

**Hochschule Flensburg**

# **B A C H E L O R – T H E S I S**

Thema:       Konzeption und prototypische Realisierung einer Microservice-Architektur mit Cloud-native Technologien

von:           Torben Schuhmacher

Matrikel-Nr.:       630298

Studiengang:       Angewandte Informatik

Betreuer/in und  
Erstbewerter/in:   Dipl.-Math. Jochen Stamp

Zweitbewerter/in:   Prof. Dr. rer. nat. Osmanbey Uzunkol

Ausgabedatum:       03.05.2021

Abgabedatum:       03.07.2021

# Hochschule Flensburg



Bachelorarbeit zum Thema:

**Konzeption und prototypische Realisierung einer  
Microservice-Architektur mit Cloud-native-Technologien**

**Name:** Torben Schuhmacher

**Matrikel-Nr.:** 630298

**Betreuer und Erstbewerter:** Dipl.-Math. Jochen Stamp

**Zweitbewerter:** Prof. Dr. rer. nat. Osmanbey Uzunkol

**Ausgabedatum:** 03.05.2021

**Abgabedatum:** 03.07.2021

# **Sperrvermerk**

Die vorliegende Bachelor-Thesis mit dem Titel:

**Konzeption und prototypische Realisierung einer Microservice-Architektur  
mit Cloud-native-Technologien**

beinhaltet interne und vertrauliche Informationen der Firma:

**Northern-Lights GmbH**

**Die Weitergabe des Inhalts der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Die Einsichtnahme ist ebenfalls untersagt, wird jedoch den Begutachtenden der Arbeit sowie die an der Bewertung beteiligten Personen ausdrücklich gestattet. Es dürfen keinerlei Kopien oder Abschriften auch in digitaler Form gefertigt werden.**

Ausnahmen bedürfen der schriftlichen Genehmigung der Firma:

**Northern-Lights GmbH**

# Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Flensburg, 3. Juli 2021

---

Name (Unterschrift)

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Zielsetzung . . . . .	1
1.2	Gliederung der Arbeit . . . . .	1
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	Microservices . . . . .	3
2.2	Docker . . . . .	4
2.3	Kubernetes . . . . .	6
2.4	Cloud-Computing . . . . .	9
<b>3</b>	<b>Analyse</b>	<b>11</b>
3.1	Qualitative Merkmale . . . . .	11
3.2	Cloud-native-Architektur . . . . .	12
3.2.1	Cloud-Ansätze und Prinzipien . . . . .	13
<b>4</b>	<b>Konzept</b>	<b>19</b>
<b>5</b>	<b>Realisierung</b>	<b>23</b>
<b>6</b>	<b>Fazit</b>	<b>28</b>
	<b>Glossar</b>	<b>29</b>
	<b>Literaturverzeichnis</b>	<b>33</b>
	<b>Abbildungsverzeichnis</b>	<b>38</b>

# **1 Einleitung**

Die Cloud-Migration ist heutzutage kaum noch wegzudenken. Immer mehr Unternehmen in Deutschland nutzen Cloud-Computing und viele andere diskutieren den Einsatz der Cloud bereits. Cloud-Migration beschreibt den Prozess der Verlagerung von Rechenressourcen wie Daten, Anwendungen und IT-Prozessen in eine Cloud-Computing-Umgebung. Große Unternehmen besitzen eine große Menge an Daten, die zu groß sind, um sie intern zu verarbeiten, ohne dass die Kosten, das Finanzvolumen überschreiten. Cloud-Computing ist darauf ausgelegt, mit Big Data umzugehen und bietet Vorteile in Kosteneinsparungen, Skalierbarkeit und einer höheren Sicherheit. Dabei ist eine genaue Planung der Architektur maßgeblich. Um den Prozess der Migration effizient bewältigen zu können, benötigen Unternehmen ausreichende Kenntnisse über die Cloud-Architektur. Mangelnde Kenntnisse führen dazu, dass der Umfang und die Komplexität der Migration falsch eingeschätzt werden, technische Hindernisse werden im Vorfeld nicht bedacht. Die längere Migrationszeit führt dadurch zu höheren Kosten als vermutet. Durch fehlendes Wissen werden wichtige Probleme in der Planungsphase und Migrationsstrategie nicht bedacht. Das führt schließlich zu einem höheren Zeit- und Kostenaufwand, was dem Ziel der Cloud-Migration widerspricht, Kosten zu senken.

## **1.1 Zielsetzung**

Das Ziel der Arbeit ist das Designen und prototypischen Entwickeln einer Cloud-nativen Architektur. Zum Verringern der Komplexität werden die verwendeten Cloud-Technologien erläutert und essenzielle Eigenschaften der Architektur vorgestellt. Das Design wird dabei anhand der zuvor analysierten Architektur-Ansätze abgebildet und mögliche Probleme, die beim Konzipieren des Designs der Architektur auftreten können, werden aufgezeigt. Es werden Teile der Architektur realisiert, wobei der Fokus auf der Containervirtualisierung von Anwendungen und deren Bereitstellung liegt.

## **1.2 Gliederung der Arbeit**

Die Arbeit ist in sechs Teile gegliedert, wobei der erste Teil die Einleitung darstellt. Hier wird das Thema der Arbeit vorgestellt und auf die Motivation der Arbeit eingegangen.

Im zweiten Teil werden die Grundlagen erläutert, die Begriffe und Konzepte erklärt, die für das weitere Verständnis der Arbeit von Bedeutung sind.

Die Analyse stellt den dritten Teil der Arbeit dar, die sich mit dem zuvor definierten Ziel auseinandersetzt. Dazu werden Anforderungen ausgearbeitet die für das konzeptionieren der Architektur benötigt werden.

Durch die Ergebnisse des dritten Teils werden im vierten Teil ein mögliches Konzept vorgestellt. Anhand der Anforderungen wird das Design gewählt und durch Diagramme grafisch dargestellt.

Der fünfte Teil der Arbeit beschreibt die Realisierung der zuvor gewählten Designentscheidung. Schließlich wird die Funktionsweise erläutert, an denen man die Zusammenhänge der einzelnen Komponenten verstehen kann.

Zum Schluss im sechsten Teil der Arbeit wird der Zusammenhang der Arbeit erläutert und ein mögliche zukünftige Perspektive vorgestellt.

## 2 Grundlagen

### 2.1 Microservices

In der Anwendungsentwicklung gibt es viele Architekturkonzepte, die sich in ihren grundlegenden Konzepten und deren Zusammenspiel innerhalb eines Softwaresystems unterscheiden. Die Software-Architektur legt dabei fest, in welchem Verhältnis Softwarekomponenten zueinanderstehen und was für Anforderungen sie umsetzen. IT-Systeme in Bezug auf Anwendungssysteme beinhalten grundsätzliche Funktionen zur Datenerhaltung in Form von Datenbanken, Verarbeitung von Daten in Form der Geschäftslogik und Präsentation der Daten in Form einer UI.

Bei einer monolithischen Software-Architektur wird die gesamte Logik der Anwendung als ein zusammenhängender Prozess bereitgestellt und ausgeführt. Anwendungsmodule sind eng miteinander gekoppelt und stark voneinander abhängig.<sup>1</sup> Die Anwendungsarchitektur kann in die drei Hauptteile User Interface, Geschäftslogik und Datenzugriffsschicht aufgeteilt werden. Dabei stellt das User Interface die Benutzeroberfläche der Anwendung dar. Funktionen und Ergebnisse der werden Anwendungen grafisch dargestellt, sodass der Benutzer möglichst einfach mit damit interagieren kann.<sup>2</sup> Der Kernbereich der Anwendung wird durch die Geschäftslogik aufgezeigt, dabei vereint sie mehrere Aufgaben in sich, beispielsweise wie Daten verändert, validiert und ermittelt werden. Sie stellt damit das Verbindungsglied zwischen dem User Interface und der Datenzugriffsschicht dar. Die Aufgabe der Datenzugriffsschicht besteht in der Erleichterung des Zugriffs von logischen Datenbank- und Dateisystemschnittstellen sowie in der Isolation der Anwendung vom Datenspeicher.<sup>3</sup> Die Monolithische Software-Architektur folgt keiner expliziten Gliederung in Teilsysteme und steht dazu im genauen Gegensatz zu dem Microservices-Architektur Ansatz.

---

<sup>1</sup>[16, S.4]

<sup>2</sup>[14, User Interface für Software]

<sup>3</sup>[16, S.5]



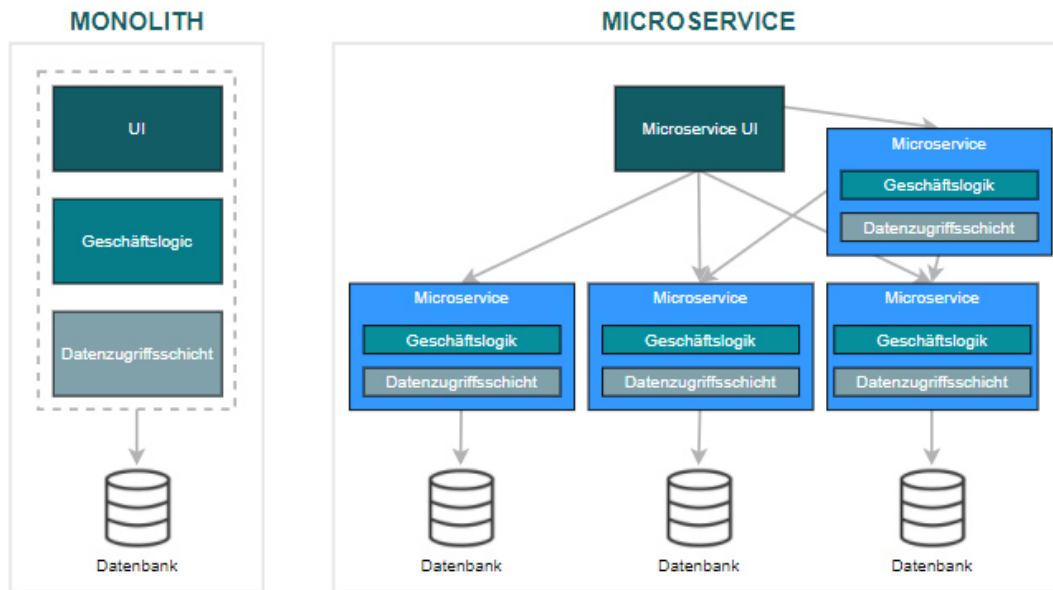


Abbildung 1: Architekturkonzepte von Monolith und Microservices

Die Microservice-Architektur ist ein Ansatz, bei dem eine Anwendung aus mehreren kleinen, lose gekoppelten Services bereitgestellt wird. Dabei übernimmt jeder Service eine andere Kernfunktion der Funktionalität der Anwendung. Hingegen der monolithischen Software-Architektur werden Microservices unabhängig voneinander entwickelt und bereitgestellt. Jeder Service stellt dabei einen eigenen Prozess dar. Datenbanken oder Datenmodelle werden nicht untereinander aufgeteilt, jeder Microservice verfügt und verwaltet seine eigene Datenbank und Daten selbst.<sup>4</sup> Cross-Cutting Concern wie die Authentifizierung finden dabei über eine externe IAM-Komponente statt.

## 2.2 Docker

Docker ist eine Software, die das Erstellen, Bereitstellen und Ausführen von containerisierten Anwendungen erleichtert. Dabei stellt jeder laufende Container einen eigenen Prozess innerhalb des Host-Betriebssystems dar, jedoch greifen Host-Rechner und die Container auf denselben laufenden Linux-Kernel zurück. Ein Docker-Container ist keine virtuelle Maschine anders als eine virtuelle Maschine beansprucht ein Container

<sup>4</sup>[16, S.10 ff.]

keine Hardware-Ressourcen von dem Host-Rechner und verfügt über kein eigenes Betriebssystem und Hypervisor. Docker isoliert Container durch die Funktion von Namespaces des Linux-Kernels in Prozessen voneinander.<sup>5</sup> Die Fähigkeit, Prozesse und Applikationen separat betreiben zu können, ist die Kernaufgabe von Docker.

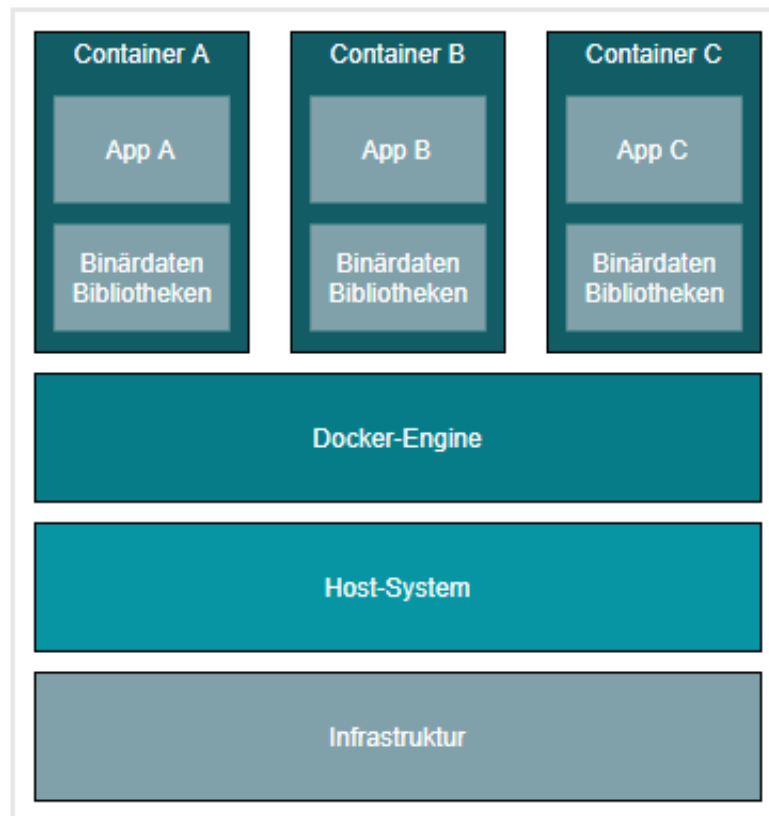


Abbildung 2: Container-Infrastruktur

Die Basis eines Docker-Containers ist das Docker-Image. Ein Image enthält alles Nötige, um die Anwendung auszuführen, einschließlich Bibliotheken, Hilfsprogrammen und statischen Daten.<sup>6</sup> Das Grundgerüst der Containererstellung ist das Image, aus dem Docker beliebig viele Container erzeugen kann.

---

<sup>5</sup>[22, S.7]

<sup>6</sup>[11, S.1]

Docker-Images werden durch ein Dockerfile erzeugt. Ein Dockerfile ist ein Textdokument, in dem der Aufbau des Docker-Images beschrieben wird. Es enthält Anweisungen wie das Kopieren von Software und Abhängigkeiten, die Ausführung von Skripten und Befehlen und das Parent-Image, auf dem die Basis des Docker-Images basieren soll. Images können auf einer Container-Registry öffentlich oder privat abgespeichert werden.<sup>7</sup> Dabei können die öffentlichen Images von jedem als Parent-Image verwendet werden.

Wenn Docker auf Basis eines Images einen Container startet, wird eine weitere beschreibbare containerspezifische Datenschicht erzeugt. Nur diese Datenschicht ist vom Prozess im Container beschreibbar, alle anderen Schichten, die Teil des Images sind, werden nur als lesbare Schichten eingebunden. Die containerspezifische Datenschicht ist nicht persistent. Durch den Neustart des Containers wird sie nicht wiederherstellbar gelöscht und der Container wird dadurch in seinen ursprünglichen Zustand zurückgesetzt. Docker nutzt für die persistente Datenerhaltung von Daten, die im laufenden Betrieb des Containers anfallen, Volumes. Volumes sind Verzeichnisse, die im lokalen Dateisystem angelegt werden. Die Daten der Volumes können von mehreren Containern geteilt werden, dies ermöglicht ein Austausch von Daten zwischen den Containern.<sup>8</sup>

## 2.3 Kubernetes

Kubernetes ist eine containerzentrierte Managementumgebung zur Verwaltung von containerisierten Arbeitslasten und Services. Dabei vereinfacht Kubernetes die deklarative Konfiguration, Automatisierung der Bereitstellung und Skalierung von Arbeitslasten.<sup>9</sup> Container werden dabei nur von Kubernetes verwaltet, für die Erstellung der containerisierten Arbeitslasten benötigt Kubernetes die Unterstützung einer Software, die eine Containervirtualisierung von Anwendung ermöglicht.

---

<sup>7</sup>[22, S.7]

<sup>8</sup>[36, Use volumes]

<sup>9</sup>[58, Was ist Kubernetes?]

Die Kubernetes-Architektur basiert auf dem Zusammenschluss verschiedener Server, den Cluster. Der Kubernetes-Cluster besteht aus zwei Teilen, dem Master-Nodes und den Worker-Nodes, den Rechenmaschinen, auf denen die containerisierten Anwendungen ausgeführt werden.<sup>10</sup> Der Master-Node stellt die Steuerungsebene des Clusters bereit, der die Kontrolle und Überwachung der Worker-Nodes gewährleistet, er besteht aus mehreren verschiedenen Komponenten, die unterschiedliche Aufgaben in sich vereinen.<sup>11</sup> Der API-Server ist eine Komponente des Masters, der die Kubernetes-API nach außen hin verfügbar macht, die Kommunikation mit dem Cluster kann nur über diese Schnittstelle erfolgen. Der API-Server kommuniziert mit dem Controller-Manager und dem Scheduler innerhalb des Masters. Im Controller-Manager werden mehrere Controller vereint, die sich um die Überwachung der Worker-Nodes, Ausfallsicherheit der Arbeitslasten, Verbindung von Services und die Verwaltung der Zugriffsrechte durch Namespaces und API-Tokens kümmert. Jeder Controller ist dabei ein separater Prozess, der einfachheitshalber zu einem einzigen Prozess zusammengefasst und ausgeführt wird.<sup>12</sup> Der primäre Datenspeicher in Kubernetes wird als Etcd bezeichnet. Etcd ist eine verteilte Datenbank für Schlüsselwertpaare, der einen sicheren Speicherplatz von kritischen Daten bereitstellt. Alle Konfigurationsdaten und Informationen zum Status der Cluster befinden sich im Etcd.<sup>13</sup>

---

<sup>10</sup>[12, S. XVIII]

<sup>11</sup>[**RedHatKube2021**]

<sup>12</sup>[38, Wie funktioniert Kubernetes?]

<sup>13</sup>[49, S.19]

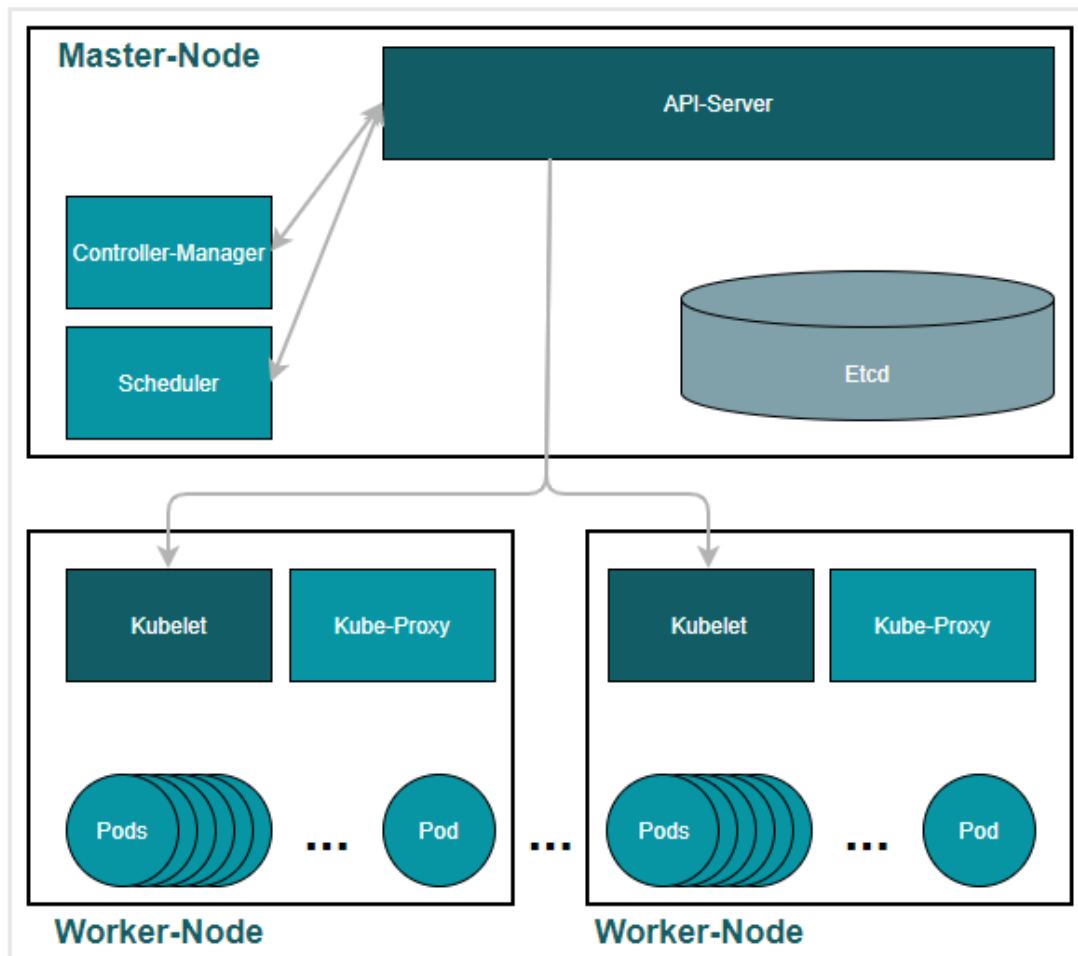


Abbildung 3: Kubernetes-Infrastruktur

Worker-Nodes sind Arbeitsmaschinen in Kubernetes die von dem Master-Node verwaltet werden. Jeder Worker-Node enthält die für den Betrieb der Pods notwendigen Dienste. Ein Pod ist eine Ansammlung aus einem oder mehrerer Container. Mithilfe von Pods wird sichergestellt, dass jeder enthaltene Container dieselben Ressourcen und dasselbe Netzwerk verwenden. Innerhalb des Systems von Kubernetes verwenden die Container im gleichen Pod dieselben Rechenressourcen. Ein Worker-Node kann mehrere Pods in sich vereinen, dabei kann der Node als eine virtuelle oder physische Maschine bereitgestellt werden.<sup>14</sup> Kubelet ist eine Anwendung, die auf jedem Node im Cluster

<sup>14</sup>[4, S.40]

ausgeführt wird. Er ist für die Kommunikation mit dem Master-Node verantwortlich, zusätzlich stellt Kubelet sicher, dass Container in einem Pod ausgeführt werden. Auf jedem Node befindet sich eine Container-Laufzeit, die für das Ausführen der Container verantwortlich ist. Dabei unterstützt Kubernetes verschiedene Container-Laufzeiten wie Docker oder containerd.<sup>15</sup> Die Komponente, die für die Kubernetes-Service-Abstraktion jedes Worker-Nodes zuständig ist der Kube-Proxy. Er verwaltet die Netzwerkregeln für das Load Balancing der Anfragen an die Pods eines Services.<sup>16</sup>

## 2.4 Cloud-Computing

Der Begriff Cloud-Computing steht für die internetbasierte Bereitstellung von Speicherplatz, Anwendungssoftware und Rechenleistung mit dynamisch skalierbaren Ressourcen. Dabei besteht die Architektur aus den Bereichen Merkmale und Service- und Deployment-Modell. Das Service-Modell beschreibt verschiedene Möglichkeiten, wie die Ressourcen der Cloud dem Anwender bereitgestellt werden können. Die Art der Cloud Infrastruktur und Bereitstellung wird durch das Deployment-Modell definiert.<sup>17</sup> Essenziell für den Cloud-Ansatz sind Eigenschaften wie das on demand Dienstleistungsmodell, der Netzwerkzugriff, Ressourcenpools, Elastizität und die nach Leistung gemessenen Services. Das on demand Dienstleistungsmodell legt fest, dass Anwender benötigte Rechenressourcen und Software jederzeit dazu buchen können, ohne auf eine Bestätigung des Cloud-Providers warten zu müssen. Dadurch wird das Bedürfnis des Erreichens von Agilität und Flexibilität beim Abrufen zusätzlicher Ressourcen gegeben. Der Netzwerkzugriff beschreibt die Eigenschaft, dass die Anwendungen über das Internet erreichbar sein müssen. Mehrere Nutzer benutzen im Cloud-Computing den gleichen Ressourcenpool an Rechenleistung, dies bedeutet, dass Rechenressourcen des Providers mehreren Nutzern gleichzeitig zugeordnet sind, dabei besteht die Möglichkeit, dedizierte Rechenleistung zu buchen.<sup>18</sup> Elastizität beschreibt die Eigenschaft, dass Anwendungen Rechenleistung nach Bedarf hoch- oder herunter skaliert werden, um mögliche Leistungsschwankungen ausgleichen zu können. Gebuchte Ressourcen des Providers wie Datenspeicherkapazitäten, Rechenleistung oder weitere Ressourcen, bei

---

<sup>15</sup>[4, S.15]

<sup>16</sup>[49, S.11]

<sup>17</sup>[45, S.6]

<sup>18</sup>[1, S.4]

dem sich die Leistung messen lässt, sollten dem Anwender transparent einsichtbar sein. Dabei sollten nur Ressourcen bezahlt werden, die auch tatsächlich verwendet werden.<sup>19</sup>

---

<sup>19</sup>[25, S.52]

## 3 Analyse

### 3.1 Qualitative Merkmale

Der Begriff Softwarequalität definiert die Gesamtheit der Eigenschaften und Funktionen eines Softwareproduktes, das bestimmte festgelegte oder vorausgesetzte Anforderungen versucht zu erfüllen.<sup>20</sup> Durch den Abgleich des Ist- Sollzustandes der gewählten Kriterien kann gemessen werden, ob das Software-Endprodukt die gewünschte Qualität erreicht. Softwarequalität kann in verschiedenen Arten von Anforderungen unterteilt werden, die verbreitetste Unterteilung ist die Unterteilung in funktionale und nicht funktionale Anforderungen. Funktionale Anforderungen legen fest, was ein Softwareprodukt für Leistungen verrichten soll, sie fallen je nach Art und Aufgabe des Produktes unterschiedlich aus. Nichtfunktionale Anforderungen geben an, wie gut das Produkt die Leistungen erbringen soll, dabei können sie mit verschiedenen Qualitätsmodellen in qualitativen Merkmalen klassifiziert werden, zudem können sie in die Kategorien Ausführungs- und Evolutionsqualitäten unterteilt werden.<sup>21</sup>

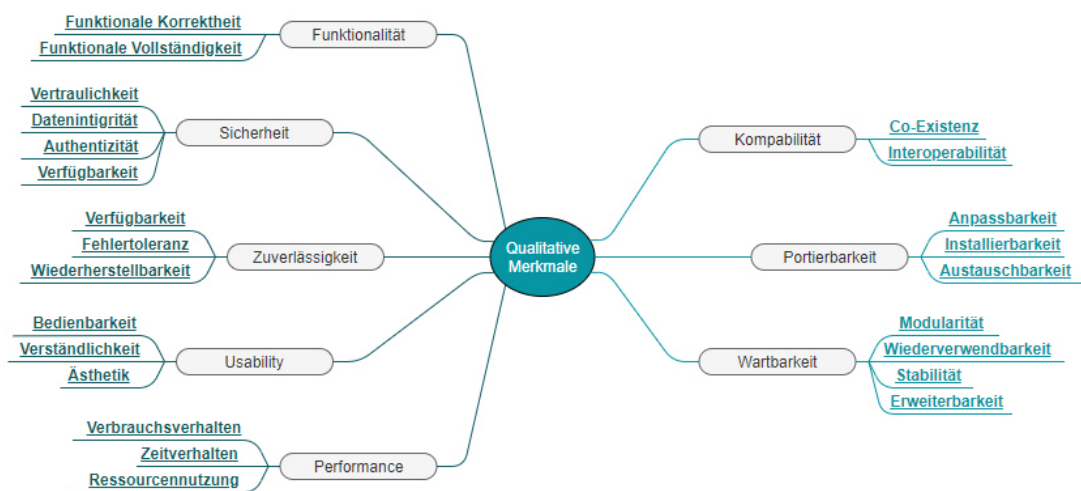


Abbildung 4: Qualitätsmodell nach ISO-25010/9162

Die Ausführungsqualität beschreibt die qualitativen Merkmale, die während des Betriebs zur Laufzeit der Software anfallen, also Zuverlässigkeit, Usability, Performance,

<sup>20</sup>[53, Softwarequalität]

<sup>21</sup>[28, Arten von Anforderungen]



Sicherheit und Funktionalität. Qualitative Merkmale, die demnach der Software ihre Struktur darstellen, werden als Evolutionsqualitäten bezeichnet, zu diesen zählen Wartbarkeit, Portierbarkeit und Kompatibilität.<sup>22</sup> Qualitative Merkmale stellen in der Softwareentwicklung sicher, dass ein gewisses Maß von Softwarequalität erreicht wird.

### 3.2 Cloud-native-Architektur

Eine Cloud-native-Architektur bezeichnet eine Architektur, dessen Ziel es ist, skalierbare, selbst verwaltende Anwendungen bereitzustellen. Cloud-native Anwendungen verwenden IT-Ressourcen, die von der Cloud-Umgebung bereitgestellt werden und verwenden sie konsequent, um eine konsistente Entwicklung und Auslieferung der Anwendung zu gewährleisten.<sup>23</sup> Dabei lassen sich verschiedene Definitionen einer Cloud-native-Architektur ableiten, darunter befindet sich eine Definition der National Institute of Standards in Technology (NIST), die den Standard der Cloud-native-Architektur gebildet haben, dadurch dass er nahezu allseits verwendet wird.<sup>24</sup>

Modularität ist eine Eigenschaft, die beschreibt, dass eine Anwendung aus mehreren separierten Modulen beziehungsweise Komponenten besteht. Cloud-native Anwendungen benötigen diese Eigenschaft, um die Möglichkeiten der Cloud-Umgebungen, die aus mehreren großen, global verteilten IT-Ressourcen bestehen, vollkommen ausnutzen zu können.<sup>25</sup> Eine Eigenschaft, bei den Rechenressourcen nach Bedarf hinzugefügt werden, um die Leistung der Anwendung zu verbessern, jedoch nicht wieder entfernt werden, nennt man Skalierbarkeit, dabei unterscheidet man zwischen dem horizontalen- und vertikalen Skalieren von Rechenressourcen. Beim horizontalen Skalieren wird die Anzahl voneinander unabhängiger Ressourcen erhöht, wenn die Anwendung mehr Speicherplatz oder Rechenleistung benötigt. Hingegen beim vertikalen Skalieren die Performance der bereitgestellten Ressourcen erhöht wird und nicht die Anzahl. Bei einem Cloud-Provider ist der Einsatz des Vertikalen-Ansatzes

---

<sup>22</sup>[43, Qualitätsmerkmale]

<sup>23</sup>[50, Der Weg zu einer Cloud-nativen Architektur]

<sup>24</sup>[45, S.5]

<sup>25</sup>[48, Was ist Cloud Native?]

begrenzt, was zu einem erhöhten Einsatz des horizontalen Skalierens führt. Cloud-native Anwendungen benötigen zusätzlich zum horizontalen Skalieren die Eigenschaft der Elastizität. Die Elastizität fokussiert sich auf das dynamische Bereitstellen und Entfernen der benötigten Ressourcen, damit die Anwendung immer genau die Ressourcen besitzt, die benötigt werden, um die geforderte Performance zu erreichen.<sup>26</sup> Der isolierte Zustand der Anwendung ist eine Eigenschaft, die einen starken Zusammenhang zu der Elastizität besitzt und versucht, den Großteil der Anwendung zustandslos zu halten. Nur durch zustandslose Anwendungen können die Vorteile der Elastizität vollständig ausgenutzt werden. Zustandslos bedeutet das Anwendungen alle Anfragen in sich vollständig und unabhängig von allen vorherigen Vorgängen behandeln. Dadurch das die Anwendung keine Information über ihren aktuellen Zustand besitzen, sind die leichter austauschbar, weil keine Zustandsinformationen mit den hinzugefügten oder entfernten Ressourcen synchronisiert werden müssen.<sup>27</sup> Das bereitstellen von Ressourcen während der Laufzeit einer Anwendung muss automatisiert stattfinden. Dazu muss ein System gegeben sein, dass die Auslastung der Anwendungen überwacht und mit dem Verwaltungsinterface des Cloud-Providers kommuniziert, um die benötigten Rechenressourcen der Anwendung automatisiert bewältigen zu können, damit das qualitative Merkmal der Fehlertoleranz gewährleistet werden kann. Aufgrund der dauerhaften Veränderung der Anzahl der Ressourcen, die eine Cloud-native Anwendung benötigt, müssen Anwendungen eine lose Kopplung besitzen. Die lose Kopplung beschreibt den geringen Grad der Abhängigkeit mehrerer Hard- oder Software-Komponenten zueinander.<sup>28</sup> Dies ermöglicht eine einfachere Bereitstellung der Ressourcen und reduziert die Anzahl des Versagens der Anwendungskomponenten.

### **3.2.1 Cloud-Ansätze und Prinzipien**

Die Cloud-Native-Architektur besteht aus mehreren Architekturprinzipien, die versuchen, die Eigenschaften von Cloud-native Anwendungen umzusetzen. Das Grundprinzip einer jeden Cloud-Native-Architektur ist das Cloud-Computing, die für die konsequente Verwendung der Services der Cloud-Umgebung steht und die eine Infrastruktur für Cloud-native Anwendung bereitstellt. Die Anwendung verwendet die Prinzipien der

---

<sup>26</sup>[1, S.23 ff.]

<sup>27</sup>[23, Was bedeutet „stateless“?]

<sup>28</sup>[47, Lose Kopplung]

Microservice-Architektur, die eine Lose-Kopplung der Komponenten voraussetzt und dadurch einen hohen Lastenausgleich durch Skalierung der einzelnen Komponenten ermöglicht. Die Containerisierung der Microservices erleichtert dabei die Installationen der Abhängigkeiten und den Betrieb der Anwendungen. Container ermöglichen den unabhängigen Betrieb mehrerer Anwendungen auf einem gemeinsamen geteilten Kernel. Die Container-Rechnerknoten können dabei über ein Container-Verwaltungs-Tool automatisiert bereitgestellt, skaliert und orchestriert werden. Ein wichtiges Prinzip, das für die Kommunikation der Anfragen zwischen den Microservices zuständig ist der Service Mesh. Ein Service Mesh ist eine dedizierte Infrastrukturschicht, die den Datenverkehr der Microservices verwaltet, dabei besteht er aus den beiden Control- und Data Plane Komponenten.<sup>29</sup> Jegliche eingehende und ausgehende Kommunikation der Microservices findet über den Service-Proxy statt, dabei ist die Data Plane die Menge aller Service-Proxy, die als eine dezentrale Ebene bereitgestellt werden. Diese flächendeckenden Daten werden dann den Control Plane zur Verfügung gestellt. Der Control Plane erfasst dabei die einkommenden Daten, die benötigt werden, um die Service-Proxy richtig zu konfigurieren.

---

<sup>29</sup>[61, Das Service Mesh]

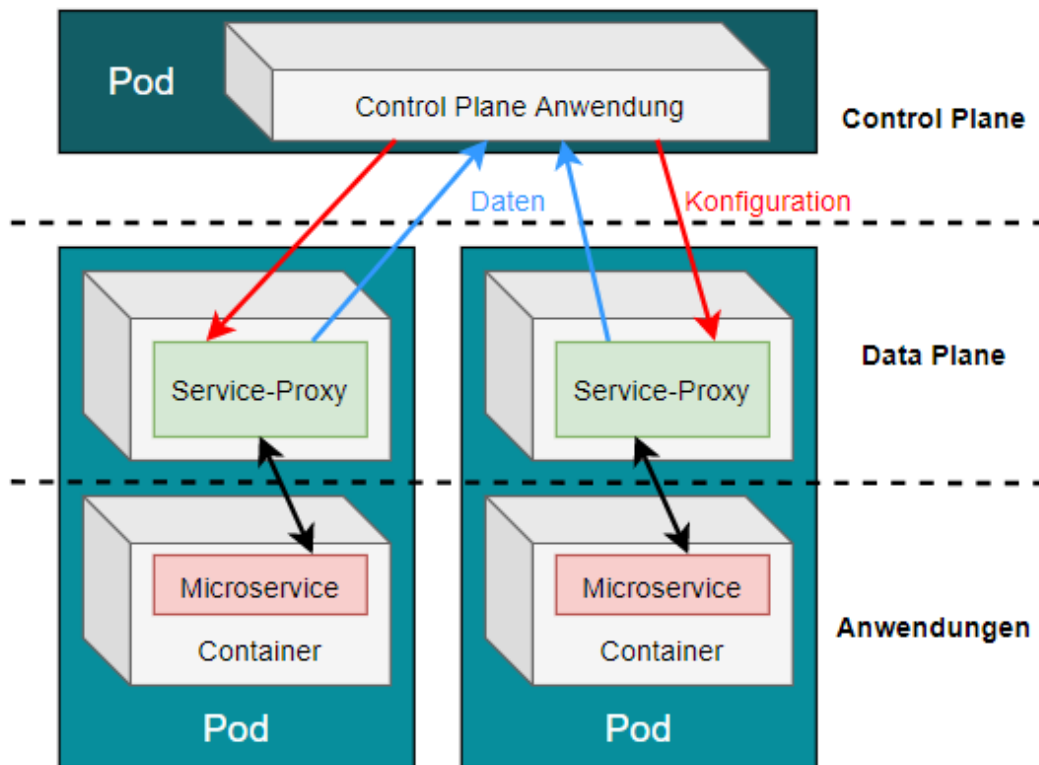


Abbildung 5: Service Mesh

Die kontinuierliche Automatisierung der Bereitstellung und Auslieferung von Updates an den Kunden wird durch die Prinzipien Continuous Delivery und Continuous Deployment, kurz CI/CD gewährleistet. CI/CD sorgt über den gesamten Anwendungs-Lifecycle hinweg für eine kontinuierliche Auslieferung und Überwachung der Integrations-, Test-, Bereitstellungs- und Implementierungsphase. Diese zusammenhängenden Phasen werden oft als CI/CD-Pipeline bezeichnet.<sup>30</sup>

Anwendungen erfahren unterschiedliche Arten von Auslastungen, die zu einer unterschiedlichen Nutzung der IT-Ressourcen führt, auf denen die Anwendungen bereitgestellt werden. Jeder Cloud-Ansatz hat dabei einen anderen Stil und erfüllt eine andere Art von Aufgabe, die durch die jeweiligen Schichten des Anwendungsstacks als IT-Ressource bereitgestellt wird.

<sup>30</sup>[59, Was versteht man unter CI/CD?]

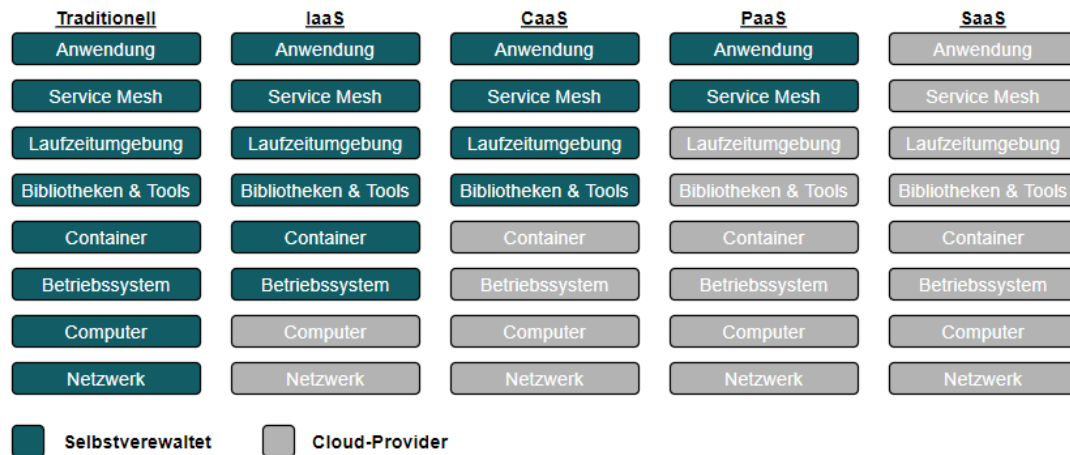


Abbildung 6: Anwendungsstack - Cloud-Ansätze

Cloud-Provider, die den Infrastruktur as a Service Ansatz (IaaS) verwenden, bieten physische und virtuelle Hardware an. Die Hardware besteht dabei aus Servern, Speicherkapazitäten und Netzwerkinfrastrukturen. Auf diesen Ressourcen können individuelle Betriebssysteme und Laufzeitumgebung, auf denen die benötigten Bibliotheken und Tools installiert sind, die für die Ausführung der Anwendungssoftware benötigt wird, bereitgestellt werden.<sup>31</sup> Der IaaS-Ansatz ist die fundamentalste Form der Cloud-Ansätze, der Anwender hat bei diesem Ansatz die meiste Kontrolle über die Art der Architektur dadurch, dass die benötigte Software selbst installiert und verwaltet wird. Der individuelle Aufbau der Architektur ist komplex, was sich sehr auf die Geschwindigkeit auswirkt, die zur Entwicklung der Architektur benötigt wird. Container as a Service (CaaS) ist ein Ansatz, bei dem der Cloud-Provider Dienstleistungen in dem Bereich der containerbasierten Virtualisierung bereitstellt. Anwender müssen keine eigene Infrastruktur für die Container-basierte-Virtualisierung selbst verwalten. Ressourcen wie Rechenleistung, Speicherplatz sowie die Container-Engine und Orchestrierungssoftware, die für die Ausführung verwendet werden, sind durch den Cloud-Provider verwaltet.<sup>32</sup> Der Anwender kann die Container und Images über eine Web-Schnittstelle, oder

<sup>31</sup>[1, S.45 ff.]

<sup>32</sup>[56, Was ist Container as a Service (CaaS)?]

API des Providers verwalten.<sup>33</sup> Beim Platform as a Service Ansatz (PaaS) stellt der Cloud-Provider flexible Serverressourcen bereit, die der Anwender für seine Anwendung nutzen kann. Dabei muss das Betriebssystem sowie die Laufzeitumgebung nicht selbst installiert und verwaltet werden.<sup>34</sup> Dies führt dazu, dass der Anwender viele ähnliche Operationen für die jeweiligen Anwendungen nicht mehr ausführen muss, was zu weniger Redundanz und Ineffizienz der Cloud-Infrastruktur führt. Die Komplexität zur Bereitstellung von Anwendungen sinkt stark bei diesem Ansatz, dadurch dass der Cloud-Provider die komplette Architektur bereitstellt, jedoch hat der Anwender kaum noch einen Einfluss auf die Konfiguration und die Art der verwendeten Software.<sup>35</sup> Der Software as a Service Ansatz (SaaS) bietet dem Anwender bereits fertige Anwendungen und Services an. Jede Schicht des Anwendungsstacks wird vom Cloud-Provider selbstverwaltet, der Anwender hat lediglich noch einen Einfluss auf die Konfiguration und Anpassung, innerhalb der definierten Grenzen der verwendeten Anwendung.<sup>36</sup> Bekannte Beispiele, die diesen Ansatz verwenden, sind die Google-Services wie Google Docs, Kalender oder die Microsoft-Office-Anwendungen.<sup>37</sup> Im Kontext der Cloud-Native-Architektur werden über den Ansatz auch APIs bereitgestellt die für die Ausführung eigener Anwendung verwendet werden können.

---

<sup>33</sup>[33, Wie funktioniert CaaS]

<sup>34</sup>[1, S.49 ff.]

<sup>35</sup>[45, S.8]

<sup>36</sup>[1, S.56]

<sup>37</sup>[45, S.8]

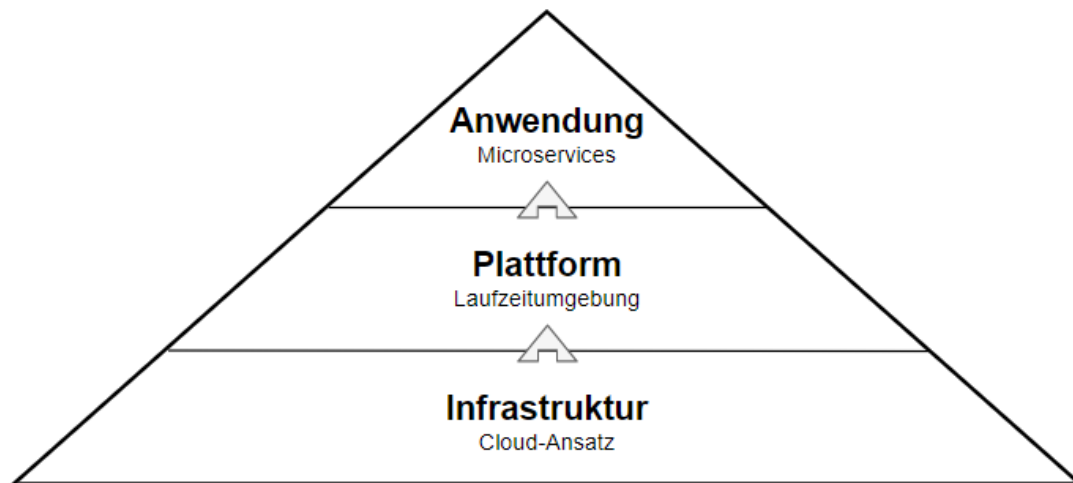


Abbildung 7: Zusammenhang: Cloud-Ansatz, Laufzeitumgebung, Microservices

Innerhalb der Cloud-Native-Architektur unterscheidet man zwischen mehreren Arten der Bereitstellung, die unterschiedliche Arten von Clouds spezifizieren. Die Public-Cloud ist das Angebot eines frei zugänglichen Cloud-Providers, der seine Cloud-Services offen über das Internet zugänglich macht. Benutzer der Public-Cloud teilen sich die Vereinigung aller Rechenressourcen des Cloud-Providers. Als Public-Cloud Provider werden üblicherweise Hyperscaler wie Amazon-Web-Services, Google Cloud Platform oder der Microsoft Azure Cloud bezeichnet die weltweite Rechenzentren beschäftigen. Das Prinzip kann jedoch auch auf jeden Provider ausgeweitet werden der Cloud-Dienste über das Internet offen bereitstellt. Dagegen steht die Private-Cloud, deren Hauptunterschied zur Public-Cloud der Datenschutz ist. Der Datenpool der Private-Cloud wird nicht mit anderen Benutzern geteilt, sie sind einem bestimmten Benutzer oder Unternehmen zugeordnet. Der Benutzer hat eine direkte Kontrolle auf die Infrastruktur der Cloud, nur autorisierte Nutzer haben Zugriff auf die Cloud. Die Art der Infrastruktur ist dabei dem Intranet eines Unternehmens sehr ähnlich. Eine Hybrid-Cloud ist eine Kombination der Public- und Private-Cloud Bereitstellung. Daten können im eigenen Intranet, oder einer Privat-Cloud gespeichert werden, zudem man über die nahezu unbegrenzten Ressourcen der Public-Cloud verfügen kann.

## 4 Konzept

Der Cloud-Native-Stack ist der Entwurf der Verbindung von Cloud-native-Technologien, die gemeinsam die Entwicklungs- und Ausführungsumgebung für Cloud-native Anwendungen darstellen, die aus den drei Ebenen Microservices Platform, Container as a Service und Infrastruktur bestehen. Der Stack enthält alle Elemente, die für die Entwicklung einer Cloud-native Anwendung benötigt werden. DevOps stellt die Schnittstelle zwischen der Softwareentwicklung und des IT-Betriebs des Cloud-native-Stack dar. Die Microservices-Plattform wird auf der CaaS Ebene ausgeführt, wobei sie für die Ausführung der containerisierten Anwendungen auf den bereitgestellten Ressourcen zuständig ist. Diese werden von der untersten Ebene der Infrastruktur virtualisiert bereitgestellt.

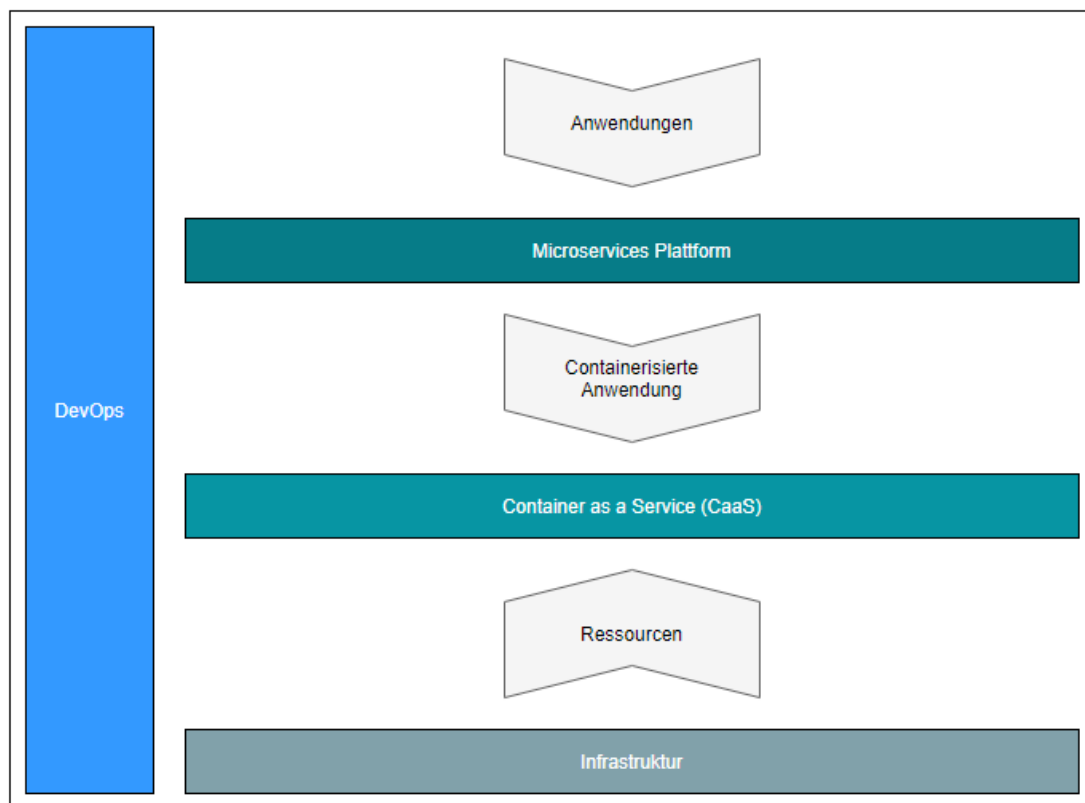


Abbildung 8: Überblick über den Cloud-native-Stack

Die Ausführungsumgebung der einzelnen Microservices ist die Laufzeitumgebung, die durch verschiedene Software und Frameworks wie Spring Boot oder WildFly Swarm.



Die Kommunikation der Microservices untereinander findet über das Service-Mesh statt, dieser verwendet dabei die Service-Discovery. Die Endpunkte werden dabei als REST-Endpunkte definiert, eine andere Schnittstellenbeschreibungssprache, die ebenso verwendet werden kann, ist Apache Thrift. Die API der Services werden über das API-Gateway nach außen hin verfügbar gemacht. Beim Aufruf eines API-Endpunktes durch einen Anwender wird vor dem empfangen, der Daten die Anfrage über die IAM-Komponente authentifiziert und autorisiert. Das Identity und Access Management sorgt für die zentrale Verwaltung von Identitäten und Zugriffsrechten des Systems. Das Identity und Access Management sorgt für die zentrale Verwaltung von Identitäten und Zugriffsrechten des Systems. Dabei ist IAM der Oberbegriff für alle Prozesse und Anwendung, die für die Administration und Verwaltung verantwortlich sind.

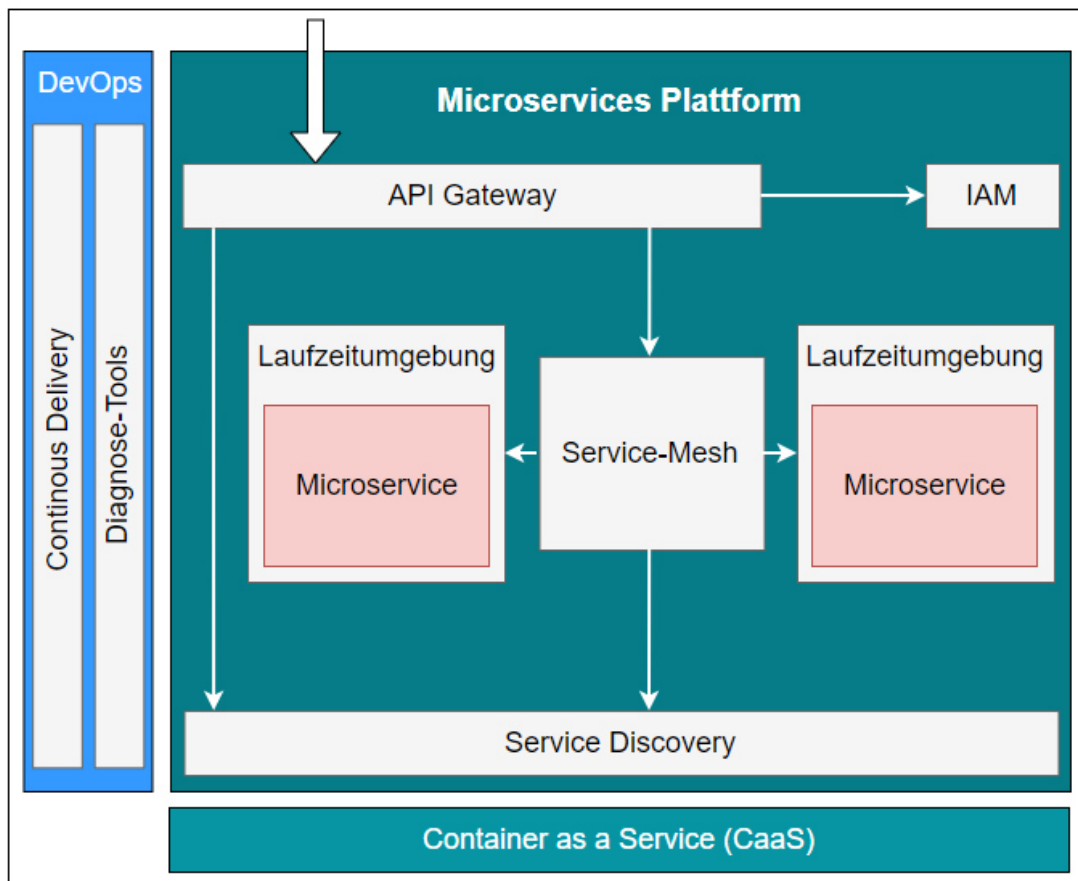


Abbildung 9: Microservices-Plattform

Die CaaS Ebene sollte weitmöglichst von der Infrastruktur Ebene getrennt werden. Damit der Anwender des Clusters als einen einzigen großen Rechner wahrnimmt, der einzelne Container zur Ausführung bereitstellt, was durch die Virtualisierung der einzelnen Cluster Komponenten gelingt. Die Betriebssystemressourcen werden über die Cloud native Laufzeit virtualisiert. Diese enthält ein minimales Betriebssystem für alle Knoten des Clusters und einer Container Laufzeit wie Docker oder rkt. Das Betriebssystem ist eine Leichtgewichtige Linux-Distributions die weniger Speicher und Anforderungen an die Prozessorgeschwindigkeit haben und dadurch optimiert darauf sind eine Container Laufzeit zu betreiben. Der übergreifende Zustand des Clusters wird mit dem Cloud Native Status verwaltet. Dieser enthält das Cloudspeichersystem, in den der Zustand des Clusters gespeichert wird und einem Konfigurations- und Koordinationsdienst wie etcd oder Consul. Das Speichersystem ist ein verteiltes Dateisystem, das je nach Auslastung elastisch skaliert wird. Die Konfiguration und Koordination Komponente verwaltet die Konfigurationswerte und den Zustand des Clusters und versucht diese Werte konsistent auf den Cluster zu verteilen. Ein oder mehrere virtuelle Netzwerke können über das physikalische Netzwerk erstellt werden, dazu muss das Netzwerk über das cloudnative Netzwerk virtualisiert werden.

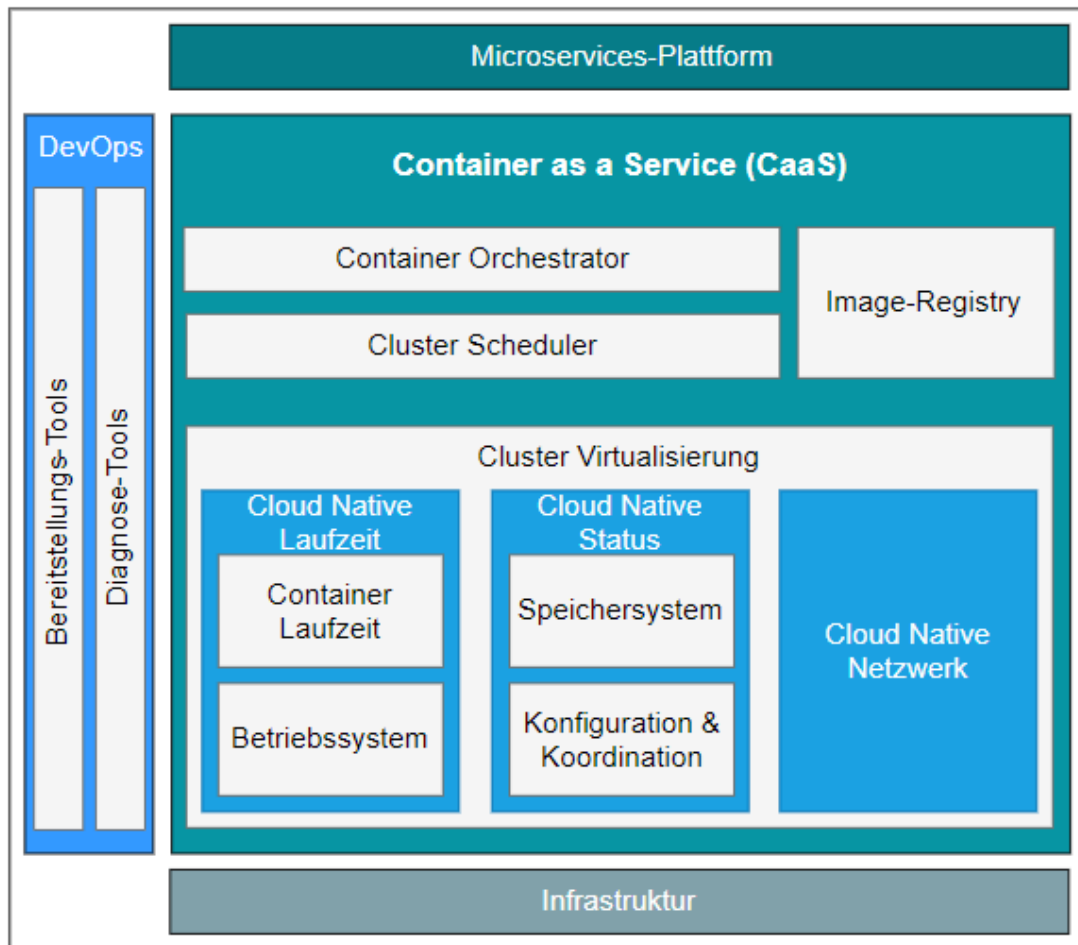


Abbildung 10: Container as a Service (CaaS)

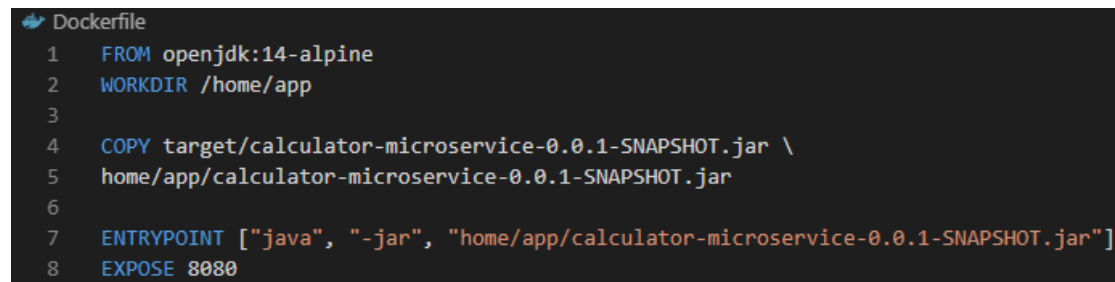
Virtualisierte Clusterressourcen werden über den Cluster Scheduler verteilt, damit die einzelnen Container zuverlässig und wirtschaftlich ausgeführt werden können. Über einen Scheduling-Algorithmus werden die Ressourcen dynamisch nach Bedarf bereitgestellt oder isoliert, dies erfolgt etwas beim Ausfall einer Ressource oder Containers. Der Cluster-Orchestrator verwaltet im Gegensatz zu dem Cluster-Scheduler, der nur einzelne Container kennt, die gesamten containerisierten Arbeitslasten. Alle betrieblichen Aufgaben wie Upgrades, Skalierung oder Konfiguration der Ressourcen und Endpunkte der Anwendungen werden dabei von dem Cluster-Orchestrator automatisiert. Die Image-Registry enthält die Container-Images, die zur Erstellung der Anwendungen von den Cluster-Orchestrator und -Scheduler genutzt werden.

## 5 Realisierung

Der Microservice ist ein Maven Projekt, das als Programmiersprache Java verwendet und zum Reduzieren der Komplexität das Open-Source-Framework Spring verwendet. Zur Ausführung des Microservices als Webanwendung führt Spring einen Apache Tomcat Server aus, der die Laufzeitumgebung, die für die Ausführung der Java-Anwendung benötigt wird, beinhaltet. Damit der Microservice als containerisierte Anwendung bereitgestellt werden kann, muss die Anwendung als JAR-Datei verpackt werden. Diese Datei enthält die Java-Klassen als ZIP-Datei sowie eine Manifest-Datei, die festlegt, wie die Java-Anwendung gestartet werden soll. Durch das Build-Management-Tool Maven ist es möglich, Projekte schnell und einfach als Archive bereitzustellen. Die Ausführung des Befehls `mvn package` verpackt die Java-Klassen der Anwendung als JAR-Datei, der Name der Datei kann dabei in den Eigenschaften des Projektes festgelegt werden, falls kein Wert festgelegt wurde, besteht der Name aus dem Projektnamen.

Zum Containerisieren der Anwendung muss vorerst ein Dockerfile erstellt werden, dass die nötigen Anweisungen enthält, die zur Erstellung des Docker-Images benötigt werden. Die Anweisung `FROM` initialisiert eine neue Build-Phase und setzt das Parent-Image des aktuellen Base-Images zu dem `openjdk:14-alpine`. Dadurch enthält das Image alle Abhängigkeiten die für die Ausführung der Java-Anwendung benötigt werden. Anschließend wird die Anweisung `WORKDIR` ausgeführt, der das aktuelle Arbeitsverzeichnis des Images zu `/home/app` setzt, dies ist für die später folgende Anweisung `ENTRYPOINT` besonders wichtig. `COPY` kopiert Dateien von einem Verzeichnis in ein anderes Verzeichnis, in dem Fall des gezeigten Dockerfiles wird die zuvor verpackte Anwendung in das Verzeichnis `home/app` des Images kopiert. Die Anweisung `ENTRYPOINT` definiert, welcher Befehl ausgeführt werden soll, wenn der Container gestartet wird. Beim Start des Containers wird das Commando `java -jar home/app/calculator-microservice-0.0.1-SNAPSHOT.jar` ausgeführt, wodurch die Anwendung gestartet wird. Die letzte Anweisung `EXPOSE` gibt an, auf welchen Netzwerkport der Container während der Laufzeit hört, dabei dient die Anweisung eher als eine Art von Dokumentation zwischen der Person, die das Docker-Image erstellt, und der Person, die für die Ausführung des Containers verantwortlich ist, damit beide Parteien wissen, welche Ports der Anwendung nach außen hin veröffentlicht werden

sollen.

A screenshot of a Dockerfile with a dark background and light-colored text. The file is titled 'Dockerfile' with a small icon. It contains eight lines of code: 1. FROM openjdk:14-alpine, 2. WORKDIR /home/app, 3. (empty line), 4. COPY target/calculator-microservice-0.0.1-SNAPSHOT.jar \, 5. home/app/calculator-microservice-0.0.1-SNAPSHOT.jar, 6. (empty line), 7. ENTRYPOINT ["java", "-jar", "home/app/calculator-microservice-0.0.1-SNAPSHOT.jar"], 8. EXPOSE 8080.

```
1 FROM openjdk:14-alpine
2 WORKDIR /home/app
3
4 COPY target/calculator-microservice-0.0.1-SNAPSHOT.jar \
5 home/app/calculator-microservice-0.0.1-SNAPSHOT.jar
6
7 ENTRYPOINT ["java", "-jar", "home/app/calculator-microservice-0.0.1-SNAPSHOT.jar"]
8 EXPOSE 8080
```

Abbildung 11: Dockerfile des Calculator-Microservices

Zum Erstellen des Images wird nun der Befehl `docker build -t eu.gcr.io/thesis-torben-schuhmacher/calculator .` ausgeführt. Die Markierung `-f` gibt den Namen des Docker-Images an, der aus einer Kombination des Registry-Namen, der Projekt-ID und dem Anwendungsnamen besteht. Der Registry-Name gibt dabei an auf welche Image-Registry das Image gepusht werden soll, `eu.gcr.io` ist der Hostname der Google Cloud Plattform Registry mit den Standort in der Europäischen Union.<sup>38</sup> Der Punkt am Ende des Befehls gibt den Pfad an, in dem sich das Dockerfile befindet. Das erstellte Image wird schließlich durch den Befehl `docker push eu.gcr.io/thesis-torben-schuhmacher/calculator` in die Container-Registry des Google Cloud Projektes gepusht.

Der Cloud-Provider, auf der die Anwendungen bereitgestellt werden, ist die Google Cloud Platform. Die bereitgestellten Services können über eine Web-Schnittstelle abgerufen und dabei beliebig hinzugefügt oder entfernt werden. Der Ansatz, der gewählt wurde, ist der CaaS-Ansatz. Durch den Google Cloud Platform Dienst Google Kubernetes Engine wird eine Infrastruktur bereitgestellt, die das bereitstellen und orchestrieren von containerisierten Anwendungen erleichtert. Bevor die containerisierten Anwendungen auf der Google Kubernetes Engine bereitgestellt werden können, muss ein Cluster erstellt werden. Dabei hat man die Auswahl zwischen einen Standard und Autopilot Cluster<sup>39</sup>, der eine vollständig bereitgestellte und verwaltete Cluster-

<sup>38</sup>[42, Lokales Image mit dem Registry-Namen taggen]

<sup>39</sup>[37, Betriebsarten]

konfiguration bereitstellt, währenddessen beim Standard Cluster die Konfiguration der Produktionsarbeitslasten selbst erfolgen müssen. Der Cluster, der gewählt wurde, ist der Standard Cluster, weil er eine bessere Konfigurationsflexibilität besitzt als der Autopilot Cluster.

Die Arbeitslasten können entweder über eine Deployment-Datei oder durch das Web-Interface der Google Kubernetes Engine bereitgestellt werden, die eine einfachere Möglichkeit Bieten Arbeitslasten bereitzustellen, dadurch aber eine geringere Flexibilität bieten als eine selbsterstellte Deployment-Datei, die mit der Auszeichnungssprache YAML serialisiert werden. Das Deployment-Objekt kann dabei durch den Befehl `kubectl create -f calculator-deployment.yml` erstellt werden. Der Selektor `kind` gibt den zu erstellenden Kubernetes Objekttypen an. Metadata enthält Informationen die zum einzigartigen identifizieren des Objektes benötigt werden, wie der Name des Objektes. Die präzisen Details des Objektes werden im `spec` Selektor beschrieben. Deployment Objekte enthalten die Felder `replicas`, das angibt wie viele Kopien es geben soll, folglich die Anzahl der Pods und `selector.matchLabels.app`, das definiert welche Pods dem Deployment-Objekt zugewiesen sind, als Schlüssel-/Wert-Paar. Das Feld `template.metadata.labels.app` kennzeichnet die Pods mit der Kennzeichnung `calculator` und `spec` gibt an das die Pods jeweils einen Container mit dem Namen `calculator` ausführen die auf den Image `eu.gcr.io/thesis-torben-schuhmacher/calculator:latest` basieren.

```

! calculator-deployment.yml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: calculator
5  spec:
6    replicas: 3
7    selector:
8      matchLabels:
9        app: calculator
10   template:
11     metadata:
12       labels:
13         app: calculator
14     spec:
15       containers:
16       - name: calculator
17         image: eu.gcr.io/thesis-torben-schuhmacher/calculator:latest

```

Abbildung 12: Deployment der Anwendung

Die Endpunkte der Arbeitslasten, die durch ein Deployment bereitgestellt wurden, sind nur innerhalb des Clusters erreichbar, wenn man die Endpunkte auch von außerhalb erreichen möchte, muss man ein Service erstellen, der die Endpunkte außerhalb des Clusters abbilden kann. Ein Service kann dabei entweder über das Web-Interface der Google Kubernetes Engine oder durch eine selbsterstellte Service-Datei zur Verfügung gestellt werden<sup>40</sup>, dabei unterscheiden sich die Service- und Deployment-Datei hauptsächlich nur durch den Selektor `spec`. Der Service, der durch die Service-Datei erstellt wird, besitzt den Namen `calculator-service`, der eine Verbindung zu den bereitgestellten Pods besitzt, die eine Übereinstimmung mit dem Label `spec.selector.app: calculator` besitzen.

---

<sup>40</sup>[51, Defining a Service]

```

! calculator-service.yml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: calculator-service
5  spec:
6    selector:
7      app: calculator
8    ports:
9      - protocol: TCP
10       port: 8080
11       targetPort: 8080
12    type: LoadBalancer

```

Abbildung 13: Bereitstellung des Services

Wenn keine IP-Adresse des `type: LoadBalancer` festgelegt ist<sup>41</sup>, wird automatisch eine kurzlebige Adresse des Clusters bereitgestellt, der Port über die Pods erreicht werden können, ist der `ports.targetPort: 8080`. Der interne `ports.port: 8080` der Anwendung wird dafür auf den Externen `ports.targetPort: 8080` gemapped. Auf welche Art und Weise die Daten zwischen den Netzwerkkomponenten ausgetauscht werden, wird durch den Selektor `protocol: TCP` bestimmt. Durch das Ausführen des Befehls `kubectl create -f calculator-service.yml` können die Endpunkte mit der gewählten Spezifikation außerhalb des Clusters verfügbar gemacht werden.

---

<sup>41</sup> [52, Type LoadBalancer]



## 6 Fazit

Das Konzipieren und realisieren einer Cloud-nativen Architektur ist nicht leicht verständlich und sehr komplex, auch wenn man sich ausgiebig mit den Technologien und Eigenschaften der Cloud beschäftigt hat. Dabei ist es schwer, einen Überblick über gängige Praktiken und Umsetzung zu behalten. Die Auswahl des richtigen Cloud-Ansatzes und -Services ist maßgebend. Die richtige Auswahl stellt dabei eine individuelle Entscheidung dar. Keine zwei gleichen Clouds oder Services werden verwendet, um dasselbe Problem zu lösen. Durch das Erkennen der Ähnlichkeiten ist es aber möglich, eine leichtere Entscheidung zu treffen. Der Fokus der Arbeit lag auf der Konzeption und nicht auf der Realisierung der Architektur, dabei wurde auf die Automatisierung der Bereitstellung der Anwendungen und die Absicherung der Endpunkte nicht genau eingegangen. Der Fokus der Arbeit lag auf der Containerisierung der Anwendungen und das Bereitstellen der Endpunkte. Auf Basis der Arbeit wäre das Konzipieren und realisieren des Sicherns der Anwendungen, Daten und Diensten der verbundenen Infrastruktur des Cloud-Computing einen interessanten Ausblick. Dadurch, dass die Migration in die Cloud eine so hohe Komplexität besitzt, sollte man abwägen, ob es sinnvoll ist, einen Teil seiner Architektur von einer on premises Lösung in die Cloud zu migrieren.

## Glossar

API	API steht für Application Programming Interface und bezeichnet eine Programmierschnittstelle, die dazu dient, Informationen zwischen einer Anwendung und Programmteilen standardisiert auszutauschen. <sup>42</sup>
Apache Thrift	Ein Framework für skalierbare sprachübergreifende Softwareentwicklung. <sup>43</sup>
Apache Tomcat	Open-Source-Webserver und Webcontainer der in Java geschriebene Web-Anwendungen ausführen kann. <sup>44</sup>
Cloud	Die internetbasierte Bereitstellung von Speicherplatz, Rechenleistung oder Anwendungssoftware als Dienstleistung. <sup>45</sup>
Cloud Provider	Ein Unternehmen, das einen oder mehrere Cloud-Services bereitstellt. <sup>46</sup>
Cluster	Ein Verbund aus zusammengeschlossenen Computern. <sup>47</sup>
Consul	Software, die das Monitorings, Konfigurationsmanagement und Service Discovery von Diensten und Knoten erleichtert. <sup>48</sup>
Container	Die Virtualisierung einer Anwendung, die alle erforderlichen Konfiguration und Abhängigkeiten enthalten. <sup>49</sup>
Container-Registry	Eine Sammlung von Container-Images für die containerbasierte Anwendungsentwicklung. <sup>50</sup>
Containerd	Ein Containervirtualisierungs-Tool das von der Cloud native Computing Foundation entwickelt wurde. <sup>51</sup>

---

<sup>42</sup>[10, Was ist eine API?]

<sup>43</sup>[30, Apache Thrift]

<sup>44</sup>[31, Apache Tomcat]

<sup>45</sup>[8, Was ist Cloud Computing?]

<sup>46</sup>[3, Was ist ein Cloud Provider?]

<sup>47</sup>[7, Was ist ein Cluster?]

<sup>48</sup>[2, HashiCorps Consul für Service Discovery, Monitoring und Konfigurationsmanagement]

<sup>49</sup>[6, Was sind Container?]

<sup>50</sup>[57, Was ist eine Container-Registry?]

<sup>51</sup>[17, Containerd (CNCF)]

Cross-Cutting Concern	Bestandteil eines Programms, das viele Abhängigkeit zu anderen Teilen des Systems besitzt. <sup>52</sup>
DevOps	Beschreibt einen Entwicklungsansatz, in dem die Zusammenarbeit zwischen Softwareentwicklung und Betrieb verbessert wird. <sup>53</sup>
Gateway	Bindeglied, dass die Kommunikation zwischen unterschiedlichen Systemen ermöglicht. <sup>54</sup>
Geschäftslogik	Abgrenzung der Logik eines Softwaresystems von der technischen Implementierung. <sup>55</sup>
Google Cloud Platform	Cloud-Infrastruktur-Plattform, von der Firma Google, auf der Anwendungen und Daten skaliert und bereitgestellt werden können. <sup>56</sup>
Host-Rechner	Ein Computer mit zugehörigem Betriebssystem, der Client bedient oder Dienste bereitstellt. <sup>57</sup>
Hyperscaler	Bezeichnet die großen Cloud-Provider Amazon, Microsoft und Google, die den Großteil des Public-Cloud-Marktes abdecken. <sup>58</sup>
Hypervisor	Ein Virtual-Machine-Monitor der verschiedenen virtuellen Instanzen Ressourcen wie CPU, RAM oder Festplattenspeicher zuweist. <sup>59</sup>
IAM	Identity- und Access Management ist die zentrale Verwaltung von Identitäten und Zugriffsrechten unterschiedlicher Systeme und Anwendungen. <sup>60</sup>
Kernel	Der zentrale Bestandteil eines Betriebssystems, auf dem alle weiteren Softwarebestandteile aufbauen. <sup>61</sup>

---

<sup>52</sup>[34, Cross-cutting concern]

<sup>53</sup>[41, S.4]

<sup>54</sup>[15, Was ist ein Gateway?]

<sup>55</sup>[39, Geschäftslogik]

<sup>56</sup>[32, Google Cloud Platform]

<sup>57</sup>[40, Hostrechner]

<sup>58</sup>[9, Was sind Hyperscaler?]

<sup>59</sup>[18, Was ist ein Hypervisor?]

<sup>60</sup>[5, Was ist IAM?]

<sup>61</sup>[44, Kernel (Betriebssystem)]

Lastverteilung (Load-Balancing)	Verteilung einer großen Menge von Anfragen auf mehrere parallel arbeitende Server. <sup>62</sup>
Laufzeitumgebung	Stellt Funktionalitäten zur Verfügung, die zur Ausführung eines Programms benötigt werden. <sup>63</sup>
Lose-Kopplung	Bezeichnet den geringen Grad der Abhängigkeiten zwischen mehrerer Hard- oder Software-Komponenten. <sup>64</sup>
Maven	Ein auf Java basierendes Build-Management-Tool, zur standardisierten Erstellung von Java-Programmen. <sup>65</sup>
Modularität	Logische Aufteilung eines Systems in mehrere Module. <sup>66</sup>
Namespace	Gruppierung ausgewählter Elemente zur einfacheren Modularisierung eines Systems. <sup>67</sup>
Proxy	Server, der als Vermittler von Anfragen dient und diese stellvertretend weiterleitet. <sup>68</sup>
REST	Programmierschnittstelle, die den Austausch von Daten auf verteilten Systemen ermöglicht. <sup>69</sup>
Rkt	Software zur Virtualisierung von Container, die für moderne Cloud-native-Umgebungen. <sup>70</sup>

---

<sup>62</sup>[46, Load balancing (computing)]

<sup>63</sup>[20, Was ist eine Laufzeitumgebung?]

<sup>64</sup>[47, Lose Kopplung]

<sup>65</sup>[29, Apache Maven]

<sup>66</sup>[24, Modularität]

<sup>67</sup>[19, Modularisierung von Code]

<sup>68</sup>[21, Was ist ein Proxy?]

<sup>69</sup>[13, Was ist REST-API?]

<sup>70</sup>[60, What is rkt?]

Service-Mesh	Dedizierte Infrastrukturschicht, die den Austausch von Daten zwischen Teilen einer Anwendung reguliert. <sup>71</sup>
Spring	Framework für die Java-Plattform zur vereinfachten Entwicklung von Java/Java EE. <sup>72</sup>
TCP	Netzwerkprotokoll, das definiert, wie Netzwerkkomponenten Daten miteinander austauschen sollen. <sup>73</sup>
Virtuelle-Maschine	Software-technische-Kapselung Rechnersystems innerhalb eines physischen lauffähigen Systems. <sup>74</sup>
ZIP	Dateiformat zur verlustfreien Komprimierung von Dateien. <sup>75</sup>

---

<sup>71</sup>[35, Das Service Mesh - Funktionsweise und Vorteile]

<sup>72</sup>[54, Das Spring Framework]

<sup>73</sup>[55, Transmission Control Protocol]

<sup>74</sup>[26, Virtuelle Maschine]

<sup>75</sup>[27, ZIP-Dateiformat]

## Literaturverzeichnis

- [1] Christoph Fehling u. a. *Cloud Computing Patterns*. Springer-Verlag Wien, 2014.
- [2] Alexander Schwartz. *HashiCorps Consul für Service Discovery, Monitoring und Konfigurationsmanagement*. <https://m.heise.de/developer/artikel/HashiCorps-Consul-fuer-Service-Discovery-Monitoring-und-Konfigurationsmanagement-3040847.html?seite=all>. Abgerufen am 29.06.2021. Dez. 2015.
- [3] Michael Hase. *Was ist ein Cloud Provider?* <https://www.it-business.de/was-ist-ein-cloud-provider-a-669650/>. Abgerufen am 28.06.2021. Dez. 2016.
- [4] Deepak Vohra. *Kubernetes Microservices with Docker*. Hrsg. von Michelle Lowman. Springer Science+Business Media New York, 2016.
- [5] *Was ist Identity- and Access Management (IAM)?* <https://www.security-insider.de/was-ist-identity-and-access-management-iam-a-612910/>. Abgerufen am 03.07.2021. Juni 2016.
- [6] Stephan Augsten. *Was sind Container?* <https://www.dev-insider.de/was-sind-container-a-573872/>. Abgerufen am 28.06.2021. Jan. 2017.
- [7] Dipl.-Ing. Thomas Drilling und Ulrike Ostler. *Was ist ein Cluster?* <https://www.datacenter-insider.de/was-ist-ein-cluster-a-588715/>. Abgerufen am 28.06.2021. März 2017.
- [8] Marcus Ladwig. *Cloud Computing Definition: Was ist eine Cloud und wie funktioniert sie?* <https://www.cloudplan.net/blogdetail/Was-ist-eine-Cloud-und-wie-funktioniert-sie>. Abgerufen am 28.06.2021. Juni 2017.
- [9] Klaus Länger. *Was sind Hyperscaler?* <https://www.it-business.de/was-sind-hyperscaler-a-664638/>. Abgerufen am 28.06.2021. Nov. 2017.
- [10] Dipl.-Ing. Stefan Luber und Stephan Augsten. *Was ist eine API?* <https://www.dev-insider.de/was-ist-eine-api-a-583923/>. Abgerufen am 28.06.2021. März 2017.
- [11] Deepak Vohra. *Docker Management Design Patterns*. Hrsg. von Todd Green. Springer Science+Business Media New York, 2017.
- [12] Deepak Vohra. *Kubernetes Management Design Patterns*. Hrsg. von Steve Anglin. Springer Science+Business Media New York, 2017.

- [13] Otto Geißler. *Was ist REST-API?* <https://www.datacenter-insider.de/was-ist-rest-api-a-714434/>. Abgerufen am 29.06.2021. Mai 2018.
- [14] Chrissi Kraus und Stephan Augsten. *Was ist ein User Interface?* <https://www.dev-insider.de/was-ist-ein-user-interface-a-735069/>. Aug. 2018.
- [15] Dirk Srocke und Andreas Donner. *Was ist ein Gateway?* <https://www.ip-insider.de/was-ist-ein-gateway-a-581268/>. Abgerufen am 28.06.2021. Aug. 2018.
- [16] Harsh Chawla und Hemant Kathuria. *Building Microservices Applikations on Microsoft Azure*. Hrsg. von Smriti Srivastava. by Springer Science+Business Media New York, 2019.
- [17] *containerd ist das fünfte Projekt mit Graduiertenstatus in der CNCF*. <https://www.heise.de/developer/meldung/containerd-ist-das-fuenfte-Projekt-mit-Graduiertenstatus-in-der-CNCF-4323678.html>. Abgerufen am 02.07.2021. März 2019.
- [18] Stefan Lubert und Jürgen Ehneß. *Was ist ein Hypervisor?* <https://www.storage-insider.de/was-ist-ein-hypervisor-a-842084/>. Abgerufen am 28.06.2021. Aug. 2019.
- [19] Chrissi Kraus und Stephan Augsten. *Was ist ein Namespace?* <https://www.dev-insider.de/was-ist-ein-namespace-a-904135/>. Abgerufen am 26.06.2021. Feb. 2020.
- [20] *Laufzeitumgebung: Was ist eine Runtime Environment?* <https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-eine-laufzeitumgebung/>. Abgerufen am 28.06.2021. Juli 2020.
- [21] Stefan Lubert und Peter Schmitz. *Was ist ein Proxy?* <https://www.security-insider.de/was-ist-ein-proxy-a-985538/>. Abgerufen am 28.06.2021. Dez. 2020.
- [22] Jan Mahn, Merlin Schumacher und Peter Siering. *Docker und Co*. Heise Medien, 2020.
- [23] MirkoK und Stephan Augsten. *Was bedeutet "stateless"?* <https://www.dev-insider.de/was-bedeutet-stateless-a-974510/>. Dez. 2020.
- [24] *Modularität*. <https://de.wikipedia.org/wiki/Modularität>. Abgerufen am 28.06.2021. Nov. 2020.

- [25] Naresh Kumar Sehgal, Pramod Chandra P. Bhatt und John M. Acken. *Cloud Computing with Security*. Springer Nature Switzerland AG, 2020.
- [26] *Virtuelle Maschine*. <https://de.wikipedia.org/wiki/Virtuelle-Maschine>. Abgerufen am 28.06.2021. Dez. 2020.
- [27] *ZIP-Dateiformat*. <https://de.wikipedia.org/wiki/ZIP-Dateiformat>. Abgerufen am 03.07.2021. März 2020.
- [28] *Anforderung (Informatik)*. [https://de.wikipedia.org/wiki/Anforderung-\(Informatik\)](https://de.wikipedia.org/wiki/Anforderung-(Informatik))cite-note-robertson-2. Abgerufen am 27.06.2021. Mai 2021.
- [29] *Apache Maven*. <https://de.wikipedia.org/wiki/Apache-Maven>. Abgerufen am 03.07.2021. Apr. 2021.
- [30] *Apache Thrift*. <https://thrift.apache.org/>. Abgerufen am 29.06.2021. Juni 2021.
- [31] *Apache Tomcat*. <https://de.wikipedia.org/wiki/Apache-Tomcat>. Abgerufen am 02.07.2021. Juni 2021.
- [32] *Cloud Infrastruktur - Google Cloud Platform*. <https://cloudwuerdig.com/cloud-infrastruktur/google-cloud-platform/>. Abgerufen am 03.07.2021. Juli 2021.
- [33] *Container-as-a-Service*. <https://www.ionos.de/digitalguide/server/knowhow/caas-container-as-a-service-anbieter-im-vergleich/>. Abgerufen am 30.06.2021. Juni 2021.
- [34] *Cross-cutting concern*. <https://en.wikipedia.org/wiki/Cross-cutting-concern>. Abgerufen am 28.06.2021. Apr. 2021.
- [35] *Das Service Mesh – Funktionsweise und Vorteile*. <https://www.redhat.com/de/topics/microservices/what-is-a-service-mesh>. Abgerufen am 28.06.2021. Juni 2021.
- [36] *Docker Docs*. <https://docs.docker.com/storage/volumes/>. Abgerufen am 29.06.2021. Mai 2021.
- [37] *Ebene der Clusterverwaltung*. <https://cloud.google.com/kubernetes-engine/docs/concepts/types-of-clusters>. Abgerufen am 01.07.2021. Juli 2021.



- [38] Dr. Thomas Fricke. *Kubernetes: Was kann das Tool?* <https://www.informatik-aktuell.de/entwicklung/methoden/kubernetes-architektur-und-einsatz-einfuehrung-mit-beispielen.html>. März 2021.
- [39] *Geschäftslogik*. <https://de.wikipedia.org/wiki/Geschäftslogik>. Abgerufen am 28.06.2021. Jan. 2021.
- [40] *Hostrechner*. <https://de.wikipedia.org/wiki/Hostrechner>. Abgerufen am 28.06.2021. März 2021.
- [41] Michael Hüttermann. *DevOps for Developers*. Hrsg. von Ben Renow-Clarke. Paul Manning, 2021.
- [42] *Images hoch- und herunterladen*. <https://cloud.google.com/container-registry/docs/pushing-and-pulling>. Abgerufen am 01.07.2021. Juli 2021.
- [43] *ISO/IEC 9126*. <https://de.wikipedia.org/wiki/ISO/IEC-9126>. Abgerufen am 27.06.2021. Juni 2021.
- [44] *Kernel (Betriebssystem)*. [https://de.wikipedia.org/wiki/Kernel-\(Betriebssystem\)](https://de.wikipedia.org/wiki/Kernel-(Betriebssystem)). Abgerufen am 28.06.2021. Jan. 2021.
- [45] Anders Lisdorf. *Cloud Computing Basics*. Hrsg. von Susan McDermott. Springer Science+Business Media New York, 2021.
- [46] *Load balancing (computing)*. <https://en.wikipedia.org/wiki/Load-balancing-computing>. Abgerufen am 28.06.2021. Juni 2021.
- [47] *Lose Kopplung*. <https://de.wikipedia.org/wiki/Lose-Kopplung>. Abgerufen am 24.07.2021. Juni 2021.
- [48] Dipl.-Ing. Stefan Lubert und Florian Karlstetter. *Was ist Cloud Native?* <https://www.cloudcomputing-insider.de/was-ist-cloud-native-a-669681/>. Abgerufen am 20.07.2021. Juni 2021.
- [49] Philippe Martin. *Kubernetes*. Hrsg. von Spandana Chatterjee. Springer Science+Business Media New York, 2021.
- [50] Michale Schäfer und Carol Gutzeit. *Der Weg zu einer Cloud-nativen Architektur*. <https://m.heise.de/developer/artikel/Der-Weg-zu-einer-Cloud-nativen-Architektur-4588019.html?seite=all>. Abgerufen am 30.07.2021. Juni 2021.

- [51] *Service*. <https://kubernetes.io/docs/concepts/services-networking/service/>. Abgerufen am 02.07.2021. Juli 2021.
- [52] *Service*. <https://kubernetes.io/docs/concepts/services-networking/service/>. Abgerufen am 02.07.2021. Juli 2021.
- [53] *Softwarequalität*. <https://www.quality.de/lexikon/softwarequalitaet/>. Abgerufen am 27.06.2021. Juni 2021.
- [54] *Spring (Framework)*. [https://de.wikipedia.org/wiki/Spring-\(Framework\)](https://de.wikipedia.org/wiki/Spring-(Framework)). Abgerufen am 03.07.2021. März 2021.
- [55] *Transmission Control Protocol/Internet Protocol*. <https://de.wikipedia.org/wiki/Transmission-Control-Protocol/Internet-Protocol>. Abgerufen am 03.07.2021. Juni 2021.
- [56] *Was ist Container as a Service (CaaS)?* <https://www.cloudcomputing-insider.de/was-ist-container-as-a-service-caas-a-852566/>. Abgerufen am 30.06.2021. Juni 2021.
- [57] *Was ist eine Container-Registry?* <https://www.redhat.com/de/topics/cloud-native-apps/what-is-a-container-registry>. Abgerufen am 28.06.2021. Juni 2021.
- [58] *Was ist Kubernetes?* <https://kubernetes.io/de/docs/concepts/overview/what-is-kubernetes/>. Abgerufen am 29.06.2021. Mai 2021.
- [59] *Was versteht man unter CI/CD?* <https://www.redhat.com/de/topics/devops/what-is-ci-cd>. Abgerufen am 01.07.2021. Juni 2021.
- [60] *What is rkt?* <https://www.openshift.com/learn/topics/rkt>. Abgerufen am 29.06.2021. Juni 2021.
- [61] Eberhard Wolff und Hanna Prinz. *Das Service Mesh*. <https://www.innoq.com/de/articles/2019/10/service-mesh/>. Abgerufen am 01.07.2021. Juni 2021.

## Abbildungsverzeichnis

1	Architekturkonzepte von Monolith und Microservices . . . . .	4
2	Container-Infrastruktur . . . . .	5
3	Kubernetes-Infrastruktur . . . . .	8
4	Qualitätsmodell nach ISO-25010/9162 . . . . .	11
5	Service Mesh . . . . .	15
6	Anwendungsstack - Cloud-Ansätze . . . . .	16
7	Zusammenhang: Cloud-Ansatz, Laufzeitumgebung, Microservices . . .	18
8	Überblick über den Cloud-native-Stack . . . . .	19
9	Microservices-Plattform . . . . .	20
10	Container as a Service (CaaS) . . . . .	22
11	Dockerfile des Calculator-Microservices . . . . .	24
12	Deployment der Anwendung . . . . .	26
13	Bereitstellung des Services . . . . .	27