, dass sich zusätzliche Nachrichtenempfänger registrieren, welche sich die Last teilen. Um die konkrete Verteilung der Nachrichten muss sich jedoch das Nachrichten-System kümmern.

#### 5.6.2.2 Proxybasierte Lastverteilung

Bei der proxybasierten Lastverteilung wird vor die Microservice Instanzen, welche auch auf verschiedenen Servern laufen können, ein Load Balancer als Proxy geschaltet. Alle Anfragen werden vom diesem Proxy Load Balancer auf die einzelnen Instanzen verteilt. Der Proxy Load Balancer ermittelt über Anfragen an die Instanzen, wie die Auslastung dieser sind. Gegebenenfalls kann der Load Balancer auch fehlerhafte Instanzen aus der Lastverteilung entfernen. Zu Bedenken ist jedoch, dass der gesamte Verkehr zu den Microservice Instanzen über den Proxy Load Balancer läuft. Dadurch stellt dieser eine einzelne Schwachstelle (Single-Point of failure) dar, wenn er ausfällt.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 147) unterscheidet bei diesem Verfahren noch zwischen zentralem Load Balancer und Load Balancer pro Microservice. Das zuvor beschriebene Verfahren bezieht sich auf Load Balancer pro Microservice. Beim zentralen Load Balancer wird ein solcher für alle Microservices geschaltet, ähnlich wie es bei einer monolithischen Architektur üblich ist.

Dies ist jedoch laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 147) zu vermeiden, da hierdurch die Konfiguration des Load Balancers sehr aufwendig würde und kein unabhängiges Deployment der Microservice mehr möglich wird.

[19] Siehe Kapitel 5.6.3

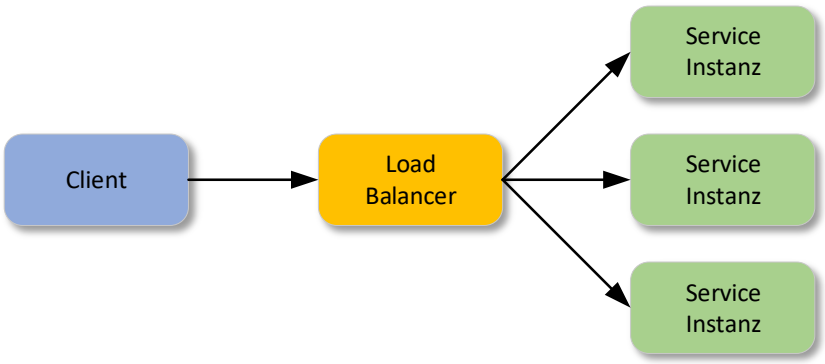



***Abbildung 6*** *Proxy-basierte Lastverteilung*

### 5.6.2.3 Lasterverteilung mit Service Discovery

Der Nachteil bei der proxybasierten Lastverteilung ist, dass die Load Balancer die vorhandenen Microservice Instanzen kennen müssen. Dynamisch hinzugefügte Instanzen werden nur durch manuellen Eingriff in die Lastverteilung einbezogen. Dieser Eingriff kann automatisch bei einem neuen Deployment erfolgen oder durch Änderung der Konfiguration des Load Balancers.

Abhilfe schafft die Einbeziehung der Diensterkennung, die dynamisch neue Microservice Instanzen erkennt. Hierbei holt sich ein Client Microservice die Adresse von vorhandenen Microservice Instanzen vom Diensterkennungsservice, bei dem diese sich zuvor dynamisch registriert haben.

Im ersten Ansatz zeigt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 148) auf, dass es sich eher um eine lastabhängige Umschaltung zu Microservice Instanzen handelt. Es gibt hierbei keinen dedizierten Load Balancer mehr. Die Umschaltung erfolgt im Client Microservice. Lediglich bei Überlastung wird eine andere Instanz gewählt.

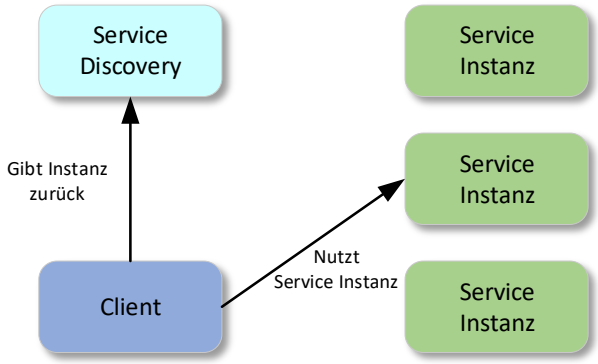



***Abbildung 7*** *Lastverteilung mit Diensterkennung*

### 5.6.2.4 Clientseitige Lastverteilung

Bei der clientseitigen Lastverteilung findet zusätzlich zu dem zuvor beschriebenen Verfahren noch eine echte Verteilung der Last im Client Microservice statt. Das bedeutet, der Algorithmus zur Umschaltung ist im Client Microservice implementiert. Der Client Microservice verwendet daher nicht nur eine Microservice Instanz, sondern mehrere gleichzeitig. Beispielsweise könnte man sich vorstellen, dass der Microservice eine Liste mit verfügbaren Instanzen vom Diensterkennungsservice speichert und falls der Zugriff auf eine gewählte Instanz nicht

funktioniert, eine andere Instanz aus der Liste als Fallback Strategie gewählt wird.
Bei diesem Verfahren entfällt der externe Load Balancer als einzelne Schwachstelle.

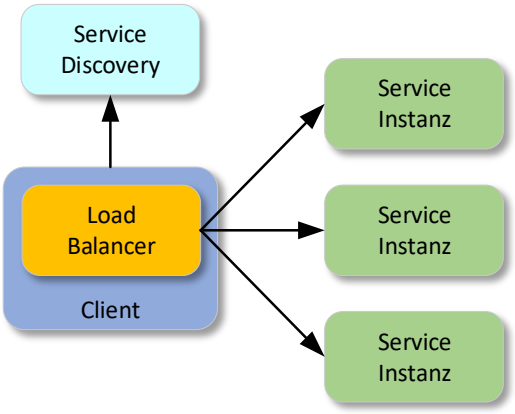



***Abbildung 8*** *Clientseitige Lastverteilung*

Untersuchungsrelevanz:
Die Lastverteilung durch Nachrichtensysteme ist vom Nachrichtensystem abhängig. Diese Anbindung wurde bereits im Kapitel 5.1.2.4 untersucht.
Bei der Lastverteilung muss unterschieden werden, ob diese von einem externen Load Balancer oder im Microservice intern umgesetzt ist. Die externe Umsetzung (Proxy basierte Lastverteilung) ist unabhängig von den Microservices und daher **nicht** untersuchungsrelevant.
Beim internen Load Balancer (Lastverteilung mit Service Discovery bzw. Clientseitige Lastverteilung) müssen sich die Microservices an keinen gemeinsamen Standard halten (im Gegensatz zur Diensterkennung), sondern können ihren eigenen Algorithmus zur Umschaltung der Service Instanzen umsetzen. Obwohl dieser Aspekt daher ebenfalls nicht untersuchungsrelvant ist, kann die Umsetzung der Algorithmen in den einzelnen Sprachen eine nicht triviale Aufgabe sein. Es soll somit zumindest untersucht werden, ob entsprechende Frameworks in den verschiedenen Technologien zur Verfügung stehen.

### 5.6.3 Puffern (Caching)

Da Microservices immer gemäß Charakteristik miteinander über Netzwerkverbindungen kommunizieren, welche unter Umständen sehr träge sein können, und damit zu einer erhöhten Latenzzeit führen, kann durch das Zwischenspeichern von bereits abgerufenen Daten mittels Caching die Verarbeitungszeit massiv verkürzt werden.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016) stellt hierzu auch fest, dass Caching benötigte Rechenleistung zum Konvertieren, Verschicken und Empfangen von Nachrichten erheblich reduziert.

Generell lassen sich beim Caching verschiedene Strategien unterscheiden, welche sich nach (Emer, 2018) danach einordnen lassen, an welchem Ort die Daten zwischengespeichert werden. In Web Applikationen allgemein lassen sich die folgenden Orte unterscheiden:

- Client (Browser)
- Netzwerk

- Server (Proxy)
- Applikation - InMemory
- Applikation - Shared Server

#### 5.6.3.1 Client (Browser)

Bei dieser Art des Cachings werden die Daten wie Bilder, Skripte und Styles im Browser des Benutzers lokal zwischengespeichert um nicht jedesmal neu vom Server abgerufen zu werden.

#### 5.6.3.2 Netzwerk

Beim Caching im Netzwerk findet das Zwischenspeichern von Daten in einem Content Delivery Netzwerk (CDN) statt. Ein Content Delivery Netzwerk ist ein Verbund von Servern, welche über das Internet zusammengeschaltet sind, um insbesondere große Mediendateien ausliefern zu können.

#### 5.6.3.3 Server (Proxy)

Um die Ladezeiten von Web-Requests zu verkürzen kann zwischen den Client und den eigentlichen WebServer ein Proxy Server geschaltet werden. Dieser speichert die Antworten auf Anfragen zwischen. Varnish, Squid und nginx sind typische Vertreter solcher Proxy Server.

#### 5.6.3.4 Applikation – InMemory

Die Applikation kann zur Beschleunigung von Zugriffen Daten selbst in einem internen Cache zwischenspeichern und damit bei wiederholten Anfragen auf die gleichen Daten die Zugriffszeit reduzieren. Es ist jedoch zu bedenken, dass der Speicher der Applikation sehr begrenzt ist.

#### 5.6.3.5 Applikation – Shared Server

Werden mehrere Instanzen der Applikation betrieben, führt dies beim cachen der Daten InMemory zu Redundanz. Abhilfe schafft hier ein verteilter Caching Server.
Dies bietet den Vorteil, dass weniger Anfragen an einen Microservice gestellt werden, welcher die Daten bereithält. Sollte dieser im Fehlerfall eventuell nicht zur Verfügung stehen, können die Daten aus dem verteilten Cache verwendet werden und machen das Microservice-System damit ein Stück robuster[20].
Die im verteilten Cache gepufferten Daten stehen auch nach einem möglichen Ausfall von anfragenden Microservices zur Verfügung und haben damit eine längere Verfügbarkeit.
Typische Vertreter solcher verteilter Caches sind Schlüssel-Wert Speicher („key-value stores“). Auf (Engines, 2818) findet man eine aktuelle Übersicht über die meist verwendeten Key-Value Stores.

Untersuchungsrelevanz:
Client, Netzwerk und Proxy Caching finden unabhängig von den Microservices statt und sind **nicht** untersuchungsrelevant.

[20] Siehe Kapitel 5.7

Das Caching in der Applikation (InMemory) kann unter Ausnutzung der Möglichkeiten des Http Protokolls für Caching stattfinden oder individuell umgesetzt sein. Sowohl Java und ASP.NET Core bieten hierfür hinreichende Möglichkeiten, weshalb auf eine weitere Untersuchung verzichtet wird.
Beim Caching in der Applikation mit Shared Server müssen die Microservices das gemeinsame Protokoll zur Anbindung an den Key/Value Store unterstützen. Daher ist dies untersuchungsrelevant. Es kann untersucht werden, ob es innerhalb der Technologien Frameworks zur Anbindung an die gängigsten Key/Value Stores gibt.

## 5.7 Robustheit und Ausfallsicherheit (Robustness and Resiliency)

In Bezug auf Microservices heißt Robustheit, diese tolerant gegenüber Fehlern zu machen. Zum einen bedeutet dies, den Microservice stabil gegenüber internen Fehlern, aber noch viel wichtiger gegenüber Fehlern anderer Microservices oder Fehlern aufgrund eines Netzwerkausfalls zu machen.

Interne Fehler werden mit den üblichen Fehlerbehandlungsmaßnahmen (z.B. Exception-handling) behandelt und sollen an dieser Stelle nicht weiter besprochen werden.

Die Behandlung der anderen Fehler lässt sich am besten dadurch erreichen, indem man die Kommunikation der Microservices untereinander robust gegenüber Fehlern macht.
Dies ist insbesondere wichtig, weil laut (Newmann, 2015, S. 223) sowohl Netzwerke, als auch Maschinen ausfallen und sich bei einem verteilten System der Fehler schnell auf das gesamte System ausdehne und es zum Totalausfall komme.

Stellt ein Microservice eine Anfrage an einen anderen Microservice und schlägt dieser fehl, so führt das meistens dazu, dass der anfragende Microservice keine Antwort mehr zurückerhält. Im Fehlerfall erhält der angefragte Microservice die Anfrage nicht mehr, um entsprechend darauf reagieren zu können.

Daher ist es prinzipiell laut (Horsdal Gammelgaard, 2017, S. 140) Aufgabe des Clients für die Robustheit zu sorgen. Der Microservice, der Anfragen an einen anderen stellt, muss entsprechende Mechanismen implementieren, um mit dieser Fehlersituation umzugehen.
Zur Umsetzung dieser Mechanismen haben sich entsprechende Entwurfsmuster heraus-gebildet.

### 5.7.1 Zeitüberschreitung (Timeout)

Das einfachste Entwurfsmuster ist die Zeitüberschreitung. Hierbei wird eine auszuführende Aktion mit einer Zeitbegrenzung ausgeführt. Dauert die auszuführende Aktion zu lange, sprich die Zeitbegrenzung wird überschritten, was typischerweise bei Anfragen über das Netzwerk an einen anderen Microservice passieren kann, so kann eine Fehlerbehandlung mit einer alternativen Code Strecke ausgeführt werden. Diese Code Strecke kann dann die im folgenden beschriebenen Entwurfsmuster verwenden.

### 5.7.2 Wiederholung (Retry)

Das Retry Entwurfsmuster findet laut (Horsdal Gammelgaard, 2017, S. 142) insbesondere Anwendung bei Anfragen mit transienten Fehlern. Also Fehlern, die „flüchtig“ und somit nur temporär auftreten. Die Ursachen hierfür können im Fall von Microservices zum Beispiel Netzwerk Überlastung oder etwa die kurzfristige Nichtverfügbarkeit eines Microservice aufgrund von Neuverteilung sein. Eine wiederholte Durchführung der Abfrage führt dann unter Umständen zum Erfolg.
Das Retry Entwurfsmuster erscheint auf den ersten Blick eine gute Lösung zu sein, um mit transienten Fehlern umzugehen. Allerdings hat es auch eine gravierende Schwäche. Dadurch, dass nun unter Umständen mehrere Microservices oder Instanzen des gleichen wiederholt auf einen fehlerhaften Microservice zugreifen, wird die Last auf diesen erheblich erhöht und dieser unter Stress gesetzt.

(Horsdal Gammelgaard, 2017, S. 143) beschreibt als Abhilfe hierfür die Einführung der sogenannten *„exponential backoff“* Strategie. Mit dieser werden nach einer konfigurierbaren Anzahl von Wiederholungsversuchen der Zeitabstand zwischen den Wiederholungen exponentiell erhöht, um dem fehlerhaften Microservice Zeit zu geben sich zu erholen. Beispielsweise könne nach 3 fehlgeschlagenen Versuchen mit einer Wiederholungsdauer von je 100ms die Wiederholungdauer auf 200ms erhöht werden. Schlägt der Versuch beim 4. Mal auch fehl, so werde die Wiederholungsdauer auf 400ms erhöht usw.
Die Anzahl der Wiederholungen solle jedoch letztlich auch begrenzt werden, da es wenig Sinn mache, unendliche Zeit auf die Antwort eines fehlerhaften Microservice zu warten.

Schließlich könnte man nun noch versuchen wie im Kapitel 5.6.2.4 die missglückte Anfrage an eine andere Instanz des Microservice zu senden.

### 5.7.3 Selbstabschaltung (Circuit Breaker)

Im Gegensatz zum Retry Entwurfsmuster wird beim Circuit Breaker Entwurfsmuster die Anzahl der Wiederholungen auf einen festen Wert begrenzt. Es wird die Annahme getroffen, dass eine wiederholte Durchführung einer fehlgeschlagenen Anfrage mit großer Wahrscheinlichkeit wieder zu einem Fehler führt. Das Entwurfsmuster verwaltet intern einen Zustandsautomaten mit dem in folgender Abbildung dargestellten Zustandsdiagramm.

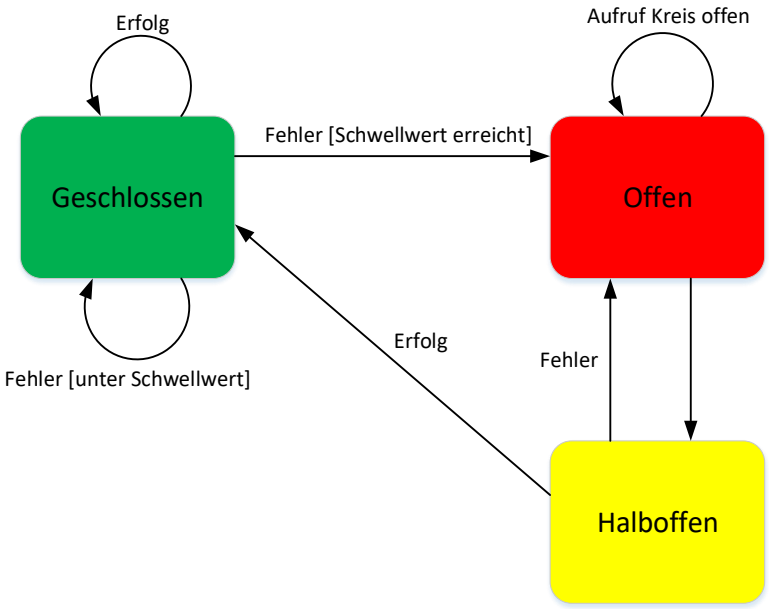



***Abbildung 9*** *Zustandsdiagramm Selbstabschaltung*

Im Zustand **Geschlossen** werden Anfragen normal ausgeführt. Sobald eine Anfrage fehlschlägt, wird ein interner Zähler erhöht und die Anfrage nach einem definierten Zeitabstand wiederholt. Ist die nächste Abfrage dann erfolgreich, wird der Zähler wieder auf 0 zurückgesetzt. Häufen sich die fehlerhaften Abfragen und der Zähler erreicht einen konfigurierten Wert, geht der Zustandsautomat in den Zustand Offen über.

Im Zustand **Offen** werden keine weiteren Abfragen mehr durchgeführt, sondern führen direkt zu einer Fehlerbehandlung. Nach Ablauf einer konfigurierbaren Zeit geht der Zustandsautomat in den Zustand Halboffen über.

Im Zustand **Halboffen** kann nun eine neue weitere Abfrage ausgeführt werden. Ist diese erfolgreich, geht der Zustandsautomat wieder in den Zustand Geschlossen über und alle weiteren Abfragen können normal durchgeführt werden. Schlägt diese jedoch erneut fehl, so geht der Zustandsautomat in den Zustand Offen über und es muss wieder abgewartet werden.

### 5.7.4 Trennwand (Bulkhead)

Der Begriff Trennwand bzw. Schott stammt aus der Schiffahrt und bezeichnet speziell im Rumpf eines Schiffes eingebaute Türen, die bei einem Wassereinbruch geschlossen werden können und damit das weitere Eindringen des Wassers verhindern. In Bezug auf Softwaresysteme wird mit Trennwänden eine Ausbreitung eines Fehlers von einem Teilsystem zu anderen Teilsystemen erreicht.

In einem Microservice-System bieten sich für die Teilsysteme insbesondere solche an, die innerhalb eines Bounded Context stehen.

Untersuchungsrelevanz:
Bei Robustheit haben sich die oben genannten Entwurfsmuster als Standard etabliert. Es ist kein technologischer Standard vorhanden. Robustheit wird innerhalb der Microservices

umgesetzt. Es gibt jedoch innerhalb der Infrastruktur Dienste zur Überwachung von Zuständen bzgl. der Robustheit. Bei diesen müssen sich dann die Microservices an einen gemeinsamen Standard halten. Insofern ist Robustheit untersuchungsrelevant.
Es wird untersucht, welche Frameworks es gibt, wie deren Qualität ist und inwieweit die Anbindung an Überwachungswerkzeuge möglich ist.

## 5.8 Sicherheit (Security)

Sicherheit ist auch bei einer Microservice-Architektur ein elementarer Aspekt der Infrastruktur. Hierbei werden die üblichen Teilaspekte Echtheitsüberprüfung, Zugriffsberechtigung und Verschlüsselung unterschieden.

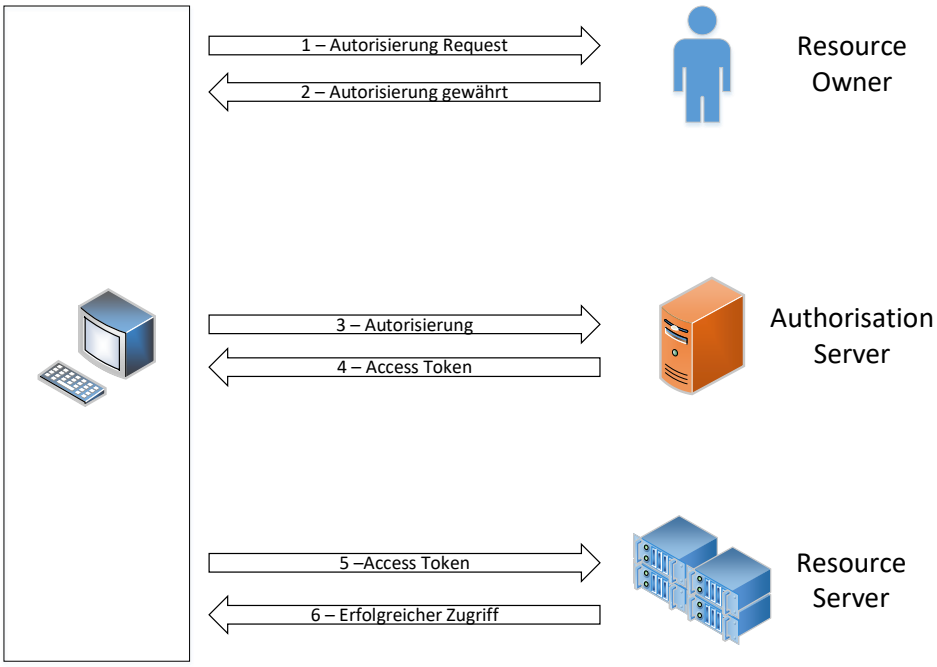
### 5.8.1 Echtheitsüberprüfung (Authentication)

Zur Echtheitsüberprüfung in IT System haben sich das OAuth2 und OpenId Connect Protokoll in Verbindung mit JSON Web Token insbesondere bei Web- und Mobile Applikationen als Standard entwickelt.

#### 5.8.1.1 OAuth2 Protokoll

Im Rahmen dieser Arbeit soll das OAuth2 Protokoll nur vereinfacht und für den Anwendungszweck einer Microservice Anwendung erläutert werden. Für die Details des Protokolls sei hier für interessierte Leser auf die Spezifikation[21] verwiesen.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 155) erklärt den prinzipiellen Ablauf des Protokolls anhand folgender vereinfachter Abbildung.




[21] Siehe https://tools.ietf.org/html/rfc6749 bzw. https://oauth.net/2/

***Abbildung 10*** *Ablauf OAuth2 Protokoll*

Es gibt die folgenden Teilnehmer:

- Client
- Ressource Owner
- Authorisation Server
- Ressource Server

In **Schritt 1** fragt der Client beim Ressource Owner nach, ob eine bestimmte Aktion durchgeführt werden darf. Die Anwendung benötigt beispielsweise Zugriff auf das Profil oder bestimmte Daten in einem sozialen Netzwerk, welches der Ressource Owner dort gespeichert hat. Meistens ist der Ressource Owner gleichzeitig der Nutzer des Systems.

In **Schritt 2** bekommt der Client eine entsprechende Antwort vom Ressource Owner zurück, sofern dieser dem Client den Zugriff auf die geschützte Ressource gewährt.

In **Schritt 3** verwendet der Client die Antwort vom Ressource Owner und stellt damit eine Anfrage beim Authorisation Server.

In **Schritt 4** überprüft der Authorisation Server die vom Client gestellte Anfrage und gibt diesem ein Access Token zurück.

In **Schritt 5** kann nun der Client das Access Token vom Authorisation Server verwenden, um auf eine geschützte Ressource zuzugreifen. Das Access Token kann beim Aufruf beispielsweise in einem Http Header untergebracht werden.

In **Schritt 6** erhält der Client die angefragte Ressource.

Das Protokoll erlaubt bezüglich der Interaktion mit dem Authorisation Server unterschiedliche Abläufe bzw. Ansätze.

Beim **Passwort Ansatz** muss der Ressource Owner Benutzername und Passwort in eine vom Client generierte Web Seite eingeben. Mit diesen Daten kann der Client dann in Schritt 3 das Access Token mit Hilfe einer Http Post Anfrage erhalten. Nachteilig hierbei ist, dass der Client sowohl Benutzernamen als auch Passwörter übertragen bekommt. Sofern der Client unsicher implementiert ist, sind diese Daten gefährdet. Und da die Login Seite vom Client generiert wird, hat der Ressource Owner keine Gewähr, ob diese tatsächlich vom vorgegebenen Client stammt.

Beim **Authorisation Ansatz** wird der Anwender vom Client in Schritt 1 auf eine Web Seite des Authorisation Servers umgelenkt. Der Nutzer wählt nun auf dieser aus, ob der Zugriff gestattet werden soll. Falls dies der Fall sein sollte, bekommt der Client einen Authorisation Code zurück, den er dann in Schritt 3 dem Authorisation Server schicken kann. Nur der Authorisation Server kann diesen gesendeten Code überprüfen und bei Gültigkeit dem Client das Access Token zurückschicken.

Beim **Implicit Ansatz** wird der Client wie beim Authorisation Ansatz ebenfalls auf eine Web Seite des Authorisation Servers umgelenkt, erhält aber nun statt eines Authorisation Codes direkt das Access Token zurück. Dieser Ansatz vereinfacht den Protokoll-Ablauf noch weiter, da

Schritt 3 und 4 entfallen. Er ist für Clients optimiert, welche in einem Browser laufen und eine Scriptsprache wie JavaScript verwenden. Er wird implizit genannt, da kein dazwischenliegender Authorisation Code benötigt wird.

Beim **Client Credentials Ansatz** wird davon ausgegangen, dass nur der Client bestimmte Informationen hat, die nur er selbst kennt. Mit diesen Informationen allein kann sich der Client beim Authorisation Server das Access Token abrufen.

#### 5.8.1.2 JSON Web Token (JWT)

Für das im vorigen Kapitel beschriebene Access Token hat sich zum Speichern der darin enthaltenen Informationen das JSON Web Token als Standard etabliert. Json (Java Script Object Notation) wird hierbei als Datenstruktur verwendet.

Das JWT besteht aus drei Teilen. Einem Header, der Payload und der Signatur. Der Header enthält den Typ des Tokens, sowie den genutzen Hashing Algorithmus. Die Signatur wird benötigt, um die Echtheit des Tokens zu überprüfen. Sie besteht aus dem kodierten Header, Payload und Secret.

Zur Überprüfung wird JWS (Json Web Signature) als digitale Signatur und zum Entschlüsseln wird JWE (Json Web Encryption) verwendet. Weiterhin können im Access Token Informationen zum Aussteller, den Ressource Owner, die Gültigkeitsdauer oder den Adressaten des Access Tokens gespeichert sein. Da das Token im Http Header verwendbar sein soll, wird es Base64 kodiert.

Das JSON Web Token kann auch mit benutzerspezifischeb Daten (Claims) angereichert werden. Es ist darauf zu achten, dass nicht zu viele Claims im Token angereichert werden. Es gibt zwar laut RFC7519 keine Größenbeschränkung bei der Verwendung von JWT, allerdings gibt es bestimmte Router (z.B. heroku oder Apache), welche Requests mit einer Headergröße von mehr als 8kbyte nicht weiterreichen.

### 5.8.2 SAML

Neben JWT gäbe es noch die Möglichkeit Nutzer im Web- und Microservice-Kontext mit Hilfe der Security Assertion Markup Language (SAML) zu authentifzieren. SAML ist ein wesentlich älteres Protokoll und bringt verschieden Nachteile mit sich.

(Schönbach, 2018) führt zunächst auf, dass SAML zum kodieren der Nachricht XML verwendet, was sprachbedingt zu einem höhren Datenvolumen führt. Obwohl SAML asymmetrische Signaturverfahren verwendet, sei diese digitale Signatur bei SAML komplex und wesentlich schwieriger in Objekte umzuwandeln als bei JSON.

(Lodderstedt, 2018) ergänzt hierzu noch, dass SAML an der inhärenten Komplexität des des unterliegenden Standards für XML Signaturen kranke und im Betrieb aufgrund der Verwendung von X.509-Zertifkaten als schwer zu beherrschen herausgestellt habe.

### 5.8.3 SWT

Eine weitere Alternative zu JWT wären Simple Web Tokens (SWT).
(Schönbach, 2018) schreibt hierzu, dass diese nur durch symmetrische Signierungsverfahren geprüft werden könne, was die Anfälligkeit für Hackerangriffe erhöhe. Sobald ein Microservice gehackt sei, ist das Secret bekannt und das ganze System wäre dann kompromietiert. Bei JWT kenne nur der Authorisation Server das Secret und dieser einzelne Service ließe sich wesentlich besser schützen als eine Vielzahl von Microservices.

#### 5.8.3.1 Echtheitsüberprüfung bei Microservices

Aus den zuvor genannten Gründen wird OAuth2 in Verbindung mit JSON Web Tokens bei Microservice Applikation verwendet. Mit dem bei OAuth2 erworbene Access Token kann nun der Nutzer eine Web Seite eines Microservice oder einen Microservice per REST aufrufen. Ein Microservice muss dann lediglich die Echtheit des Access Tokens überprüfen und nicht mehr den Nutzer selbst.

Die Überprüfung erfolgt mit Hilfe eines Authorisation Servers. Dieser speichert einen öffentlichen und einen privaten Schlüssel. Beim ersten Aufruf muss der Microservice den öffentlichen Schlüssel beim Authorisation Server herunterladen und kann diesen lokal zwischenspeichern. Es müssen dann keine weiteren Zugriffe mehr auf den Authorisation Server erfolgen. Beim Aufruf von weiteren Microservices, muss das Access Token dann jeweils weitergereicht werden.

(Schönbach, 2018) gibt weiterhin jedoch zu bedenken, dass Tokens ein Ablaufdatum besitzen sollten, da sonst das entziehen von Nutzerrechten unmöglich wird. Ebenso sollten keine sicherheitskritischen Informationen wie Passwörter im Token enthalten sein, da diese sich mit Base64 leicht dekodieren ließen.

### 5.8.4 Autorisierung (Authorisation)

Im Fall von Microservices bedeutet Autorisierung eine Zugriffsbeschränkung von geschützten Ressourcen. Wie bereits beschrieben, können im Access Token noch weitere Informationen codiert sein. Diese vom Authorisation Server stammenden Informationen werden über die Signatur des Access Tokens abgedeckt. Sofern das Access Token gültig ist, sind auch die darin enthaltenen Daten gültig. Typischerweise werden benutzerspezifische Daten wie etwa eine Gruppenzugehörigkeit, Benutzerrollen oder entsprechende Zugriffsrechte verwendet. Diese werden in dem Zusammenhang auch als *Claims* bezeichnet. Die Verwaltung der Claims findet nicht im Microservice statt, sondern werden vom Authorisation Server geliefert, während der Microservice nur aufgrund vorhandener Claims den Zugriff auf geschützte Ressourcen erlaubt bzw. verbietet. Hiermit wird laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 156) sichergestellt, dass eine Änderung der Zuordnung von Benutzern und Claims nicht zu einer Änderung von Microservices führt.

### 5.8.5 Verschlüsselung (Kryptographie)

Verschlüsselung bedeutet für Microservices, dass diese ganz allgemein untereinander verschlüsselte Daten austauschen wollen. Also, dass entweder die Datenverbindung verschlüsselt sein soll oder andererseits, dass bereits verschlüsselte Daten übertragen und dann entschlüsselt werden müssen.

Man kann somit die Verschlüsselung der Verbindung und die Verschlüsselung der zu übertragenen Daten betrachten.
Die Verschlüsselung der Verbindung erfolgt in der Praxis meist über Zertifikate, welche sich technologieunabhängig vom Microservice in der Infrastruktur konfigurieren lassen.

Das Ver- und Entschlüsseln von Daten ist notwendig, wenn verschlüsselt abgelegte Daten übertragen werden sollen.
Hierbei werden symmetrische und asymmetrische Verschlüsselungsverfahren unterschieden.

Bei **symmetrischen Verfahren** werden zur Ver- und Entschlüsselung der gleiche Schlüssel verwendet, welche zuvor zwischen Sender und Empfänger ausgetauscht werden müssen.

Bei **asymmetrischen Verfahren** gibt es einen geheimen Schlüssel, welcher vom Sender zum Verschlüsseln und einen öffentlichen Schlüssel, welcher vom Empfänger zum Entschlüsseln verwendet wird.

Untersuchungsrelevanz:
Bei der Echtheitsüberprüfung müssen sich die Microservices an vorgegebene Sicherheitsprotokolle halten. SAML und SWT sollten aus den genannten Gründen nicht eingesezt werden und werden daher **nicht** untersucht.
OAuth2 mit OpenId Connect und Json Web Tokens sind untersuchungsrelevant.
Es kann hier untersucht werden, wie die Unterstützung dieser Standards in den Technologien aussieht. Bei der Autorisierung müssen sich die Microservices an keinen gemeinsamen Standard halten, sondern müssen ihre eigene Implementierung mitbringen. Daher ist Autorisierung **nicht** untersuchungsrelevant.
Die Verschlüsselung mit Zertifikaten wird von der Infrastruktur selbst übernommen und ist unabhängig von den Microservices und somit ebenfalls **nicht** untersuchungsrelevant.
Bezüglich Kryptographie müssen sich die Technologien an die verfügbaren Algorithmen halten. Diese sind standardisiert und werden bereits in anderen Architekturen erfolgreich eingesetzt. Es wird davon ausgegangen, dass diese zuverlässig in den Technologien umgesetzt wurden und werden daher **nicht** untersucht.
Die Verschlüsselung der Kommunikation zwischen den Microservices findet nicht statt, da dies viel zu aufwendig wäre. Die Ver- und Entschlüsselung zu den Clients mittels SSL wird vom API Gateway übernommen (Vitz, 2018). (siehe Kapitel 5.5)

## 5.9 Instrumentierung

Bei der Instrumentierung können die Teilaspekte Logging und Monitoring unterschieden werden.

### 5.9.1 Logging

Beim Logging geht es insbesondere darum ein Protokoll des Software Prozesses zu erstellen.

Die Applikation soll somit ein Protokoll erstellen, welches Informationen über aufgetretene Ereignisse und Fehler enthält. Diese sollen den Entwicklern bei der Fehlersuche und Administratoren beim Betreiben der Applikation dienen. Bei einer monolithischen Applikation hat es genügt, in ein oder mehrere Log-Dateien zu **protokollieren**.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 245) erklärt hierzu, dass dieser Ansatz bei einer Microservice-Applikation nicht ausreichend sei und nennt dafür folgende Gründe:

- Die meisten Anfragen an Microservices können nur durch das Zusammenspiel von mehreren Microservices behandelt werden. Um den Gesamtablauf verstehen zu können, ist die Log-Datei eines einzelnen Microservice nicht ausreichend.
- Da die Last oft auf mehrere Instanzen des Microservice verteilt wird, hilft es nur wenig die Informationen aus der Log-Datei einer einzelnen Instanz zu betrachten.
- Beim Ausfall einer Instanz des Microservice gehen auch die lokal gespeicherten Log-Dateien verloren.

Abhilfe schafft daher nur das Schreiben der Log-Daten in einen zentralen Server. Hierbei sollten die Microservices laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 245) nicht einfach nur Texte, sondern weitere semantische Informationen in einem **einheitlichen** Datenformat protokollieren. Als einheitliches Datenformat eigne sich Json, um das Übertragen und Parsen der Daten zu vereinfachen.
Da Microservices sich gegenseitig aufrufen, entsteht das Problem, dass der Ursprung einer Anfrage nicht nachvollzogen werden kann.
(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 248) empfiehlt hierzu die Verwendung einer CorrelationID. Hiermit kann beispielsweise die Anfrage eines einzelnen Benutzers nachvollzogen werden, welche sich über mehrere Microservices erstreckt. Man müsse diese bei jeglicher nachgelagerten Kommunikation der Microservices mit weiterreichen. Implementieren ließe sich das, indem die CorrelationID entweder im Header oder in der Payload der Anfrage übertragen wird.

### 5.9.2 Monitoring

Beim Monitoring geht es nicht nur um die unmittelbare systematische Erfassung von Protokollen, sondern auch um die Beobachtung und Überwachung von Vorgängen oder Prozessen.

In Bezug auf Microservices bedeutet dies, Metriken zu sammeln, die dabei helfen sollen, Transparenz zu schaffen, was den aktuellen Zustand der Applikation bzw. der Microservices betrifft und wie sich dieser über die Zeit verhält. Im einfachsten Fall handelt es sich um Health Checks.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 249) nennt als weitere Beispiele die Anzahl der verarbeiteten Anfragen über eine bestimmte Zeit, die Dauer der Bearbeitung der Anfragen, oder generelle Systemwerte wie die CPU- bzw. Speicherauslastung. Mit Hilfe der gesammelten Werte ließen sich Schwellwerte überprüfen. Bei Über- bzw. Unterschreitung könnten dann Aktionen ausgeführt werden. Dies könnte im einfachsten Fall das Auslösen eines Alarms sein. Es ließen sich damit aber auch Maßnahmen zur Beeinflussung der Performanz einleiten, indem beispielsweise neue Instanzen eines Microservices hinzu- bzw. abgeschaltet werden.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 250) hebt hier weiterhin hervor, dass die relevanten Informationen in einem einheitlichen Format und grundlegende Informationen verpflichtend sein sollten.
In der Regel bietet der Microservice eine Schnittstelle an, die von einem externen Monitoring Dienst zyklisch abgefragt wird. Diese Dienste bieten dann auch entsprechende Benutzerschnittstellen zur Visualisierung der abgerufenen Daten für die verschiedenen Interessensgruppen an. Zu diesen können neben den Entwicklern und Administratoren auch Stakeholder aus den unterschiedlichen Fachbereichen gehören.

(Scholl, Swanson, & Frenandez, 2016, S. 210) erklären, dass prinzipiell 3 grundlegende Komponenten überwacht werden müssen. Diese seien:

- Die Host Maschine (Host)
- Die Container
- Die Dienste innerhalb des Containers

Beim Host ist die Überwachung insbesondere wichtig um frühzeitig erkennen zu können, ob eine Skalierung notwendig wird. Es müssen die Metriken CPU, RAM, Netzwerk und Disk I/O überwacht werden.
Metriken der Container können mit Hilfe von Control groups und Namespaces beispielsweise bei Linux Betriebssystemen in gemounteten Verzeichnissen ausgelesen werden. Docker Container böten die Möglichkeit über eine Restful Remote API diese Daten auszulesen. Toolhersteller wie z.B. Datadog bieten hierzu entsprechende Agents an, welche pro Host in einem Container laufen und diese Remote API nutzen um Daten an ein zentrales Dashboard zu schicken.

Prinzipiell kann somit beim Monitoring unterschieden werden, ob zur Realisierung Instrumentierungscode notwendig ist oder nicht. Beim Monitoring von Container und System ist kein Instrumentierungscode notwendig, während beim Monitoring der Applikation selbst Instrumentierungscode notwendig ist.

Untersuchungsrelevanz:
Beim Logging müssen bzw. sollten sich die Microservices an ein einheitliches Datenformat halten und in der Lage sein, die Log Daten in eine zentrale Log-Senke zu schreiben. Als Log Senken können alle möglichen Speicher zur Persistierung verwendet werden. Aufgrund der Vielzahl der Möglichkeiten wird dies **nicht** untersucht.
Beim Monitoring müssen sich die Microservices an einen gemeinsamen Standard halten. Untersuchungsrelevant ist jedoch nur das Monitoring der Applikation selbst.
Das Monitoring des Hosts bzw. der Container ist **nicht** untersuchungsrelevant, da dies unabhängig von der Technologie der Microservices ist.

## 5.10 Software Verteilung (Deployment)

Bei der Software Verteilung für Microservices geht es um deren Ausbringung in ihre Ausführungsumgebung.
(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 257) beschreibt hier folgende Möglichkeiten:

- Installation über Skripte
- Statt der Aktualisierung der vorhandenen Software auf einem Server, werden komplett neue Server aufgesetzt. Container Technologien unterstützen diesen Ansatz.
- Installationswerkzeuge verarbeiten Skripte, welche den Zustand des Systems nach der Installation beschreiben.
- Paket Manager, die es ermöglichen Software zentralisiert auf eine große Anzahl Server zu verteilen.

Untersuchungsrelevanz:
Die Software Verteilung erfolgt in eine Ausführungsumgebung, z.B. wie bereits in Kapitel 2.2.2.3 beschrieben wurde, in eine Container Technologie. Dies bezieht sich aber auf die Ausführungsumgebung selbst und nicht auf den Prozess der Ausbringung. Dieser soll gemäß den Merkmalen unabhängig möglich sein. Es können auch keine Software Frameworks in den einzelnen Technologien untersucht werden. Daher ist die Software Verteilung **nicht** untersuchungsrelevant.

## 5.11 Testen

Das Testen von Microservices ist ein Bestandteil der Deployment Pipeline und damit auch Bestandteil der Infrastruktur. Der Hauptgrund für das Testen im Allgemeinen ist das minimieren des Risikos, dass Fehler im produktiven System auftreten. Bei einer Microservice-Applikation im speziellen ergibt sich die größte Schwierigkeit beim Testen dadurch, dass hierbei das gesamte System im Zusammenspiel mit allen Microservices getestet werden muss und im Fehlerfall das

Auffinden des Fehlers. Aus diesem Grund werden, wie beim monolithischen Architekturstil, verschiedene Testarten nach der Testpyramide unterschieden, um mögliche Fehler besser eingrenzen zu können.

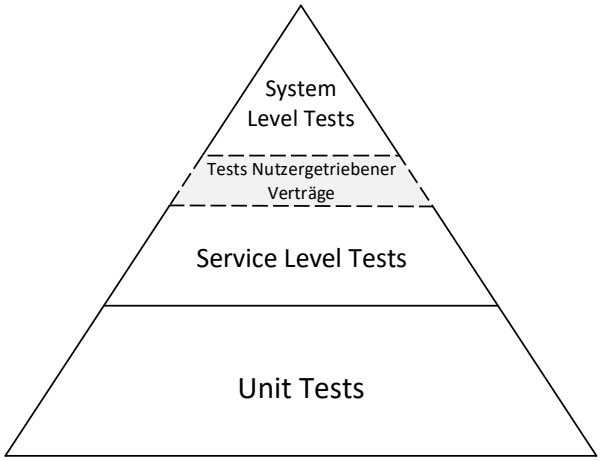



***Abbildung 11*** *Testpyramide für Microservices*

### 5.11.1 Unit Tests

Auf der untersten Ebene der Testpyramide sind Unit Tests angesiedelt. Sie werden auf dieser Ebene nach den gleichen Methoden wie bei monolithischen System geschrieben. Unit Tests testen eine einzelne Funktionalität innerhalb eines einzigen Microservice, um mögliche Fehler auf einzelne Funktionalitäten begrenzen zu können. Es werden Tests geschrieben, die beispielsweise Business Logik auf Klassenebene testen und es findet noch keine Kommunikation über das Netzwerk statt. Um die Aufrufe an andere Microservices bzw. an eine Datenbank zu simulieren, werden *Mock-Objekte* oder *Mocks* eingesetzt.

Mock-Objekte sind Platzhalter für echte Objekte, welche innerhalb von Modulttest verwendet werden. Umgangssprachlich werden diese auch Mocks genannt.

Ein Mock ersetzt somit das echte Objekt im Test durch ein Objekt, welches die gleiche Schnittstelle besitzt und bei einem bestimmten Aufruf einer Methode dieser Schnittstelle ein gültiges Ergebnis zurückliefert.

### 5.11.2 Service Level Tests

Auf der nächsten Ebene, nach den Unit Tests, erfolgen die Service Level Tests. Bei Service Level Tests werden Microservices isoliert vom Gesamtsystem getestet. Hierbei geht es insbesondere darum den Microservice über seine Schnittstellen nach außen zu testen. Hierzu werden die umliegenden Microservices durch Mock-Objekte oder Stubs ersetzt.

Als Stub bezeichnet man ein Stück Programmcode, welches anstelle eines anderen Programmcodes verwendet wird, der entweder noch nicht zur Verfügung steht, auf einem anderen Rechner installiert ist oder eventuell in einem anderen Speicherbereich läuft.

Ein Stub simuliert, im Vergleich zu einem Mock-Objekt, den gesamten Microservice. Ein Stub könnte beispielsweise bei einem bestimmten Aufruf immer einen konstanten Wert zurückgeben und ermöglicht damit das Testen des eigentlich abhängigen Microservice.

In der Praxis bedeutet dies für Service Level Tests somit, dass ein Microservice von außen direkt über seine API getestet wird und sowohl die Antworten des Microservice unter Test, wie die Aufrufe an die Mock-Objekte umliegenden Microservices entsprechend überprüft werden. Da Service Level Tests nur einen Microservice überprüfen, sind sie einerseits präziser als die im nachfolgenden beschriebenen System Level Tests und zur Ursachenforschung von Fehlern besser geeignet. Andererseits sind sie auch schneller, da alle Aufrufe an andere Microservices gemockt werden.

Bezüglich Datenbanken erklärt (Horsdal Gammelgaard, 2017, S. 160), dass Abfragen an diese bereits auf dieser Ebene schon an real existierende Datenbanken und nicht an Mock-Datenbanken erfolgen sollten.

### 5.11.3 System Level Tests

Auf obererster Ebene sind die System Level Tests angesiedelt. Mit System Level Tests wird das gesamte System mit allen Microservices als Ganzes getestet. Es werden weder Mock-Objekte, noch Stubs verwendet. Ziel ist es möglichst viel Code abzudecken und komplette Geschäftsanwendungsfälle im Rahmen von Akzeptanztests sicherzustellen.

(Horsdal Gammelgaard, 2017, S. 157) schreibt hierzu, dass diese Tests sehr unpräzise und eine Problemanalyse bei Fehlern sich als schwierig erwiesen. In der Praxis könne diese die vorhandene Benutzerschnittstelle verwenden, deren Verwendung automatisiert werden könne. Sie schaffen zwar auf der einen Seite großes Vertrauen im Erfolgsfall, weil hierbei das komplette Zusammenspiel überprüft werde, aber sind auf der anderen Seite sehr langsam. Nachteilig sei auch, dass für die Durchführung das gesamte System mit allen Microservices als Testumgebung vorhanden sein müsse.
Insofern sollten nur die wichtigsten Geschäfts-Anwendungsfälle als Tests umgesetzt werden und um solche erweitert werden, die kritische Fehlerszenarios nachbilden würden.
Für die Umsetzung gelten die gleichen Regeln wie bei monolithischen System.

### 5.11.4 Tests Nutzergetriebener Verträge (Consumer-Driven Contract Tests)

In **Abbildung 11** gibt es noch eine weitere Testart. Diese wurde grau und gestrichelt dargestellt, da diese zusätzlich stattfinden sollte, und von der Fachliteratur zwar genannt, aber bisher noch nicht Bestandteil der offiziellen Testpyramide sind. Sie sollten nach den Service Level und vor den System Level Tests ausgeführt werden.

Bei der Kommunikation zwischen Microservices werden bestimmte Annahmen über Anfragen und Antworten gemacht. Änderungen in einem Microservice können dazu führen, dass die gemachten Schnittstellenverträge zu anderen benutzenden Microservices nicht mehr eingehalten werden.

Folgende Dinge gehören nach (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 233) dazu:

- Die Datenformate, also in welchem Format die anderen Microservices die Daten erwarten.

- Die Schnittstelle, welche bestimmte Operationen zur Verfügung stellt.
- Abläufe und Protokolle, die definieren, welche Operationen in welcher Reihenfolge mit welchen Ergebnissen durchgeführt werden können.
- Meta-Informationen bei den Aufrufen, die zum Beispiel eine Benutzerauthentifizierung umfassen können.
- Bestimmte nichtfunktionale Aspekte, wie etwa die Latenzzeit oder ein bestimmter Durchsatz.

Werden Änderungen an einem Vertrag gemacht, kann dies zu einem Fehler führen, der unter Umständen viel später erst und in Produktion auftritt. Dies führt dazu, dass Änderungen deshalb an Systembestandteilen nicht vorgenommen werden, weil unklar ist, wie deren genaue Nutzung aussieht und man das Risiko nicht eingehen möchte einen Schnittstellenvertrag zu brechen. Abhilfe schaffen hier Tests nutzergetriebener Verträge, bei denen die nutzergetriebenen Verträge getestet werden.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 234) unterscheidet als Erklärung für diese Art von Tests folgende Verträge:

- Den **Anbieter Vertrag** (Provider Contract), der alles umfasst, was der Service-Anbieter zur Verfügung stellt. Für jeden Anbieter existiert ein solcher Vertrag.
- Den **Nutzer Vertrag** (Consumer Contract), welcher die Funktionalitäten umfasst, die der Service-Nutzer aus der Obermenge der vom Service-Anbieter zur Verfügung gestellten Funktionalitäten tatsächlich nutzt. Für jeden Service gibt es pro Service-Nutzer einen solchen Vertrag. Diese Verträge ändern sich, wenn sich die Schnittstelle des Service Nutzers ändert.
- Den **Nutzergetriebenen Vertrag**, welche alle Nutzer-Verträge zusammenfasst und daher alle Funktionalitäten des angebotenen Services verwendet, welche irgendein Service-Nutzer verwendet.

Aus dem nutzergetriebenen Vertrag wird nun offensichtlich, welche Bestandteile des Anbieter Vertrags wirklich genutzt werden. Änderungen an der Schnittstelle des Service-Anbieters können ohne Risiko durchgeführt werden, weil anhand dieses Vertrages leicht ermittelt werden kann, welche Bestandteile entweder gar nicht oder von welchem Service-Nutzer verwendet werden. Diese Überprüfung lässt sich nun mit Hilfe von Tests automatisieren. Somit hat man für jeden nutzergetriebenen Vertrag einen ausführbaren Test.

In der Praxis sieht es beispielsweise so aus, dass in einem ersten Schritt der Service Nutzer einen Test implementiert, welcher die Schnittstelle eines anderen Microservice nutzt. Der Test enthält hierzu entsprechende Aufrufe und die erwarteten Antworten. Mit Hilfe geeigneter Frameworks wird nun der Http Verkehr mit aufgezeichnet und eine Datei mit dem Vertag daraus generiert. Alle erstellten Vertragsdateien werden an einem zentralen Ort gespeichert. In einem zweiten Schritt verwendet der Service-Anbieter diese Vertragsdateien um gegen seine eigne Schnittstelle zu testen.

Untersuchungsrelevanz:
Unit Tests sind auf die Microservices selbst begrenzt und müssen sich an keinen gemeinsamen Standard mit anderen Microservices halten. Daher sind diese **nicht** untersuchungsrelevant.
Bei Service Level und System Level Tests werden Microservices von außen über ihre Schnittstelle getestet. Die Implementierung der Tests ist technologieunabhängig und man muss sich an keinen gemeinsamen Standard halten. Entsprechend sind diese Tests **nicht** untersuchungsrelevant.
Bei nutzergetriebene Tests ist es notwendig, ein den Microservice übergreifenden Standard zum Speichern der Verträge zu verwenden. Daher ist diese Kategorie untersuchungsrelevant. Es kann untersucht werden, wie gut die Verwendung der entsprechenden Frameworks funktioniert.

## 5.12 Orchestrierung (Clustering)

Der Zusammenschluss (Clustering) von Microservices benötigt ein Ecosystem um diese deployen zu können, die verschiedene Dienste zu managen, mehrere Instanzen des gleichen Microservice betreiben zu können und die Konnektivität zwischen den Microservices herzustellen. Damit diese Aufgaben automatisiert möglich sind benötigt man Orchestrierungstools.

(Scholl, Swanson, & Frenandez, 2016, S. 120) erklärt hierzu, dass eine Microservice Cluster Umgebung neben den Host Agents, in welchen die Microservices in Containers betrieben werden, und dem in Kapitel 5.5 beschriebenen API-Gateway noch die Infrastruktur-komponenten Service Discovery Registry, Management Service, Cluster State Store und Artifact Store/Image Registry enthält. **Abbildung 12** zeigt den grundsätzlichen Aufbau einer Cluster Umgebung.

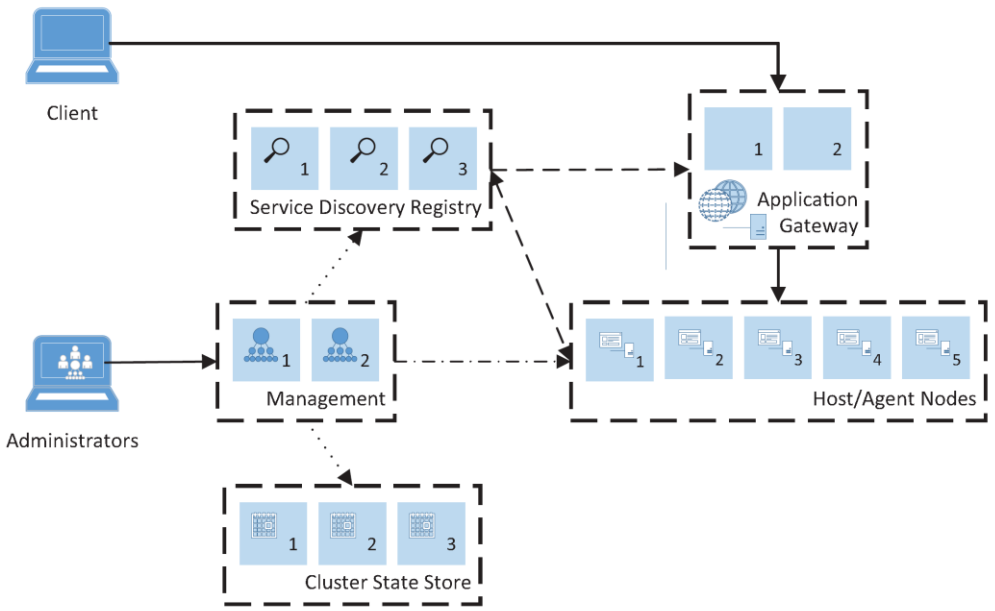



***Abbildung 12*** *Konzeptioneller Aufbau einer Cluster Umgebung mit Orchestrierungskomponenten für eine Microservice Architektur mit Containern aus (Scholl, Swanson, & Frenandez, 2016)*

### 5.12.1 Management Services

Die Management Services bieten einen zentralen Zugriffspunkt in Form einer API oder Benutzerschnittstelle um Dienste auf den Host/Agent Nodes auswählen und deployen zu können.

### 5.12.2 Cluster State Store

Der Cluster State Store wird von den Management Services benötigt um den Zustand des gesamten Clusters, aber auch Informationen zu den einzelnen Nodes zu speichern.

### 5.12.3 Service Discovery Store

Der Service Discovery Store ist ein zentraler Speicher für die Diensterkennung (siehe Kapitel 5.2.2).

### 5.12.4 Artifact Store / Image Registry

Im Artifact Store bzw. Image Registry werden Deployment Artifakte und Container Images gespeichert. Da Container Images oft auch in öffentlichen Registries, wie beispielsweise Docker Hub gespeichert sein können, ist diese Infrastrukturkomponente nicht in **Abbildung 12** dargestellt.

Untersuchungsrelevanz:
Orchestrierung bzw. Clustering wird erst bei einer größeren Anzahl von Microservices relevant. Die wichtigsten Vertreter dieser Technologien wie Docker Swarm, Kubernetes, Mesos werden in (Scholl, Swanson, & Frenandez, 2016) sehr ausführlich beschrieben. Deren Infrastrukturkomponenten werden der Vollständigkeit halber erwähnt. Da die Microservices (abgesehen von der Diensterkennung) keinen Zugriff auf diese Infrastrukturkomponenten haben ist dieser Aspekt **nicht** untersuchungsrelevant.

## 5.13 Dokumentation

Ein weiterer Aspekt, der auch Teil der Infrastruktur sein sollte, ist Dokumentation. Hierbei geht es weniger um schriftliche Dokumente, da diese aufgrund der häufigen Änderung von Microservices schnell veralten würden, sondern es geht um automatische vom Microservice oder eventuell auch vom Build System generierte Dokumentation.

Die Dokumentation soll insbesondere Auskunft über die API und deren Verwendung geben.

Untersuchungsrelevanz:
Bei Dokumentation ist es empfehlenswert einen gemeinsamen Standard zu haben, um die gesammelten Daten leicht über eine grafische Benutzerschnittstelle darstellen zu können. Daher ist dieser Aspekt untersuchungsrelevant.
Es kann untersucht werden, ob es bereits einen einheitlichen etablierten Standard gibt bzw. welche Frameworks in den einzelnen Technologien eventuell Unterstützung bieten.

## 5.14 Dienst Management

### 5.14.1 Steuerung

Bei diesem Aspekt geht es darum, dass Microservices eine einheitliche Schnittstelle anbieten sollten, um während der Laufzeit eines Microservice die Möglichkeit für einen Eingriff in das Laufzeitverhalten desselben zu haben und Metadaten abrufen zu können.

Für den Eingriff in das Laufzeitverhalten gibt es laut (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 262) hierzu verschiedene mögliche Maßnahmen:

- Sofern der Microservice in einer virtuellen Maschine läuft, kann diese heruntergefahren bzw. neu gestartet werden. Es müssen dann keine speziellen Vorkehrungen für den einzelnen Microservice getroffen werden.
- Der Microservice läuft als Betriebssystem Dienst und kann entsprechend gestartet, gestoppt bzw. neu gestartet werden.
- Der Microservice bietet eine Schnittstelle an, welche über Http und Rest angesprochen werden kann. Diese Schnittstelle muss der Microservice selbst implementieren und anbieten.

(Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 262) ergänzt hierzu noch, dass die technische Umsetzung dieser Steuerungsmechanismen kein großes Problem darstelle, sie müsse nur vorhanden und für alle Microservices identisch umgesetzt sein, damit der Aufwand für den Betrieb des Systems reduziert wird.

Ein anderes Szenario zur externen Steuerung wäre beispielsweise, wenn alle Service Instanzen eines Microservice Konfigurationsparametern neu einlesen müsse, falls sich diese ändern.

### 5.14.2 Metadaten

Zum Abrufen von Metadaten schlägt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 161) vor, dass jeder Microservice eine einheitliche Schnittstelle anbietet, welche zur Laufzeit beispielsweise folgende Informationen bereitstellt:

- Grundlegende Informationen wie der Name des Service und die verantwortlichen Ansprechpartner.
- Informationen über den Quell-Code, wie etwa wo dieser im Versionskontrollsystem abgelegt ist und welche Bibliotheken er verwendet.
- Mit welchen anderen Microservices kommuniziert wird.
- Aktuell verwendete Konfigurationsparameter oder Feature Toggles.

Weiterhin empfiehlt (Wolff, Microservices Grundlagen flexibler Softwarearchitekturen, 2016, S. 162) den Einsatz eines Service Templates, welches beispielhaft zeigt, wie die Dokumentation erzeugt wird, da dies die Umsetzung einer standard-konformen Dokumentation erleichtert.

Untersuchungsrelevanz:
Bei Dienstmanagement ist es notwendig einen gemeinsamen Standard für diese Aufgabe zu definieren. Daher ist dieser Aspekt untersuchungsrelevant.
Es kann untersucht werden, ob es in den einzelnen Technologien Unterstützung für diese Aufgabe gibt und ob diese je nach Infrastruktur hinreichend gelöst ist.

## Zusammenfassung

Zum Abschluss dieses Kapitels wird eine Übersicht über die zu untersuchenden Aspekte im Praxisteil gegeben. Die mit grünen Häkchen versehenen Aspekte sind untersuchungsrelevant, jene mit rotem X sind nicht untersuchungsrelevant.

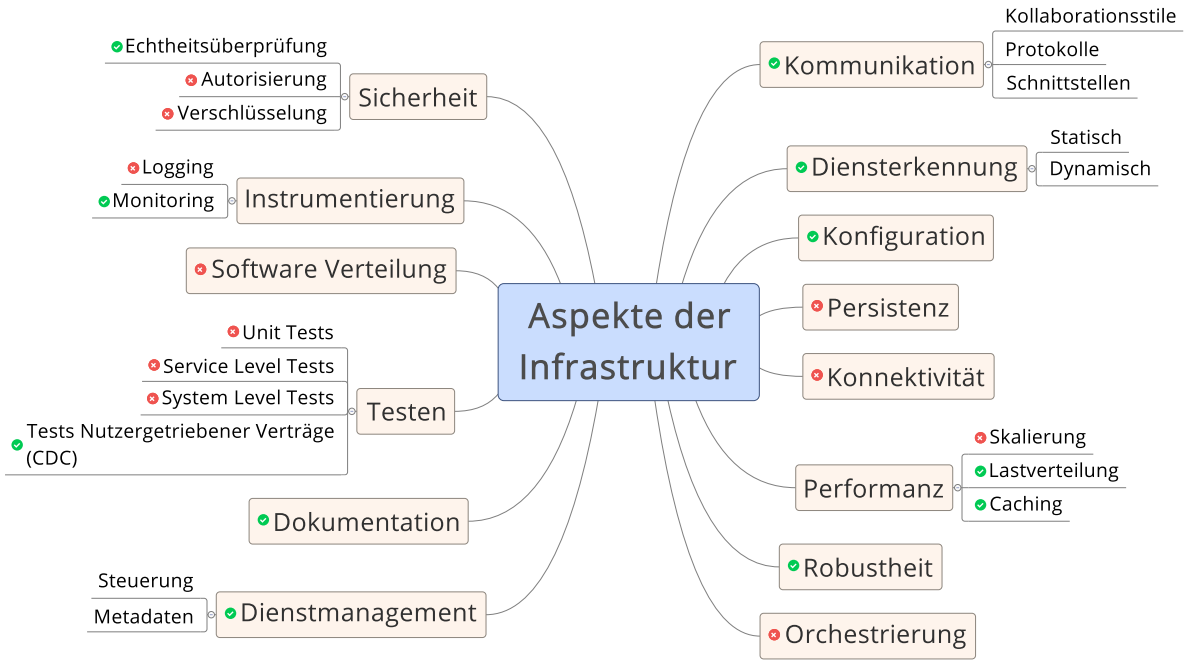



***Abbildung 13*** *Aspekte der Infrastruktur*

# 6 VERGLEICHSKRITERIEN FÜR SOFTWARE FRAMEWORKS ZUR UNTERSTÜTZUNG VON ASPEKTEN EINER MICROSERVICES-INFRASTRUKTUR

Das erste Teilkapitel dieses Abschnitts beschreibt zunächst, wie die Qualität von Softwareprodukten im Allgemeinen bewertet wird und im Anschluss in einem weiteren Teilkapitel, welche Vergleichskriterien für die Bewertung von Open Source Frameworks zur Unterstützung von Aspekten der Infrastruktur von Microservices herangezogen werden können.

## 6.1 Bewertung der Qualität von Softwareprodukten

Das Deutsche Institut für Normung hat 2010 als neuen Leitfaden für Qualitätskriterien und die Bewertung von Softwareprodukten die Normenreihe 25xx eingeführt. Die ISO 25010 definiert im speziellen die für die Bewertung von Softwareprodukten anwendbaren Qualitätsmerkmale.

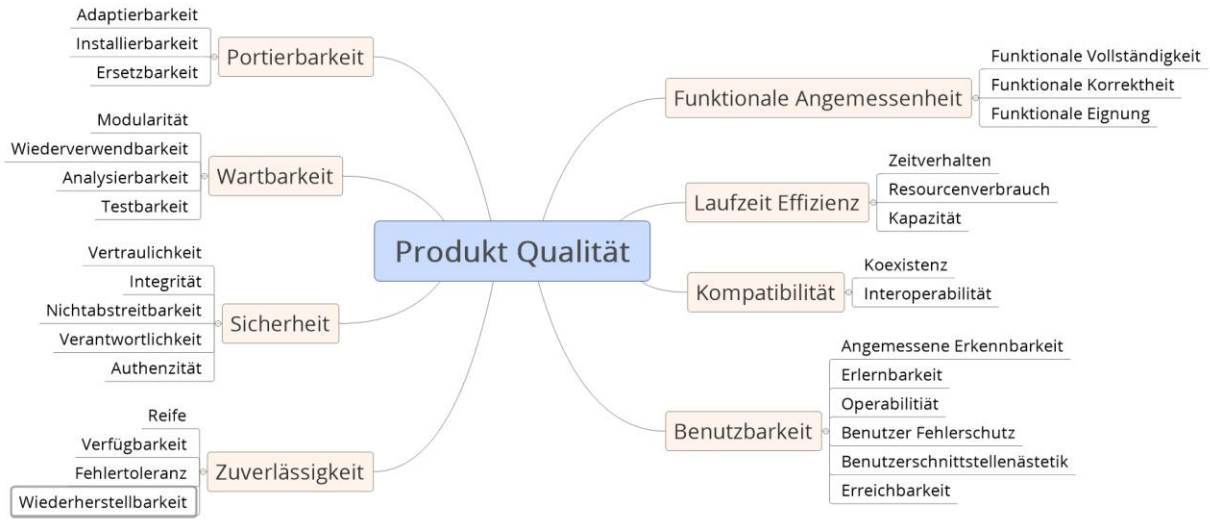



***Abbildung 14*** *Qualitätskriterien nach ISO/IEC 25010 gemäß (Arendt, 2015)*

## 6.2 Ermittlung von Kriterien zur Bewertung von Open Source Frameworks

Das Model des vorigen Kapitels findet Anwendung für Software Produkte im Allgemeinen. Bei der Auswahl der Kriterien im Rahmen dieser Arbeit müssen folgende Bedingungen berücksichtigt werden:

1) Es werden technisch vollkommen unterschiedliche Frameworks betrachtet.
2) Es soll eine große Menge an Frameworks untersucht werden.

Das bedeutet, dass bei der Auswahl der Kriterien nur solche Verwendung finden können, welche sich einerseits allgemein auf eine große Anzahl von Frameworks anwenden lassen und Metriken bieten, welche mit überschaubarem Aufwand ermittelbar sind, aber andererseits auch genügend große Aussagekraft zum Vergleichen liefern.

Die Qualitätskriterien mit ihren Subcharakteristika des Models aus dem vorigen Kapitel erfüllen diese Bedingungen überwiegend nicht.

Im Folgenden werden daher anhand verschiedener Beispiele aus der Praxis Kriterien betrachtet. Die mit grünem Häkchen versehenen Kriterien sind verwendbar, während die mit rotem Kreuz nicht anwendbar sind. Es erfolgt nach jedem Unterkapitel insbesondere eine Begründung, weshalb die ausgeschlossenen Kriterien nicht verwendet werden.

### 6.2.1 Kriterien nach (Eggert, 2018)

(Eggert, 2018) schlägt als Kriterien zur Bewertung von PHP Frameworks die folgenden Kriterien vor:

- Aktualität ✓
- Verbreitung ✓
- Dokumentation ✓
- Qualitätssicherung ✓
- Community ✓
- Lizenz ✓
- Technik ✗
- Bugs ✓
- Features ✗

Die Kriterien Technik und Features sind nicht mit Bedingung 2 vereinbar.

### 6.2.2 Kriterien nach (Doumack, 2018)

(Doumack, 2018) verwendet zur Bewertung von Reporting Frameworks folgende Kriterien:

- Funktionalität ✗
- Benutzbarkeit ✗
- Qualität ✓
- Sicherheit ✗
- Performanz ✗
- Skalierbarkeit ✗
- Architektur ✗
- Support ✓
- Dokumentation ✓
- Adoption ✓
- Community ✓
- Professionalität ✗

Das Kriterium *Funktionalität* entspricht dem Kriterium *Features* aus (Eggert, 2018) und kann nicht verwendet werden. Das Kriterium *Benutzbarkeit* entspricht dem Kriterium *Dokumentation* sowie den in (Jackson, Crouch, & Baxter, 2018) genannten Kriterien *Erlernbarkeit* und *Installierbarkeit*. Das Kriterium *Sicherheit* wird nicht untersucht, da die Microservices in einer geschützten Umgebung und isoliert betrieben werden und davon ausgegangen werden darf, dass die Frameworks keine Sicherheitslücken eröffnen. Die Kriterien *Performanz* und *Skalierbarkeit* setzen umfangreiche Messungen voraus und sind daher nicht mit Bedingung 2 vereinbar. Das Kriterium *Architektur* misst die Erweiterbarkeit bzw. Konfigurierbarkeit und ist ebenfalls nicht mit Bedingung 2 vereinbar. Das Kriterium *Professionalität* untersucht den

Entwicklungsprozess und die Projektorganisation und ist nicht anwendbar, da es sich im Rahmen dieser Arbeit um Open Source Frameworks handelt und keine Kapitalgesellschaften hinter der Entwicklung stehen.

### 6.2.3 Kriterien nach (Jackson, Crouch, & Baxter, 2018)

(Jackson, Crouch, & Baxter, 2018) beschreiben bezüglich der Hauptkriterien Benutzbarkeit und Nachhaltigkeit bzw. Wartbarkeit folgende Subkriterien:

- Verständlichkeit ✓
- Dokumentation ✓
- Baubarkeit ✗
- Installierbarkeit ✓
- Erlernbarkeit ✓
- Identität ✗
- Copyright ✗
- Lizenz ✓
- Governance ✗
- Community ✓
- Erreichbarkeit ✗
- Testbarkeit ✓
- Portierbarkeit ✓
- Support ✓
- Analysierbarkeit ✗
- Änderbarkeit ✗
- Evolvierbarkeit ✓
- Interoperabilität ✗

Das Kriterium *Baubarkeit* wird nicht verwendet, da die Frameworks überwiegend in kompilierter Form eingebunden werden. Das Kriterium *Identität* untersucht, ob das Framework eine eigene Domäne, ein Logo und registrierte Waren- bzw. Dienstleistungsmarke hat und wird aufgrund der geringen Aussagekraft für den Vergleich nicht herangezogen. Das Kriterium *Copyright* wird nicht verwendet, da die Frameworks Open Source sind und hinreichend mit dem Kriterium Lizenz abgedeckt werden. Das Kriterium *Governance* entspricht dem Kriterium *Professionalität* aus (Doumack, 2018) und wird ebenfalls nicht verwendet. Das Kriterium *Erreichbarkeit* beschreibt, wo ein Framework verfügbar gemacht wurde. Es entspricht teilweise dem Kriterium *Installierbarkeit* und hat zu wenig Aussagekraft. Das Kriterium *Analysierbarkeit* ermittelt die Code Qualität und Struktur der Frameworks und ist nicht mit Bedingung 2 vereinbar. Das Kriterium *Änderbarkeit* beschreibt, wie leicht Änderungen gemacht werden dürfen, wer diese durchführen darf und wie diese dokumentiert werden und sind ebenfalls nicht mit Bedingung 2 vereinbar. Das Kriterium *Interoperabilität* beschreibt, ob und wie sich die Frameworks in einen offenen Standard halten und wird nicht verwendet, da es nicht mit Bedingung 1 und 2 vereinbar ist.

## Zusammenfassung

**Abbildung 15** zeigt eine Übersicht der verwendeten Kriterien. Das Kriterium Qualität und Testbarkeit wurde zum Kriterium Tests zusammengefasst. Das Kriterium Bugs wurde in Korrekturrate umbenannt, da der Begriff besser verständlich ist.

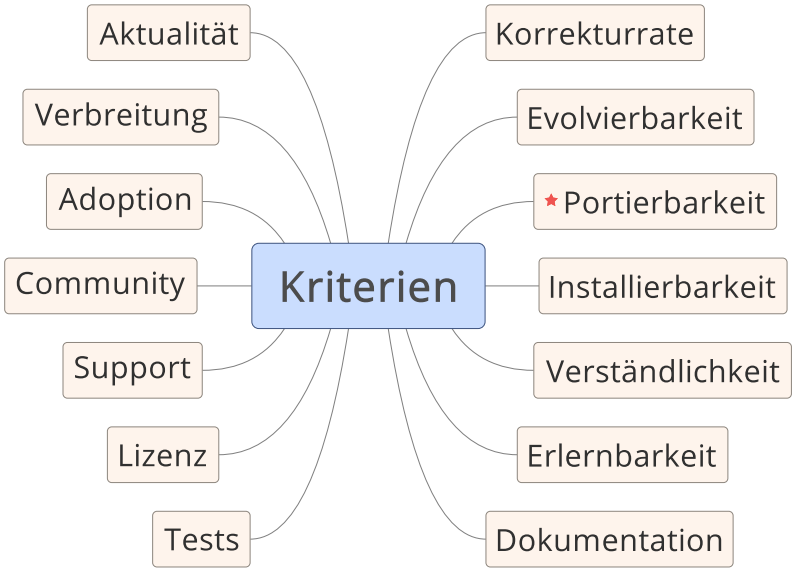



***Abbildung 15*** *Vergleichskriterien zur Bewertung von Open Source Frameworks*

## 6.3 Kriterienkatalog mit Fragen zur Bewertung

Zur Bewertung der Aspekte wird ein Kriterienkatalog definiert. Die Definition erfolgt in Anlehnung an die Beschreibung aus den 3 angegebenen Quellen.
Es werden je nach Kriterium entweder konkrete Metriken definiert oder jeweils maximal 4 ja/nein Fragen formuliert. Je nach positiv beantworteter Frage werden 1-4 Punkte vergeben. Es können pro Kriterium daher maximal 4 Punkte erzielt werden.

### 6.3.1 Verbreitung

Es wird als Quelle das Ranking von Google Trends der betrachteten Frameworks betrachtet. Es werden, sofern vorhanden, jeweils die 4 verbreitesten Java und .NET Frameworks verglichen.
Rang 1 (4 Punkte)
Rang 2 (3 Punkte)
Rang 3 (2 Punkte)
Rang 4 (1 Punkt)
alle übrigen (0 Punkte)

### 6.3.2 Portierbarkeit

Portierbarkeit ist ein KO Kriterium. Es wird zwingend erwartet, dass das Framework Cross-Plattform fähig ist. Java Frameworks erfüllen dieses Kriterium gänzlich, während im Fall von .NET dieses Kriterium nur für Frameworks erfüllt ist, welche auf .NET Core aufbauen. (siehe

hierzu Kapitel 3.2.6). Es werden nur solche Frameworks untersucht, welche .NET Core unterstützen!

### 6.3.3 Aktualität

Wie aktuell ist das neueste stabile Release?
weniger als 1 Woche (4 Punkte)
weniger als 1 Monat (3 Punkte)
weniger als 6 Monat (2 Punkte)
weniger als 1 Jahr (1 Punkt)
weniger als 2 Jahre (0 Punkte)

### 6.3.4 Adoption

Das Kriterium Adoption beschreibt, wie gut das Framework von der Community, dem Markt und der Industrie angenommen wurde. (Doumack, 2018) misst hier die Anzahl veröffentlichter Bücher, bzw. ob es veröffentlichte Berichte über einen praktischen Einsatz gibt. Es wird im Folgenden die Anzahl der Github Stars gemessen.
> 5000 (4 Punkte)
> 2500 (3 Punkte)
> 1000 (2 Punkte)
> 500 (1 Punkt)
< 500 (0 Punkte)

### 6.3.5 Community

Anzahl von eindeutigen Quellcode-Beitragenden in den letzten 6 Monaten?
>= 50 (4 Punkte)
20-49 (3 Punkte)
10-19 (2 Punkte)
1-9 (1 Punkt)
1 (0 Punkte)

### 6.3.6 Support

Verhältnis geschlossener oder beantworteter Anfragen zu gesamten Anfragen
> 95% (4 Punkte)
> 90% (3 Punkte)
> 85% (2 Punkt)
> 80% (1 Punkte)
< 80% (0 Punkte)

### 6.3.7 Lizenz[22]

Lizenz ohne Copyleft Effekt [z.B. MIT] (4 Punkte)
Lizenz mit beschränktem Copyleft Effekt [z.B. Apache License] (3 Punkte)
Lizenz mit Wahlmöglichkeiten/Lizenz mit Sonderrechten [z.B. LGPL] (2 Punkte)
Lizenz mit strengem Copyleft Effekt [z.B. GPL, CPL, EUPL] (1 Punkt)
keine Lizenz (0 Punkte)

### 6.3.8 Tests

Gibt es Tests? (2 Punkt)
Gibt es einen automatisierten Buildprozess zur Ausführung der Tests (2 Punkte)

### 6.3.9 Korrekturrate

Verhältnis Anzahl behobener Bugs zu Anzahl gemeldeter Bugs.
> 90% (4 Punkte)
> 70% (3 Punkte)
> 50% (2 Punkte)
> 30% (1 Punkt)
< 30% (0 Punkte)

### 6.3.10 Evolvierbarkeit

Gibt es definierte Meilensteine bzw. einen Releaseplan? (2 Punkt)
Werden neue Versionen angekündigt (1 Punkt)
Gibt es Angaben über die Finanzierung des Projektes (1 Punkt)

### 6.3.11 Installierbarkeit

Framework lässt sich über Paket-Manager (z.B. Nuget, Maven) bzw. über Installerwebsite [z.B. Spring Initializr] installieren (4 Punkte)
Framework lässt sich in kompilierter Form herunterladen [Zip bzw Tar File] (2 Punkt)
Es gibt eine detaillierte Anleitung zur Installation (1 Punkt)

### 6.3.12 Verständlichkeit

Gibt es eine abstrakte Beschreibung des Verwendungszwecks? (1 Punkt)
Gibt es Architekturübersichtsdiagramme? (1 Punkt)
Gibt es Beispielimplementierungen? (1 Punkt)
Gibt es Beschreibungen von möglichen Einsatzszenarios? (1 Punkt)

[22] Siehe hierzu (it-economics, 2018) und (Luckart, 2018)

### 6.3.13 Erlernbarkeit

Gibt es Video Tutorials (2 Punkt)
Gibt es eine Quickstart Anleitung (1 Punkt)
Gibt es schriftliche Beschreibungen zu den wichtigsten Use Cases (1 Punkt)

### 6.3.14 Dokumentation

Gibt es eine ausführliche strukturierte Dokumentation? (1 Punkt)
Gibt es eine Projektwebsite? (1 Punkt)
Gibt es eine API Dokumentation? (1 Punkt)
Gibt es Bücher (Quelle Amazon)? (1 Punkte)

# 7 BEWERTUNG VON OPEN SOURCE FRAMEWORKS IN JAVA UND .NET ZUR ANBINDUNG AN OPEN SOURCE INFRASTRUKTURKOMPONENTEN EINER MICROSERVICE ARCHITEKTUR MIT HILFE AUSGEWÄHLTER VERGLEICHSKRITERIEN

**Abbildung 16** enthält eine Übersicht der Infrastrukturkomponenten, welche in der Fachliteratur zu Microservices erwähnt werden und welche in diesem Kapitel untersucht werden.

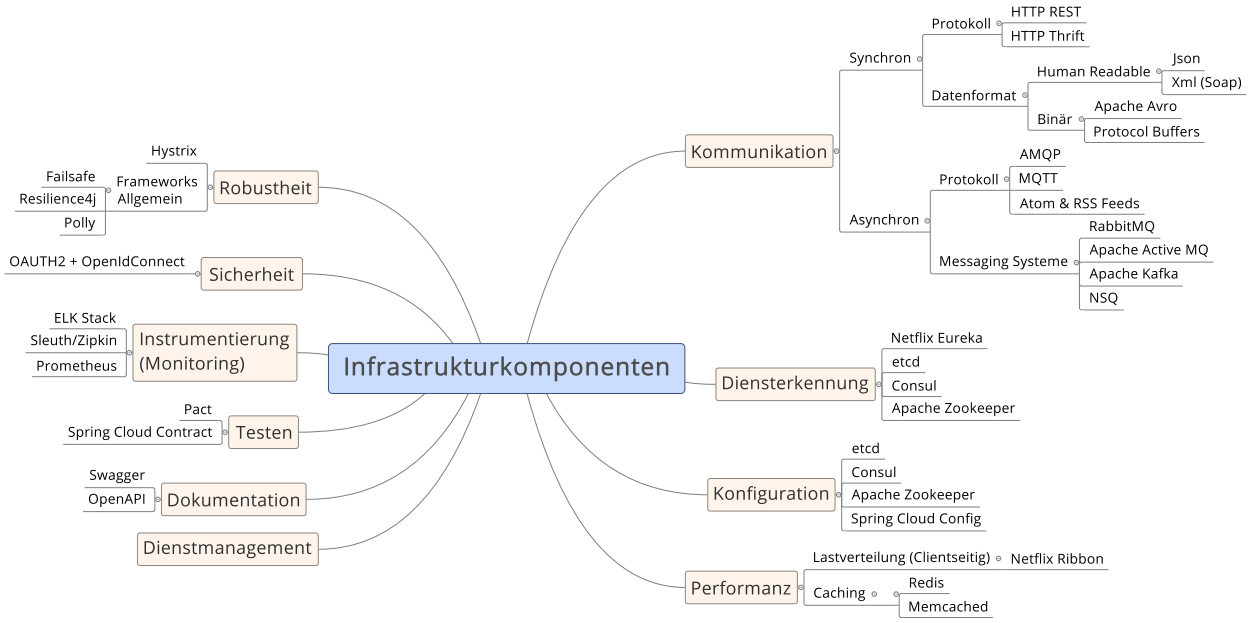



***Abbildung 16*** *Open Source Infrastrukturkomponenten*

Die einzelnen Teilkapitel enthalten jeweils eine Tabelle mit der Auswertung der Frameworks nach den vorgegebenen Kriterien und eine kurze Auswertung. Die Zahlen, Daten und Fakten, welche als Grundlage zur Bewertung verwendet wurden, sind im Anhang zu finden. Alle gefundenen Frameworks werden aufgelistet und die Verbreitung anhand Google Trends ermittelt. Zu den meist verbreitetsten Frameworks ist eine Steckbrief in Tabellenform zu finden. Sofern vorhanden, werden auch entsprechende Links zu Benchmarks angegeben.

## 7.1 Kommunikation

### 7.1.1 Http Rest

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring Boot** | 4 | 3 | 4 | 4 | 4 | 3 | 4 | 4 | 3 | 4 | 4 | 4 | 4 | **3,76** |
| **Akka** | 3 | 3 | 4 | 2 | 4 | 3 | 2 | 4 | 3 | 4 | 3 | 4 | 4 | 3,3 |
| **ASP.Net Core Web API** | 2 | 2 | 4 | 2 | 4 | 3 | 4 | 4 | 3 | 4 | 4 | 4 | 4 | 3,4 |
| **ServiceStack** | 1 | 2 | 3 | 2 | - | 1 | 2 | - | 1 | 4 | 4 | 4 | 3 | 2,5 |

***Tabelle 3*** *Vergleich Frameworks Http Rest*

Java bietet eine große Anzahl an Frameworks zur Unterstützung von http Rest an.
.NET bietet wesentlich weniger diesbezüglich an und portierbar sind noch weniger.
ASP.NET Core Web API kann mit den beiden „großen“ Java Frameworks gut mithalten, während das teilweise kostenpflichtige ServiceStack aufgrund der Aktualität, Lizenz und Evolvierbarkeit schlechter ausfällt.

### 7.1.2 Http Thrift[23]

Thrift stammt ursprünglich von Facebook und ist gemäß (Bayer, 2018) im Apache Inkubator enthalten und unterstützt mehrere Sprachen. Darunter auch Java und C#

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Apache Thrift Java** | 4 | 0 | 0 | 1 | 0 | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0,84 |
| **Apache Thrift NetCore** | 3 | 0 | 0 | 1 | - | 3 | 2 | 4 | 0 | 0 | 0 | 0 | 0 | 1,33 |
| **Thrift-netcore** | 2 | 0 | 0 | 0 | 4 | 2 | 4 | - | 0 | 4 | 0 | 0 | 0 | **1,5** |

***Tabelle 4*** *Vergleich Frameworks http Thrift*

Thrift ist ein Protokoll, welches sowohl von Java als auch .NET wenig verbreitet ist.
Für Java existiert offenbar nur das offizielle von Apache zur Verfügung gestellte Framework, während es für .NET zusätzlich noch ein Framework eines einzelnen Entwicklers gibt.
Die Frameworks sind alle kaum beschrieben. Es gibt 2 Bücher, welche Beispiele für Java

[23] Siehe https://de.wikipedia.org/wiki/Apache_Thrift

haben. Die Frameworks von Apache sind nur schlecht installierbar und es gibt keinen Support, daher fällt die Bewertung hierfür sehr schlecht aus.

### 7.1.3 Json[24]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gson** | 4 | 3 | 4 | 1 | 3 | 3 | 0 | 3 | 0 | 4 | 2 | 4 | 3 | 2,61 |
| **Jackson** | 3 | 3 | 3 | 1 | 4 | 3 | 0 | 3 | 0 | 4 | 3 | 3 | 2 | 2,46 |
| **Newtonsoft Json.NET** | 2 | 3 | 4 | 1 | 3 | 4 | 4 | 3 | 0 | 4 | 3 | 4 | 3 | **2,69** |
| **LitJson** | 1 | 2 | 0 | 1 | 0 | 4 | 4 | 2 | 0 | 4 | 1 | 2 | 2 | 1,77 |

***Tabelle 5*** *Vergleich Frameworks Json*

Das .NET Framework Newtonsoft Json.NET ist den Java Frameworks knapp überlegen.

### 7.1.4 Xml (Soap[25])

Xml selbst ist in den Platform Frameworks sowohl in Java als auch .Net eingebaut.
In .Net Core wird Xml allerdings erst seit kurzem unterstützt. Mehr Details hierzu findet man in Anhang C.4.2. Ein Vergleich zu Frameworks bzgl. Xml selbst wäre an dieser Stelle zu aufwendig.

Es gibt zahlreiche weitere Implementierungen zu Xml, welche in Anhang C.4 aufgelistet sind. Zu den meisten Implementierungen wurden keine entsprechenden kompatiblen Frameworks in .NET gefunden.

Eine wichtige Implementierung im Zusammenhang mit Microservices stellt das Soap Protokoll zur Realisierung von Web Services dar.
In Java gibt es hierfür das Metro JAX-WS Framework.
In .NET existierte hierfür in der Vergangenheit das von Microsoft angebotenen „Windows Communication Foundation“ (WCF) Framework. Im Augenblick gibt es große Anstrengungen dessen Funktionalität in .NET Core zu übernehmen. Einerseits durch einen von Microsoft angebotenen „Compatibility Pack“ und andererseits durch ein Community Projekt für .NET Core. Weitere Informationen hierzu findet man im Anhang C.4.2.

[24] Siehe https://de.wikipedia.org/wiki/JavaScript_Object_Notation

[25] Siehe https://de.wikipedia.org/wiki/SOAP

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Metro JAX-WS | 4 | 3 | 0 | 1 | 4 | 1 | 2 | 3 | 1 | 4 | 4 | 4 | 3 | 2,61 |
| WCF .NET Core | 3 | 3 | 1 | 2 | 4 | 4 | 4 | 4 | 3 | 4 | 4 | 4 | 4 | **3,38** |

***Tabelle 6*** *Vergleich Frameworks Xml*

Bezüglich Xml (Soap) gewinnt .NET deutlich vor Java. Dies liegt an der geringen Adoption, der Lizenz und der Evolvierbarkeit. In .NET steht eine größere Community hinter dem Framework, welche ein sehr großes Interesse an der Fertigstellung des WCF Frameworks hat. Es besteht auch die Möglichkeit, dass im Java Bereich eher andere Techniken Verwendung finden als Soap Web Services.

### 7.1.5 Apache Avro[26]

Das offizielle Apache Github Repository von Avro bietet sprachabhängige Implementierungen für das Protokoll. Die Implementierung für C# ist veraltet und nicht kompatibel.
Für Java gibt es nur das unten aufgeführte Framework von Apache.
Für .NET existieren 2 Frameworks, welche kompatibel zu .NET Core sind. Weitere Informationen hierzu sind in Anhang C.5.2 zu finden.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Apache Avro Java | - | 1 | 0 | 1 | 0 | 3 | 2 | - | 0 | 4 | 2 | 3 | 3 | **1,72** |
| Microsoft.Avro | - | 1 | 0 | 0 | 0 | 2 | 2 | 2 | 0 | 4 | 0 | 0 | 1 | 1,00 |
| Apache-Avro-Core | - | 0 | 0 | 0 | 0 | 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0,41 |

***Tabelle 7*** *Vergleich Frameworks Apache Avro*

Das Java Framework hat zwar eine geringe Aktualität und auch Adoption, ist aber im Vergleich zu den .NET Frameworks wesentlich besser. Diese sind weder verständlich, noch erlernbar und Dokumentation ist nahezu nicht vorhanden oder schwer auffindbar. Das von Microsoft zur Verfügung gestellte Projekt ist zwar vorhanden, wird aber kaum genutzt. Das andere Projekt wurde von einem einzelnen Entwickler zur Verfügung gestellt und erfüllt nahezu keine Kriterien.

[26] Siehe https://de.wikipedia.org/wiki/Apache_Avro

### 7.1.6 Protocol Buffers[27]

Für Protocol Buffers gibt es von Google ein Github Repository mit Implementierungen für diverse Sprachen. Neben Java auch für C#. Die Implementierung für C# unterstützt auch .NET Core. Für .NET existiert noch ein weiteres Framework, welches allerdings nicht .NET Core kompatibel ist. Die Auswertung von Java und .NET anhand des einen Github Repositories sind in **Tabelle 8** abgebildet. Die Daten hierzu sind in Anhang C.6 hinterlegt.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Protocol Buffers (Java/.NET)** | - | 2 | 4 | 2 | 4 | 3 | 4 | 3 | 1 | 4 | 3 | 4 | 3 | **3,08** |

***Tabelle 8*** *Auswertung Protocol Buffers Java und .NET*

Sowohl Java und .NET unterstützen Protocol Buffers gleich gut in sehr gutem Maß.

### 7.1.7 AMQP[28]

AMQP ist ein Protokoll, welches hauptsächlich zur Anbindung an RabbitMQ verwendet wird. Sowohl für Java und .NET wurde jeweils nur ein Framework gefunden, welches explizit zur Verwendung mit AMQP gedacht ist. Frameworks zur Anbindung an RabbitMQ verwenden implizit das AMQP Protokoll und werden in Kapitel 7.1.10 betrachtet.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring AMQP** | 4 | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 1 | 4 | 3 | 4 | 4 | **3,25** |

***Tabelle 9*** *Auswertung Frameworks AMQP*

Ein Vergleich von Frameworks ist für AMQP leider nicht möglich. Es existiert zwar ein einziges entsprechendes Framework für .NET, welches auch .NET Core unterstützt. Dort sind jedoch die wichtigsten Features noch nicht vorhanden. (siehe hierzu Anhang C.7.2.1).
Das in Spring Boot enthaltene AMQP Projekt erfüllt die Kriterien in sehr gutem Maß.

### 7.1.8 MQTT[29]

MQTT ist ein Protokoll, welches im Bereich „Internet of Things“ große Bedeutung hat.

[27] Siehe https://de.wikipedia.org/wiki/Protocol_Buffers

[28] Siehe https://de.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

[29] Siehe https://de.wikipedia.org/wiki/MQTT

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Fusesource mqtt-client** | - | 1 | 1 | 0 | 0 | 3 | 2 | 0 | 0 | 4 | 1 | 4 | 0 | 1,17 |
| **Eclipse Paho MQTT** | - | 2 | 1 | 1 | 3 | 1 | 4 | 3 | 0 | 4 | 1 | 3 | 4 | **2,25** |
| **MQTTnet** | - | 4 | 0 | 1 | 3 | 4 | 4 | 4 | 0 | 4 | 1 | 1 | 1 | **2,25** |
| **M2qtt Dotnet Core** | - | 3 | 0 | 0 | - | 1 | 0 | - | 0 | 4 | 4 | 1 | 1 | 1,40 |

***Tabelle 10*** *Vergleich Frameworks MQTT*

Bei den MQTT Frameworks sind Java und .NET gleich gut. Das offizielle Java Framework von Eclipse ist besser erlernbar und hat auch die bessere Dokumentation, während das .NET Framework MQTTnet wesentlich aktueller ist und die bessere Lizenz hat.

### 7.1.9 Atom und RSS Feeds[30]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Rome** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 1 | 4 | 4 | 2 | 2 | **2,67** |
| **Ms.Syndication.ReaderWriter** | - | 3 | 0 | 1 | 4 | 4 | 2 | 4 | 1 | 4 | 2 | 1 | 0 | 2,17 |
| **CodeHallow FeedReader** | - | 3 | 0 | 1 | 3 | 4 | 2 | 3 | 0 | 4 | 2 | 1 | 0 | 1,92 |

***Tabelle 11*** *Vergleich Frameworks Atom und RSS Feeds*

Bei Atom und RSS Feeds gewinnt Java deutlich vor .NET. Das Java Framework ist besser verständlich, leicher erlernbar und bietet mehr Dokumentation. Das von .NET angebotene Microsoft.Syncication.ReaderWriter Projekt ist eine Portierung der Klassen aus dem WCF Framework, welches diese Funktionalität eingebaut hatte. Es existiert, weil die .NET Entwicklergemeinde ein starkes Interesse daran hat, dass diese für .NET Core zur Verfügung gestellt wird.

[30] Siehe https://de.wikipedia.org/wiki/Atom_(Format) und https://en.wikipedia.org/wiki/Atom_(Web_standard)

### 7.1.10 RabbitMQ[31]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RabbitMq Java Client | 2 | 4 | 0 | 1 | 4 | 1 | 2 | 4 | 1 | 4 | 1 | 2 | 3 | 2,23 |
| Apache Camel RabbitMQ | 3 | 3 | 2 | 4 | - | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | **3,58** |
| RabbitMQ .NET Client | 4 | 2 | 1 | 1 | 4 | 3 | 0 | 4 | 1 | 4 | 1 | 1 | 1 | 2,08 |
| EasyNetQ | 3 | 3 | 2 | 1 | 4 | 4 | 2 | 3 | 1 | 4 | 3 | 4 | 2 | 2,78 |

***Tabelle 12*** *Vergleich Frameworks RabbitMQ*

Das Camel Framework zur unterstützung von RabbitMQ gewinnt deutlich vor den anderen Frameworks. Dies liegt an einer sehr starken Community, der hohen Evolvierbarkeit und der besten Kriterien bezüglich Installierbarkeit, Verständlichkeit, Erlernbarkeit und Dokumentation.

### 7.1.11 Apache Active MQ[32]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ActiveMQ Java Client | 4 | 3 | 2 | 1 | - | 3 | 2 | 4 | 0 | 4 | 2 | 4 | 4 | **2,75** |

***Tabelle 13*** *Vergleich Frameworks ActiveMQ*

Für die direkte Unterstützung von ActiveMQ wurden nur die offiziellen von Apache zur Verfügung gestellten Frameworks gefunden. Das .NET Framework erfüllt bislang nicht das Kriterium der Portabilität. Java bietet hierfür einzig allein Unterstützung. Für .NET Projekte muss auf die Low-Level Protokolle wie AMQP, Stomp, JMS ausgewichen werden. (siehe Kapitel 7.1.7)

[31] Siehe https://de.wikipedia.org/wiki/RabbitMQ

[32] Siehe https://de.wikipedia.org/wiki/Apache_ActiveMQ

### 7.1.12 Apache Kafka[33]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring Kafka** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 3 | 4 | 1 | 4 | 4 | **2,92** |
| **Vertx-Kafka-Client** | - | 3 | 0 | 1 | 3 | 3 | 4 | 3 | 0 | 4 | 1 | 4 | 2 | 2,33 |
| **Confluent-Kafka-Dotnet** | - | 2 | 0 | 1 | 4 | 3 | 4 | 3 | 1 | 4 | 2 | 1 | 1 | 2,17 |

***Tabelle 14*** *Vergleich Frameworks Apache Kafka*

Die Unterstützung für Apache Kafka ist mit Java Frameworks wesentlich besser als in .NET. Für .NET steht bislang nur das Framework von Confluent zur Verfügung, welches .NET Core unterstützt.

### 7.1.13 NSQ[34]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **JavaNSQClient** | - | 1 | 0 | 0 | 3 | 4 | 2 | 3 | 0 | 4 | 0 | 0 | 0 | 1,42 |
| **Nsq-java** | - | 1 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 4 | 0 | 0 | 0 | 0,75 |
| **NSQCore** | - | 1 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 4 | 1 | 0 | 0 | 0,84 |
| **ZeroNSQ** | - | 3 | 0 | 0 | 4 | 4 | 2 | 3 | 0 | 4 | 1 | 0 | 0 | **1,75** |

***Tabelle 15*** *Vergleich Frameworks NSQ*

Die Unterstützung für NSQ ist sowohl von Java als auch NSQ noch sehr verbesserungswürdig. Trotzdem liegt hier .NET vorne. Dies liegt insbesondere daran, dass ZeroNSQ wesentlich aktueller ist. Die aktuelle Version, welche zur Verfügung steht, ist eine Prerelease Version. Diese wird jedoch laut WebSite bereits in Produktion eingesetzt. Die Java Frameworks erscheinen vollkommen veraltet. Die Qualität der zur Verfügung gestellten Unterlagen ist bei allen Frameworks ausgesprochen schlecht.

[33] Siehe https://en.wikipedia.org/wiki/Apache_Kafka

[34] Siehe http://nsq.io/

## Zusammenfassung

Von den untersuchten Protokollen für für den Aspekt Kommunikation werden 7 besser von Java Frameworks und 4 besser von .NET Frameworks unterstützt. Bei zwei Protokollen ist die Unterstützung gleich gut. Spring AMQP und Apache ActiveMQ werden von .NET nicht unterstützt!

## 7.2 Diensterkennung

### 7.2.1 Netflix Eureka

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring Cloud Netflix Eureka** | - | 3 | 2 | 2 | 4 | 3 | 4 | 3 | 3 | 4 | 3 | 4 | 3 | **3,17** |
| **Steeltoe Discovery** | - | 3 | 0 | 1 | 4 | 3 | 4 | 3 | 1 | 4 | 3 | 4 | 2 | 2,67 |

***Tabelle 16*** *Vergleich Frameworks Netflix Eureka*

Netflix Eureka wird für .NET nur von Steeltoe Discovery unterstützt. Das von Netflix zur Verfügung gestellte Framework, welches in Spring Cloud integriert wurde, gewinnt deutlich. Dies liegt insbesondere an der Adoption, was sicher auf den noch geringen Bekanntheitsgrad von Steeltoe Discovery zurückzuführen ist.

### 7.2.2 Etcd[35]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Jetcd Core** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 1 | 1 | **2,25** |
| **Etcd4j** | - | 2 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 1 | 2 | 0 | 2,08 |
| **EtcdGrpcClient** | - | 2 | 0 | 0 | - | 4 | 4 | - | 0 | 4 | 1 | 2 | 0 | 1,7 |

***Tabelle 17*** *Vergleich Frameworks Etcd*

Die Unterstützung von Etcd ist mit Java Frameworks wesentlich besser als mit .NET. .NET bietet ein einziges .NET Core kompatibles Framework hierfür an. Die aktuelle Version wurde auf Nuget zur Verfügung gestellt, ist jedoch kein offizielles Release. Die Beteiligung und das Interesse an diesem Projekt sind sehr gering.

[35] Siehe https://wikitech.wikimedia.org/wiki/Etcd

### 7.2.3 Consul[36]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring Cloud Consul** | - | 3 | 0 | 1 | 4 | 3 | 4 | 3 | 0 | 4 | 2 | 4 | 3 | **2,58** |
| **Consul Client** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 1 | 2 | 0 | 2,16 |
| **Microdot** | - | 3 | 1 | 1 | 4 | 3 | 2 | 4 | 0 | 4 | 3 | 1 | 0 | 2,16 |
| **Consul.NET** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 1 | 4 | 1 | 1 | 0 | 2,16 |

***Tabelle 18*** *Vergleich Frameworks Consul*

Zur Unterstützung von Consul gibt es sowohl für Java, als auch .NET eine ganze Reihe von Frameworks. Siehe Anhang D.3. Das in Spring Clound integrierte Framework für Consul für Java erfüllt die Kriterien wesentlich besser als alle anderen Frameworks.

### 7.2.4 Apache Zookeeper[37]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Apache Zookeeper** | 4 | 2 | 3 | 1 | - | 3 | 2 | 2 | 2 | 4 | 2 | 4 | 4 | **2,75** |
| **Spring Cloud Zookeeper** | 3 | 3 | 1 | 1 | 4 | 3 | 2 | 4 | 0 | 4 | 2 | 2 | 3 | 2,46 |
| **Zookeeper .NET Client** | 2 | 2 | 0 | 1 | 3 | 3 | 2 | 3 | 0 | 4 | 1 | 0 | 0 | 1,75 |

***Tabelle 19*** *Vergleich Frameworks Apache Zookeeper*

Die Java Frameworks erfüllen die Kriterien zur Unterstützung von Apache Zookeeper deutlich besser als das einzige zur Verfügung stehende .NET Framework.

## Zusammenfassung

Die Kriterien für den Aspekt Diensterkennung werden von den Java Frameworks eindeutig besser erfüllt als von den vergleichbaren .NET Frameworks.

[36] Siehe https://www.consul.io/

[37] Siehe https://en.wikipedia.org/wiki/Apache_ZooKeeper

## 7.3 Konfiguration

Etcd und Consul können ebenfalls als Infrastrukturkomponenten eingesetzt werden um den Aspekt Konfiguration abzudecken. Diese wurden bereits im vorangehenden Kapitel untersucht.

### 7.3.1 Spring Cloud Config[38]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring Cloud Config** | 4 | 3 | 1 | 2 | 4 | 3 | 4 | 3 | 0 | 4 | 2 | 4 | 3 | **2,85** |
| **Steeltoe Configuration** | 3 | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 1 | 4 | 2 | 2 | 2 | 2,54 |

***Tabelle 20*** *Vergleich Frameworks Spring Cloud Config*

Das Steeltoe Framework in .NET bietet einzig und alleine einen Configuration Provider zur Unterstüzung von Spring Cloud Config an. Trotzdem erfüllt es die Kriterien nicht ganz so gut, wie das orginal Java Framework.

#### Zusammenfassung

Die Java Frameworks erfüllen die Kriterien bezüglich des Aspekts Konfiguration jeweils besser als die vorhandenen .NET Frameworks.

## 7.4 Performanz

### 7.4.1 Netflix Ribbon

Zur Umsetzung von Clientseitiger Lastverteilung wurde nur das von Netflix angebotene Java Framework Ribbon gefunden. Etwas Vergleichbares im .NET Bereich ist bisher nicht bekannt. Eine Umsetzung des Ribbon Frameworks in .NET mit Steeltoe ist jedoch gemäß (Brown & Horan, 2018, S. 31) angedacht.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Netflix Ribbon** | - | 3 | 2 | 1 | 1 | 3 | 2 | 2 | 0 | 4 | 3 | 4 | 3 | **2,15** |

***Tabelle 21*** *Auswertung Frameworks Ribbon*

[38] Siehe https://cloud.spring.io/spring-cloud-config/

### 7.4.2 Redis[39]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Jedis** | 4 | 1 | 4 | 2 | 4 | 4 | 4 | 4 | 0 | 4 | 1 | 2 | 2 | **2,77** |
| **Spring Redis** | 3 | 3 | 1 | 1 | 0 | 3 | 4 | 0 | 1 | 4 | 2 | 4 | 3 | 2,23 |
| **StackExchange.Redis** | 2 | 3 | 3 | 4 | 3 | 4 | 2 | 3 | 0 | 4 | 1 | 3 | 1 | 2,54 |
| **ServiceStack.Redis** | 2 | 3 | 2 | 1 | 0 | 1 | 2 | 0 | 1 | 4 | 4 | 4 | 3 | 2,08 |

***Tabelle 22*** *Vergleich Frameworks Redis*

Das Java Framework Jedis erfüllt die Kriterien am besten. Es besticht durch die hohe Adoption, einen hervorragenden Support und eine große Korrekturrate. Das .NET Framework StackExchange.Redis erfüllt die Kriterien ebenfalls sehr gut und besticht durch eine ungewöhnlich große Community.

### 7.4.3 Memcached[40]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Xmemcached** | - | 3 | 1 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 1 | 2 | 2 | **2,41** |
| **Spymemcached** | - | 0 | 0 | 0 | 0 | 4 | 2 | 1 | 0 | 4 | 1 | 0 | 1 | 1,08 |
| **EasyCaching** | - | 3 | 0 | 0 | 4 | 4 | 4 | 4 | 0 | 4 | 2 | 2 | 1 | 2,33 |
| **EnyimMemcachedCore** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 1 | 0 | 2,16 |

***Tabelle 23*** *Vergleich Frameworks Memcached*

Das Java Framework Xmemcached erfüllt die Kriterien am besten. Die .NET Frameworks sind nur wenig schlechter, während das Java Framework Spymemcached vollkommen veraltet ist und keine aktive Community mehr hat.

[39] Siehe https://de.wikipedia.org/wiki/Redis

[40] Siehe https://de.wikipedia.org/wiki/Memcached

## Zusammenfassung

Java bietet mit Ribbon einzig und allein gegenwärtig clientseitige Lastverteilung an.
Die gängigsten Key-/Value Stores werden von Java jeweils besser unterstützt als von .NET.

## 7.5 Robustheit

Um Robustheit in den Microservices zu Unterstützen hat sich Hystrix als Standard etabliert. Dieses bietet auch die Möglichkeit die Zustände der Circuit Breaker mittels eines Dashboards zu überwachen.

### 7.5.1 Hystrix

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Netflix Hystrix** | - | 2 | 4 | 1 | 4 | 3 | 4 | 3 | 1 | 4 | 3 | 4 | 2 | **2,92** |
| **Steeltoe.Circuit Breaker** | - | 2 | 0 | 1 | 4 | 3 | 4 | 3 | 1 | 4 | 3 | 2 | 2 | 2,42 |
| **Hystrix.Dotnet** | - | 2 | 0 | 1 | 3 | 4 | 4 | 2 | 0 | 4 | 2 | 2 | 0 | 2 |

***Tabelle 24*** *Vergleich Frameworks Hystrix*

Das orginal von Netflix angebotene Hystrix Framework in Java gewinnt deutlich vor den .NET Frameworks. Es hat eine extrem hohe Adoption. Hystrix wird zwar in beiden .NET Frameworks angeboten, aber eine Anbindung an das Dashboard ist nur mit dem Steeltoe Framework möglich.

### 7.5.2 Frameworks Allgemein

Nicht alle untersuchten Frameworks bieten alle Implementierungen für Robustheit. Daher wird in **Tabelle 25** eine Übersicht darüber gegeben.

| Framework / Pattern | Timeout | Retry | Circuit Breaker | Bulkhead |
|---|---|---|---|---|
| **Failsafe** | ✗ | ✓ | ✓ | ✗ |
| **Resilience4j** | ✓ | ✓ | ✓ | ✓ |
| **Spring Circuit Breaker** | ✗ | ✗ | ✓ | ✗ |
| **Vertx Circuit Breaker** | ✗ | ✗ | ✓ | ✗ |
| **Akka** | ✗ | ✗ | ✓ | ✗ |
| **Polly** | ✓ | ✓ | ✓ | ✓ |

***Tabelle 25*** *Frameworks Robustheit Allgemein und implementierte Patterns.*

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Failsafe** | 4 | 3 | 2 | 2 | 2 | 3 | 4 | 2 | 1 | 4 | 1 | 2 | 1 | 2,38 |
| **Resilience4j** | 3 | 4 | 1 | 2 | 4 | 3 | 4 | 4 | 0 | 3 | 2 | 2 | 1 | 2,62 |
| **Polly** | 2 | 3 | 3 | 2 | 4 | 4 | 4 | 4 | 2 | 4 | 4 | 4 | 2 | **3,15** |

***Tabelle 26*** *Vergleich Frameworks Robustheit Allgemein*

Das .NET Framework Polly setzt nicht nur die meinesten beschriebenen Patterns bezüglich Robstheit um, sondern erfüllt von allen Frameworks die Kriterien am besten. Polly hat eine hohe Adoption, bietet sehr guten Support, ist sehr gut verständlichlich und erlenbar. Die Java Frameworks erfüllen die Kriterien auch gut, bieten jedoch, abgesehen von Resilience4j, nicht alle beschriebenen Patterns bzgl. Robustheit an.

### Zusammenfassung

Die Kriterien für den Aspekt Robustheit werden vom .NET Framework Polly am besten erfüllt. Polly bietet jedoch im Gegensatz zu Hystrix keine Möglichkeit zur Anbindung an ein Dashboard zur Überwachung der Circuit Breaker. Hystrix selbst implementiert alle beschrieben Patterns bezüglich Robustheit und erfüllt die Kriterien besser als die vergleichbaren .NET Frameworks.

## 7.6 Sicherheit

In Anhang H - werden einerseits Frameworks aufgelistet, mit denen sich Identity Provider, welche das OpenIdConnect Protokoll implementieren und andererseits Client Frameworks mit denen Microservices an diese anbinden können. Diese Frameworks dienen hauptsächlich dazu User zu authentifizieren, also das Login mit Hilfe des OpenID Connect Protokolls durchzuführen. Es existieren hierfür sowohl Frameworks in Java als auch .NET.
Identity Provider müssen nicht zwingend in Java bzw. .NET implementiert sein. Diese können in einer beliebigen Technolgie umgesetzt sein, sofern diese das OpenIDConnect Protokoll implementieren.

Der Login Vorgang findet typischerweise im API-Gateway statt und endet mit einem Json Web Token, welches dann für den Aufruf an andere Microservices benötigt wird.

Jeder Microservice hat dann die Aufgabe dieses Token zu überprüfen und entsprechend weiterzureichen. In Anhang H.2 sind entsprechende Frameworks zum Verarbeiten von Json Web Tokens aufgelistet.

Mit Hilfe der im Token enthaltenen Claims kann jeder Microservice eine entsprechende Autorisierung durchführen. Diese Aufgabe ist in den entsprechenden Technologien bereits in den Basis Frameworks sowohl in Java als auch .NET hinreichend gut umgesetzt und **nicht** Microservice spezifisch.

Unterstützung würde man sich jedoch für das automatische Weiterleiten des Tokens (Token Relay) wünschen.

Spring Cloud Security bietet hier die Möglichkeit mit Hilfe von Annotationen das Token an den nächsten aufrufenden Service durchzureichen[41].
.NET bietet hierzu im Augenblick Framework Unterstützung.

## Zusammenfassung

Ein Vergleich von Frameworks für Sicherheit mit Hilfe der aufgeführten Kriterien ist nicht gut möglich, da die Funktionalität für Sicherheit in den Basis Frameworks enthalten ist.
Diese Funktionalität ist auch nicht Microservice spezifisch und lässt sich ebenfalls schlecht vergleichen.

Sofern Sicherheit mit Hilfe eines standardisierten Protokolls (OAuth2 und OpenID Connect) realisiert wird, sollte es wenig Probleme bei der Umsetzung geben.

Token Forwarding bzw. Token Relay wird von Java Frameworks wie z.B. Spring Clound Security bereits angeboten, während .NET hier noch nicht so weit ist.

Ebenfalls negativ aufgefallen ist bei .NET, dass sich die API bei der Umstellung von .NET Core 1.x auf .NET Core 2.x erheblich geändert hat. Dies erschwert die Einarbeitung enorm, da viele Methoden obsolet sind und in der Dokumentation mühsam nach den alternativen Methodenaufrufen gesucht werden muss.

[41] Siehe Spring Cloud Security Dokumenation

## 7.7 Testen (CDC)

Zur Umsetzung von Consumer Driven Contracts gibt es zum aktuellen Zeitpunkt 2 Technologien:

- Pact
- Spring Cloud Contract

### 7.7.1 Pact[42]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Pact-Jvm** | 4 | 3 | 0 | 2 | 4 | 3 | 4 | 3 | 2 | 4 | 3 | 4 | 1 | **2,85** |
| **Pact-Net** | 3 | 4 | 0 | 1 | 4 | 4 | 4 | 4 | 1 | 4 | 3 | 4 | 1 | **2,85** |

***Tabelle 27*** *Vergleich Frameworks Pact*

Obwohl das Java Framework Pact-Jvm bereits länger existiert, eine größere Community und auch eine höhere Evolvierbarkeit hat, erfüllt das .NET Framework Pact-Net die Kriterien insgesamt gleich gut. Hervorzuheben ist insbesondere, dass das Framework für .NET Core just an dem Tag der Untersuchung eine erste Release Version hervorbrachte.

### 7.7.2 Spring Cloud Contract[43]

Spring Cloud Config ist ein Teil des Spring Cloud Frameworks. Es ist insbesondere für die Verwendung in der Java Virtual Machine ausgelegt. Zur Anbindung von .NET aus wurden keine Möglichkeiten gefunden.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Spring-Cloud-Contract** | 4 | 4 | 0 | 3 | 4 | 3 | 4 | 4 | 0 | 4 | 3 | 4 | 3 | **3,08** |

***Tabelle 28*** *Bewertung Spring-Cloud-Contract*

[42] Siehe https://docs.pact.io/

[43] Siehe https://cloud.spring.io/spring-cloud-contract/

## Zusammenfassung

Consumer Driven Contracts werden seit kurzem auch von .NET Seite aus unterstützt.
Als Defacto Standard hat sich Pact etabliert, welches auch für andere Technologien Unterstützung anbietet. Sowohl Java als auch .NET erfüllen die Kriterien gleich gut. Das Spring Cloud Contract Framework ist nur für Java verfügbar.

## 7.8 Instrumentierung (Monitoring)

### 7.8.1 Logging Frameworks Allgemein

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Log4j 2** | 4 | 3 | 0 | 0 | - | 3 | 4 | 3 | 0 | 4 | 4 | 4 | 4 | 2,75 |
| **Logback** | 3 | 2 | 2 | 2 | - | 1 | 4 | 2 | 0 | 4 | 2 | 0 | 3 | 2,08 |
| **NLog** | 2 | 4 | 0 | 1 | 4 | 4 | 4 | 4 | 0 | 4 | 2 | 4 | 3 | 2,77 |
| **Serilog** | 1 | 2 | 2 | 2 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 4 | 2 | **2,83** |

***Tabelle 29*** *Vergleich Frameworks Logging Allgemein*

Bei den Logging Frameworks im Allgemeinen schneiden die .NET Frameworks besser ab, als die Java Frameworks. Dies liegt insbesondere am hervorragenden Support und der sehr hohen Korrekturrate.

### 7.8.2 ELK Stack[44]

Der ELK Stack besteht aus Elasticsearch als Suchmaschine und Datenbank, Logstash zur Logdatenaufbereitung und Aggregation, und Kibana als Visualisierungstool.

Logstash kann die Daten aus verschiedenen Quellen, wie Datenbank, Message Queues und Logdateien lesen.

(Wolff, Docker: zentralisiertes Logging mit dem ELK-Stack, 2018) schlägt vor, die Logdateien in ein Shared Volume zu schreiben, welches einerseits von den Containern mit den Microservices und andereseits von einem Container mit Logstash Installation geteilt wird.

Alle Microservices schreiben Logdateien in dieses Shared Volume und der Logstash Container wertet diese Dateien aus und speichert diese in einer Elasticsearch Instanz.

[44] Siehe https://www.elastic.co/guide/index.html

Das Erzeugen von Logdateien kann über die bekannten Logging Frameworks erfolgen. Allgemeine Logging Frameworks wurden im vorangegangen Abschnitt bereits untersucht und in Anhang J.1 findet sich eine Auflistung möglicher Logging Frameworks für Java und .NET.

### 7.8.3 Zipkin[45]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Brave** | - | 3 | 1 | 2 | 4 | 3 | 4 | 3 | 0 | 4 | 2 | 1 | 0 | 2,25 |
| **Spring Cloud Sleuth** | - | 3 | 1 | 3 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 4 | 3 | **2,91** |
| **Zipkin4Net** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 0 | 0 | 2,08 |

***Tabelle 30*** *Vergleich Frameworks Zipkin*

Die Kriterien werden vom Java Framework Spring Cloud Sleuth am besten erfüllt. Das .NET Framework Zipkin4Net erfüllt die Kriterien Erlernbarkeit und Dokumentation sehr schlecht.

### 7.8.4 Prometheus[46]

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Java_Client** | - | 3 | 0 | 1 | 4 | 3 | 4 | 4 | 0 | 4 | 1 | 3 | 1 | **2,33** |
| **Prometheus-net** | - | 2 | 0 | 2 | 4 | 4 | 2 | 4 | 0 | 4 | 1 | 2 | 0 | 2,08 |

***Tabelle 31*** *Vergleich Frameworks Prometheus*

Für Prometheus existiert nur jeweils ein Framework für Java, als auch .NET.
Die Java Client Library läßt sich sehr gut in andere vorhandene Java Frameworks einbinden.
In Anhang J.5.1 findet sich hierzu eine Auflistung.

Das Java Framework zur Anbindung an Prometheus erfüllt die Kriterien besser als das vergleichbare .NET Framework. Es ist etwas aktueller und auch besser erlernbar. Das .NET Framework schneidet bei den Tests schlechter ab und bietet fast keine Dokumentation.

[45] Siehe https://zipkin.io/

[46] Siehe https://prometheus.io/

## 7.9 Dokumentation

### 7.9.1 Swagger und OpenAPI[47]

Swagger und OpenAPI sind Spezifiaktionen zur Beschreibung einer RestAPI.
Diese Beschreibung ermöglicht es sowohl Menschen als auch Computern den Aufbau und die Verwendung einer vorhandenen RestAPI zu verstehen.
OpenAPI ist der Nachfolger von Swagger. Unter Swagger versteht man auch die zur Erstellung von Servern, Zugriffsclients und Testtools vorhandene Werkzeugkette.

| Framework / Kriterium | Verbreitung | Aktualität | Adoption | Community | Support | Lizenz | Tests | Korrekturrate | Evolvierbarkeit | Installierbarkeit | Verständlichkeit | Erlernbarkeit | Dokumentation | Gesamt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Springfox** | - | 2 | 3 | 3 | 4 | 3 | 4 | 4 | 0 | 4 | 3 | 3 | 3 | 3 |
| **Jobby-Swagger** | - | 2 | 1 | 2 | 4 | 3 | 4 | 4 | 0 | 4 | 2 | 2 | 3 | 2,58 |
| **Swashbuckle.AspNetCore** | - | 3 | 2 | 2 | 3 | 4 | 4 | 3 | 0 | 4 | 1 | 4 | 1 | 2,58 |
| **NSwag** | - | 3 | 2 | 3 | 4 | 4 | 4 | 3 | 1 | 4 | 4 | 4 | 1 | **3,08** |

***Tabelle 32*** *Vergleich Frameworks Swagger und OpenAPI*

Die Kriterien für Swagger und OpenAPI werden von dem .NET Framework NSwag am besten unterstützt. Springfox erfüllt ebenfalls die Kriterien recht gut.

#### Zusammenfassung

Für den Aspekt Dokumentation und Metadaten hat sich OpenAPI als Standard etabliert. Dieser wird bereits von sehr vielen Frameworks auch in anderen Technolgien unterstützt. Im vorliegenden Fall erfüllt das .NET Framework NSwag die Kriterien am besten hierfür.

[47] Siehe https://swagger.io/

## 7.10 Dienstmanagement

Für Dienstmanagement gibt es zurzeit keinen einheitlichen Standard.

Sowohl Java als auch .NET bieten Frameworkunterstützung für Steuerung bzw. Metadaten an. Da die Funktionalität vollkommen unterschiedlich ist und diese teilweise in den Basis Frameworks enthalten ist, lässt sich ein Vergleich dieser mit Hilfe der Vergleichskriterien nicht durchführen. In Anhang L - erfolgt eine Auflistung von Frameworks in den beiden Technolgien, welche diesen Aspekt prinzipiell unterstützen.

### Zusammenfassung

Für den Aspekt Dienstmanagement gibt es sowohl für Java als auch .NET nur sehr wenig Frameworks die hierfür Unterstützung bieten.

Für Java wurde nur das Spring Boot Actuator Modul gefunden, welches Teil des Spring Boot Frameworks ist. Die Qualtität dieses Frameworks lässt sich mit den beschriebenen Kriterien nicht untersuchen. Es bietet gemäß Dokumentation[48] recht viele Endpunkte zur Verwendung an.

Bei .NET bietet das Steeltoe Framework einige Endpunkte um diesen Aspekt zu unterstützen. Zusätzlich existieren zwei weitere Projekte auf Github, mit denen sich ein Health Check Endpoint dynamisch hinzufügen lässt. Allerdings wurde bei beiden Projekten noch keine offizielle Release Version zur Verfügung gestellt.

[48] Siehe https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html

# 8 GESAMTAUSWERTUNG

**Tabelle 33** zeigt in welchen Aspekten und Teilaspekten, welche Technologie die höhere Punktzahl erreicht hat.

| Aspekt | Teilaspekt | Technologie |
|---|---|---|
| Kommunikation | http Rest | Java |
| | http Thrift | .NET |
| | Json | .NET |
| | Xml (Soap) | .NET |
| | Apache Avro | Java |
| | Protocol Buffers | = |
| | AMQP | Java |
| | MQTT | = |
| | Atom & RSS Feeds | Java |
| | RabbitMQ | Java |
| | Apache Active MQ | Java |
| | Apache Kafka | Java |
| | NSQ | .NET |
| Diensterkennung | Netflix Eureka | Java |
| | Etcd | Java |
| | Consul | Java |
| | Apache Zookeeper | Java |
| Konfiguration | Spring Cloud Config | Java |
| Performanz | Netflix Ribbon | Java |
| | Redis | Java |
| | Memcached | Java |
| Robustheit | Hystrix | Java |
| | Frameworks Allgemein | .NET |
| Sicherheit | | Java |
| Testen | Pact | = |
| Instrumentierung | Logging Frameworks Allgemein | .NET |
| | Zipkin | Java |
| | Prometheus | Java |
| Dokumentation | Swagger und OpenAPI | .NET |
| Dienstmanagement | | |

***Tabelle 33*** *Technologiegewinner nach Teilaspekten*

Folgende Aspekte werden von .NET zurzeit nicht abgedeckt:

- Kommunikation – AMQP
- Kommunikation – Apache ActiveMQ
- Performanz – Netflix Ribbon
- Konfiguration – Spring Cloud Contract

**Tabelle 34** zeigt zusammenfassend in welchen Aspekten, welche Technolgie die höchste Punktzahl erreicht hat.

| Aspekt | Technologie |
|---|---|
| Kommunikation | Java |
| Diensterkennung | Java |
| Konfiguration | Java |
| Performanz | Java |
| Robustheit | = |
| Sicherheit | Java |
| Testen | = |
| Instrumentierung | Java |
| Dokumentation | .NET |
| Dienstmanagement | |

***Tabelle 34*** *Technologiegewinner nach Aspekten*

# 9 FAZIT UND AUSBLICK

Die Java Frameworks dominieren die überwiegende Anzahl an Aspekten. Diese sind:

- Kommunikation
- Diensterkennung
- Konfiguration
- Performanz
- Sicherheit
- Instrumentierung

.NET ist nur im Aspekt Dokumentation besser. In den Aspekten Robustheit und Testen sind beide Technolgien gleich gut vertreten. Über den Aspekt Dienstmanagement konnte keine Aussage getroffen werden.

**Zusammenfassend lässt sich nun sagen, dass Java in Bezug auf die Anbindung an Open Source Infrastrukturkomponenten die wesentlich besseren Frameworks bietet im Vergleich zu .NET.**

Mit dem Spring Framework hat man als Entwickler nahezu das Rundum Sorglos Paket gebucht. .NET bietet in fast allen Aspekten ebenfalls entsprechende Möglichkeiten zur Anbindung. Ursache für das schlechtere Abschneiden könnte darin begründet sein, dass .NET noch an der Umstellung vom .NET Full Framework auf .NET Core leidet. Viele Frameworks für .NET Core sind noch sehr jung und die .NET Entwicklergemeinde hat noch nicht akut den Bedarf danach.

Um die Frage des Architekten nach einer polyglotten Microservice Architektur mit Java und .NET zu beantworten, kann man prinzipiell zuversichtlich verlauten lassen, dass dies prinzipiell gut möglich ist, jedoch mit den in Kapitel 8 genannten Ausnahmen.
Verwendet eine Microservice Architektur, welche überwiegend Microservices in Java implementiert, genau eine der Infrastrukturkomponenten, welche zu diesen Ausnahmen gehört, dann ist dringend erstmal davon abzuraten einen neuen Microservice in .NET zu implementieren.

Darüberhinaus sind noch folgende Dinge zu bedenken. Im Rahmen dieser Arbeit wurden insgesamt zwar mehr als 80 Frameworks untersucht, dies bedeutet jedoch nicht, dass dies alle verfügbaren Frameworks sind. Möglicherweise existieren noch weitere Frameworks, welche nicht gefunden wurden. Es stellt auch nur eine Momentaufnahme zum aktuellen Zeitpunkt dar.

Weiterhin wurden im Rahmen dieser Arbeit nur Open Source Infrastruktur Komponenten untersucht, da in der Java Welt überwiegend solche eingesetzt werden.

Microsoft bietet selbst auch Infrastrukturkomponenten an, welche kostenpflichtig sind und nicht in diese Kategorie fallen. Diese wurden bei der Untersuchung nicht in Betracht gezogen.
Es liegt die Vermutung nahe, dass es zur Anbindung an kommerzielle Infrastruktur-komponenten mit Hilfe von .NET Frameworks die besseren Möglichkeiten gibt, als mit Java Frameworks. Dies zu Untersuchen wäre eine geeignete Aufgabenstellung für eine weitere Masterarbeit.

Des Weiteren bietet diese Masterarbeit die ideale Basis um mit Hilfe der genannten Frameworks eine konkrete Beispiel Applikation zu bauen. Anhand dieser Applikation könnte man im Rahmen einer weiteren Bachelor- bzw. Masterarbeit das Zusammenspiel der Microservices untereinander und mit den Infrastrukturkomponenten in der Praxis testen und entsprechende Benchmarks durchführen.

Schließlich läßt sich als Fazit sagen, dass die Entwicklung in den Technologien in Bezug auf Microservice Architekturen rasant vorangeht. Die Anzahl der veröffentlichen Bücher über das Thema nimmt stetig zu und manche Frameworks sind sehr stark in der Weiterentwicklung. (z.B. Steeltoe)

Um der Möglichkeit der Polyglotten Microservice Architektur noch mehr Rechnung zu tragen, bedarf es grundsätzlich noch weiterer Standards. Die Masterarbeit hat dies anhand des Aspekts Dienstmanagement aufgedeckt.

Wir dürfen gespannt in die Zukunft blicken und sollten Microservices nicht als kurzfristigen Hype betrachten.

# Anhang A - Microservice in Java mit Spring Boot

(Long & Bastiani, 2017) beschreibt sehr ausführlich, wie man einen einfachen Microservice erstellt, der „Hello World!“ zurückliefert.

Bevor man beginnt, muss das Java Development Kit[49] heruntergeladen und installiert werden, sowie die JAVA_HOME Umgebungsvariable für die Java Binaries direkt im Anschluss gesetzt werden.

Zum Erstellen der Projektinfrastruktur für ein Java Programm mit Spring Boot gibt es das Open Source Projekt *SpringInitializer*[50]*.* Es generiert Maven und Gradle Projekte mit den dazugehörigen Abhängigkeiten, den Rumpf der Java Einstiegsklasse und den Rumpf eines Unit Tests. *SpringInitializr* ist auch als Web-Seite verfügbar. Man wählt dort zunächst den Projekt Typ aus (z.B. Maven Project), die Programmiersprache (z.B. Java), die Version des Spring Boot Frameworks (z.B. 1.5.8) und dann die Group (z.B. com.helloWorld), den Namen des später zum Download bereitstehenden Artifakts (z.B. Helloworld) und die abhängigen Module (z.B. Web zum Erstellen von Restful Services). **Abbildung 17** zeigt eine Darstellung der Web-Seite mit den benötigten Eingaben zum Erzeugen des HelloWorld Artifakts.

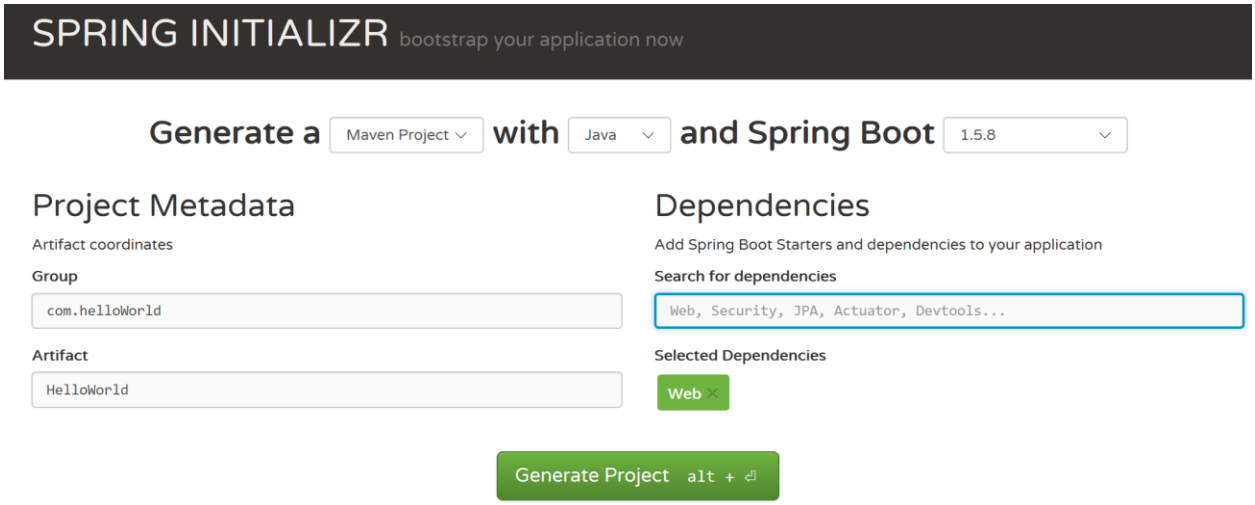



***Abbildung 17*** *SpringInitializr Web-Seite mit Eingaben für HelloWorld*

Mit dem Button „Generate Project“ kann man sich anschließend die *HelloWorld.zip* Datei herunterladen und entpacken.

Das Projekt enthält nun insbesondere die `HelloWorldApplication` Klasse mit dem in **Abbildung 18** dargestellten Inhalt.

[49] Siehe http://www.oracle.com/technetwork/java/javase/downloads/index.html

[50] Siehe http://start.spring.io/

```java
package com.helloWorld.HelloWorld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HelloWorldApplication {

	public static void main(String[] args) {
		SpringApplication.run(HelloWorldApplication.class, args);
	}
}
```

***Abbildung 18*** *HelloWorldApplication Klasse*

Mit dem Befehl `mvnw clean install` werden alle Abhängigen Module heruntergeladen und das Programm ist damit prinzipiell lauffähig.

Damit „HelloWorld!“ zurückgeliefert wird, muss noch ein RestApi Controller hinzugefügt werden. **Abbildung 19** zeigt den hierfür notwendigen Code.

```java
package com.helloWorld.HelloWorld;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

	@GetMapping("/greeting")
	public String getGreeting() {
		return "Hello World!";
	}
}
```

***Abbildung 19*** *GreetingController Klasse*

Die mit `@RestController` annotierte Klasse GreetingController enthält eine Methode `getGreeting`, welche den String „Hello World!“ zurückliefert.
Die Annotation `@GetMapping(„/greeting“)` legt die Route fest, unter der der Endpunkt erreichbar ist.
Mit dem Befehl `mvnw spring-boot:run` startet die Applikation und der Endpunkt ist unter der Adresse *http://localhost:8080/greeting* erreichbar und liefert das gewünschte Ergebnis „Hello World!“ zurück.

# Anhang B - Microservices in C# mit ASP.NET Core

(Hoffman, 2017, S. 3) beschreibt ausführlich, wie man einen einfachen Microservice erstellt, der „Hello World!“ zurückgibt.
Um ans Ziel zu kommen, müssen vorab folgende Dinge installiert werden[51]:

- .NET Core Framework
- Software Development Kit (SDK)
- dotnet Kommandozeilen Werkzeug.

Auf der Kommandozeile hat man nun das Kommando `dotnet` zur Verfügung.

Mit dem Befehl `dotnet --version` kann man sich z.B. die aktuell installierte Version anzeigen lassen.

Zum Erstellen der Projektinfrastruktur sind bereits einfache Projekttemplates eingebaut.
Diese lassen sich mit dem Befehl `dotnet new` ausgeben.

```
Templates                                 Short Name      Language      Tags
----------------------------------------------------------------------------
Console Application                       console         [C#], F#      Common/Console
Class library                             classlib        [C#], F#      Common/Library
Unit Test Project                         mstest          [C#], F#      Test/MSTest
xUnit Test Project                        xunit           [C#], F#      Test/xUnit
ASP.NET Core Empty                        web             [C#]          Web/Empty
ASP.NET Core Web App                      mvc             [C#], F#      Web/MVC
ASP.NET Core Web API                      webapi          [C#]          Web/WebAPI
Solution File                             sln                           Solution
```

***Abbildung 20*** *Eingebaute Projekt Templates von dotnet*

Mit dem Befehlt `dotnet new console` wird automatisch die *HelloWorld.csproj* Datei, sowie die minimal benötigte C# Klasse *Programm.cs* für eine einfache Konsolen-Applikation im aktuellen Verzeichnis erstellt. **Abbildung 21** zeigt die generierte *Programm.cs* Datei.

```
using System;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

***Abbildung 21*** *Programm.cs für HelloWorld Konsolen Applikation aus (Hoffman, 2017, S. 7)*

[51] Siehe hierzu https://www.microsoft.com/net/learn/get-started/windows

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>

</Project>
```

***Abbildung 22*** *HelloWorld.csproj aus (Hoffman, 2017, S. 8)*

Mit dem Befehl `dotnet restore` werden die Projekt-Abhängigkeiten analysiert und die notwendigen NuGet Pakete heruntergeladen. Dieser Schritt ist immer dann notwendig, wenn .csproj Dateien verändert wurden.

Mit dem Befehl `dotnet run` wird die Konsolen-Applikation ausgeführt und liefert die Ausgabe „Hello World!“ zurück.

Um einen Microservice zu implementieren, kann man sich von Anfang an mit Hilfe der entsprechenden Projekttemplates die Projektinfrastruktur generieren lassen.
Entweder mit dem in **Abbildung 20** aufgeführten Templates für ASP.NET, mit Yeoman[52] oder direkt aus der Entwicklungsumgebung Visual Studio heraus[53].
Oder durch Erweiterung der soeben erstellten Konsolen Applikation.

Dieser Schritt soll nun kurz gezeigt werden, um zu verdeutlichen, wie wenig Aufwand hierfür notwendig ist, um von einer .Net Core Konsolen Applikation zu einem minimal lauffähigen Microservice mit ASP.NET Core zu kommen.

Hierzu müssen in der *HelloWorld.csproj* Datei einige neue NuGet Pakete hinzugefügt werden. **Abbildung 23** zeigt die veränderte *HelloWorld.csproj* Datei.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Mvc"
      Version="1.1.1" />
    <PackageReference Include="Microsoft.AspNetCore.Server.Kestrel"
      Version="1.1.1"/>
    <PackageReference Include="Microsoft.Extensions.Logging"
      Version="1.1.1"/>
    <PackageReference Include="Microsoft.Extensions.Logging.Console"
      Version="1.1.1"/>
    <PackageReference Include="Microsoft.Extensions.Logging.Debug"
      Version="1.1.1"/>
    <PackageReference
      Include="Microsoft.Extensions.Configuration.CommandLine"
      Version="1.1.1"/>
  </ItemGroup>
</Project>
```

***Abbildung 23*** *HelloWorld.csproj Datei für die Verwendung von ASP.NET aus (Hoffman, 2017, S. 10)*

[52] Siehe hierzu http://yeoman.io

[53] Siehe Kapitel 3.2.9

Die hinzugefügten Pakete sind insbesondere für ASP.NET MVC und für den Kestrel Server, welche im folgenden http Anfragen[54] bearbeitet.
Damit das Programm nun auf http Anfragen „Hello World!“ zurückgibt, muss die *Programm.cs* Datei wie in **Abbildung 24** erweitert werden.

```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.Configuration;


namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            var config = new ConfigurationBuilder()
                .AddCommandLine(args)
                .Build();


            var host = new WebHostBuilder()
                .UseKestrel()
                .UseStartup<Startup>()
                .UseConfiguration(config)
                .Build();

            host.Run();
        }
    }
}
```

***Abbildung 24*** *Programm.cs zur Verarbeitung von http Anfragen aus (Hoffman, 2017, S. 10)*

In der neuen *Main* Methode des Programms, wird mit Hilfe der *ConfigurationBuilder* Klasse das Konfigurationssubsystem initialisiert um Konfigurationsparameter von der Kommandozeile entgegenzunehmen. An dieser Stelle ließen sich auch noch andere Konfigurationsparameter zur Verwendung konfigurieren, beispielsweise aus einer Json Datei, aus Umgebungsvariablen oder von externen Diensten.

Mit Hilfe der *WebHostBuilder* Klasse wird der Web Host zum Verarbeiten von http Anfragen initialisiert und zur Ausführung gebracht.
Mit der dem Aufruf von `UseKesterel()`, wird der Web Server initialisiert, mit `UseStartup<Startup>()` wird die im Folgenden noch beschriebene Startup Klasse eingebunden und mit `UseConfiguration(config)` wird die zuvor initialisierte Konfiguration geladen. Aufgabe der C# Klasse *Startup* in *Startup.cs* ist es, alle http Anfragen mit dem statischen Text „Hello World!“ zu beantworten. **Abbildung 25** zeigt die neue *Startup* Klasse.

[54] Siehe hierzu Kapitel 5.1.1

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.AspNetCore.Http;

namespace HelloWorld {
  public class Startup
  {
      public Startup(IHostingEnvironment env)
      {
      }

      public void Configure(IApplicationBuilder app,
        IHostingEnvironment env, ILoggerFactory loggerFactory)
      {
        app.Run(async (context) =>
        {
          await context.Response.WriteAsync("Hello, world!");
        });
      }
  }
}
```

***Abbildung 25*** *Startup.cs aus (Hoffman, 2017, S. 11)*

Die Klasse *Startup* enthält neben einem Konstruktor die Methode `Configure`.
Diese ruft auf der übergebenen Schnittstelle `IApplicationBuilder` die `Run` Methode auf und übergibt dieser eine namenlose Funktion, welche asynchron ausgeführt wird und als Übergabeparameter den Ausführungskontext `context` der http Anfrage übergeben bekommt. Mit Hilfe des Lambda Ausdrucks (`=>`) wird die Funktion implementiert und schreibt in die Antwort (`Response`) der Anfrage den statischen Text „Hello, world!“.

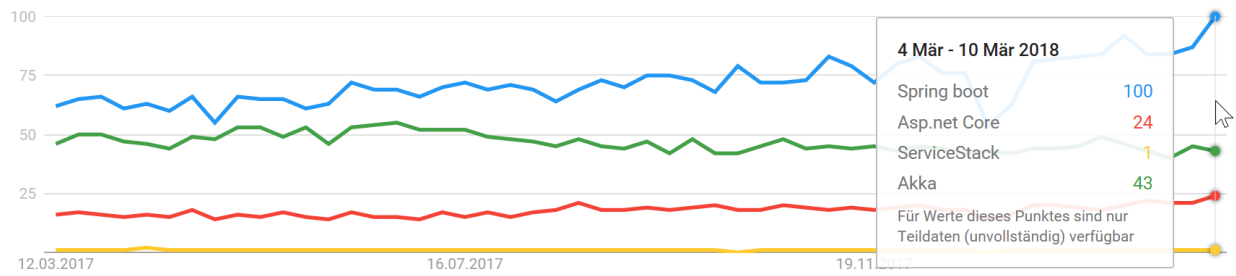
Nachdem man mit dem Aufruf `dotnet restore` die NuGet Pakete aktualisiert hat und mit dem Aufruf `dotnet run` die Applikation startet, bekommt man die in **Abbildung 26** dargestellte Ausgabe. Diese besagt insbesondere, dass nun ein http Endpunt unter der Adresse *http://localhost:5000* erreichbar ist. Jegliche Anfragen an diesen werden nun mit „Hello, world!“ beantwortet.

```
Hosting environment: Production
Content root path: C:\Projekte\HelloWorld\bin\Debug\netcoreapp1.1
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

***Abbildung 26*** *Ausgabe von HelloWorld mit Startup.cs*

# Anhang C - Frameworks Kommunikation

## C.1 Http Rest




### C.1.1 Java

Vergleich Java Restful Frameworks

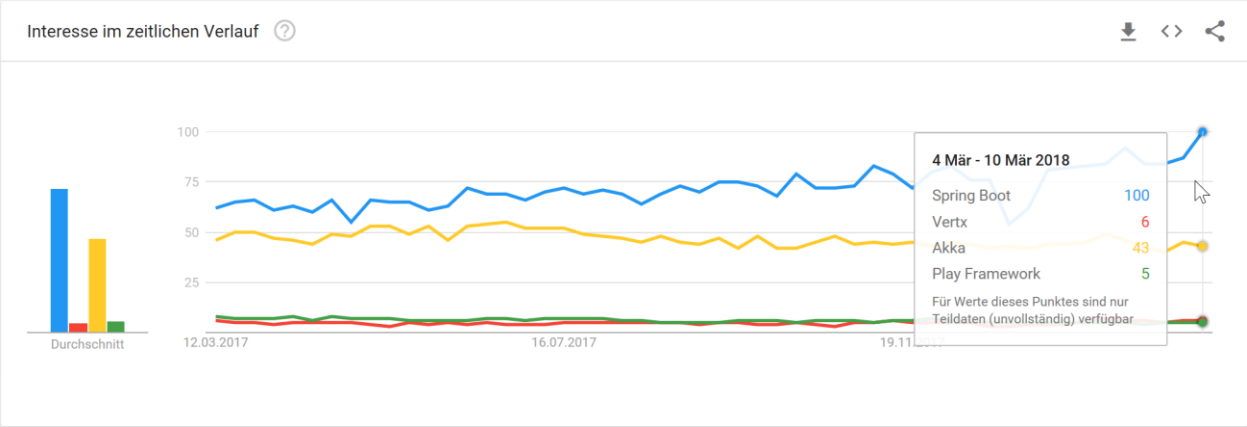
https://www.quora.com/Whats-the-best-RESTful-web-framework-to-use-with-Java
https://www.gajotres.net/best-available-java-restful-micro-frameworks/

- Restlet (1)
- Jersey (<1)
- Dropwizard (3)
- Light-rest-4j (also known as Light Java) (0)
- Play Framework (5)
- RESTEasy (1)
- Spark Framework (1)
- Spring Boot (100)
- Jooby/Untertow (4)
- SeedStack-Filter (0)
- Sprint Boot Reactor (0)
- Pippo-Undertow (0)
- Pippo-Jetty (0)
- FJinal (0)
- Akka-http (43)
- RatPack (2)
- Pippo-Tomcat (0)
- Bootique+Jetty/Jersey (0)
- SeedStack-Jersey2 (0)
- Baseio (0)
- Ninja Framework (1)
- Spring Boot Undertow (<1)
- Sprint Boot Tomcat (<1)

- Payra Micro (0)
- WildFly Swarm (<1)
- ActFramework (3)
- Vertx (6)

https://github.com/networknt/microservices-framework-benchmark

Platz 2 bzgl. Performance: Light Java, Platz 3 ActFramework, Platz 5 Vertx, Platz 7 dotnet




### C.1.1.1 Spring Boot

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-projects/spring-boot | | |
| Aktuelle Version | 2.0.0 | | |
| Releasedatum | 01.03.2018 | | |
| Github Stars | 21.906 | | |
| Contributer | > 100 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 484 | 11.941 | 96% |
| Uncommented Open Issues | 96 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Release 2.0.1 ist angekündigt. |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven, Gradle, SpringInitializr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

### C.1.1.2 Akka

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/akka/akka | | |
| Aktuelle Version | 2.5.11 | | |
| Releasedatum | 08.03.2018 | | |
| Github Stars | 8.269 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 658 | 11.453 | 95% |
| Uncommented Open Issues | 168 | | 96% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://akka.io/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

## C.1.2 .NET


- ASP.NET Core WEB API (100)
- Service Stack (11)
- NancyFx (7)
- Spring.NET (2)
- Giraffe .net (0)

### C.1.2.1 ASP.NET Core

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/aspnet/Mvc | | |
| Aktuelle Version | 2.0.3 | | |
| Releasedatum | 15.11.2017 | | |
| Github Stars | 5.105 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 342 | 7.126 | 95% |
| Uncommented Open Issues | 61 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | Meilensteine vorhanden. |
| Neue Versionen angekündigt | ☐ Ja | ☐ Nein | Version 3.0.0 wird bereits angekündigt. |
| Angaben zur Finanzierung | ☐ Ja | ☐ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

### C.1.2.2 ServiceStack

Die neueste Version von ServiceStack ist für .NET Core verfügbar.
Es gibt eine Free Version, welche begrenzte Funktionalität hat.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/ServiceStack/ServiceStack | | |
| Aktuelle Version | 5.0.2 | | |
| Releasedatum | 06. 01.2018 | | |
| Github Stars | 4.154 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☒ GPL |
| Open Issues / Closed Issues | | | Issues werden nicht auf Github getrackt. |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | Roadmap Seite ohne Inhalt. |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://servicestack.net/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☒ Ja | ☐ Nein | Überwiegend veraltet. (von 2015) |

# C.2 Http Thrift


## C.2.1 Java

- Apache Thrift Java

### C.2.1.1 Apache Thrift Java

Apache Thrift Java ist nur Teil einer Programmbibliothek.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/thrift/tree/master/lib/java | | |
| Aktuelle Version | | | Es gibt kein offizielles Release. |
| Releasedatum | | | Letzte Aktivität vor 7 Tagen. |
| Github Stars | 4.500 | | Nur für Gesamtbibliothek |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | | | Es gibt kein Issue Tracker. |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☐ Ja | ☒ Nein | |
| Download von Zip / Tar | ☐ Ja | ☒ Nein | Clonen von Repository möglich und selbst builden ☹ |
| Installationsanleitung | ☐ Ja | ☒ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☒ Ja | ☐ Nein | Es gibt ein Buch mit Beispielen in Java. |

## C.2.2 .NET

- Apache Thrift NetCore
- Thrift-netCore

### C.2.2.1 Apache Thrift NetCore

Apache Thrift Netcore ist nur Teil einer Programmbibliothek.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/thrift/tree/master/lib/netcore | | |
| Aktuelle Version | | | Es gibt kein offizielles Release. |
| Releasedatum | | | Letzte Aktivität vor 2 Monaten. |
| Github Stars | 4.500 | | Nur für Gesamtbibliothek |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | | | Es gibt kein Issue Tracker |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☐ Ja | ☒ Nein | |
| Download von Zip / Tar | ☐ Ja | ☒ Nein | Clonen von Repository möglich und selbst builden ☹ |
| Installationsanleitung | ☐ Ja | ☒ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

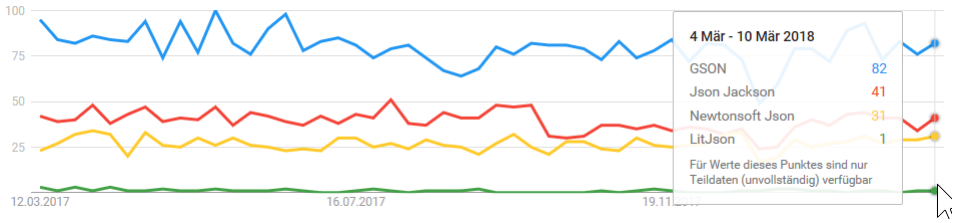
### C.2.2.2 Thrift-netCore

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/gboucher90/thrift-netcore | | |
| Aktuelle Version | 0.9.3.2 | | Es gibt keine Release Version. |
| Releasedatum | | | |
| Github Stars | 2 | | |
| Contributer | 1 | | Keine Aktivität die letzten 6 Monate. |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 0 | 1 | 100% |
| Uncommented Open Issues | | | Es wurden keine Issues angegeben. |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.3 Json

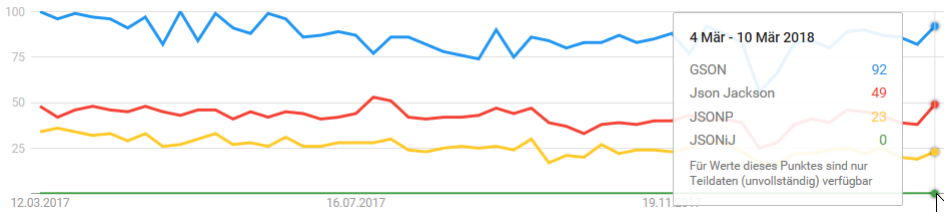
Benchmark .NET vs Java Json Serializer:
https://github.com/ngs-doo/json-benchmark




### C.3.1 Java

Performance Vergleich Java Frameworks Json:

- https://blog.takipi.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/
- https://www.developer.com/lang/jscript/top-7-open-source-json-binding-providers-available-today.html




- GSON (92)
- Jackson (49)
- Json.simple (0)
- JSONP (21)
- JSON-lib (<1)
- Flexjson (0)
- Json-io (0)
- Genson (2)
- JSONiJ (0)

### C.3.1.1 Gson

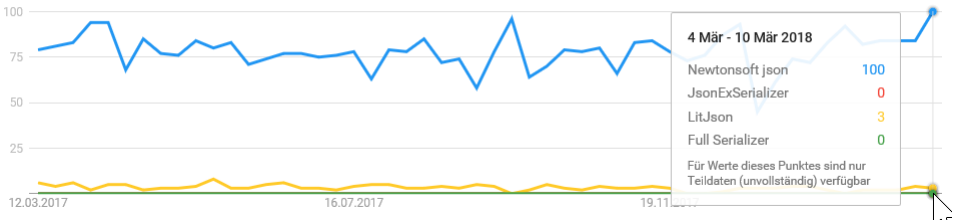
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/google/gson | | |
| Aktuelle Version | 2.8.2 | | |
| Releasedatum | 20.09.2017 | | |
| Github Stars | 11.942 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 324 | 936 | 74% |
| Uncommented Open Issues | 109 | | 91% |
| Tests | ☐ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☐ Nein | Releaseplan wurde angelegt, aber nicht verwendet. |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☐ Ja | ☒ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | Auf separaten Github Projekt. |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | Auf YouTube. |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | Es gibt ein Buch, aber von 2013 |

### C.3.1.2 Jackson

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/FasterXML/jackson-core | | |
| Aktuelle Version | 2.9.4 | | |
| Releasedatum | 24.01.2018 | | |
| Github Stars | 3.671 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 247 | 1.720 | 87% |
| Uncommented Open Issues | 34 | | 98% |
| Tests | ☐ Ja | ☒ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | Es gibt eine separate Github Seite für Ideen |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | Jackson Cookbook, Learnpub (nicht mehr erhältlich) |

## C.3.2 .NET




- LitJSON (3)
- JsonExSerializer (0)
- Newtonsoft JSON.NET (100)
- Full Serializer (0)

### C.3.2.1 Newtonsoft JSON.NET

ASP.NET Core verwendet intern JSON.NET. Daher kompatibel zu .NET Core.
Es gibt einen offiziellen Patch von Microsoft für .NET Standard 2.0.

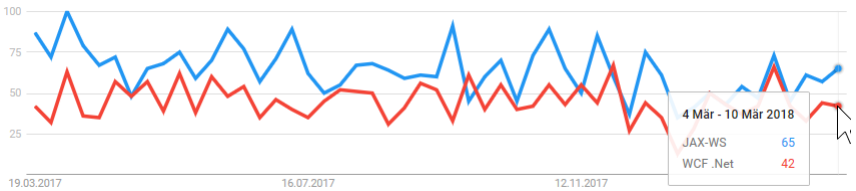
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/JamesNK/Newtonsoft.Json | | |
| Aktuelle Version | 11.0.1 | | |
| Releasedatum | 17.02.2018 | | |
| Github Stars | 5.287 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 159 | 1.099 | 87% |
| Uncommented Open Issues | 89 | | 93% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://www.newtonsoft.com/json |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.3.2.2 LitJSON

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/LitJSON/litjson | |
| Aktuelle Version | 0.12.0 | |

| | | | | |
|---|---|---|---|---|
| Releasedatum | 24.01.2018 | | | |
| Github Stars | 403 | | | |
| Contributer | < 10 | | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL | unlicense.org → kompatibel zu MIT |
| Open Issues / Closed Issues | 39 | 40 | 50% | |
| Uncommented Open Issues | 19 | | 76% | |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☒ Ja | ☐ Nein | | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☐ Ja | ☒ Nein | | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☒ Ja | ☐ Nein | | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://litjson.net/ | |
| API Referenz | ☒ Ja | ☐ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

# C.4 Xml (Soap)




Xml Implementierungen:

- JAXP
- StAX
- SAX
- DOM
- JDOM
- JAXP
- SOAP

Siehe hierzu: www.torsten-horn.de/techdocs/java-xml.htm

## C.4.1 Java

Xml Frameworks:

- SAX
- DOM4j
- JDOM

- JAXP
- Xerces C
- XML4C

Soap wird in Java mit JAX-WS umgesetzt. Siehe hierzu:

- https://docs.oracle.com/javase/7/docs/technotes/guides/xml/jax-ws/index.html
- https://javaee.github.io/metro-jax-ws/

JAX-WS ist Teil der Java SE und Java EE Platformen.

### C.4.1.1 Metro JAX-WS

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://javaee.github.io/metro/ | | |
| Aktuelle Version | 2.3.0 | | |
| Releasedatum | 03.08.2017 | | |
| Github Stars | 53 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☒ GPL |
| Open Issues / Closed Issues | 243 | 971 | 80% |
| Uncommented Open Issues | 14 | | 99% |
| Tests | ☐ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Neues Release 2.3.1 angekündigt. |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

## C.4.2 .Net

Xml ist Teil vom .NET Framework. Es gibt 2 Basisklassen im Namespace System.Xml, welche diese Aufgaben erledigen:
- XmlDocument
- XDocument (ab .NET 3.5)

Gemäß https://www.heise.de/developer/meldung/NET-Standard-2-0-ist-fertig-und-NET-Core-2-0-auf-der-Zielgeraden-3796910.html ist der Namespace System.Xml nun auch vollständig im .NET Standard 2.0 und somit in .NET Core enthalten.
Siehe hierzu auch https://www.c-sharpcorner.com/article/what-is-new-in-net-core-2-0/
Es gibt eine Umsetzung von SAX in .NET, welche jedoch nicht .NET Core 2 kompatibel ist. (https://github.com/rasmusjp/sax.net)

Soap wurde bisher (vor .NET Core) über das schwergewichtige und komplexe WCF Framework behandelt. Eine Unterstützung hierfür in .NET Core ist vorhanden, aber nicht mit dem vollen Komfort und Umfang den man mit WCF hatte.
Siehe hierzu https://stackify.com/soap-net-core/ und https://github.com/dotnet/wcf/issues/2382
Microsoft bietet seit November 2017 ein Compatibility Pack an, welches das Entwickeln von WebServices mit WCF Klassen für Soap ermöglicht. Siehe hierzu: https://www.heise.de/developer/meldung/Microsoft-veroeffentlicht-Windows-Compatibility-Pack-fuer-NET-Core-3892367.html
Allerdings sind diese Klassen nur unter Windows verfügbar und damit nicht kompatibel.
Es gibt jedoch eine Community Projekt um WCF nach .NET Core zu migrieren.

### C.4.2.1 WCF .NET Core

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/dotnet/wcf | | |
| Aktuelle Version | 2.0.0 | | prerelease 2.1.0 vom 27.01.2018 für .NET Core 2.1 |
| Releasedatum | 14.08.2017 | | |
| Github Stars | 954 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 166 | 2.504 | 94% |
| Uncommented Open Issues | 41 | | 95% |
| Tests | ☒ Ja | ☐ Nein | Inklusive Coverage Report. 😊 |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

Bezüglich Verständlichkeit, Erlernbarkeit und Dokumentation sind alle Kriterien erfüllt.
Trotz der vollen Punktzahl bezüglich Erlenbarkeit, sei jedoch angemerkt, dass WCF mit eines der schwer erlernbarsten Frameworks im .Net Bereich ist.

## C.5 Apache Avro

Über die Verbreitung der Frameworks kann keine adäquate Aussage gemacht werden.

### C.5.1 Java

Für Java wurde nur das offizielle von Apache verfügbare Avro Framework für Java gefunden.

### C.5.1.1 Apache Avro für Java

Auf dem Apache Github Repository für Avro sind für verschiedene Sprachen Implementierungen vorhanden. (Java 45%, C13.9%, C# 11.1%, C++ 8.9%, Python 7.3%, JavaScript 4,7%, Others 9.1%) Die Wertung kann prozentual für Java vorgenommen werden.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/avro | | |
| Aktuelle Version | 1.8.2 | | |
| Releasedatum | 15.05.2017 | | |
| Github Stars | 388 / 862 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | | | |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://avro.apache.org/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.5.2 .NET

Das offizielle Framework für C# von Apache ist veraltet und nicht .NET Core kompatibel.
Der Link in der Dokumentation für C# ist defekt!
Es gibt 2 Frameworks die .NET Core kompatibel sind.

- Microsoft-Avro
- Apache-Avro-Core

### C.5.2.1 Microsoft-Avro

Microsoft-Avro ist das aus dem HDInsight SDK extrahierte Avro Core, welcher portiert wurde um dem .NET Standard zu genügen und mit .NET Core 2.0 läuft.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/dougmsft/microsoft-avro | | |
| Aktuelle Version | 0.1.0 | | |
| Releasedatum | 01.05.2017 | | |
| Github Stars | 7 | | |
| Contributer | 1 | | letzte Aktivität im April 2017 |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 6 | 12 | 66% |

| | | | |
|---|---|---|---|
| Uncommented Open Issues | 18 | | Es wurden keine Anfragen beantwortet! (0 Punkte) |
| Tests | ☐ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☐ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☐ Nein | |
| Installation über Paketmanager | ☐ Ja | ☐ Nein | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

API Referenz vorhanden, aber nicht da wo man es erwartet ☹

https://msdn.microsoft.com/en-us/library/microsoft.hadoop.avro.aspx

Es gibt auch Samples, aber nicht da wo man es erwartet. ☹

https://docs.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-dotnet-avro-serialization

### C.5.2.2 Apache-Avro-Core

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/welly87/Apache-Avro-Core | | |
| Aktuelle Version | | | Es gibt keine Release Version. |
| Releasedatum | | | |
| Github Stars | 5 | | |
| Contributer | 1 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 5 | 0 | 0% |
| Uncommented Open Issues | 3 | | 40% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☐ Ja | ☒ Nein | |
| Download von Zip / Tar | ☐ Ja | ☒ Nein | |
| Installationsanleitung | ☐ Ja | ☒ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# C.6 Protocol Buffers

## C.6.1 Java und .NET

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/google/protobuf | | |
| Aktuelle Version | 3.5.1 | | |
| Releasedatum | 21.12.2017 | | |
| Github Stars | 24.282 | | ☺ |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL Google Lizenz |
| Open Issues / Closed Issues | 679 | 3.674 | 84% |
| Uncommented Open Issues | 167 | | 96% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Google |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | ☺ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# C.7 AMQP


## C.7.1 Java

- Spring AMQP

### C.7.1.1 Spring AMQP

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://projects.spring.io/spring-amqp/ | | |
| Aktuelle Version | 2.0.2 | | |
| Releasedatum | 29.01.2018 | | |
| Github Stars | 384 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 3 | 718 | 100% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☐ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Automatisierter Build | ☐ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☐ Nein | Neue Version 2.0.3 bereits angekündigt. |
| Angaben zur Finanzierung | ☐ Ja | ☐ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven,Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://projects.spring.io/spring-amqp/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | Es gibt ein explizites eBook |

## C.7.2 .NET

- AMQPNetLite.Core

### C.7.2.1 AMQPNetLite.Core

AMQPNetLite.Core unterstützt im Augenblick nur .NET Core 1.0.
Die wichtigsten Features (Serializer, Listener) noch nicht unterstützt.
Das Framework erfüllt damit nicht das Kriterium der Portabilität.

# C.8 MQTT

Projektseite mit vielen Links zu MQTT https://github.com/hobbyquaker/awesome-mqtt
Offizielle Projektseite: http://mqtt.org/
Bezüglich Verbreitung lässt sich mit Google Trends kein Resultat erzielen.
Die Auswahl wurde anhand der Github Stars getroffen.

## C.8.1 Java

- Fusesource mqtt-client (776)
- Eclipse paho.mqtt.java (552)
- Vert.x MQTT (57)
- Xenqtt (21)
- MeQanTT (57)

### C.8.1.1 Fusesource mqtt-client

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/fusesource/mqtt-client | | |
| Aktuelle Version | 1.14 | | |
| Releasedatum | 31.05.2016 | | |
| Github Stars | 776 | | |
| Contributer | 1 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 70 | 28 | 29% |
| Uncommented Open Issues | 70 | | 29% |
| Tests | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.8.1.2 Eclipse paho.mqtt.java

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/eclipse/paho.mqtt.java | | |
| Aktuelle Version | 1.2.0 | | |
| Releasedatum | 04.08.2017 | | |
| Github Stars | 552 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL Eclipse License |
| Open Issues / Closed Issues | 109 | 388 | 78% |
| Uncommented Open Issues | 39 | | 92% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Failed ☹ |
| Releaseplan / Meilensteine | ☐ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☐ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☐ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | MQTT mit Eclipse Paho |

## C.8.2 .NET

- MQTTnet
- M2qtt Dotnet Core
- LagoVista.MQTT (kein explizites Repository auffindbar)
- GnatMQ (reines Broker Framework, keine Client Funktionalität)
- D2L.MQTT.Packets (keine Angaben bzgl. .Net Core)

### C.8.2.1 MQTTnet

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/chkr1011/MQTTnet | | |
| Aktuelle Version | 2.7.1 | | |
| Releasedatum | 13.03.2018 | | |
| Github Stars | 214 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 20 | 176 | 90% |
| Uncommented Open Issues | 20 | | 90% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.8.2.2 M2qtt Dotnet Core

M2qtt ist ein Fork des Eclipse Paho Frameworks von C#.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/mohageg/paho.mqtt.m2mqtt | | |
| Aktuelle Version | 1.0.7 | | Nur auf Nuget. Nicht auch Github geführt. |
| Releasedatum | 01.09.2017 | | |
| Github Stars | 13 | | Framework wurde für .NET Core geforked. |
| Contributer | | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL Eclipse License |
| Open Issues / Closed Issues | | | Kein Issue Tracker |
| Uncommented Open Issues | | | |
| Tests | ☐ Ja | ☒ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# C.9 Atom und RSS Feeds

Über die Verbreitung kann keine Aussage getroffen werden.

## C.9.1 Java

- Rome

### C.9.1.1 Rome

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/rometools/rome | | |
| Aktuelle Version | 1.9.0 | | |
| Releasedatum | 25.11.2017 | | |
| Github Stars | 403 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 22 | 358 | 94% |
| Uncommented Open Issues | 6 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Rome 2 bereits angekündigt. |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip/Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.9.2 .NET

- SyndicationFeedReaderWriter (aus dem .NET WCF Framework übernommen)
- CodeHollowReader

### C.9.2.1 SyndicationFeedReaderWriter

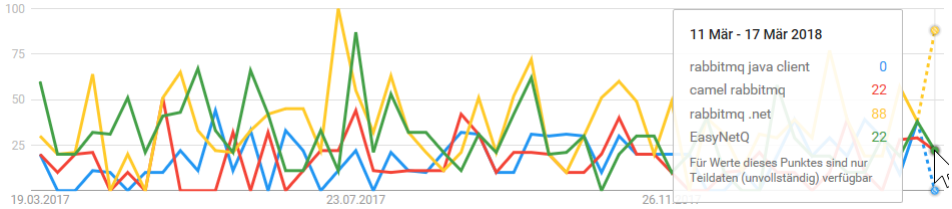
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/dotnet/SyndicationFeedReaderWriter | | |
| Aktuelle Version | 1.0.2 | | |
| Releasedatum | 13.11.2017 | | |
| Github Stars | 55 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 1 | 28 | 97% |

| | | | |
|---|---|---|---|
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Projekt wird von Microsoft betrieben |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip/Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.9.2.2 CodeHollow FeedReader

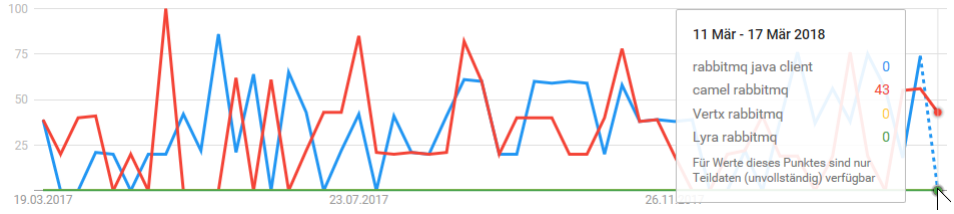
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/codehollow/FeedReader | | |
| Aktuelle Version | 1.1.1 | | Nicht auf Github geführt. |
| Releasedatum | 03.03.2018 | | |
| Github Stars | 26 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 2 | 8 | 80% |
| Uncommented Open Issues | 1 | | 90% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | nuget |
| Download von Zip/Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# C.10 RabbitMQ




## C.10.1 Java

- RabbitMq Java Client
- Apache Camel-rabbitmq
- Spring AMQP (siehe C.7.1.1)
- Vertx RabbitMQ Client
- Lyra




### C.10.1.1 RabbitMq Java Client

Offizieller RabbitMQ Java Client von Pivotal

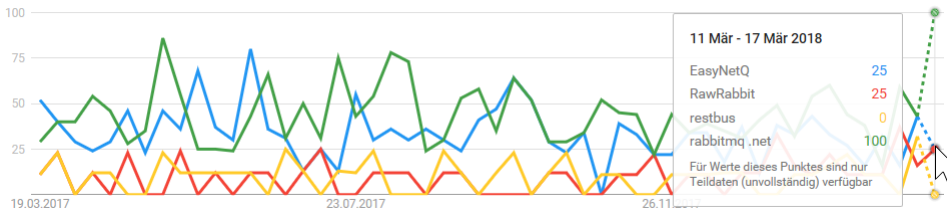
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/rabbitmq/rabbitmq-java-client | | |
| Aktuelle Version | 5.2.0 | | |
| Releasedatum | 13.03.2018 | | |
| Github Stars | 493 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☒ GPL |
| Open Issues / Closed Issues | 3 | 349 | 99% |
| Uncommented Open Issues | 1 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Pivotal |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | Nicht explizit zu RabbitMQ |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://www.rabbitmq.com/api-guide.html |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.10.1.2 Apache Camel-rabbitmq

RabbitMQ ist Teil des Apache Camel Frameworks

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/camel | | |
| Aktuelle Version | 2.21.0 | | |
| Releasedatum | 11.03.2018 | | |
| Github Stars | 1.727 | | |
| Contributer | >=50 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 461 | 11857 | 96% (Jira Board auf Projektwebseite) |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Version 3.0 |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Apache Foundation |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube explizit zu RabbitMq |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://camel.apache.org/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | z.B. Camel in Action |

## C.10.2 .NET




- RabbitMQ .NET Client
- EasyNetQ (Prerelease auf Nuget erhältlich für .NET Core)
- RawRabbit
- Restbus

### C.10.2.1 RabbitMQ .NET Client

Offizieller RabbitMQ .Net Client von Pivotal

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/rabbitmq/rabbitmq-dotnet-client | |
| Aktuelle Version | 5.0.1 | |
| Releasedatum | 26.09.2017 | |
| Github Stars | 576 | |
| Contributer | < 10 | |

| | | | |
|---|---|---|---|
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 20 | 375 | 95% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☐ Ja | ☒ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Build failed ☹ |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Pivotal |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☐ Nein | |

### C.10.2.2 EasyNetQ

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/EasyNetQ/EasyNetQ | | |
| Aktuelle Version | 2.3.1 | | |
| Releasedatum | 20.02.2018 | | |
| Github Stars | 1.559 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 103 | 652 | 86% |
| Uncommented Open Issues | 19 | | 97% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | Roadmap vorhanden. |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://easynetq.com/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.11 Apache Active MQ

Für ActiveMQ wurden nur die von Apache zur Verfügung gestellten Client Frameworks für Java und .NET gefunden. Das .NET Framework erfüllt nicht das Kriterium der Portabilität. Das letzte

Release ist vom 07.04.2016. Es muss diesbezüglich auf die Low-Level Protokolle wie AMQP, STOMP etc. ausgewichen werden.

## C.11.1 Java

- ActiveMQ Client

### C.11.1.1 ActiveMQ

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/activemq | | |
| Aktuelle Version | 5.15.3 | | |
| Releasedatum | 29.01.2018 | | |
| Github Stars | 1.051 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 604 | 6657 | 92% Issue Tracker auf Apache |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://activemq.apache.org/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

# C.12 Apache Kafka

Bezüglich der Verbreitung liefert Google Trends keine verwertbaren Resultate.

## C.12.1 Java

Für Java existiert das offizielle von Apache zur Verfügung gestellte Framework, welches vermutlich auch einen entsprechenden Client zur Anbindung an Kafka bietet. Eine Untersuchung speziell hierfür ist wenig sinnvoll.

Es existieren sowohl für Spring als auch für Vertx Client Frameworks.

- Spring Kafka
- Vertx-kafka-client

### C.12.1.1 Spring Kafka

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-projects/spring-kafka | | |
| Aktuelle Version | 2.1.4 | | |
| Releasedatum | 27.02.2018 | | |
| Github Stars | 381 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 24 | 578 | 96% |
| Uncommented Open Issues | 5 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Version 2.1.5 |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | Udemy |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

### C.12.1.2 Vertx-Kafka-Client

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/vert-x3/vertx-kafka-client | | |
| Aktuelle Version | 3.5.1 | | |
| Releasedatum | 11.02.2018 | | |
| Github Stars | 21 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 17 | 78 | 82% |
| Uncommented Open Issues | 6 | | 94% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://www.vertx.io/docs/vertx-kafka-client/java/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.12.2 .NET

Für .NET existiert nur das Confluent-kafka-dotnet Framwork, welches .NET Core unterstützt. Alle anderen von Apache empfohlenen .NET Clients, inklusive das von Microsoft angebotene Framework, sind nicht .NET Core kompatibel.

- Confluent-kafka-dotnet

### C.12.2.1 Confluent-kafka-dotnet

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/confluentinc/confluent-kafka-dotnet | | |
| Aktuelle Version | 0.11.3 | | |
| Releasedatum | 04.12.2017 | | |
| Github Stars | 429 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 128 | 337 | 72% |
| Uncommented Open Issues | 11 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Confluent |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# C.13 NSQ

Google Trends liefert zu den Frameworks keine Daten.
Die Verbreitung wird anhand der Github Stars beurteilt, aber nicht mit einbezogen.

## C.13.1 Java

- Nsq-j (8)
- JavaNSQClient (59)
- Nsq-java (45)

### C.13.1.1 JavaNSQClient

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/brainlag/JavaNSQClient | | |
| Aktuelle Version | 1.0.0.RC4 | | |
| Releasedatum | 01.04.2017 | | |
| Github Stars | 59 | | |
| Contributer | 0 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 9 | 37 | 80% |
| Uncommented Open Issues | 3 | | 93% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.13.1.2 Nsq-java

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/nsqio/nsq-java | | |
| Aktuelle Version | 1.1 | | |
| Releasedatum | 01.08.2015 | | |
| Github Stars | 45 | | |
| Contributer | 0 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL keine |
| Open Issues / Closed Issues | 7 | 9 | 56% |
| Uncommented Open Issues | 4 | | 25% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## C.13.2 .NET

- NSQCore (7)
- ZeroNSQ (0)

### C.13.2.1 NSQCore

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/Webjet/NSQCore | | |
| Aktuelle Version | 1.0.0 | | |
| Releasedatum | 01.08.2017 | | |
| Github Stars | 7 | | |
| Contributer | 1 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 0 | 0 | 0% |
| Uncommented Open Issues | 0 | | 0% |
| Tests | ☐ Ja | ☒ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### C.13.2.2 ZeroNSQ

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/leypascua/ZeroNsq | |
| Aktuelle Version | 0.1.2.20b | Prerelease |
| Releasedatum | 20.02.2018 | |
| Github Stars | 0 | |
| Contributer | 1 | |

| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL | |
|---|---|---|---|---|
| Open Issues / Closed Issues | 1 | 8 | 89% | |
| Uncommented Open Issues | 0 | | 100% | |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☐ Ja | ☒ Nein | | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☐ Ja | ☒ Nein | | |
| Quickstarts | ☐ Ja | ☒ Nein | | |
| Use Cases | ☐ Ja | ☒ Nein | | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | | |
| Projektwebseite | ☐ Ja | ☒ Nein | | |
| API Referenz | ☐ Ja | ☒ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

# Anhang D - Frameworks Diensterkennung

## D.1 Netflix Eureka

Sowohl für Java als auch .NET wurde jeweils nur ein Framework gefunden, welche Eureka unterstützt. Google Trends liefert keine verwertbaren Resultate bezüglich Verbreitung.

### D.1.1 Java

- Spring Cloud Netflix

#### D.1.1.1 Spring Cloud Netfix

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-netflix | | |
| Aktuelle Version | 2.0.0.M7 | | |
| Releasedatum | 27.02.2018 | | |
| Github Stars | 2.032 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 383 | 2.402 | 86% |
| Uncommented Open Issues | 33 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage Report 65% 😊 |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Netflix |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://cloud.spring.io/spring-cloud-netflix/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### D.1.2 .NET

- Steeltoe Discovery

#### D.1.2.1 Steeltoe Discovery

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/SteeltoeOSS/Discovery | | |
| Aktuelle Version | 2.0.1 | | |
| Releasedatum | 01.03.2018 | | |
| Github Stars | 96 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 6 | 20 | 76% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | .NET Foundation, Pivotal |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://steeltoe.io/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## D.2 Etcd

Etcd ist ein Key/Value Store. Kubernetes Cluster speichern Konfigurationsdaten bezüglich Diensterkennung und Cluster Management in Etcd. Goolge Trends liefert bezüglich Verbreitung keine vernünftigen Daten. Die Auswahl zur Untersuchung wird anhand der Github Stars getroffen.

### D.2.1 Java

- Spring Cloud Etcd (65)
- Etcd4j (206)
- Jetcd Core (205)

#### D.2.1.1 Jetcd Core

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/coreos/jetcd | | |
| Aktuelle Version | 0.0.2 | | |
| Releasedatum | 17.02.2018 | | |
| Github Stars | 205 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 27 | 272 | 91% |
| Uncommented Open Issues | 4 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | Work In Progress !! |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### D.2.1.2 Etcd4j

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/jurmous/etcd4j | | |
| Aktuelle Version | 2.15.0 | | |
| Releasedatum | 23.01.2018 | | |
| Github Stars | 206 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 9 | 165 | 95% |
| Uncommented Open Issues | 2 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Build failed ☹ |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## D.2.2 .NET

Für .NET Core wurden gegenwärtig nur 2 Frameworks zur Unterstützung von etcd gefunden.

- EtcdConfigurationProvider (0)
- EtcdGrpcClient (5)

EtcdConfigurationProvider wurde von einem einzelnen Entwickler in Github angelegt und verspricht etcd Unterstützung in AspNetCore. Das Framework wird nicht untersucht, da das Github Projekt nahezu keinen Inhalt (SourceCode, Dokumentation, Commits, etc.) enthält.

### D.2.2.1 EtcdGrpcClient

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/UnderNotic/EtcdGrpcClient | | |
| Aktuelle Version | 0.0.1 | | |
| Releasedatum | < 4 Monate | | |
| Github Stars | 5 | | |
| Contributer | 1 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 0 | 0 | - |
| Uncommented Open Issues | 0 | | - |
| Tests | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# D.3 Consul

Consul ist ein End-to-End Service Discovery Framework, welches nicht nur ein Key-Value-Store beinhaltet, sondern zusätzlich Health Checking, Failure Detection und DNS.
Google Trends liefert keine verwertbaren Resultate. Für die Verbreitung wird die Anzahl der Github Stars verwendet.

## D.3.1 Java

- Consul-Client (301)
- Consul-Api (209)
- Dropwizard Consul Bundle (49)
- Spring Cloud Consul (335)
- Marathon-Consul (Register Marathon Tasks as Consul Service Discovery)
- Mesos-Consul (Mesos to Consul Bridge)
- Vertx Consul Client

### D.3.1.1 Spring Cloud Consul

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-consul | | |
| Aktuelle Version | 2.0.0.M6 | | |
| Releasedatum | 26.02.2018 | | |
| Github Stars | 335 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 51 | 355 | 87% |
| Uncommented Open Issues | 6 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://cloud.spring.io/spring-cloud-consul/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### D.3.1.2 Consul-Client

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/rickfast/consul-client | | |
| Aktuelle Version | 1.1.1 | | |
| Releasedatum | 02.03.2018 | | |
| Github Stars | 301 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 14 | 302 | 96% |
| Uncommented Open Issues | 5 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## D.3.2 .NET

- Consul.NET (409)
- CondenserDotNet (64)
- ServiceStack-Discovery-Consul (53)
- Winton.Extensions.Configuration.Consul (52)
- Microdot (546)
- Fakta (25)

### D.3.2.1 Microdot

| Teilkriterium / Frage | Wert | | | Anmerkung |
|---|---|---|---|---|
| Github Url | https://github.com/gigya/microdot | | | |
| Aktuelle Version | 1.9.0 | | | |
| Releasedatum | 05.03.2018 | | | |
| Github Stars | 546 | | | |
| Contributer | < 10 | | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL | |
| Open Issues / Closed Issues | 8 | 136 | | 94% |
| Uncommented Open Issues | 2 | | | 97% |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☐ Ja | ☒ Nein | | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☐ Ja | ☒ Nein | | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☐ Ja | ☒ Nein | | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | | |
| Projektwebseite | ☐ Ja | ☒ Nein | | |
| API Referenz | ☐ Ja | ☒ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

### D.3.2.2 Consul.NET

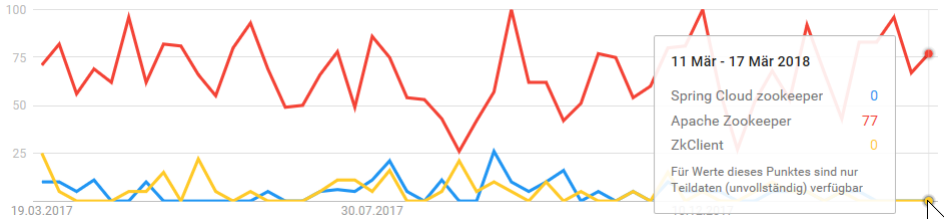
| Teilkriterium / Frage | Wert | | | Anmerkung |
|---|---|---|---|---|
| Github Url | https://github.com/PlayFab/consuldotnet | | | |
| Aktuelle Version | 0.7.2.4 | | | |
| Releasedatum | 25.01.2018 | | | |
| Github Stars | 409 | | | |
| Contributer | < 10 | | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL | |
| Open Issues / Closed Issues | 10 | 99 | | 91% |
| Uncommented Open Issues | 4 | | | 96% |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☒ Ja | ☐ Nein | | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | | PlayFab |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☐ Ja | ☒ Nein | | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☐ Ja | ☒ Nein | | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | | |
| Projektwebseite | ☐ Ja | ☒ Nein | | |

| | | | |
|---|---|---|---|
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# D.4 Apache Zookeeper

Zookeeper ist ein Key-Value-Store zur Koordination von Services.

## D.4.1 Java




- Apache Zookeeper (offizielle Projektseite, enthält auch Java Client)
- Spring Cloud Zookeeper (221)
- Vertx-zookeeper Cluster Manager (kein expliziter Client, verwendet jedoch Zookeeper)
- ZkClient (734)

### D.4.1.1 Apache Zookeeper

Das Github Repository enthält 68,6% Java Code. Entsprechend prozentual erfolgt auch die Wertung bei Github Stars.

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/zookeeper | | |
| Aktuelle Version | 3.4.11 | | |
| Releasedatum | 08.11.2017 | | |
| Github Stars | 2.8001 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 1081 | | Gesamt: 2965 → 64% |
| Uncommented Open Issues | | | Kein Issue Tracker auf Github |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://zookeeper.apache.org/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

### D.4.1.2 Spring Cloud Zookeeper

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-zookeeper | | |
| Aktuelle Version | 2.0.0.M6 | | |
| Releasedatum | 27.02.2018 | | |
| Github Stars | 221 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 17 | 146 | 90% |
| Uncommented Open Issues | 4 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Failed ☹ |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☐ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | Nicht explizit zu Spring Cloud Zookeeper |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://cloud.spring.io/spring-cloud-zookeeper/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### D.4.1.3 ZkClient

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/sgroschupf/zkclient | | |
| Aktuelle Version | 0.10 | | |
| Releasedatum | 08.11.2016 | | |
| Github Stars | 734 | | |
| Contributer | 0 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 20 | 48 | 71% |
| Uncommented Open Issues | 8 | | 88% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☐ Ja | ☒ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## D.4.2 .NET

Für Apache Zookeeper existiert nur ein Framework, welches .NET Core kompatibel ist. Der offizielle von Apache angebotene Client ist nicht .NET Core kompatibel.
Das Github Repository von Apache Zookeeper .NET async Client enthält auch Clients für andere Sprachen. Der C# CodeAnteil beträgt 9%. Entsprechend prozentual erfolgt auch die Wertung bei Github Stars.

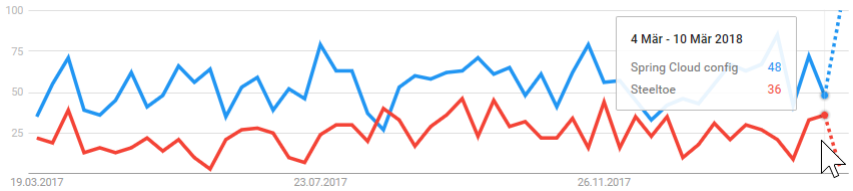
- Apache Zookeeper .NET async Client

### D.4.2.1 Apache Zookeeper .NET async Client

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/shayhatsor/zookeeper | | |
| Aktuelle Version | 3.4.10.1 | | |
| Releasedatum | 28.10.2017 | | |
| Github Stars | 11 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 7 | 19 | 73% |
| Uncommented Open Issues | 2 | | 92% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget. |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang E - Frameworks Konfiguration

## E.1 Spring Cloud Config




### E.1.1 Java

- Spring Cloud Config

#### E.1.1.1 Spring Cloud Config

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-config | | |
| Aktuelle Version | 2.0.0.M8 | | |
| Releasedatum | 02.03.2018 | | |
| Github Stars | 802 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 173 | 772 | 82% |
| Uncommented Open Issues | 25 | | 97% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | 75% Coverage 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://cloud.spring.io/spring-cloud-config/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### E.1.2 .NET

- Steeltoe Configuration

#### E.1.2.1 Steeltoe Configuration

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/SteeltoeOSS/Configuration | | |
| Aktuelle Version | 2.0.0 | | |
| Releasedatum | 15.02.2018 | | |
| Github Stars | 53 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |

| | | | |
|---|---|---|---|
| Open Issues / Closed Issues | 0 | 22 | 100% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Pivotal |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://steeltoe.io/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang F - Frameworks Performanz

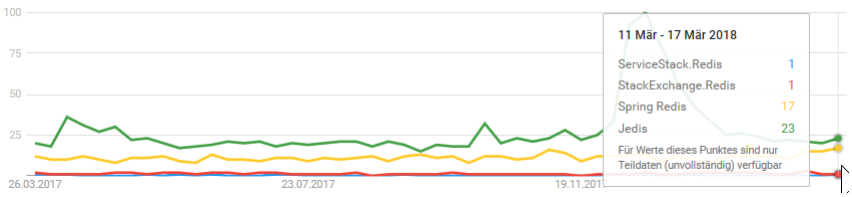
## F.1 Netflix Ribbon

### F.1.1 Java

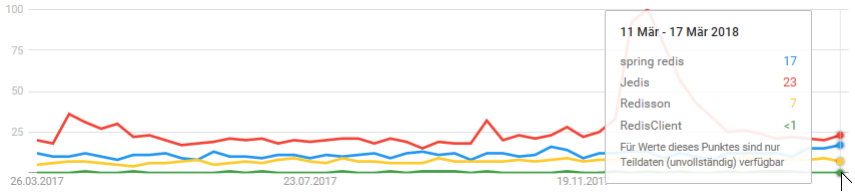
#### F.1.1.1 Netflix Ribbon

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/Netflix/ribbon | | |
| Aktuelle Version | 2.2.5 | | |
| Releasedatum | 12.03.2018 | | |
| Github Stars | 1.945 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 141 | 229 | 62% |
| Uncommented Open Issues | 71 | | 81% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle, Spring Initalizr |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | https://github.com/Netflix/ribbon/wiki |
| Projektwebseite | ☒ Ja | ☐ Nein | https://spring.io/guides/gs/client-side-load-balancing/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## F.2 Redis




### F.2.1 Java




- Spring Redis (728)
- Aredis (6)
- JDBC-Redis (6)

- Jedipus (90)
- Jedis (6.345)
- JRedis (282)
- Lettuce (1.102)
- Redis-protocol (280)
- Redisson (4.677)
- RJC (24)
- Vertx-redis-client (39)

### F.2.1.1 Jedis

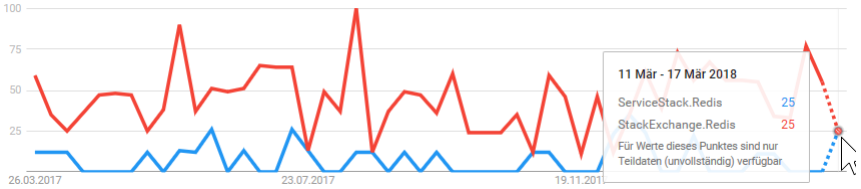
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/xetorthio/jedis | | |
| Aktuelle Version | 2.9.0 | | |
| Releasedatum | 11.09.2016 | | |
| Github Stars | 6.345 | | |
| Contributer | < 20 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 135 | 1.656 | 92% |
| Uncommented Open Issues | 23 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### F.2.1.2 Spring Redis

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-projects/spring-data-redis | | |
| Aktuelle Version | 2.0.5 | | |
| Releasedatum | 01.03.2018 | | |
| Github Stars | 728 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | | | Kein Issue Tracker auf Github. |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☒ Ja | ☐ Nein | Version 2.1.0 |

| | | | |
|---|---|---|---|
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | Pakt Verlag |

## F.2.2 .NET




- ServiceStack.Redis
- StackExchange.Redis
- CacheManager

### F.2.2.1 StackExchange.Redis

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/StackExchange/StackExchange.Redis | | |
| Aktuelle Version | 1.2.6 | | |
| Releasedatum | 01.08.2017 | | |
| Github Stars | 2.654 | | |
| Contributer | >= 50 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 244 | 559 | 70% |
| Uncommented Open Issues | 56 | | 93% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| Bücher | ☐ Ja | ☒ Nein | |

### F.2.2.2 ServiceStack.Redis

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/ServiceStack/ServiceStack.Redis | | |
| Aktuelle Version | 5.0.2 | | |
| Releasedatum | 02.01.2018 | | |
| Github Stars | 1.493 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☒ GPL |
| Open Issues / Closed Issues | | | Kein Issue Tracker auf Github |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | ServiceStack |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | Live Demo auf gistlyn.com 😊 |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://servicestack.net/redis |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

## F.3 MemCached

Google Trends liefert keine verwertbaren Resultate für die angegebenen Frameworks.
Die Verbreitung wird anhand der Github Stars beurteilt, aber nicht mit einbezogen.

### F.3.1 Java

- Spymemcached (459)
- Folsom (59)
- Xmemcached (581)
- Simple-Spring-Memcached (161)

### F.3.1.1 Xmemcached

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/killme2008/xmemcached | | |
| Aktuelle Version | 2.4.2 | | |
| Releasedatum | 23.02.2018 | | |
| Github Stars | 581 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 2 | 74 | 97% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### F.3.1.2 Spymemcached

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/dustin/java-memcached-client | | |
| Aktuelle Version | 2.11.4 | | |
| Releasedatum | 01.07.2014 | | |
| Github Stars | 459 | | |
| Contributer | 0 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 29 | 18 | 38% |
| Uncommented Open Issues | 20 | | 57% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://code.google.com/archive/p/spymemcached/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## F.3.2 .NET

- EnyimMemcachedCore (45)
- EasyCaching (89)

### F.3.2.1 EasyCaching

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/dotnetcore/EasyCaching | |
| Aktuelle Version | 0.12 | |

| | | | |
|---|---|---|---|
| Releasedatum | 02.03.2018 | | |
| Github Stars | 89 | | |
| Contributer | 1 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 0 | 23 | 100% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | 87% Coverage 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | http://easycaching.readthedocs.io/en/latest/ |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### F.3.2.2 EnyimMemcachedCore

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/cnblogs/EnyimMemcachedCore | | |
| Aktuelle Version | 2.1.0.5 | | |
| Releasedatum | 8.03.2018 | | |
| Github Stars | 45 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 1 | 46 | 98% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang G - Frameworks Robustheit

## G.1 Hystrix

In Java existiert das orginal Netflix Hystrix Projekt, eine Integration in Spring und Vertx.
Für .NET gibt es eine Umsetzung in Steeltoe und das Hystrix.Dotnet Framework.
Google Trends liefert keine verwertbaren Resultate.

### G.1.1 Java

- Netflix Hystrix (12.926)

#### G.1.1.1 Netflix Hystrix

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/Netflix/Hystrix | | |
| Aktuelle Version | 1.5.13 | | |
| Releasedatum | 13.07.2017 | | |
| Github Stars | 12.926 | | 😊 |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 205 | 1.374 | 87% |
| Uncommented Open Issues | 78 | | 95% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Failed ☹ |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Netflix |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☐ Nein | |

### G.1.2 .NET

- Hystrix.Dotnet (45)
- Steeltoe.CircuitBreaker (49)

#### G.1.2.1 Steeltoe.Circuit Breaker

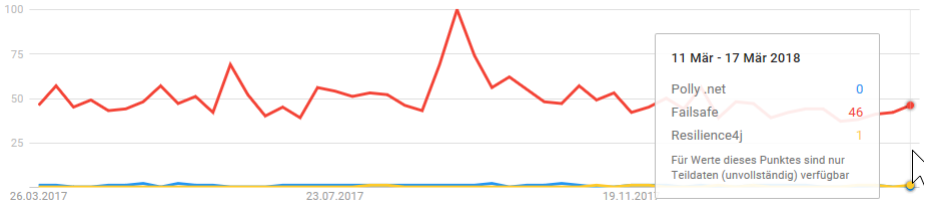
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/SteeltoeOSS/CircuitBreaker | | |
| Aktuelle Version | 2.0.0 | | |
| Releasedatum | 15.02.2018 | | |
| Github Stars | 49 | | |
| Contributer | <10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 2 | 5 | 71% |

| | | | |
|---|---|---|---|
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Pivotal |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### G.1.2.2 Hystrix.Dotnet

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/Travix-International/Hystrix.Dotnet | | |
| Aktuelle Version | 1.0.4 | | |
| Releasedatum | 01.10.2018 | | |
| Github Stars | 45 | | |
| Contributer | < 10 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 9 | 18 | 67% |
| Uncommented Open Issues | 3 | | 89% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage 63% |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# G.2 Frameworks Allgemein




## G.2.1 Java

- Resilience4j (801)
- Failsafe (2096)
- Spring Circuit Breaker (Integriert)
- Vertx Circuit Breaker (26)
- Akka (Integriert)

### G.2.1.1 Failsafe

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/jhalterman/failsafe | | |
| Aktuelle Version | 1.0.5 | | |
| Releasedatum | 10.01.2018 | | |
| Github Stars | 2096 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 43 | 82 | 66% |
| Uncommented Open Issues | 14 | | 89% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Zahlreiche Stakeholder Firmen aufgeführt. |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### G.2.1.2 Resilience4j

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/resilience4j/resilience4j | |
| Aktuelle Version | 0.12.1 | |
| Releasedatum | 15.03.2018 | |
| Github Stars | 804 | |
| Contributer | < 20 | |

| | | | | |
|---|---|---|---|---|
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL | |
| Open Issues / Closed Issues | 16 | 200 | 93% | |
| Uncommented Open Issues | 3 | | 99% | |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage 100% 😊 | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven (JFrog)/Gradle | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☐ Ja | ☒ Nein | | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☒ Ja | ☐ Nein | | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | http://resilience4j.github.io/resilience4j/ | |
| Projektwebseite | ☐ Ja | ☒ Nein | | |
| API Referenz | ☐ Ja | ☒ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

## G.2.2 .NET

- Polly

### G.2.2.1 Polly

| Teilkriterium / Frage | Wert | | Anmerkung | |
|---|---|---|---|---|
| Github Url | https://github.com/App-vNext/Polly | | | |
| Aktuelle Version | 5.8.0 | | | |
| Releasedatum | 31.01.2018 | | | |
| Github Stars | 3.700 | | | |
| Contributer | < 20 | | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL | BSD |
| Open Issues / Closed Issues | 19 | 401 | 95% | |
| Uncommented Open Issues | 2 | | 100% | |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☒ Ja | ☐ Nein | | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | Roadmap vorhanden. | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | .NET Foundation | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☒ Ja | ☐ Nein | | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://www.thepollyproject.org/ | |
| API Referenz | ☐ Ja | ☒ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

# Anhang H - Frameworks Sicherheit

## H.1 OpenId Connect[55]

### H.1.1 Java

Provider (Authorisation Server)

- Connect2id Server
- Gluu Server
- MITREid Connect
- OIDC OP Overlay for Shibboleth IdP
- Cobalt

Client Frameworks

- Apache mod_auth_openidc
- OpenID-Connect-Java-Spring-Server

### H.1.2 .NET

Provider (Authorisation Server)

- IdentityServer
- SimpleIdentityServer

Client Frameworks

- IdentityModel.OidcClient
- Spring Cloud Security

## H.2 JWT

### H.2.1 Java

- Jose4j
- Nimbus JOSE+JWT
- Java JWT
- Resteasy
- Apache Oltu-Jose
- Apache CXF

[55] Siehe http://openid.net/developers/certified/

### H.2.2 .NET

- Microsoft.AspNetCore
- Microsoft.IdentityModel.Tokens

# Anhang I - Frameworks Consumer Driven Contracts

## I.1 Pact


### I.1.1 Java

Für Java gibt es zurzeit nur das Pact-Jvm Framework

#### I.1.1.1 Pact-Jvm

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/pact-foundation/pact-net | | |
| Aktuelle Version | 3_5_14_2.11 | | |
| Releasedatum | 19.03.2018 | | |
| Github Stars | 493 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 150 | 507 | 77% |
| Uncommented Open Issues | 23 | | 96% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### I.1.2 .NET

Für .NET Core existiert das Pact.NET Framework, welches seit kurzem auch für .NET Core verfügbar ist.

#### I.1.2.1 Pact.NET

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/pact-foundation/pact-net | |
| Aktuelle Version | 2.2.5 | |
| Releasedatum | 27.03.2018 | |
| Github Stars | 257 | |
| Contributer | < 10 | |

| | | | |
|---|---|---|---|
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 7 | 137 | 95% |
| Uncommented Open Issues | 1 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☒ Ja | ☐ Nein | Seek Foundation |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# I.2 Spring Cloud Contract

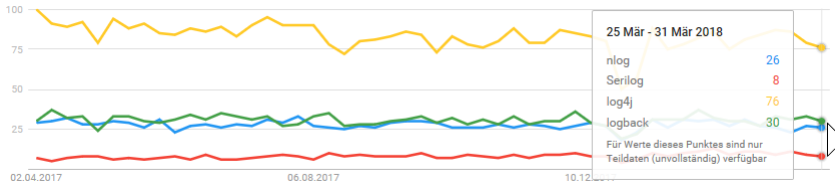
Für Spring Clound Config gibt es zurzeit nur eine Implementierung in Java.

## I.2.1 Java

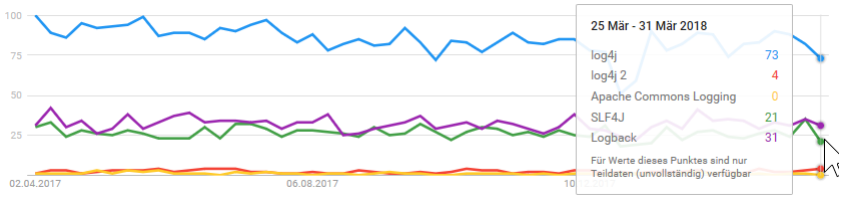
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-contract | | |
| Aktuelle Version | V2.0.0.M8 | | |
| Releasedatum | 23.03.2018 | | |
| Github Stars | 262 | | |
| Contributer | < 50 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 38 | 561 | 94% |
| Uncommented Open Issues | 13 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | 60% Coverage 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang J - Frameworks Monitoring

## J.1 Logging Frameworks Allgemein




## J.2 Java




- Log4j
- Log4j 2
- Apache Commons Logging
- SLF4J
- Logback

Obwohl Log4J einen höheren Trend aufweist als Log4J 2 wird letzteres Framework untersucht, da es für Log4J seit mehr als 5 Jahren kein neues Release mehr gab.

### J.2.1.1 Log4J 2

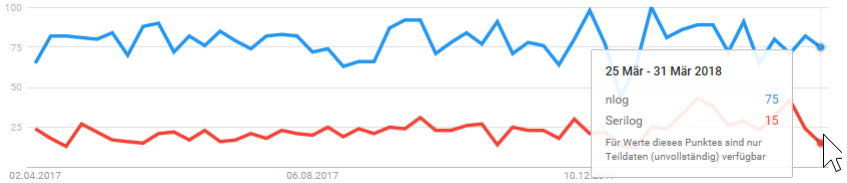
| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/apache/logging-log4j2 | | |
| Aktuelle Version | Log4j-2.11.0-rc1 | | |
| Releasedatum | 11.03.2018 | | |
| Github Stars | 441 | | |
| Contributer | 0 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 552 | 1735 | 76% |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Projektwebseite | ☒ Ja | ☐ Nein | https://logging.apache.org/log4j/2.x/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☒ Ja | ☐ Nein | |

### J.2.1.2 Logback

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/qos-ch/logback | | |
| Aktuelle Version | V_1.3.0-alpha | | |
| Releasedatum | 11.02.2018 | | |
| Github Stars | 1.222 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☐ Apache | ☒ GPL |
| Open Issues / Closed Issues | 923 | 999 | 52% |
| Uncommented Open Issues | | | |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://logback.qos.ch/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## J.3 .NET




Zum gegenwärtigen Zeitpunkt sind nur 2 Third-Party Frameworks verfügbar, welche sich in ASP.NET Core integrieren lassen.[56]

- NLog
- Serilog

### J.3.1.1 NLog

| Teilkriterium / Frage | Wert | Anmerkung |
|---|---|---|
| Github Url | https://github.com/NLog/NLog.Web | |
| Aktuelle Version | 4.5.0 | |

[56] Siehe https://www.c-sharpcorner.com/article/logging-framework-in-asp-net-core-2-0/

| | | | | |
|---|---|---|---|---|
| Releasedatum | 27.03.2018 | | | |
| Github Stars | 76 | | | |
| Contributer | < 10 | | | |
| Lizenz | ☐ MIT | ☐ Apache | ☐ GPL | BSD |
| Open Issues / Closed Issues | 14 | 253 | 95% | |
| Uncommented Open Issues | 5 | | 98% | |
| Tests | ☒ Ja | ☐ Nein | | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage 57% | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget | |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | | |
| Installationsanleitung | ☐ Ja | ☐ Nein | | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube | |
| Quickstarts | ☒ Ja | ☐ Nein | | |
| Use Cases | ☒ Ja | ☐ Nein | Stackify | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://nlog-project.org/ | |
| API Referenz | ☒ Ja | ☐ Nein | | |
| Bücher | ☐ Ja | ☒ Nein | | |

### J.3.1.2 Serilog

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/serilog/serilog | | |
| Aktuelle Version | V2.6.0 | | |
| Releasedatum | 04.12.2017 | | |
| Github Stars | 1.851 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 15 | 1.129 | 99% |
| Uncommented Open Issues | 2 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | Pluralsight, YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://serilog.net/ |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# J.4 Zipkin

Google Trends liefert für die Verbreitung keine verwertbaren Resultate. Es werden daher die Anzahl der Github Stars als Auswahlkriterium verwendet.

## J.4.1 Java

- Brave (971)
- Cassandra-zipkin-tracing (27)
- Dropwizard Zipkin (35)
- Htrace (74)
- Spring Cloud Sleuth (631)
- Wingtips (287)

### J.4.1.1 Brave

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/openzipkin/brave | | |
| Aktuelle Version | 4.18.2 | | |
| Releasedatum | 12.03.2018 | | |
| Github Stars | 971 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 51 | 621 | 92% |
| Uncommented Open Issues | 10 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | https://github.com/openzipkin/brave-webmvc-example |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### J.4.1.2 Spring Cloud Sleuth

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/spring-cloud/spring-cloud-sleuth | | |
| Aktuelle Version | 1.3.3 | | |
| Releasedatum | 27.03.2018 | | |
| Github Stars | 631 | | |
| Contributer | < 50 | | |
| Lizenz | ☐ MIT | ☒ Apache ☐ GPL | |
| Open Issues / Closed Issues | 7 | 916 | 99% |
| Uncommented Open Issues | 2 | | 100% |

| | | | |
|---|---|---|---|
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | 70% Coverage 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://cloud.spring.io/spring-cloud-sleuth/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## J.4.2 .NET

- Zipkin4net (108)

### J.4.2.1 Zipkin4net

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/openzipkin/zipkin4net | | |
| Aktuelle Version | 1.2.0 | | |
| Releasedatum | 11.03.2018 | | |
| Github Stars | 111 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 17 | 182 | 91% |
| Uncommented Open Issues | 7 | | 96% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# J.5 Prometheus

Für Prometheus existieren sowohl für Java als auch .NET zurzeit nur die offiziellen von Prometheus zur Verfügung gestellten Client Frameworks. Über die Verbreitung kann mit Hilfe von Google Trends keine Aussage getroffen werden.

## J.5.1 Java

- Client_Java

### J.5.1.1 Client_Java

Das Client_Java Framework bietet unterstützung für:

- Caffeine
- Dropwizard
- Graphite
- Guava
- Hibernate
- Hotspot
- Jetty
- Log4j & log4j2
- Logback
- Spring boot
- Spring web
- Vertx

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/prometheus/client_java | | |
| Aktuelle Version | 0.3.0 | | |
| Releasedatum | 27.03.2018 | | |
| Github Stars | 422 | | |
| Contributer | < 10 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 14 | 349 | 96% |
| Uncommented Open Issues | 4 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☐ Ja | ☒ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## J.5.2 .NET

- Prometheus-net

### J.5.2.1 Prometheus-net

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/prometheus-net/prometheus-net | | |
| Aktuelle Version | 2.0.0 | | |
| Releasedatum | 26.02.2018 | | |
| Github Stars | 120 | | |
| Contributer | < 20 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 0 | 82 | 100% |
| Uncommented Open Issues | 0 | | 100% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☐ Ja | ☒ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☐ Ja | ☒ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang K - Frameworks Dokumentation und Metadaten

## K.1 Swagger und OpenAPI

Google Trends liefert keine vernünftigen Resultate zur Ermittlung der Verbreitung. Es werden Github Stars als Auswahlkriterium verwendet.

### K.1.1 Java

Für Java existieren folgende Client Frameworks:

- Assertj-swagger (107)
- Binder-swagger-java (56)
- Dropwizard-swagger (120)
- Elide (314)
- Jobby-swagger (745)
- Restlet-framework (556)
- Springfox (2.513)
- Swagger-codegen-maven-plugin (24)
- Swagger2markup (1.109) (Markdown Generation)
- Swagger2markup-gradle-plugin (45)
- Swagger-maven-plugin (451)

#### K.1.1.1 Springfox

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/springfox/springfox | | |
| Aktuelle Version | 2.8.0 | | |
| Releasedatum | 15.01.2018 | | |
| Github Stars | 2.513 | | |
| Contributer | < 50 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 204 | 2.133 | 91% |
| Uncommented Open Issues | 11 | | 99% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage 94% 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☐ Ja | ☒ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | http://springfox.github.io/springfox/ |
| API Referenz | ☒ Ja | ☐ Nein | |

| | | | |
|---|---|---|---|
| Bücher | ☐ Ja | ☒ Nein | |

### K.1.1.2 Jobby-Swagger

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/jooby-project/jooby | | |
| Aktuelle Version | 1.3.0 | | |
| Releasedatum | 22.02.2018 | | |
| Github Stars | 746 | | |
| Contributer | < 20 | | |
| Lizenz | ☐ MIT | ☒ Apache | ☐ GPL |
| Open Issues / Closed Issues | 40 | 1.011 | 96% |
| Uncommented Open Issues | 17 | | 98% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | Coverage 100% 😊 |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Maven/Gradle |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☐ Ja | ☒ Nein | |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☒ Ja | ☐ Nein | https://jooby.org/ |
| API Referenz | ☒ Ja | ☐ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

## K.1.2 .NET

Für .NET Core existieren folgende Frameworks:

- Swashbuckle.AspNetCore (1.295)
- NSwag (1.139) https://github.com/RSuter/NSwag
- QSwag (19)
- Nancy.Swagger (103)

### K.1.2.1 Swashbuckle.AspNetCore

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/domaindrivendev/Swashbuckle.AspNetCore | | |
| Aktuelle Version | 2.3.0 | | |
| Releasedatum | 11.03.2018 | | |
| Github Stars | 1.295 | | |
| Contributer | < 20 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 113 | 558 | 83% |
| Uncommented Open Issues | 43 | | 94% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☐ Ja | ☒ Nein | |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |

| | | | |
|---|---|---|---|
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☐ Ja | ☒ Nein | |
| Beispielimplementierungen | ☐ Ja | ☒ Nein | |
| Einsatzszenarios | ☐ Ja | ☒ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube, Pluralsight |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

### K.1.2.2 NSwag

| Teilkriterium / Frage | Wert | | Anmerkung |
|---|---|---|---|
| Github Url | https://github.com/RSuter/NSwag | | |
| Aktuelle Version | 11.16.1 | | |
| Releasedatum | 23.03.2018 | | |
| Github Stars | 1.141 | | |
| Contributer | < 50 | | |
| Lizenz | ☒ MIT | ☐ Apache | ☐ GPL |
| Open Issues / Closed Issues | 204 | 1.051 | 84% |
| Uncommented Open Issues | 37 | | 97% |
| Tests | ☒ Ja | ☐ Nein | |
| Automatisierter Build | ☒ Ja | ☐ Nein | |
| Releaseplan / Meilensteine | ☒ Ja | ☐ Nein | Roadmap |
| Neue Versionen angekündigt | ☐ Ja | ☒ Nein | |
| Angaben zur Finanzierung | ☐ Ja | ☒ Nein | |
| Installation über Paketmanager | ☒ Ja | ☐ Nein | Nuget |
| Download von Zip / Tar | ☐ Ja | ☐ Nein | |
| Installationsanleitung | ☐ Ja | ☐ Nein | |
| Beschreibung Verwendungszweck | ☒ Ja | ☐ Nein | |
| Architekturübersichtsdiagramme | ☒ Ja | ☐ Nein | |
| Beispielimplementierungen | ☒ Ja | ☐ Nein | |
| Einsatzszenarios | ☒ Ja | ☐ Nein | |
| Video Tutorials | ☒ Ja | ☐ Nein | YouTube |
| Quickstarts | ☒ Ja | ☐ Nein | |
| Use Cases | ☒ Ja | ☐ Nein | |
| Strukturierte Dokumentation | ☒ Ja | ☐ Nein | |
| Projektwebseite | ☐ Ja | ☒ Nein | |
| API Referenz | ☐ Ja | ☒ Nein | |
| Bücher | ☐ Ja | ☒ Nein | |

# Anhang L - Frameworks Dienstmanagement

## L.1 Java

- Spring Boot Actuator (Teil des Spring Boot Frameworks)

## L.2 .NET

- Steeltoe Management (https://steeltoe.io/docs/steeltoe-management/ )
- Dotnet.Microservice (https://github.com/lynxx131/dotnet.microservice )
- HealthChecks (https://github.com/dotnet-architecture/HealthChecks )

# ABBILDUNGSVERZEICHNIS

# TABELLENVERZEICHNIS

# LITERATURVERZEICHNIS

Arendt, D. T. (26. November 2015). *Software-Qualität*. Von https://www.uni-marburg.de/fb12/arbeitsgruppen/swt/lehre/files/sevo1516/sevo_ws1516_08_qual.pdf abgerufen

Aslett, M. (2013). *Database Landscape Map*. Abgerufen am 3. Mai 2017 von 451 Resarch Group: http://files.meetup.com/1789394/Matt%20Aslett%20-%20DB%20landscape.pdf

Bayer, T. (12. März 2018). *Protocol Buffers, Etch, Avro und Thrift im Vergleich*. Von https://www.predic8.de/protocol-buffers-etch-thrift-vergleich.htm abgerufen

Brown, Z., & Horan, M. (20. März 2018). *Open Source Summit*. Von http://schd.ws/hosted_files/ossna2017/81/Building%20Microservices%20w-.NET%20Core%20and%20Steeltoe.pdf abgerufen

Container, W. (25. Mai 2016). *Windows Pro*. Von https://www.windowspro.de/marcel-kueppers/container-windows-server-2016-funktionsweise-typen-anwendungen abgerufen

de la Torre, C. (2017). *Modernize existing .NET Applications with Azure Cloud and Windows Containers.* Redmond: Microsoft.

de la Torre, C., Wagner, B., & Rousos, M. (2018). *.NET Microservices: Architecture for Containerized .NET Applications.* Redmond: Microsoft.

Doumack, A. (6. März 2018). *Diplomarbeit Evaluierung von Reporting Frameworks am Beispiel der ausgewählten Open-Source-Anwendung.* Von http://blogs.gm.fh-koeln.de/faeskorn/files/2008/11/diplom_eval_ado.pdf abgerufen

Eggert, R. (6. März 2018). *10 Auswahlkriterien für PHP Frameworks*. Von https://www.thewebhatesme.com/allgemein/10-auswahlkriterien-fur-php-frameworks/ abgerufen

Emer, J. C. (21. März 2018). *Client,Network, Server and Application Caching on the Web*. Von https://hackernoon.com/client-network-server-and-application-caching-on-the-web-2fcdcd856886 abgerufen

Engines, D. (5. März 2818). *DB-Engines Ranking von Key-Value Stores*. Von https://db-engines.com/de/ranking/key-value+store abgerufen

Fabrizio Montesi, J. W. (26. Februar 2018). *Circuit Breakers, Discovery, and API Gateways in Microservices*. Von https://arxiv.org/abs/1609.05830 abgerufen

Fowler, M. (9. Mai 2011). *Tolerant Reader*. Abgerufen am 12. Juli 2017 von https://martinfowler.com/bliki/TolerantReader.html

Fowler, M., & Lewis, J. (2015). Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr? *Objectspectrum*(1).

Fowler, S. S. (2016). *Production Ready Microservices.* Sebastopol: O'Reilly.

Gucer, V., & Narain, S. (2015). *Creating Applications in Bluemix Using the Microservices Approach.* IBM Redbooks.

Hoffman, K. (2017. August 2017). *Building Microservices with ASP.NET Core.* England: O'Reilly Media.

Horsdal Gammelgaard, C. (2017). *Microservices in .NET Core.* Shelter Island: Manning Publications.

it-economics. (7. März 2018). *Software Lizenzen - Ein Überblick*. Von https://www.it-economics.de/blog/2016-3/pt9oi6kzodgcl1tmsfc1yn1vzq4usb abgerufen

Jackson, M., Crouch, S., & Baxter, R. (6. März 2018). *Software Evaluation: Criteria-based Assessment.* Von https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf abgerufen

Lodderstedt, T. (23. März 2018). *OpenID Connect: Login mit OAuth, Teil 1 – Grundlagen*. Von https://www.heise.de/developer/artikel/OpenID-Connect-Login-mit-OAuth-Teil-1-Grundlagen-2218446.html?seite=all abgerufen

Long, J., & Bastiani, K. (2017). *Claud Native Java.* Sebastopol CA: OReilly.

Luckart, D. H.-D. (7. März 2018). *Open Source und Freie Software*. Von https://saar.infowiss.net/projekte/ident/highlights/index-3/lizenzen/ abgerufen

MarketPlace, e. (15. November 2017). *Eclipse MarketPlace*. Von https://marketplace.eclipse.org/content/spring-tools-aka-spring-ide-and-spring-tool-suite#group-details abgerufen

Newmann, S. (2015). *Building Microservices: Design Fine-Grained Systems.* Sebastopol: OReilly books.

Pivotal Software. (2017). *Spring Cloud*. Von http://projects.spring.io/spring-cloud/ abgerufen

Richardson. (20. Februar 2018). *Microservices: From Design to Deployment.* Von https://www.nginx.com/resources/library/designing-deploying-microservices/ abgerufen

Scholl, B., Swanson, T., & Frenandez, D. (2016). *Microservices with Docker on Microsoft Azure.* Crawfordsville: Pearson Education.

Schönbach, B. (22. März 2018). *Nutzer-Authentifizierung in Microservice-Umgebungen*. Von https://www.heise.de/developer/artikel/Nutzer-Authentifizierung-in-Microservice-Umgebungen-3651632.html?seite=all abgerufen

sdxCentral. (2016). *What is Docker Container?* (sdxCentral) Abgerufen am 30. Juni 2017 von https://www.sdxcentral.com/cloud/containers/definitions/what-is-docker-container-open-source-project/

Serra, J. (1. Januar 2017). *What is Polyglot Persistence?* Abgerufen am 3. Mai 2017 von http://www.jamesserra.com/archive/2015/07/what-is-polyglot-persistence/

TechGlimpse. (5. März 2018). *Waht is Docker, Difference between Docker and VM, Installation of Docker and its usage*. Von https://techglimpse.com/docker-installation-tutorial-centos/ abgerufen

Video2Brain, T. W. (21. Juni 2017). *Einführung in Windows Container*. Von https://www.video2brain.com/de/tutorial/einfuehrung-in-windows-container abgerufen

Vitz, M. (26. Februar 2018). *Microservice Infrastruktur*. Von https://www.sigs-datacom.de/uploads/tx_dmjournals/vitz_JS_06_16_5Xuz.pdf abgerufen

Wikipedia. (10. November 2017). *Java Version History*. Von https://en.wikipedia.org/wiki/Java_version_history abgerufen

Wikipedia. (18. Oktober 2017). *NetFramework*. Von https://de.wikipedia.org/wiki/.NET_Framework abgerufen

Wolff, E. (2016). *Microservices Grundlagen flexibler Softwarearchitekturen.* Heidelberg: dpunkt.verlag.

Wolff, E. (30. März 2018). *Docker: zentralisiertes Logging mit dem ELK-Stack*. Von https://entwickler.de/online/development/logging-leicht-gemacht-elk-stacks-128685.html abgerufen

Zytrax. (18. November 2016). *DNS*. Abgerufen am 12. Juli 2017 von http://www.zytrax.com/books/dns/