Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

**Björn Freund**

## Konzeption einer Big Data-Architektur unter Einhaltung von DevOps-Vorgaben

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Björn Freund

# Konzeption einer Big Data-Architektur unter Einhaltung von DevOps-Vorgaben

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Zukunft
Zweitgutachter: Prof. Dr. Steffens

Eingereicht am: 25.01.2019

**Björn Freund**

**Thema der Arbeit**
Konzeption einer Big Data-Architektur unter Einhaltung von DevOps-Vorgaben

**Stichworte**
DevOps, Big Data, Cluster, Cloud-Computing

**Kurzzusammenfassung**
Seit einigen Jahren hat das Wachstum an Daten inzwischen Dimensionen angenommen, wodurch die Handhabung dieser Daten nicht mehr durch klassische Datenverarbeitungssysteme möglich ist. Diese Problematik und deren Lösungsansätze werden unter dem Begriff Big Data zusammengefasst.
Parallel dazu hat sich der Begriff DevOps in der Software-Entwicklung fest etabliert. Dabei handelt es sich um einen Ansatz, um den gesamtem Auslieferungsprozess eines Software-Produktes kontinuierlich zu verbessern und dadurch dessen Qualität und Effizienz zu steigern.
Diese Bachelorarbeit beschäftigt sich mit der Frage, ob es sinnvoll ist, diese beiden Konzepte zu kombinieren. Dabei werden mögliche Schnittstellen und Gemeinsamkeiten erörtert und eine Cloud-basierte Lösung prototypisch ausgearbeitet.

**Björn Freund**

**Title of the paper**
Conception of a Big Data architecture in compliance with DevOps standards

**Keywords**
DevOps, Big Data, Cluster, Cloud-Computing

**Abstract**
For some years now the growth of data has taken on dimensions, whereby classical data processing systems aren't able to handle this data anymore. This difficulty and it's solutions are summarized under the term Big Data.
In parallel the term DevOps has well-established in software development. It is an approach to continuously improve the overall delivery process of a software product in order to increase quality and efficiency.
This bachelor thesis deals with the question of whether it makes sense to combine these two

concepts. Possible interfaces and similarities are discussed and a cloud-based prototype is worked out.

# Inhaltsverzeichnis

# Tabellenverzeichnis

# Abbildungsverzeichnis

# 1. Einleitung

In der heutigen Zeit gehören Big Data und DevOps zu den aktuellen IT-Trends. Big Data wird dabei als Schlüsseltechnologie der digitalen Transformation bezeichnet (vgl. Redaktion Digitales Wirtschaftswunder, 2017), weil durch den verbreiteten Einsatz von verschiedensten IT-Systemen kontinuierlich große und unstrukturierte Datenmengen erzeugt werden und erst die Technologien aus dem Big Data-Kontext die Speicherung und Verarbeitung dieser Datenmengen in adäquater Zeit ermöglichen. Zu diesen Datenmengen gehören z.B. Ereignisprotokolle, die durch Sensoren oder die Überwachung von Online-Diensten erzeugt werden oder Inhalte, die durch Personen auf Social-Media-Plattformen generiert werden.

Neben diesen internen Daten, die in den meisten Fällen von Unternehmen selber erzeugt und meist unter Verschluss gehalten werden, gibt es auch externe Daten, die z.B. durch Open-Data-Portale bereitgestellt werden. Dabei kann es sich um Daten aus der Forschung, Medizin, Verkehr, Wetter und anderen Bereichen handeln. Durch die Verarbeitung bzw. Analyse der internen und externen Daten, können Informationen extrahiert werden, die z.B. Unternehmen dabei helfen, Maßnahmen zur Steigerung der Kundenzufriedenheit bzw. zur Steigerung des Profits zu ergreifen. Dabei gilt: Je schneller ein Unternehmen die jeweiligen Daten erheben, analysieren und interpretieren kann, desto schneller kann es auf Veränderungen reagieren.

Um in der heutigen Zeit konkurrenzfähig zu sein, reicht nicht nur eine hohe Geschwindigkeit bei der Verarbeitung von Daten, sondern auch bei der Bereitstellung von Diensten. Neben der Kundenanforderung, dass ein Online-Dienst, z.B. eine Online-Shopping-Plattform, zu jeder Zeit erreichbar sein muss, müssen auch konkrete Änderungen und weitere Kundenwünsche schnell und zuverlässig in einer Anwendung abgebildet und implementiert werden können, damit die Kunden nicht das Interesse an dem Dienst verlieren und unter Umständen zu einem Konkurrenten gehen. Die kontinuierliche Bereitstellung, bei der gleichzeitigen Möglichkeit zur kontinuierlichen Weiterentwicklung von Anwendungen, stellt ein Unternehmen vor neue, große, technische, aber auch organisatorische Aufgaben. Viele Konzepte und Werkzeuge zur Bewältigung dieser Aufgaben werden unter dem Begriff *DevOps* zusammengefasst. Die

Anwendung dieser Konzepte und Werkzeuge steigert die Agilität eines Unternehmens bezogen auf Kundenwünsche und die Qualität und Robustheit der Anwendungen.

## 1.1. Ziel der Arbeit

Ziel dieser Arbeit ist herauszufinden, inwieweit sich die beiden Kontexte Big Data und DevOps begünstigen und und daraus ein Konzept für eine Architektur zu entwickeln. Big Data und DevOps sind zwei Begriffe, die in der IT-Welt fest etabliert sind. Big Data in Bezug auf die Verarbeitung großer Datenmengen und DevOps in Bezug auf das Software Engineering. Neue Technologien, die vor allem auf Cloud-Systemen basieren, helfen bei der Umsetzung von Lösungen aus beiden Bereichen und werden mit in diese Arbeit einfließen.

## 1.2. Gliederung

Die Arbeit beginnt mit Kapitel 1. In diesem Kapitel werden das Thema der Arbeit, die Zielsetzung und die Gliederung vorgestellt. In Kapitel 2 werden die Grundlagen hinter dem Begriff Big Data erläutert. Dabei werden mehrere Definitionsansätze, der Big Data-Lebenszyklus und Werkzeuge und Technologien, z.B. NoSQL-Datenbanken, für die Verarbeitung großer Datenmengen vorgestellt.
In Kapitel 3 wird dargelegt, worum es sich bei dem Begriff DevOps handelt. Nach einer Definition und einer Übersicht der Herausforderungen, die mit DevOps-Ansätzen bewältigt werden sollen, folgt die Darstellung eines Lebenszyklus einer Anwendung, die nach DevOps-Prinzipien betrieben wird. Danach werden mögliche Technologien, die bei der Umsetzung von DevOps helfen, vorgestellt. Am Ende des dritten Kapitels folgt eine Gegenüberstellung von Big Data und DevOps, um mögliche Gemeinsamkeiten bzw. Synergien darzustellen.

Kapitel 4 stellt die Grundlagen des Cloud Computings vor, weil Cloud-Systeme aktuell im Trend liegen und sie eine Möglichkeit für die Realisierung von Big Data- und DevOps-Architekturen darstellen. Am Ende des Kapitels folgt eine Übersicht darüber, wie und welche Eigenschaften des Cloud Computings sich mit denen von Big Data und DevOps verknüpfen lassen.

In Kapitel 5 wird auf Grundlage der vorherigen Kapitel ein Konzept für eine Architektur ausgearbeitet, auf der Big Data-Anwendungen mit Hilfe von DevOps-Vorgaben entwickelt und automatisiert bereitgestellt werden sollen. Dafür wird zunächst ein Beispielszenario mit einer Beispielanwendung vorgestellt, um einen praktischen Bezug herzustellen. Weiterhin werden

Anforderungen spezifiziert, bevor das Konzept ausgearbeitet wird. Im Anschluss folgt Kapitel 6, welches sich mit der Realisierung befasst und zwei Lösungen vorstellt. Im Anschluss daran folgt die Evaluation in Kapitel 7 unter Einbezug des Beispielszenarios und der Anforderungen aus Kapitel 5. Abschließend folgt Kapitel 8 mit einer Zusammenfassung der Arbeit und einem abschließenden Fazit.

# 2. Big Data

Dieses Kapitel gibt einen kurzen Überblick darüber, worum es sich bei dem Begriff *Big Data* handelt. Nach einer Begriffsdefinition in Abschnitt 2.1 folgt eine Darstellung des Big Data-Lebenszyklus in Abschnitt 2.2. Im Anschluss werden in Abschnitt 2.3 Werkzeuge und Technologien zur Umsetzung vorgestellt.

## 2.1. Definition

Big Data ist ein Begriff für den es keine offizielle, einheitliche Definition gibt (vgl. Fasel und Meier, 2016, S.3). Es haben sich jedoch in den vergangenen Jahren Definitionsansätze etabliert. Diese Definitionsansätze machen deutlich, dass es sich bei dem Begriff Big Data nicht nur um große Datenmengen handelt, sondern um einen Kontext, der sich kontinuierlich weiterentwickelt (vgl. Klein u. a., 2013, S.319f.). Im Folgenden werden die beiden bekanntesten Ansätze vorgestellt:

### 2.1.1. Erster Definitionsansatz

Der erste Definitionsansatz beschreibt Big Data als eine Datenmenge, die so groß ist, dass sie mit herkömmlichen Mitteln der elektronischen Datenverarbeitung nicht in angemessener Zeit gespeichert, verwaltet oder verarbeitet werden kann (vgl. Merv, 2011). Bei diesen Datenmengen handelt es sich um interne Unternehmensdaten und externe Daten (vgl. Sagiroglu und Sinanc, 2013, S.46) aus der Wirtschaft, der Gesellschaft und der Forschung. Durch die Analyse dieser Datenmengen werden Informationen generiert, aus denen neue Erkenntnisse gewonnen werden können (vgl. Merv, 2011).

### 2.1.2. Der V-Definitionsansatz

Die Frage nach der Definition von Big Data wird in der heutigen Zeit mit dem V-Definitionsansatz beantwortet. Bei dem *V* handelt es sich um Wörter, die mit dem Buchstaben V beginnen und die Eigenschaften von Big Data beschreiben (vgl. Fasel und Meier, 2016, S.6):

**Volume (große Menge):** Mit Daten-Volumina, deren Größen im Terabyte- ($10^{12}$ Byte) bis Zettabyte-Bereich ($10^{21}$ Byte) liegen, sind die Datenbestände äußerst groß. Der klassische Definitionsansatz ist damit auch erfüllt, weil klassische Datenbanksysteme, wie z.B. SQL-Datenbanken, bei der Berechnung solcher Größen sehr viel Zeit benötigen.

**Velocity (Geschwindigkeit):** Die großen Datenbestände müssen in sehr kurzer Zeit bearbeitet werden können. Das bedeutet in der Praxis, dass die Analyse und Auswertung von sehr großen Datenströmen in Echtzeit ausgeführt wird. Diese Daten müssen auch allen Beteiligten in kürzester Zeit zur Verfügung stehen.

**Variety (Vielfalt):** Die Quelldaten sind meist sehr unstrukturiert. Zum einen haben die Daten unterschiedliche Typen, da neben Dokumenten und Tabellen auch Bilder und andere Multimedia-Dateien gespeichert werden und zum anderen stammen die erhobenen Daten aus unterschiedlichen Quellsystemen, welche die Daten unterschiedlich strukturieren und bereitstellen.

Der Definitionsansatz der drei V hat sich in den letzten Jahren weiterentwickelt. Aus den Erkenntnissen, die in den letzten Jahren gewonnen werden konnten, wurden dem Ansatz zwei Eigenschaften hinzugefügt (vgl. Meier und Kaufmann, 2016, S.13):

**Value (Wert):** Die aus den Analysen gewonnen Informationen stellen einen Mehrwert dar, da sich mit den gewonnenen Erkenntnissen der Unternehmenswert steigern lässt.

**Veracity (Wahrhaftigkeit):** Je umfangreicher Datenbestände sind, desto schwieriger ist es, ihre Aussagekraft einzuschätzen. Spezifische Algorithmen helfen dabei, die Qualität der Datenbestände zu bewerten.

## 2.2. Big Data-Lebenszyklus

Der Big Data-Lebenszyklus ist ein allgemeines Modell, welches sich in sechs grundlegende Phasen aufteilt. Dadurch, dass es allgemein gehalten wurde, dient es nicht nur einer bestimmten Gemeinschaft, sondern allen, die im Big Data-Kontext arbeiten. Es hilft in der Praxis vor allem dabei die Phase, in der sich Projekt bzw. Daten befinden, und die Fragen, die sich aus der jeweiligen Phase ergeben, zu bestimmen und basierend darauf Entscheidungen zu treffen (vgl. Pouchard, 2015, S.183ff.).
Abbildung 2.1 repräsentiert den Big Data-Lebenszyklus:

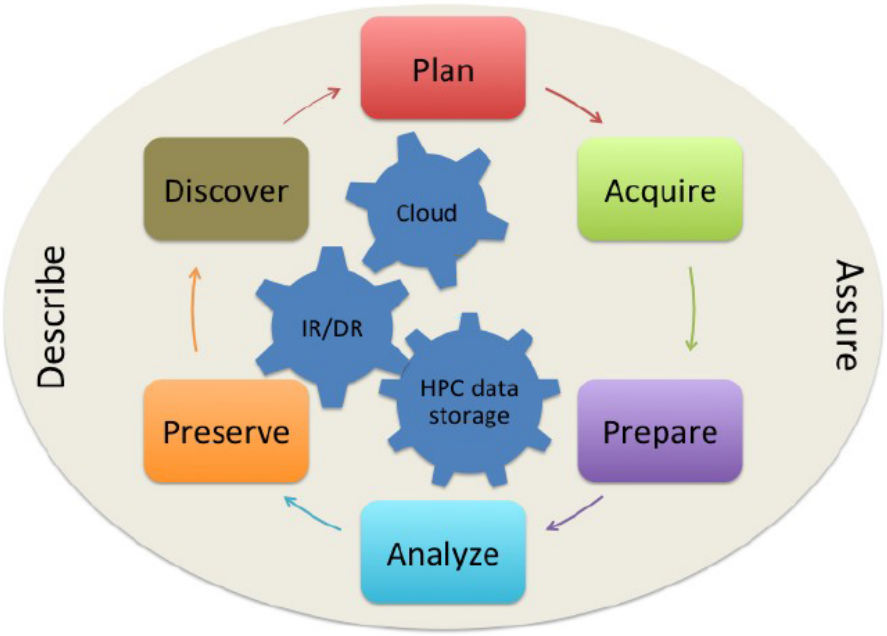


Abbildung 2.1.: Big Data-Lebenszyklus
Quelle: Übernommen aus Pouchard, 2015, S.184

Es handelt sich um ein iteratives Modell, d.h., dass das Modell nach Ablauf der letzten Phase wieder bei der ersten Phase beginnt. Der Lebenszyklus endet erst, wenn es keine Anwendung mehr für ein betroffenes Projekt bzw. die betroffenen Daten gibt.

Die beiden äußeren Aktivitäten „Describe“ (Beschreiben) und „Assure“ (Sicherstellen) sind Teil jeder Phase. „Describe“ gibt vor, dass die Daten und Prozesse der einzelnen Phasen dokumentiert werden, um so die Entdeckung neuer Informationen zu beschleunigen. „Assure“ bedeutet, dass die Verarbeitung und Analyse der Daten beschrieben werden. Dies hilft bei der Qualitätssicherung der Daten und erhöht damit das Vertrauen.

Die Zahnräder in der Mitte stellen die Infrastruktur dar. Dabei handelt es sich um Cloud-Plattformen (Cloud), interne- und externe Datenquellen (IR/DR) und Datenspeicher auf Hochleistungsrechnern (HPC data storage). Die sechs kreisförmig angeordneten Aktivitäten beschreiben die Phasen, die während des Lebenszyklus durchlaufen werden:

**Plan (Planen):** Aufgrund der großen Datenmengen, müssen die technischen Voraussetzungen für die Speicherung und Bearbeitung der ermittelt und umgesetzt werden. Je nach

Anwendungsfall kann es vorkommen, dass die Rohdaten gespeichert werden müssen, da sie nicht reproduzierbar sind. Das ist äußerst vorteilhaft, wenn z.B. forensische Analysen auf den Datenbeständen betrieben werden. Außerdem muss definiert werden, welche Daten analysiert und wie die Ergebnisse hinterher dargestellt werden sollen.

**Acquire (Beschaffen):** Die Daten werden erzeugt, generiert bzw. aufgenommen. Die Beschaffung der Daten erfolgt über verschiedenste Quellen. Zum Beispiel lassen sich große Datenbestände aus Sensoren, Instrumenten wie Massenspektrometern, Computersimulationen oder Downloads aus externen Quellen beschaffen. Die Rohdaten enthalten oft Anteile, die für die eigenen Auswertungen nicht relevant sind und werden bei der Beschaffung herausgefiltert. Die Herausforderung liegt darin, die Filter so zu gestalten, dass keine wichtigen Informationen verloren gehen (vgl. Labrinidis und Jagadish, 2012, S.2032).

**Prepare (Vorbereiten):** Um die Daten für Analysen und Visualisierungen bereitzustellen, werden sie weiter verarbeitet und für entsprechende Analyse- und Visualisierungsplattformen nutzbar gemacht (vgl. Heer und Shneidermann, 2012, S.21). Die Komplexität dieses Arbeitsschritts hängt von der Qualität der Daten ab.

**Analyze (Analysieren):** Für die Informationsgewinnung analysieren Data Scientists die vormals verarbeiteten Daten. Hierbei kommen vor allem statistische Methoden und maschinelles Lernen zum Einsatz. Dabei spielt die Reproduzierbarkeit der Ergebnisse eine wichtige Rolle, um die Validität und die Bedeutung der Daten bewerten zu können (vgl. James u. a., 2014, S.7).

**Preserve (Aufbewahren):** Ergebnisse werden für die langfristige Nutzung erhalten.

**Discover (Entdecken):** Hierbei handelt es sich um eine Reihe von Verfahren, die sicherstellen, dass Datensätze, die für eine bestimmte Analyse oder Sammlung relevant sind, gefunden und bereitgestellt werden.

## 2.3. Werkzeuge und Technologien

In der heutigen Zeit gibt es eine sehr große Anzahl an Applikationen, die den Umgang mit großen Datenmengen ermöglichen und erleichtern. Einen Eindruck darüber gibt die *Big Data-Landscape* (vgl. Abbildung 2.2):

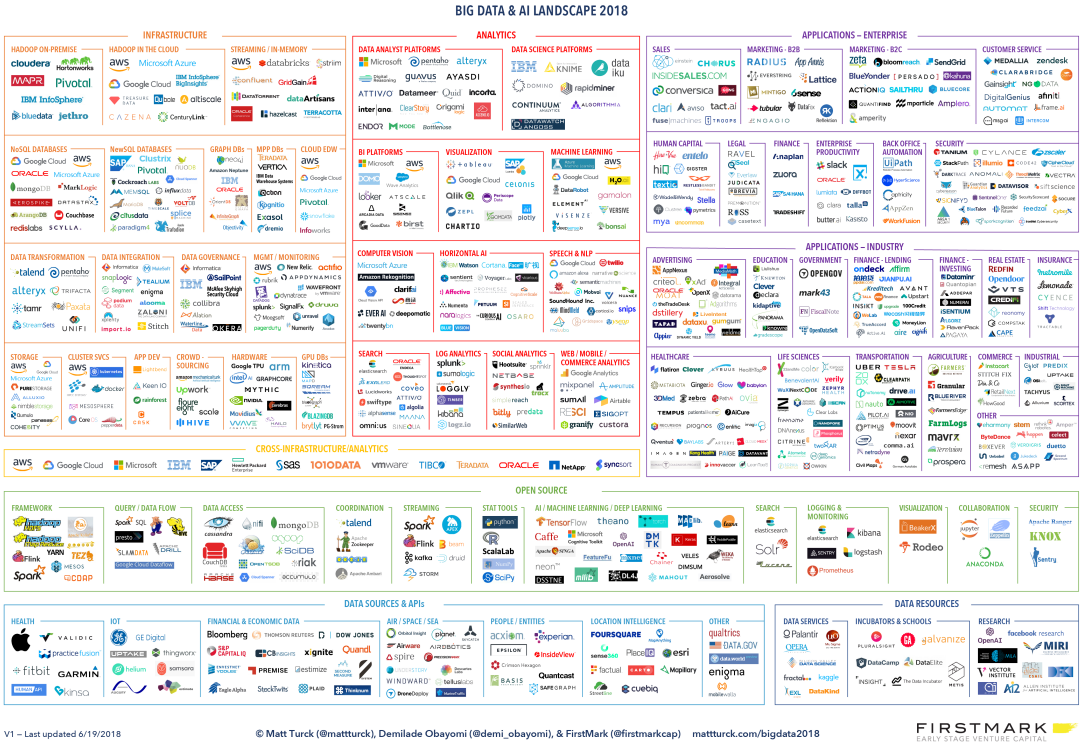


Abbildung 2.2.: Big Data & AI Landscape
Quelle: Übernommen aus Turck, 2018

Dabei handelt es sich um eine Sammlung von Anwendungen, Diensten und Frameworks, mit denen es möglich ist, große Datenmengen in adäquater Zeit zu speichern und zu analysieren. Die Grafik illustriert, dass es eine sehr große Anzahl an Werkzeugen gibt, wobei diese Werkzeuge stetig weiterentwickelt werden und sich deren Anzahl mit der Zeit weiter erhöht, was bestätigt, dass ein sehr großes Interesse an der Verarbeitung großer Datenmengen besteht.

### 2.3.1. Konsistenzmodelle

Aufgrund der schon in Abschnitt 2.1 beschriebenen Problematik der stetig wachsenden Menge an Daten, haben sich die Anforderungen an die Datenbanksysteme geändert. Um diese Anforderungen zu erfassen, hilft die Erläuterung der gängigen Konsistenzmodelle *ACID* und *BASE*, deren Charakteristika im Einklang mit dem *CAP-Theorem* stehen.

#### 2.3.1.1. CAP-Theorem

Das CAP-Theorem ist der Beweis dafür, dass nur zwei der drei im Akronym *CAP* beschriebenen Eigenschaften in stark verteilten Systemen in einem Fehlerfall eingehalten werden können. Bei diesen Eigenschaften handelt es sich um Consistency (Konsistenz), Availability (Verfügbarkeit), Partition Tolerance (Fehlertoleranz beim Ausfall einer oder mehrerer Computer) (vgl. Brewer, 2000).

#### 2.3.1.2. ACID

*ACID* ist ein Akronym, welches die Charakteristika der Transaktionen in diesem Konsistenzmodell beschreibt. Eine Transaktion ist eine Sequenz von Operationen, die als eine logische Einheit zusammengefasst werden (vgl. Haerder und Reuter, 1983, S.289). Die schon erwähnten relationalen Datenbanksysteme machen Gebrauch von diesen Transaktionen (vgl. Sadalage und Fowler, 2013, S.19). Hierbei liegt der Fokus vor allem auf der Konsistenz (Consistency) der Daten (vgl. Meier, 2017, S.34). Das heißt, dass *ACID-Systeme*, bedingt durch das CAP-Theorem, im Fehlerfall gleichzeitig nur die Eigenschaften CA oder CP ermöglichen können.
In der Vergangenheit wurde größtenteils mit relationalen Datenbanksystemen gearbeitet, da diese ein sehr hohes Maß an Konsistenz (vgl. Fasel und Meier, 2016, S.11) und eine standardisierte Abfragesprache (vgl. Unterstein und Matthiessen, 2012, S.35) bieten. Sie erfüllen die Eigenschaften von ACID. Man spricht bei diesen Datenbanksystemen auch von SQL-Datenbanken (SQL = Structured Query Language).

**Atomicity (Atomarität):** Eine Transaktion wird vollständig oder gar nicht ausgeführt. Im Falle eines Fehlers während einer Transaktion, werden alle bisher ausgeführten Operationen rückgängig gemacht (vgl. Haerder und Reuter, 1983, S.289).

**Consistency (Konsistenz):** Die Daten müssen sich vor und nach einer Transaktion in einem konsistenten Zustand befinden. Während einer Transaktion können sie auch inkonsistent sein (vgl. Haerder und Reuter, 1983, S.289).

**Isolation (Isolierung):** Transaktionen werden nicht nebenläufig ausgeführt. So wird verhindert, dass sich Transaktionen gegenseitig stören (vgl. Haerder und Reuter, 1983, S.290).

**Durability (Dauerhaftigkeit):** Daten werden nach einer Transaktion dauerhaft gespeichert (vgl. Haerder und Reuter, 1983, S.290)..

#### 2.3.1.3. BASE

Der Begriff *BASE* ist ebenfalls ein Akronym, welches die Charakteristika des dazugehörigen Konsistenzmodells beschreibt. Bei diesem Konsistenzmodell liegt der Fokus auf der Erreichbarkeit, was bedeutet, dass nur CA- und AP-Systeme möglich sind. Viele NoSQL-Datenbanksysteme, auf die im weiteren Verlauf Bezug genommen wird, erfüllen zum größten Teil dieses Konsistenzmodell (vgl. Sadalage und Fowler, 2013, S.56). Aufgrund seiner Eigenschaften eignet sich *BASE* vor allem für die Umsetzung massiv verteilter Systeme.

**Basically Available (Grundsätzlich verfügbar)** : Das System muss zu jedem Zeitpunkt verfügbar sein.

**Soft State (Weicher Zustand):** Der Zustand des Systems kann sich ändern, auch wenn zu einem Zeitpunkt am System nichts geändert wurde. Das System befindet sich in diesem Zustand, sobald eine Änderung noch nicht an alle Knoten weitergegeben wurde.

**Eventually Consistent (Irgendwann konsistent):** Die Daten erreichen im Laufe der Zeit einen konsistenten Zustand. Das bedeutet im Umkehrschluss, dass das System nicht zu jedem Zeitpunkt konsistent ist.

#### 2.3.1.4. ACID vs. BASE

*BASE*-Systeme stellen eine Alternative zu *ACID*-Systemen dar (vgl. Banothu u. a., 2016, S.3704). Damit wird das Spektrum an Möglichkeiten, Daten zu halten und bereitzustellen, erweitert. Da diese beiden Konsistenzmodelle zwei vollkommen unterschiedliche Anforderungsprofile abdecken, wird das eine Konzept nicht das andere ablösen. Eher wird es dazu kommen, dass solche Systeme miteinander kombiniert werden, um die Vorteile aus beiden nutzen zu können. In Abbildung 2.3 werden die grundlegenden Unterschiede beider Konsistenzmodelle aufgezählt und gegenübergestellt:

| ACID | BASE |
|---|---|
| • Konsistenz hat oberste Priorität (strong consistency)<br>• meistens pessimistische Synchronisationsverfahren mit Sperrprotokollen<br>• Verfügbarkeit bei überschaubaren Datenmengen gewährleistet<br>• einige Integritätsregeln sind im Datenbankschema gewährleistet (z. B. referenzielle Integrität) | • Konsistenz wird verzögert etabliert (weak consistency)<br>• meistens optimistische Synchronisationsverfahren mit Differenzierungsoptionen<br>• hohe Verfügbarkeit resp. Ausfalltoleanz bei massiv verteilter Datenhaltung<br>• kein explizites Schema vorhanden |

Abbildung 2.3.: ACID vs. BASE
Quelle: Übernommen aus Fasel und Meier, 2016, S.35

Diese Darstellung verdeutlicht, dass *ACID*-Systeme Konsistenz als oberste Priorität haben und *BASE*-Systeme hohe Verfügbarkeit bei großen Datenmengen anstreben.

### 2.3.2. Datenhaltung mit NoSQL-Datenbanken

Der Begriff NoSQL ist eine Abkürzung und bedeutet '*Not only SQL*' (vgl. Fasel und Meier, 2016, S.V). Er steht für Datenbanksysteme, die anders als SQL-Datenbanksysteme, keine standardisierte Abfragesprache, also kein SQL, haben (vgl. Pokorny, 2011, S.280), sondern eigene Abfragesprachen und APIs (application programming interface), um auf die Daten zuzugreifen und diese zu manipulieren. Im Allgemeinen sind NoSQL-Datenbanken Open Source (vgl. Sadalage und Fowler, 2013, 10), das bedeutet, dass deren Quelltexte öffentlich gemacht wurden und sie kostenfrei zu erwerben sind. Eine weitere Charakteristik ist die Tatsache, dass diese Datenbanksysteme ihre Daten nicht in Form von Tabellen, also schemalos, abspeichern (vgl. Meier und Kaufmann, 2016, S.20).

Die meisten NoSQL-Datenbankensysteme sind in ihrer Architektur darauf ausgelegt, in Clusters, also einer Ansammlung von mehreren Instanzen der jeweiligen Datenbank, die in diesem Kontext als Knoten (Nodes) bezeichnet werden, betrieben zu werden (vgl. Sadalage und Fowler, 2013, S.10), wodurch erst die Bearbeitung sehr großer Datenmengen ermöglicht wird. Außerdem wird dadurch auch das horizontale Skalieren unterstützt (vgl. Meier und Kaufmann, 2016, S.20), wodurch hohe Verfügbarkeit durch Replikation und Partitionierung der Daten begünstigt wird. NoSQL-Datenbanksysteme eignen sich daher als Komponenten für Web-Anwendungen,

da diese auch '*basically available*' sein sollen.

Es haben sich, neben dem relationalen Datenmodell, in den letzten Jahren mehrere Datenmodelle, die mit NoSQL-Datenbanken umgesetzt wurden, etabliert. Bei diesen Datenmodellen handelt es sich um Key-Value, Column Family, Document und Graph (vgl. Sadalage und Fowler, 2013, S.13). Einige NoSQL-Datenbanksysteme setzen dabei auf die In-Memory-Methode, wobei der Hauptspeicher, auch Arbeitsspeicher genannt, als Datenspeicher dient, um höhere Geschwindigkeiten bei Zugriffen zu ermöglichen (vgl. Klein u. a., 2013, S.322), was im Gegensatz zur On-Disk-Methode steht, wo die Daten im Festplattenspeicher persistiert werden und die Zugriffe länger dauern. Es gibt auch Hybride-Datenbanksysteme, die von beiden Speichermedien Gebrauch machen.

Abbildung 2.4 illustriert die Komplexität und die Skalierbarkeit der einzelnen Datenmodelle. Bei steigender Komplexität sinkt die Skalierbarkeit von NoSQL-Systemen. Key-Value-Stores sind dabei die einfachsten Systeme (vgl. Sadalage und Fowler, 2013, S.81), da sie die unkomplizierteste Datenstruktur haben. Ein Datensatz besteht dabei nur aus einem Schlüssel und einem dazugehörigen Wert, wobei nur Schlüssel-basierte Abfragen möglich und diese dadurch sehr performant sind (vgl. Kapitel 4) basieren (Meier, 2017, S.43; Fasel und Meier, 2016, S.113;). Ein Key-Value-Store organisiert seine Daten also genauso wie eine Map. Dadurch lassen sie sich am einfachsten skalieren (vgl. Moniruzzaman und Hossain, 2013, S.5).

Danach kommen die Column-Family-Stores. Hier werden die Daten in komplexeren, multidimensionalen Maps abgespeichert. Für jeden Wert, der hinterlegt wird, existiert zusätzlich ein Zeitstempel (vgl. Gudivada u. a., 2014, S.194). Dies ermöglicht eine automatische Versionierung der Daten (vgl. Fasel und Meier, 2016, S.118). Attribut-orientierte Abfragen, wie z.B. Aggregationen, sind in diesen Systemen äußerst performant.

Document Databases halten die Daten in Form eines Schlüssels, der auf ein Dokument referenziert. Ein Dokument hält Schlüssel-Wert-Paare in einer hierarchischen Struktur (vgl. Sadalage und Fowler, 2013, S.89). Der jeweilige Schlüssel ist der Attributbezeichner. In einer Document Database können sich die Attribute der einzelnen Dokumente unterscheiden (vgl. Fasel und Meier, 2016, S.115). Oft werden die Daten in Form von JSON Dokumenten abgespeichert (vgl. Gudivada u. a., 2014, S.194).

Am komplexesten sind die Graph Databases. Sie sind, im Gegensatz zu den anderen Datenmodellen, nicht auf Clustering ausgerichtet (vgl. Sadalage und Fowler, 2013, S.26) und einige folgen dem *ACID*-Konsistenzmodell (vgl. Grolinger u. a., 2013, S.7). Hier werden Entitäten in Form von Knoten und Beziehungen, zwischen den Knoten, in Form von Kanten dargestellt (vgl. Fasel und Meier, 2016, S.111).

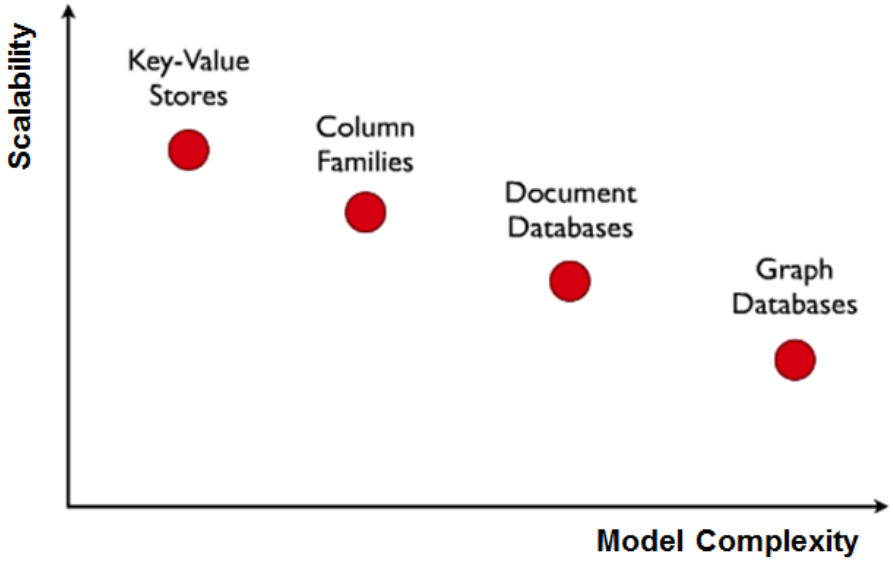


Abbildung 2.4.: NoSQL Complexity
Quelle: Übernommen aus Hsieh, 2014

### 2.3.3. Analysewerkzeuge und Anwendungen

Um auf die Daten zuzugreifen und diese zu analysieren, gibt es verschiedene Möglichkeiten. Die meisten NoSQL-Datenbanksysteme bieten eine API an, die es ermöglicht, Daten über die gängigen Programmiersprachen, wie z.B. Java oder Python, zu ändern, abzurufen und zu visualisieren oder ein command line interface (CLI), wodurch Zugriffe mit Hilfe von Skriptsprachen wie z.B. Bash (Linux) und Powershell (Windows) möglich sind und diese zu automatisieren. Jede API bzw. CLI ist individuell und es gibt keine Standards.

Neben der Möglichkeit, mittels APIs und CLIs auf die NoSQL-Datenbanken zuzugreifen und die Daten auf diese Weise zu prozessieren, gibt es Frameworks, welche die Analyse und Berechnung von Daten über verteilte Systeme ermöglichen (vgl. Apache, 2014, S.1ff.). Das Basiskonzept dieser Frameworks nennt sich Mapreduce. Dabei handelt es sich um ein Programmiermodell,

dass auf den Funktionen *map* und *reduce* aus der funktionalen Programmierung basiert (vgl. Dean und Ghemawat, 2008, S.107).

# 3. DevOps

In diesem Kapitel wird erläutert, worum es sich bei dem Begriff DevOps handelt. Zunächst wird in Abschnitt 3.1 erläutert, wie DevOps definiert wird. Im Anschluss wird in Abschnitt 3.2 auf die Anforderungen, die an DevOps gerichtet sind, eingegangen. In Abschnitt 3.3 wird der Begriff *DevOps-Lebenszyklus* erläutert. Weiterhin werden in Abschnitt 3.4 typische Werkzeuge zur Umsetzung von DevOps vorgestellt. Im letzten Abschnitt 3.5 werden schließlich Gemeinsamkeiten und Beziehungen zwischen *Big Data* und *DevOps* zusammengefasst.

## 3.1. Definition von DevOps

Der Begriff DevOps ist ein Kunstwort, das sich aus den beiden Bezeichnungen für die IT-Bereiche Development (Entwicklung) und Operations (Betrieb) zusammensetzt (vgl. Alt u. a., 2017, S.13). Da es zu diesem Begriff keine klare Definition gibt, haben sich mehrere Ansätze für eine Definition ergeben. Zum Beispiel wird DevOps als Sammlung von Praktiken angesehen. Diese Praktiken sollen unter Gewährleistung hoher Qualität die Zeit zwischen einer begangenen Änderung in einem System und ihrer Überführung in die Produktivumgebung verkürzen (vgl. Bass u. a., 2015, S.4).

Ein weiteres Beispiel für einen Definitionsansatz ist, dass DevOps als eine Kombination von Denkweisen, Werkzeugen und Praktiken angesehen wird. Diese Kombination hilft dabei, dass Unternehmen ihre Applikationen einfach und schnell bereitstellen und weiterentwickeln können. Daraus ergibt sich ein Vorteil anderen Unternehmen gegenüber, die DevOps nicht nutzen (vgl. Amazon, 2018).

Durch Einführung von agilen und schlanken Praktiken wird DevOps auch als Wandel in der IT-Kultur verstanden. Die Zusammenarbeit zwischen Entwicklung und Betrieb soll insbesondere durch Automatisierung verbessert werden (vgl. Gartner, 2018).

Andere sehen DevOps als ein anderes Selbstverständnis von Entwicklung und Betrieb an. Dieses Selbstverständnis trennt die Bereiche nicht mehr nach Ihrer Abteilung, sondern nach

den Diensten, die sie betreuen. Daraus ergibt sich, dass hinter jedem Service ein Team steht, welches Spezialisten aus dem Betrieb und der Entwicklung hat, was die Kommunikationswege, die einen Service betreffen, stark verkürzt (vgl. Wolff, 2015, S.235ff.).

Ein weiterer Ansatz ist die Beschreibung durch das Akronym *CAMS* (vgl. Willis, 2010):

**Culture (Kultur):** An erster Stelle stehen die Menschen und die Prozesse. DevOps bedeutet für viele Unternehmen und deren Mitarbeiter eine Veränderung. Damit diese Veränderung funktionieren kann, müssen alle Menschen sich darauf einlassen und die Prozesse müssen kompatibel sein bzw. angepasst werden.

**Automation (Automatisierung):** Die Automatisierung des Auslieferungsprozesses mit unterstützenden Werkzeugen. Durch die Automatisierung jeder Phase sinkt die Anfälligkeit für Fehler.

**Measurement (Messen):** Um sich kontinuierlich zu verbessern, ist es wichtig, kontinuierlich Daten über die gesamte Auslieferungskette zu sammeln.

**Sharing (Teilen):** Es ist wichtig, Ideen und Probleme zu teilen. Daraus lässt sich wiederum Wissen generieren. Dieses Wissen trägt zur eigenen Verbesserung bei.

Es haben sich zwar unterschiedliche Vorstellungen vom Thema DevOps gefestigt, jedoch weisen diese im Kern einige Überschneidungen auf. Die Effizienz eines Unternehmens soll gesteigert werden, mit dem Ziel, die Qualität des eigenen Produkts zu erhöhen (vgl. Hasselbring, 2015, Folie 6). Das führt zu höheren Einnahmen, weil das Produkt dadurch einen höheren Mehrwert für den Kunden hat (vgl. Peschlow, 2016, S.9). Die Grenzen zwischen allen beteiligten Abteilungen sollen aufgehoben werden. Neben Betrieb und Entwicklung z.B. auch die Qualitätsicherung (vgl. Stähler, 2016). Das verstärkt die Zusammenarbeit durch Verkürzung der Kommunikationswege zwischen den einzelnen Abteilungen. Durch maximale Transparenz und Automatisierung werden die einzelnen Bereiche entlastet, wodurch sie sich wieder mehr auf ihre Kernaufgaben fokussieren können (vgl. Fischer, 2015, Teil 1).

Bei DevOps handelt es sich also um ein Selbstverständnis von Entwicklung und Betrieb (vgl. Wolff, 2015, S.238), das sich von dem klassischen Selbstverständnis unterscheidet und oft ein Umdenken in der Unternehmensorganisation erfordert, um den Auslieferungsprozess zu verbessern, indem Engpässe eliminiert und sich wiederholende Prozesse automatisert werden.

## 3.2. Anforderungen an DevOps

Traditionell sind die Bereiche Entwicklung (Development) und IT-Betrieb (Operations) voneinander getrennt (vgl. Wolff, 2015, S.235). Durch die komplett unterschiedlichen Zielsetzungen dieser beiden Bereiche, haben sich situativ Hürden entwickelt. Der IT-Betrieb wird nach der Verfügbarkeit der Server und Applikationen bewertet (vgl. Hüttermann, 2012, S.6) und an der Kosteneffizienz gemessen (vgl. Wolff, 2015, S.235). Der Betrieb ist also dafür Zuständig, die Systeme am Laufen zu erhalten und Veränderungen, z.B. durch neue Software, möglichst zu vermeiden, weil Veränderungen den Betrieb gefährden können. Es besteht grundsätzlich eine Angst vor Veränderungen.
Die Entwicklung wird anhand der Geschwindigkeit bemessen, d.h. wie lange die Erstellung neuer Funktionen dauert (Hüttermann, 2012, S.235; Wolff, 2015, S.235). Ziel der Entwicklung ist es, viele neue Funktionen in Produktion zu bringen. Die Entwicklung hat also einen erhöhten Bedarf an Veränderungen, der mit der Angst vor Veränderungen durch den Betrieb im Konflikt steht (vgl. Hüttermann, 2012, S.6).

Neben den unterschiedlichen Zielsetzungen sollen noch weitere Hürden durch den DevOps-Ansatz eliminiert werden (vgl. Hüttermann, 2012, S.17f.):

**Unterschiedliche Teams:** Die unterschiedlichen Bereiche haben ein gesteigertes Interesse daran, nur die eigenen Ziele voranzutreiben.

**Keine einheitliche Sprache:** Die Kommunikation zwischen den Bereichen ist problematisch, weil die einzelnen Bereiche unterschiedliche Sprachen und Frameworks benutzen.

**Angst:** Die Angst vor Veränderungen und davor, dass Operationen der anderen Bereiche negative Auswirkungen auf den eigenen Bereich haben.

Diese Hürden sind auch auf die anderen beteiligten Bereiche, wie z.B. QA, zu übertragen. Im weiteren Verlauf wird jedoch nur noch auf die Bereiche Development und Operations eingegangen, weil deren Anteil in einer DevOps-Umgebung am größten ist.

DevOps soll dabei helfen, genau diese Hürden zu eliminieren und dadurch, die wie in Abschnitt 3.1 beschriebene erhöhte Effizienz zu erreichen.

### 3.2.1. Traditioneller Auslieferungsprozess

Ein Produkt bzw. ein Software-Produkt durchläuft innerhalb seines Auslieferungsprozesses mehrere unterschiedliche Abteilungen (vgl. Humble und Farley, 2010, S.5). Diese Abteilungen

sind organisatorisch bedingt voneinander getrennt (vgl. Humble und Farley, 2010, S.439) und haben unterschiedliche Vorgaben. Daraus ergibt sich auch der Umstand, dass alle Abteilungen zwar am selben Produkt arbeiten, aber dies in unterschiedlichen Systemen. Oft haben diese Systeme keine Schnittstellen zueinander. Es müssen manuelle Schritte vorgenommen werden, um den Auslieferungsprozess voranzutreiben.

Die Entwicklungs-, Test- und Produktivumgebungen sind oft nicht vereinheitlicht (vgl. Kim u. a., 2017, S.107), wodurch es zu Kompatibilitätsproblemen kommen kann. Der Auslieferungsprozess ist also nicht automatisiert, was eine potenzielle Fehlerquelle darstellt und die Auslieferung verzögern kann und sich somit die Vorlaufzeit (time to market) erhöht (vgl. Wolff, 2015, S.3). Die Grenzen zwischen den Abteilungen machen es anspruchsvoller, die Ursache bei einem Fehler zu finden, da alle beteiligten Abteilungen nur ein ihrem eigenen Einsatzbereich Fehlersuchen betreiben können.

Wenn die Software ausgeliefert wurde, ist der Prozess aber noch nicht abgeschlossen. Nach dem Bereitstellen des Produkts wird es genutzt und Feedback eingeholt, um zu prüfen, ob noch Verbesserungen vorgenommen werden müssen (vgl. Kim u. a., 2017, S.123). Dieser Schritt ist auch nicht automatisiert, was weiteren Administrationsaufwand bedeutet.

## 3.3. DevOps-Lebenszyklus

Abbildung 3.1 zeigt, wie ein Software-Lebenszyklus aussehen kann, wenn Software-Entwicklung und Bereitstellung nach DevOps-Philosophie betrieben wird. Der Lebenszyklus ist in mehrere Phasen unterteilt. Die grauen Phasen liegen im Verantwortungsbereich der Entwickler und die grünen Phasen im Bereich des Betriebs, wobei beide zusammenarbeiten (vgl. Atlassian, 2018) bzw. das Softwareprodukt von einem Team betrieben wird, welches das Know-How aus beiden Bereichen in sich vereint. Das verkürzt die Kommunikationswege gegenüber einer Organisation, in der die Bereiche komplett getrennt voneinander operieren.

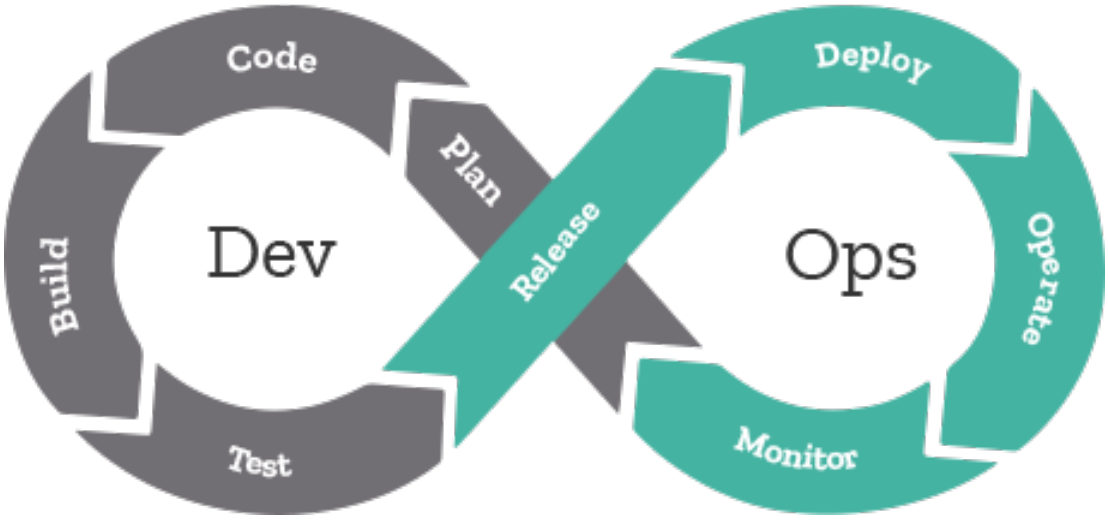


Abbildung 3.1.: DevOps Lebenszyklus
Quelle: Übernommen aus Tvisha (2018)

Im Folgenden werden die dargestellten Phasen erläutert:

**Plan (Planen):** Dabei handelt es sich um das Projektmanagement. In dieser Phase wird das Software-Produkt gestaltet und geplant. Es fließen vorhandene Erkenntnisse und Anwender-Feedback ein, um Optimierungen vorzunehmen.

**Code ("Coden"):** Die Ergebnisse der Plan-Phase werden in die Tat umgesetzt. Es wird Quelltext (Code) entwickelt und am Ende dieser Phase wird der Quelltext eingecheckt. Diese Phase wird auch oft als *Check-In* oder *Commit* bezeichnet und stellt die Entwicklungsphase dar.

**Build (Bauen):** Nach dem Einchecken des Quelltextes werden Unit-Tests ausgeführt und sobald diese abgeschlossen sind, wird aus dem Quelltext ein Softwarepaket gebaut.

**Continuous Testing (Test):** Nachdem das Softwarepaket gebaut wurde, wird es weiteren Tests unterzogen. Bei diesen Tests handelt es sich um vordefinierte und automatisierte Routinen, die von der Qualitätssicherung festgelegt wurden.

**Release (Freigeben):** Ist die Qualitätssicherung abgeschlossen, wird die Software in eine Staging-Umgebung geladen. Die Staging-Umgebung stellt eine Testumgebung bereit, die der Produktivumgebung sehr ähnlich ist. Hier werden automatisierte Tests unter realen

Bedingungen ausgeführt und explorative Tests ermöglicht. Sind alle Tests erfolgreich abgeschlossen, wird die Software zur Bereitstellung freigegeben.

**Deploy (Bereitstellen):** Es wird die endgültige Entscheidung getroffen, ob die Software bereitgestellt werden soll. Wird der Bereitstellungsprozess gestartet, dann wird die neue Software automatisch auf der Produktivumgebung verteilt und in einen einsatzbereiten Zustand versetzt.

**Operate (Betreiben):** Nach dem Ausrollen der Software wird diese auf der Produktivumgebung ausgeführt und betrieben.

**Monitor (Überwachen):** Alle Phasen werden mit Hilfe von Überwachungs- und Aufzeichnungs-Tools begleitet. Die gesammelten Daten aus diesen Tools werden aufbereitet und fließen, genau wie das Feedback der Product Owner und Endanwender, in die Plan-Phase der nächsten Iteration ein.

Aufgrund des hohen Automatisierungsgrades wird die Effizienz des Auslieferungsprozesses stark gesteigert, da er nur noch Minuten, statt Tagen oder Wochen in Anspruch nimmt und die gesamte Prozesskette reproduzierbar ist (vgl. Wolff, 2015, S.4), was den Aufbau von identischen Produktiv-, Test- und Entwicklungsumgebungen und den Aufbau neuer Auslieferungsprozesse stark vereinfacht. Aufgrund der hohen Auslieferungsgeschwindigkeit ergibt sich die Möglichkeit, Aktualisierungen funktions-orientiert zu veröffentlichen und parallel zu entwickeln, bzw. die Anzahl der möglichen Deploys wird zu erhöhen (vgl. Humble und Farley, 2010, S.109). Die Agilität eines Entwicklungsteams soll damit gesteigert werden.

## 3.4. DevOps-Werkzeuge

Die Umsetzung von DevOps erfordert eine Kombination aus Konzepten, die dabei helfen sollen, die Zusammenarbeit zu fördern. Dazu gehören z.B. das agile Projektmanagement, welches die Kommunikation und Zusammenarbeit innerhalb des Entwicklerteams, vor allem aber auch zum Kunden verbessert, Configuration Management zur Automatisierung beim Aufbau der Infrastrukturen, Continuous Delivery, um Änderungen kontinuierlich und automatisch bereitstellen zu können, Continuous Monitoring, um die Transparenz der Systeme zu erhöhen und die Fehlersuche zu vereinfachen und Continuous Feedback, um aus gewonnen Erkenntnissen das Software-Produkt kontinuierlich zu verbessern.

### 3.4.1. Agile Methoden

Die Agilen Methoden, wie z.B. *Extreme Programming* und *Scrum* (vgl. Alt u. a., 2017, S.21), bilden einen wichtigen Bestandteil von DevOps, da mit deren Hilfe schlechte Zusammenarbeit und unterschiedliche Zielsetzungen vermieden werden können (vgl. Hüttermann, 2012, S.18ff.). Die Basis der agilen Methoden bildet das *Agile Manifest*. Dabei handelt es sich um Werte, welche klassischen, schwergewichtigen Software-Entwicklungsprozessen, wie z.B. dem Wasserfallmodell (vgl. Royce, 1987, S.328ff.), entgegenwirken (vgl. Kim u. a., 2017, S.346), um die Flexibilität und Reaktionsgeschwindigkeit auf Änderungen im Software-Entwicklungsprozess zu erhöhen (vgl. Trepper, 2012, S.69). Folgende vier Leitsätze wurden als Werte des agilen Manifests definiert:

| |
|---|
| Individuen und Interaktionen mehr als Prozesse und Werkzeuge |
| Funktionierende Software mehr als umfassende Dokumentation |
| Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung |
| Reagieren auf Veränderung mehr als das Befolgen eines Plans |

Tabelle 3.1.: Leitsätze des agilen Manifests
Quelle: Übernommen aus Beck u. a. (2001)

Diese vier Sätze beschreiben wichtige Bestandteile des agilen Projektmanagements und stellen diese in Beziehung. Dabei gilt, dass alle Bestandteile wichtig sind, jedoch die Aspekte auf der jeweils linken Seite eines Satzes wichtiger sind, als diese auf der rechten Seite (vgl. Trepper, 2012, S.68). Der erste Satz beschreibt, dass die Menschen und deren Interaktionen wichtiger als die konkrete Planung sind, der zweite Satz, dass die Funktionstüchtigkeit der Software wichtiger als deren Dokumentation ist, der dritte Satz, dass es wichtiger ist, auf den Kunden einzugehen, als sich konsequent auf den vorher verhandelten Vertrag zu beziehen und nur danach zu arbeiten und der vierte Satz sagt aus, dass ein Plan zwar wichtig ist, es aber noch wichtiger ist, flexibel auf Änderungen zu reagieren (Cockburn, 2002, S.217ff; Wolff, 2015, S.13).

Agile Methoden helfen vor allem dabei, die Kommunikation zwischen Kunden, Projektmanagern und Entwicklerteams zu erhöhen und durch Feedback, welches direkt in die Entwicklung einfließt, die Qualität der Erzeugnisse bzw. bereitgestellten Dienste zu erhöhen. Der Dienst soll zu jeder Zeit um Funktionen erweitert werden können, ohne dabei die Verfügbarkeit des Dienstes zu beeinträchtigen.

### 3.4.2. Continuous Delivery

Continuous Delivery (CD) ist ein Konzept aus der Software-Entwicklung, wobei der Schwerpunkt auf der Bereitstellung von Software in die Produktionsumgebung (zu jeder Zeit) liegt (vgl. Fowler, 2013). Die Basis für CD bildet *Continuous Integration* (CI). Dabei handelt es sich um ein Vorgehensmodell, bei dem alle Beteiligten an einem Versionsstand arbeiten und durch Automatisierung kontinuierlich Software gebaut und getestet wird (vgl. Wolff, 2015, S.14).

Andersherum gilt CD als Erweiterung von CI. Dies wird deutlich, wenn man diese beiden Konzepte miteinander vergleicht. Das Prinzip sieht vor, dass eine erfolgreich abgeschlossene Phase die nächste Phase auslöst. Dabei sind alle Phasen automatisiert. Am Anfang steht die Commit-Phase, die vom Entwickler manuell eingeleitet wird. Dabei handelt es sich um das Einchecken einer Änderung in eine Versionsverwaltung bzw. ein Code Repository. Dieses Einchecken löst dann einen automatischen *Build* aus, welcher bei Erfolg automatisierte Integrationstests ansteuert. Sobald es innerhalb dieses Ablaufs zu einem Fehler kommt, wird der verantwortliche Entwickler benachrichtigt. An der Stelle ist Continuous Integration zu Ende. CD erweitert den Ablauf um eine Phase, in der die neue Version der Software in eine Staging-Umgebung geladen wird. Hierbei handelt es sich um eine produktivnahe Umgebung, in der Benutzerakzeptanztests und Leistungstests ausgeführt werden. Sobald die letzten Tests erfolgreich abgeschlossen wurden, ist die neue Version fertig für die Bereitstellung. An diesem Punkt haben die Entscheidungsträger die Möglichkeit, die Bereitstellung einzuleiten oder sie abzulehnen (vgl. Bass u. a., 2015, S.80).

Im Falle des Continuous Deployment ist der letzte Schritt auch automatisiert, was bedeutet, dass die neue Version automatisch bereitgestellt wird, sobald alle vorherigen Phasen erfolgreich abgeschlossen wurden (vgl. Bass u. a., 2015, S.80).

In der Praxis wird CD mit Hilfe einer *Continuous Delivery Pipeline* umgesetzt. Bei dieser Pipeline handelt es sich um eine Gruppe von Technologien und Werkzeugen, mit denen der Software-Lebenszyklus abgebildet, automatisiert und überwacht wird.

### 3.4.3. Continuous Monitoring

Beim Continuous Monitoring bzw. der kontinuierlichen Überwachung handelt es sich um die Aufzeichnung und Überwachung aller Phasen, Anwendungen und Systeme, die an den Auslieferungssprozess gebunden sind (vgl. Soni, 2016, S.31). Es werden Daten und Metriken

geliefert (vgl. Sharma, 2014, S.6), um kritische Probleme erkennen zu können (vgl. Farroha und Farroha, 2014, S.6). Dadurch wird den Stakeholdern eine schnelle Reaktion ermöglicht und die Konformität der Software gesichert.

Die folgende Tabelle enthält mögliche Zielsetzungen, die bei der Überwachung gesetzt werden können, und deren Quellsysteme:

| Überwachungsziel | Quelle |
|---|---|
| Fehlererkennung | Anwendung und Infrastruktur |
| Leistungsabbauerkennung | Anwendung und Infrastruktur |
| Kapazitätsplanung | Anwendung und Infrastruktur |
| Benutzerreaktion auf Geschäftsangebote | Anwendung |
| Eindringklingserkennung | Anwendung und Infrastruktur |

Tabelle 3.2.: Überwachungsziele
Quelle: Übernommen aus Bass u. a., 2015, S.129ff.

### 3.4.4. Continuous Feedback

Feedback ist ein zentrales Schlüsselelement von DevOps (vgl. Virmani, 2015, S.78). Das komplette Feedback wird allen Stakeholdern, außer den Kunden, bereitgestellt, damit jeder vom Anderen lernen kann (vgl. Humble und Molesky, 2011, S.4), was die Transparenz und Zusammenarbeit fördern. Kontinuierliches Feedback wird teilweise erst durch Continuous Monitoring ermöglicht, da dessen Daten in das Feedback einfließen (vgl. Balalaie u. a., 2016, S.43), genau wie das Feedback aller Stakeholder. Das Feedback ist damit äußerst wichtig, um aus vergangenen Fehlern zu lernen und sich kontinuierlich zu verbessern. Je schneller das Feedback ist, desto höher ist der Grad der Verbesserung (vgl. Cockburn, 2002, S.66). Dadurch dass das Feedback aller Phasen und das der Kunden zentral gesammelt wird, haben die Entwickler, welche für die frühen Phasen verantwortlich sind, Zugriff auf das Kunden-Feedback, welches erst nach der letzten Phase eingeholt werden kann, und können es dann direkt als Beitrag zur Verbesserung nutzen (vgl. Soni, 2016, S.66).

### 3.4.5. Configuration Management

Mit Hilfe des Configuration Managements (CM) wird DevOps erst ermöglicht (Smeds u. a., 2015, S.171; Spinellis, 2012, S.86). Dabei handelt es sich um einen fortlaufenden Prozess, in dem das Verhalten von Systemen zentral koordiniert und gleichzeitig dokumentiert wird (vgl. Couch

und Sun, 2004, S.215f.). In der Umsetzung sieht das so aus, dass der gewünschte Systemzustand einer Infrastruktur, also einer Gruppe von Maschinen, durch eine domänenspezifische Sprache beschrieben wird (vgl. Humble und Molesky, 2011, S.3). Für die Beschreibung der Zustände werden Text-basierte *Configuration Files* erstellt, in denen die Systemzustände in Form von Code spezifiziert werden (*Infrastructure as Code* (IaC)). IaC ist durch das Cloud Computing (vgl. Kapitel 4) entstanden und verfolgt das Ziel, Infrastrukturkomponenten regelbasiert und vollautomatisch zu installieren und zu konfigurieren (vgl. Alt u. a., 2017, S.30). Mit Hilfe dieser Configuration Files lassen sich, voll automatisiert, die spezifizierten Konfigurationen und Prozesse auf einer beliebigen Teilmenge der Maschinen beliebig wiederholen (vgl. Couch und Sun, 2004, S.216.), was den gesamten Software-Lebenszyklus effizienter und robuster macht. Grund dafür ist, dass durch die Automatisierung die operativen Schritte zur Verwaltung der Systeme nicht mehr manuell durchgeführt werden müssen und dadurch weniger Fehler entstehen können.

## 3.5. Bezug zu Big Data

Bei Big Data und DevOps handelt es sich um zwei unterschiedliche Kontexte. Jedoch weisen sie einige Gemeinsamkeiten auf, mit denen sie sich in Beziehung bringen lassen. Zum einen haben beide einen iterativen Lebenszyklus, der erst endet, wenn die jeweilige Anwendung nicht mehr betrieben wird. Zum anderen empfiehlt es sich, für die Umsetzung bzw. für den Aufbau von DevOps- und Big Data-Systemen Infrastrukturen zu verwenden, die auf Cloud Computing (vgl. Kapitel 4) basieren (Virmani, 2015, S.79; Spinellis, 2012, S.86; Assunção u. a., 2015, S.4;), da diese Infrastrukturen äußerst flexibel sind und andere Eigenschaften aufweisen, die bei der Ausführung von Big Data- und DevOps-Anwendungen helfen.

Von einer Architektur, welche die Vorteile beider Konzepte in sich trägt, würden neben Develepement und Operations auch die Data Analysts und Data Scientists profitieren. Die Bereitstellung und Auslieferung von Anwendungen, die für einen Big Data-Anwendungsfall geschrieben wurden, um z.B. Daten zu analysieren und zu visualisieren, lassen sich mit Hilfe von Continuous Delivery automatisieren.

Die Entwicklung von Big Data-Applikationen profitiert auch durch Konzepte wie agile Methoden, Continuous Feedback und Continuous Monitoring. Die Applikationen können zu jeder Zeit um Funktionen erweitert werden und die Ergebnisse, aus der Überwachung und den Feedback-Prozessen, helfen mögliche Schwachstellen zu finden und in die Planung einfließen

zu lassen.

Beim Continuous Monitoring werden Daten erzeugt. Das Volumen der erzeugten Daten hängt von der Größe der zu überwachenden Infrastruktur ab, wodurch es bei großen Systemen, aufgrund der Vielzahl an Aufzeichnungen, zu Big Data wachsen kann und damit bei der Speicherung, Auswertung und Visualisierung auf die Charakteristika von Big Data geachtet werden muss.

# 4. Cloud Computing

In diesem Kapitel wird der Begriff Cloud Computing kurz erläutert. Nach einer Definition des Begriffs in Abschnitt 4.1 folgt in Abschnitt 4.2 eine Übersicht der Eigenschaften von Cloud Computing. Im Anschluss wird im letzten Abschnitt 4.5 wird aufgezeigt, welche Beziehungen Cloud Computing zu Big Data und DevOps hat.

## 4.1. Definition

Cloud Computing ist ein Begriff, der in den späten neunziger Jahren auf einer Konferenz zum ersten Mal genannt wurde (Chellappa, 1997). Es gab seitdem mehrere Definitionsansätze für diesen Begriff. In einigen Definitionsansätzen werden Eigenschaften wie *"Illusion von unendlichen Computer-Ressourcen, die auf Abruf zur Verfügung stehen"* (vgl. Armbrust u. a., 2009, S.1) und *"Flexibilität und Skalierbarkeit von Infrastrukturen"* (vgl. Rittinghouse und Ransome, 2010, S.xiii) genannt. Der Begriff *Cloud* deutet an, dass Dienste von einem Anbieter im Internet bereitgestellt werden (vgl. Baun u. a., 2011, S.1-2). Diese und noch mehr Eigenschaften finden sich in der aktuell weitläufig anerkannten Definition von Cloud Computing (vgl. Reinheimer, 2018, S.4-5), die vom National Institute of Standards and Technology (NIST) für Cloud Computing aufgestellt wurde, wieder:

> "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics,three service models, and four deployment models." (Mell und Grance, 2011)

Es handelt sich also um ein Modell, bei dem Computerressourcen schnell, bequem und bedarfsorientiert bereitgestellt und abgerufen werden.

## 4.2. Charakteristika

Bei den Charakteristika handelt es sich um die wesentlichen Eigenschaften, die durch eine Cloud Computing-Plattform gewährleistet sein müssen (vgl. Vossen u. a., 2013, S.20-21).

**On-demand self-service:** Dem Verbraucher wird die Möglichkeit gegeben, sich seine Dienste selber zusammenzustellen. Er kann darüber entscheiden, welche Dienste und Ressourcen er benötigt und diese seinem Bedarf dynamisch anpassen, ohne, dass die Interaktion mit einem Mitarbeiter des CSP (Cloud Service Provider) erforderlich ist. Der Verbraucher bekommt damit die Möglichkeit, seine gemieteten Cloud-Services eigenständig seinem Bedarf anzupassen.

**Broad network access:** Die bereitgestellten Dienste sind über ein weitgehendes Netzwerk, typischerweise das Internet, erreichbar. Dabei stellen die CSP zur Nutzung ihrer Angebote häufig Schnittstellen, die auf Standards, z.B. HTTP und REST, basieren, zur Verfügung. Diese Schnittstellen ermöglichen die Anbindung einer großen Vielfalt an Endgeräten, z.B. Computer, Smartphones, Tablets, bzw. Client-Applikationen, z.B. Webbrowsern oder eigens entwickelter Client-Software. Ziel ist es, dem Verbraucher eine hohe Erreichbarkeit zu ermöglichen.

**Resource pooling:** Die bereitgestellten Dienste werden dabei von der physischen Hardware- und Software abstrahiert, wodurch sich die Ressourcen zu einem gemeinsamen virtuellen Vorrat zusammenfassen lassen und sie den Verbrauchern dynamisch zugewiesen werden können. Das ermöglicht dem Verbraucher, erforderliche Komponenten bedarfsorientiert zu reservieren und auch wieder freizugeben. Der Verbraucher hat dabei keine Kenntnis darüber, auf welcher Hardware seine Dienste ausgeführt werden, da die Komponenten, die er reserviert hat nur aus einem gemeinsamen virtuellen Pool entstammen.

**Rapid elasticity:** Die Cloud-Infrastruktur ist in der Lage, Computerressourcen schnell bereitzustellen. Das bedeutet für den Verbraucher, dass er die Möglichkeit hat, bei steigendem Ressourcenbedarf, mit wenig Aufwand mehr Ressourcen zu beziehen und diese schnell zum Einsatz zu bringen. In den meisten Fällen muss der Verbraucher die Ressourcenmenge manuell anpassen, jedoch gibt es für einige Serviceklassen die Möglichkeit, einen steigenden Bedarf zu erfassen und durch Regeln die Anzahl der Ressourcen automatisch anpassen zu lassen.

**Measured service:** Cloud-Dienste werden oft nutzungsbezogen verrechnet. Dadurch ist es für den CSP erforderlich die Nutzung der Ressourcen genau zu messen. Die Daten, die

aus dieser Messung hervorgehen, helfen einerseits die entstandenen Kosten zu ermitteln, aber auch dem Verbraucher einen Überblick darüber zu verschaffen, was er tatsächlich nutzt. Beide Seiten erhalten dadurch mehr Transparenz. Mit Hilfe dieser Daten kann der Ressourcenverbauch überwacht, kontrolliert und gemeldet werden.

## 4.3. Servicemodelle

Im Cloud Computing haben sich drei grundlegende Servicemodelle etabliert. Bei diesen Servicemodellen werden die Cloud Services in drei Abstraktionsebenen eingeteilt (vgl. Reinheimer, 2018, S.9). Jede Abstraktionsebene steht für eine Klasse an Diensten, die bereitgestellt werden.

Den Zusammenhang zwischen den drei Abstraktionsebenen illustriert Abbildung 4.1. Zu sehen ist eine Darstellung einer Vor-Ort-Infrastruktur (On-Premise Environment), also einer vom Verbraucher verwalteten IT-Landschaft, und die Darstellung der drei Servicemodelle in Schichten. Die grünen Schichten markieren dabei die Systemkomponenten, die vom Verbraucher verwaltet werden und die blauen zeigen jene, die vom CSP verwaltet werden. Dies verdeutlicht, auf welcher Abstraktionsebene die jeweiligen Servicemodelle, also die vom CSP bereitgestellten Dienste, beginnen. Außerdem verdeutlicht die Abbildung, dass die Servicemodelle aufeinander aufbauen. Zum Beispiel werden bei einem Service der Klasse *Software as a Service* automatisch alle Schichten aus den beiden übrigen Servicemodellen vom CSP verwaltet.

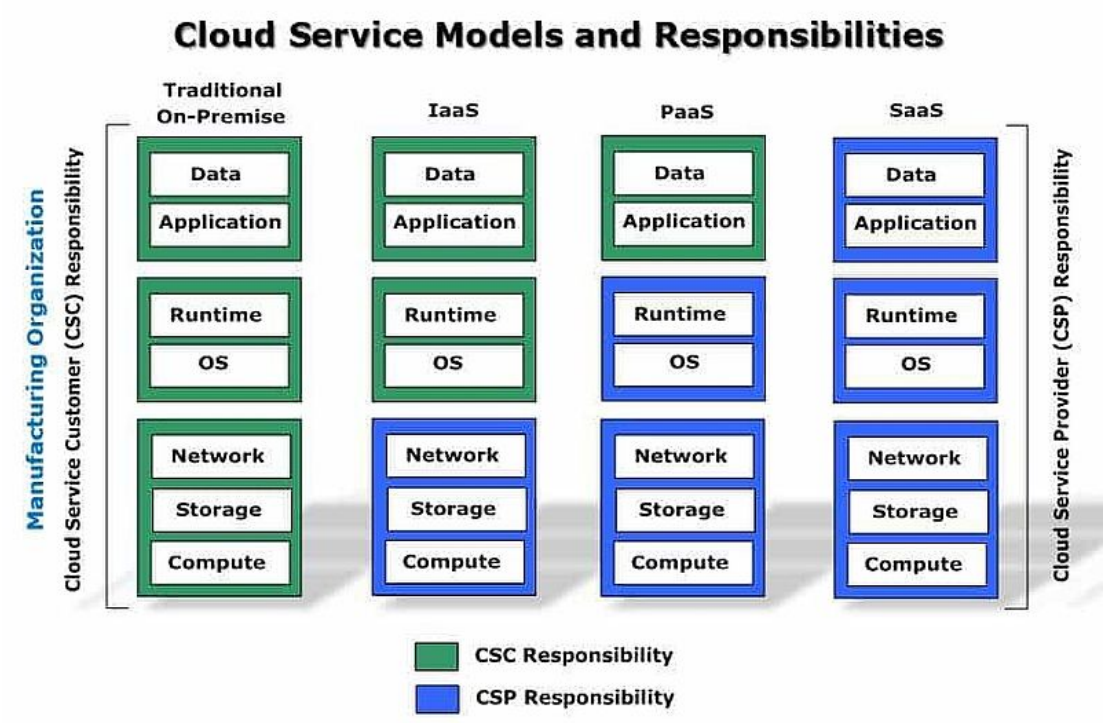


Abbildung 4.1.: Cloud Computing Servicemodelle
Quelle: Übernommen aus Novkovic, 2018

**Infrastructure as a Service (IaaS):** Durch den CSP werden Infrastrukturdienste angeboten. Dabei handelt es sich um Computerressourcen, wie z.B. Rechenleistung, Speicherplatz, Netzwerke oder virtuelle Maschinen. Ein Verbraucher kann mit Hilfe der bereitgestellten Dienste virtuelle IT-Systemlandschaften aufbauen und nach Bedarf anpassen. Während alle bereitgestellten Komponenten vom CSP verwaltet werden und er für dessen Funktionalität und Absicherung verantwortlich ist, fängt die Verantwortung für den Verbraucher auf der Schicht des Betriebssystems an und zieht sich über die darüber liegenden Schichten.

**Platform as a Service (PaaS):** Der Verbraucher kann zwischen verschiedenen Laufzeitumgebungen wählen, in denen er seine Applikation ausführen kann. Diese Laufzeitumgebungen werden auch als Plattform bezeichnet. Dabei ist zu beachten, dass die Verantwortung des Verbrauchers erst auf der Anwendungsebene beginnt und er somit an die Vorgaben, welche die Laufzeitumgebung betreffen, des CSP gebunden ist. Der CSP legt z.B. die

Programmiersprache und die Programmbibliotheken fest. Der Verbraucher kann ein paar Einstellungen an der Plattform vornehmen.

**Software as a Service (SaaS):** Hierbei liegt die komplette Verantwortung über die Funktionalität und die Sicherheit der Komponenten aller Schichten beim CSP. Dem Verbraucher werden Dienste in Form von Software angeboten und er hat nur die Möglichkeit Software zu konfigurieren und seine Daten zu hinterlegen. Diese Dienste sind häufig über Webbrowser zu erreichen.

**Big Data as a Service (BDaaS):** In den letzten Jahren wurde ein weiteres Servicemodell, namens *Big Data as a Service* (BDaaS), entwickelt (vgl. Skourletopoulos u. a., 2016; Xinhua u. a., 2014; Zheng u. a., 2013). Dabei werden Datenstrukturen und -definitionen in einen Service gekapselt und somit für den Verbraucher abgeschirmt (vgl. Vossen u. a., 2013, S.739). Der Verbraucher muss sich nur um die Auswahl und die Konfiguration seiner Services kümmern. BDaaS beinhaltet Services zur Speicherung, zur Analyse und zur Visualisierung von Daten.

## 4.4. Bereitstellungsmodelle

Für das Cloud Computing gibt es vier ursprüngliche Bereitstellungsmodelle. Diese Bereitstellungsmodelle unterscheiden sich in der Art und Weise, welchen Verbrauchern die Cloud-Services angeboten werden (vgl. Vossen u. a., 2013, S.30-31). Es handelt sich um eine allgemeine Klassifizierung der Berechtigungsstruktur, d.h. welchen Verbrauchern Zugriff auf die jeweilige Cloud gewährleistet wird.

**Public Cloud:** Bei einer Public Cloud (öffentliche Cloud) werden die Cloud Dienste der breiten Öffentlichkeit zur Verfügung gestellt. Jeder ist in der Lage sich bei einer solchen Cloud anzumelden und je nach Nutzungsgebühr die Dienste zu beziehen. Eine Public Cloud wird von einem CSP verwaltet und das zugehörige Rechenzentrum liegt meist in den Räumlichkeiten des CSP selbst.

**Private Cloud:** Im Gegensatz zur Public Cloud steht die Private Cloud. Sie ist nur für eine Organisation verfügbar. Diese Form der Cloud-Umgebungen wird oft von der Organisation im eigenen Rechenzentrum oder auf der Hardware eines Dienstleisters betrieben, wobei sich die Hardware in den jeweiligen Räumlichkeiten der Betreiber befindet. Unabhängig vom Standort der Hardware haben nur Mitglieder der Organisation, welche die Dienste der Private Cloud bezieht, Zugriff.

**Community Cloud:** Die Community Cloud (gemeinschaftliche Cloud) trägt Eigenschaften von *Public und Private Cloud* in sich. Hierbei handelt es sich auch um eine nicht-öffentliche Cloud, die ‚im Gegensatz zur Private Cloud, von mehreren Organisationen mit ähnlichen Anforderungen genutzt wird. Diese Form der Cloud-Umgebungen wird oft von einer der beteiligten Organisationen oder einem Dienstleister betrieben. Die Organisationen profitieren von dieser Form, da sie über die Isolation einer nicht-öffentlichen Cloud verfügen und sich die Kosten für den Betrieb teilen können.

**Hybrid Cloud:** Eine Hybrid Cloud beschreibt den Zusammenschluss von mindestens zwei Clouds. Organisationen sind durch dieses Bereitstellungsmodell in der Lage, ihre Dienste auf mehreren unterschiedlichen Clouds zu verteilen. Diese Art der Bereitstellung ermöglicht es Organisationen, die Hardware der eigenen Private Cloud mit der Hardware einer anderen Cloud zu erweitern und so z.B. höhere Lasten bewältigen oder zur Sicherung eine Replik der eigenen Infrastruktur erstellen zu können.

## 4.5. Bezug zu Big Data und DevOps

Cloud Computing, Big Data und DevOps begünstigen sich aufgrund ihrer Eigenschaften gegenseitig. Alle drei Kontexte helfen dabei, sowohl Ressourcen wie Informationen, Daten und Menschen miteinander zu verbinden. Dadurch kann die Zusammenarbeit gestärkt und gleichzeitig eine Steigerung der Effizienzerreicht werden.

Abbildung 4.2 / Tabelle 4.1 verdeutlicht, dass sich einige der wichtigsten Eigenschaften des Cloud Computing mit denen der anderen Kontexte in Relation setzen lassen:

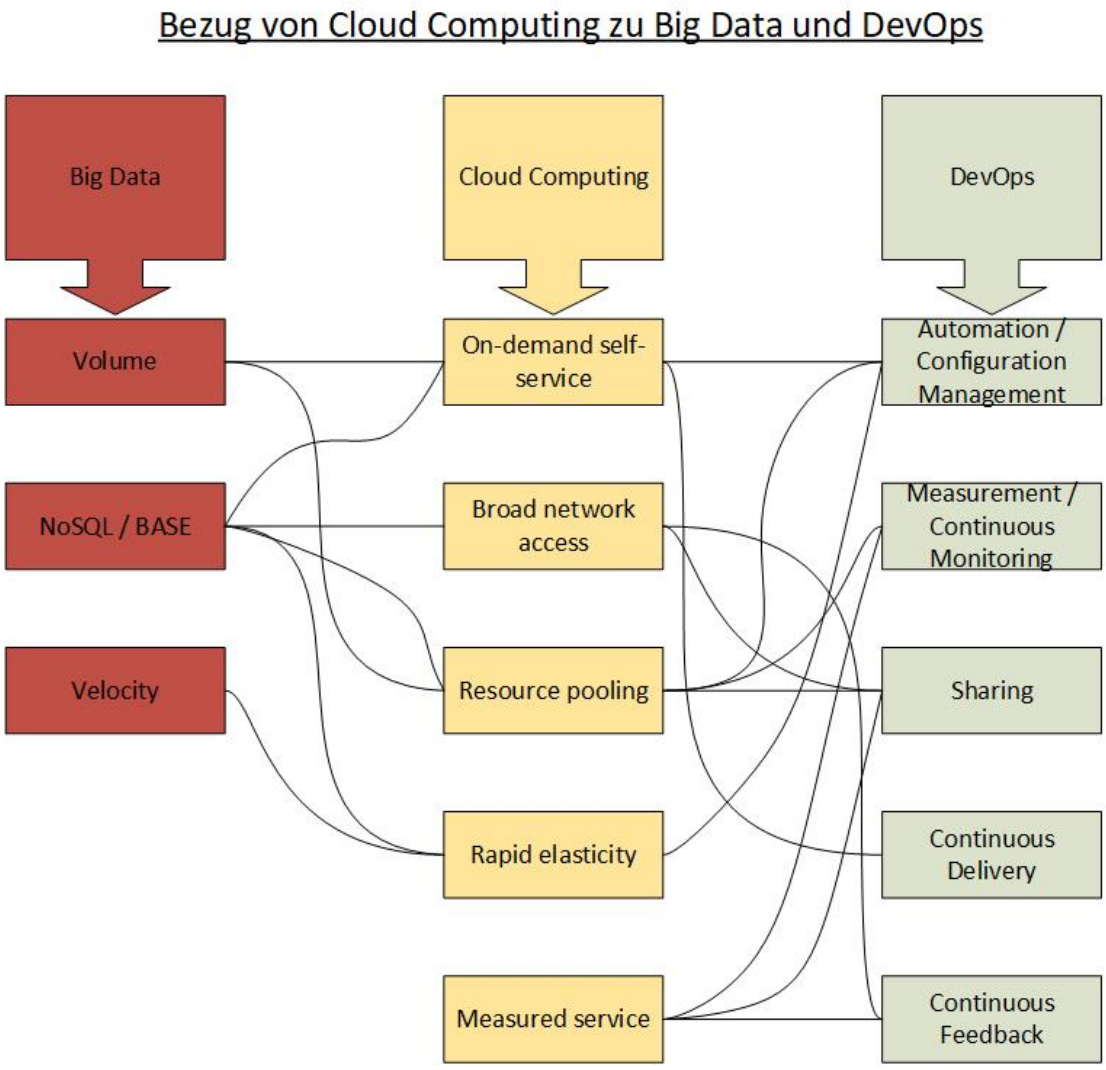


Abbildung 4.2.: Cloud Computing Relationen

| **Cloud Computing** | On-Demand Self-Service | Broad Network Access | Resource Pooling | Rapid Elasticity | Measured Service |
|---|---|---|---|---|---|
| **Big Data** | | | | | |
| Volume | x | | x | | |
| Velocity | | | | x | |
| NoSQL / BASE | x | x | x | x | |
| **DevOps** | | | | | |
| Automation | x | | x | x | |
| Measurement | | | x | | x |
| Sharing | | x | | | |

Tabelle 4.1.: Cloud Computing Relationen

### 4.5.1. Cloud Computing und Big Data

Cloud Computing ist vorteilhaft für die Umsetzung von Big Data-Anwendungen. Mit Big Data Anwendungen sind Systeme gemeint, die dabei helfen, die Phasen des Big Data-Lebenszyklus abzubilden. In den meisten Fällen werden zur Speicherung und Verarbeitung von großen Datenmengen NoSQL-Systeme verwendet.

Bei NoSQL-Systemen handelt es sich um schlanke, d.h. ressourcenschonende Systeme, die auf horizontale Skalierung ausgelegt sind. Aufgrund dieser Eigenschaften profitiert der Big Data-Kontext von den Cloud Computing-Eigenschaften *On-Demand Self Service*, *Ressource Pooling* und *Rapid Elasticity*, da es einerseits möglich ist, NoSQL-Systeme auf einer Cloud Computing-Umgebung bedarfsorientiert bereitzustellen und andererseits die bereitgestellten Anwendungen durch Skalierung schnell und dynamisch innerhalb einer Gruppe von Ressourcen anzupassen. Dadurch verkürzt sich auf der eien Seite die Dauer des Bereitstellungsprozesses von Informationen, was der Big Data-Anforderung *Velocity* zugutekommt, und auf der anderen Seite lässt sich aufgrund des - laut Definition - scheinbar unendlichen Vorrats an Ressourcen einer Anwendung beliebig viel Speicher zuordnen, wodurch *Volume* ebenfalls begünstigt wird. Ein weiterer Vorteil für NoSQL-Systeme ist die Eigenschaft des *Broad Network Access*, da hierdurch die Vorgabe *Basically Available* aus dem BASE-Theorem begünstigt wird.

### 4.5.2. Cloud Computing und DevOps

Cloud Computing ist eine gute Basis für den Aufbau von DevOps-Infrastrukturen. Dabei sind die drei Cloud Computing-Eigenschaften *On-Demand Self Service*, *Ressource Pooling* und *Rapid Elasticity* gute Voraussetzungen, um die nachhaltige Automatisierung und Bereitstellung von Continuous Delivery Pipelines zu ermöglichen, da beliebige Anwendungen und Services schnell und einfach bereitgestellt und kombiniert werden können.

Die Eigenschaft *Measured Service* bietet die Möglichkeit, Systeme und Applikationen in Echtzeit zu überwachen, was im DevOps-Kontext *Continuous Monitoring* zugutekommt. Die aus der Messung resultierenden Daten geben einen Überblick über den Status der Systeme, auf denen die Anwendungen betrieben werden, woraus sich Feedback ableiten lässt. Die Möglichkeit der hohen Erreichbarkeit von Cloud Computing-Systemen ist maßgeblich für die Zusammenarbeit und ist damit eine gute Voraussetzung für die DevOps-Anforderung *Sharing*.

# 5. Konzeption

In diesem Kapitel wird basierend auf den Erkenntnissen der Abschnitte 3.5 und 4.5 ein Architekturkonzept für die Bereitstellung einer Big Data-Anwendung in einer agilen Projektumgebung ausgearbeitet. Dafür wird in Abschnitt 5.1 zunächst ein Beispielszenario mit einer Beispielapplikation vorgestellt und es werden die Anforderungen an die Architektur spezifiziert. In Abschnitt 5.2 folgt die technische Konzeption der Architektur.

## 5.1. Analyse

Um ein technisches Konzept zu entwerfen, ist ein geeignetes Beispielszenario erforderlich. Es hilft, die fachlichen Aspekte zu erfassen und funktionale und technische Anforderungen zu ermitteln. Das Beispielszenario wird in die Realisierung und Evaluation einfließen. Das Beispielszenario wurde entwickelt, um einen praktischen Bezug zur Realität herzustellen.

### 5.1.1. Beispielszenario

Zunächst wird das Beispielszenario kurz beschrieben. Anschließend folgen eine Erläuterung der verwendeten Quelldaten, der Säuberung der Rohdaten, zu welchem Zweck die Daten gespeichert werden und eine Veranschaulichung der Architektur der Beispielanwendung.

#### 5.1.1.1. Kurzbeschreibung

Die Deutsche Bahn AG betreibt in einigen deutschen Großstädten einen Bike-Sharing-Service namens *Call a Bike* (vgl. Deutsche Bahn AG, 2019). Dazu gehören z.B. Hamburg, Berlin und München. An Stationen, die an verschiedenen Orten in den Städten aufgebaut wurden, werden Fahrräder für den Leihbetrieb zur Verfügung gestellt. Benutzer können an jeder Station Fahrräder ausleihen und auch wieder abgeben. Das ermöglicht den Benutzern, Fahrräder zu buchen und damit zwischen den Stationen zu pendeln. Die Buchungsdaten werden dabei erfasst und gespeichert.

Damit die Buchungsdaten analysiert werden können, müssen diese zunächst heruntergeladen und in ihrer Rohform in einem zentralen Speicher abgelegt werden. Relevante Daten werden anschließend extrahiert und in einer Document-Database gespeichert. Für die Analyse der Buchungsdaten wurde ein Prototyp einer Applikation namens *Data Analyzer*, welche die Daten nach bestimmten Kriterien aus der Document-Database ausliest und weiterverarbeitet, geschrieben. Die Ergebnisse der Analyse werden für die Visualisierung und Weiterverarbeitung in leichtgewichtigen Key-Value-Stores gespeichert. Da die Applikation stetig weiterentwickelt und um Funktionen erweitert werden soll, müssen kurze und zuverlässige Bereitstellungs- und Freigabeprozesse ermöglicht werden.

#### 5.1.1.2. Quelldaten

Die Deutsche Bahn AG betreibt ein eigenes Open-Data-Portal (vgl. Deutsche Bahn AG, 2019), mit dem sie einen stetig wachsenden Teil ihrer Mobiltäts- und Infrastrukturdaten in verschiedenen Formaten für die Weiterverarbeitung, bereitstellt. Darunter befinden sich auch die Buchungsdatensätze des Call a Bike-Dienstes (vgl. Deutsche Bahn AG, 2016), welche in Form von CSV-Dateien (Comma-separated values), einem Text-basiertem Format zur Speicherung von tabellarischen Daten, bereitgestellt werden. Die verwendete Datei *Buchungen Call a Bike (Stand 07/2016)* hat eine Größe von ungefähr 6,1 Gigabyte und enthält 12,7 Millionen Zeilen, sowie 47 Spalten.

#### 5.1.1.3. Relevante Daten

Zu den relevanten Daten gehören die Buchungen von Fahrrädern in der Stadt Hamburg. Die CSV-Tabelle ist aufgrund ihrer Dimensionen sehr unübersichtlich und enthält viele Daten, die für die Analyse nicht benötigt werden. Aus diesem Grund werden irrelevante Spalten herausgefiltert. Tabelle 5.1 zeigt die Datenstruktur, mit der in der Document-Database weitergearbeitet wird. Buchungen mit gleicher Start-Station und Endstation werden ebenfalls herausgefiltert, weil diese nicht als Reise klassifiziert werden.

| Spalte | Beschreibung |
|---|---|
| _id | Identifikationsnummer der Buchung |
| DATE_FROM | Startzeitpunkt der Buchung |
| DATE_UNTIL | Endzeitpunkt der Buchung |
| DATE_BOOKED | Buchungstag |
| START_RENTAL_ZONE | Start-Station |
| END_RENTAL_ZONE | Endstation |

Tabelle 5.1.: Relevante Spalten aus den Buchungsdatensätzen

Um die Daten weiter einzugrenzen, werden die Datensätze nach *VEHICLE_MODEL_TYPE=="Fahrrad"* und *CITY_RENTAL_ZONE=="Hamburg"* selektiert und anschließend in die Document-Database importiert.

Der *Data Analyzer* ermittelt die Anzahl der *Buchungen pro Tag* und die *Durchschnittliche Tripdauer pro Tag*. Die Ergebnisse jeder Abfrage werden in einem eigenen Key-Value-Store abgespeichert. Hier können die Daten für die Visualisierung durch andere Applikationen abgerufen und für die Weiterverarbeitung verwendet werden. Tabelle 5.2 zeigt die Datenstruktur der Ergebnisse.

| Abfrage | Datenstruktur (Key\|Value) |
|---|---|
| Buchungen pro Tag | Datum\|Anzahl |
| Durchschnittliche Tripdauer pro Tag | Datum\|AVG(Tripdauer) |

Tabelle 5.2.: Datenstruktur der Ergebnisse

#### 5.1.1.4. Anwendungsarchitektur

Abbildung 5.1 zeigt die Systemarchitektur der Anwendung. Die erhobenen Daten liegen für die Säuberung in einem Dateisystem. Von dort werden die Daten in eine Document-Database, vorzugsweise MongoDB, importiert. Der *Data Collector* repräsentiert den Import in die MongoDB.
MongoDB ist eine hochverfügbare und skalierbare Document-Database (MongoDB, Inc, 2019), welche Daten im JSON-Format abspeichert und eine API bietet, wodurch die Anbindung von Applikationen sehr einfach ist. Von hier liest der *Data Analyzer*, bei dem es sich um eine Java-

Applikation handelt, die Daten aus und speichert die Ergebnisse in Redis-Key-Value-Stores. Redis bietet ebenfalls eine API für den Zugriff.

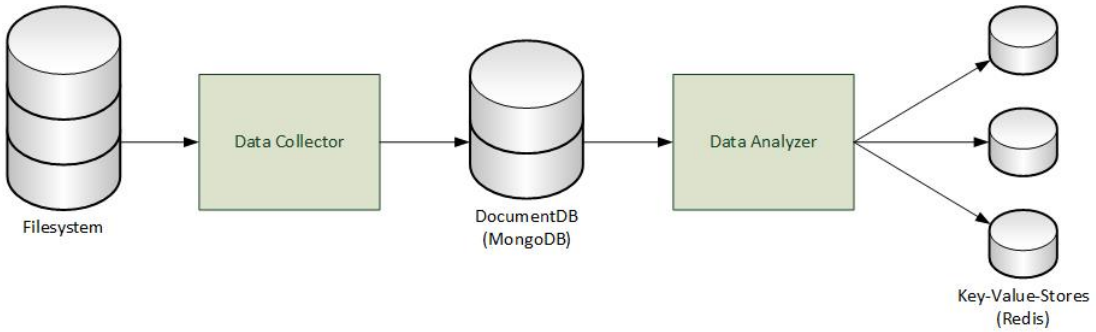


Abbildung 5.1.: Systemarchitektur der Anwendung

Abbildung 5.2 ist eine Darstellung des *Data Analyzer* in Form eines UML-Klassendiagramms.

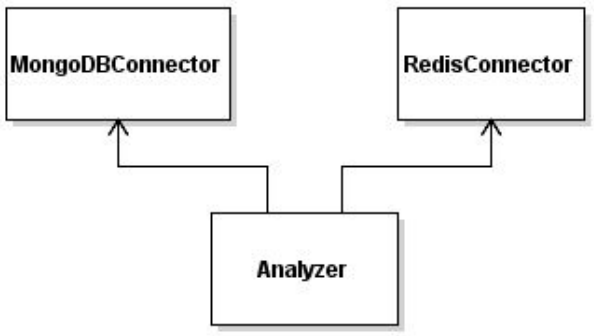


Abbildung 5.2.: UML-Klassendiagramm Data Analyzer

#### 5.1.1.5. Bewertung des Beispielszenarios

Das Beispielszenario deckt einen großen Teil des Big Data-Lebenszyklus ab. Es werden Daten erhoben (*Acquire*). Relevante Daten werden als Teilmenge der erhobenen Daten für die Analyse vorbereitet und abgespeichert (*Prepare*). Nach der Analyse (*Analyze*) der relevanten Daten werden die Ergebnisse abgespeichert (*Preserve*) und für die Visualisierung zugänglich gemacht. Auf die Planung (*Plan*) und Entdeckung von Informationen (*Discover*) wird nicht eingegangen, weil der Fokus auf der technischen Umsetzung liegt und diese beiden Phasen keinen direkten technischen Hintergrund haben.

Das Beispielszenario ist grundsätzlich übertragbar auf ähnliche Szenarien, bei denen die Entwicklung und Bereitstellung einer Big Data-Architektur mit Hilfe von DevOps-Konzepten unterstützt werden soll, jedoch wird im Rahmen dieser Arbeit nur Bezug auf die Gegebenheiten des vorgestellten Szenarios eingegangen.

### 5.1.2. Anforderungen

Ein Architekturkonzept für die Bereitstellung von Big Data-Anwendungen auf voll automatisierten Infrastrukturen, ist für alle, die große und sich stetig ändernde Datenmengen verarbeiten und analysieren wollen, interessant. Der Fokus des Konzepts liegt in der Integration einer Anwendung (z.B. *Data Analyzer* aus dem Beispielszenario) in eine agile Projektumgebung, die Gebrauch von Werkzeugen und Konzepten aus dem DevOps-Kontext macht.

#### 5.1.2.1. Anforderungen durch Big Data

Bei der Entwicklung und Bereitstellung von Big Data-Anwendungen muss auf die Anforderungen, die sich aus den Eigenschaften des V-Definitionsansatzes (vgl. Unterabschnitt 2.1.2) ableiten lassen, geachtet werden:

**Volume:** Um Große Datenbestände speichern zu können, müssen genug Speicherressourcen zur Verfügung stehen. Für die Verwaltung und den Zugriff auf die Daten müssen zusätzlich Datenbanken bereitgestellt werden. Damit hat *Volume* nur technische Anforderungen an die Architektur.

**Velocity:** Die Bearbeitung der Daten und Bereitstellung der Ergebnisse müssen schnell durchgeführt werden können. *Velocity* stellt damit grundlegend eine nicht-funktionale Anforderung dar, da sie ein Zeichen für die Ausführungsqualität einer Big Data-Anwendung darstellt. Für die Ausführung von Anwendungen, die große Datenbestände analysieren und verarbeiten sollen, müssen entsprechende Laufzeitumgebungen bereitgestellt werden, die untereinander und mit Datenbanken kommunizieren können.

Die übrigen Eigenschaften des V-Definitionsansatzes helfen bei der Spezifizierung der Anforderungen nicht und werden deshalb nicht weiter in Betracht gezogen.
Weitere Anforderungen ergeben sich bei der Verwendung von NoSQL-Datenbanken (vgl. Unterabschnitt 2.3.2), welche nach dem *BASE*-Konsistenzmodell arbeiten. Diese Datenbanken sind auf eine hohe Erreichbarkeit und Skalierbarkeit ausgelegt.

#### 5.1.2.2. Anforderungen durch DevOps

Die Architektur soll die Anwendung von agilen Methoden ermöglichen, um die Flexibilität bei der Entwicklung und der Bereitstellung zu erhöhen. Die Architektur muss dadurch den Betrieb eines schnellen Auslieferungsprozesses ermöglichen, der die Implementation und Bereitstellung von Änderungen zu jeder Zeit zuverlässig und fehlerfrei ermöglicht. Mit Hilfe der Eigenschaften des CAMS-Definitionsansatzes (vgl. Abschnitt 3.1) lassen sich weitere Anforderungen spezifizieren:

**Culture:** Die Umgebung der Architektur bilden, neben der Hardware, Menschen und Prozesse, welche durch die Verwendung von agilen Methoden kurze Entwicklungszyklen anstreben.

**Automation:** Der Auslieferungsprozess soll komplett automatisiert werden. Zwischen dem Einchecken einer Änderung und ihrer Bereitstellung in der Produktivumgebung sollen möglichst keine manuellen Schritte durchgeführt werden. Die Bereitstellung der entsprechenden Systeme soll ebenfalls automatisiert werden.

**Measurement:** Die Architektur muss in der Lage sein, sich selber zu überwachen. Es sollen eigenständig System- und Event-Daten gesammelt werden, aus denen Informationen über den Zustand der Plattformen und Anwendungen gesammelt und eingesehen werden können.

**Sharing:** Die Architektur soll zu jeder Zeit für alle Beteiligten erreichbar sein. Dabei sollen die Quelldaten und Quelltexte der Anwendungen und die Ergebnisse der Anwendungen und Messungen allen, die Interesse an den Daten und Ergebnissen haben, zur Verfügung stehen.

#### 5.1.2.3. Anforderungsprofil an die Architektur

Auf Basis der Abschnitte 5.1.1, 5.1.2.1 und 5.1.2.2 wird nun das Anforderungsprofil spezifiziert und in Tabelle 5.3 festgehalten:

| ID | Beschreibung |
|---|---|
| BD-1 | Speicherung großer Datenmengen |
| BD-2 | Verwaltung von Systemressourcen (Speicher, CPU, etc.) |
| BD-3 | Bereitstellung von Datenbanken |
| BD-4 | Bereitstellung von Laufzeitumgebungen |
| BD-5 | Schnelle Bereitstellung von Datenbanken & Laufzeitumgebungen |
| BD-6 | Beliebiges Kombinieren von Datenbanken & Laufzeitumgebungen |
| BD-7 | Hohe Verfügbarkeit der Systeme |
| BD-8 | Beliebige Skalierung der Systeme |
| DO-1 | Unterstützung von agilen Methoden |
| DO-2 | Automatisierung der Auslieferungsprozesse |
| DO-3 | Bereitstellung durch zentrale Infrastrukturen |
| DO-4 | Überwachung aller Systemressourcen |
| DO-5 | Reproduzierbarkeit der Infrastrukturen |

Tabelle 5.3.: Anforderungsprofil an die Architektur

Das Anforderungsprofil ist eine Zusammenfassung der Anforderungen, die sich aus dem Big Data- (*BD-**) und dem DevOps-Kontext (*DO-**) gebildet haben. Der bedarfsorientierte Aufbau von Big Data-Infrastrukturen soll ermöglicht werden. Es ist nötig, dass Ressourcen für das Speichern von Daten und Ausführen von Plattformen und Anwendungen, bereitgestellt, skaliert und eigenständig verwaltet werden können.
Bei der Entwicklung und Bereitstellung soll das Arbeiten mit modernen Praktiken aus dem Software-Engineering ermöglicht werden. Die Architektur soll die Zusammenarbeit fördern, indem sie für alle Beteiligten gleichermaßen erreichbar ist und alle Daten, die aus Überwachung, Feedback und Analysen entstanden sind, verfügbar machen.

### 5.1.3. Einschätzung des Anforderungsprofils

Viele der Anforderungen können durch die Eigenschaften des *Cloud Computings* (vgl. Abschnitt 4.2) abgedeckt werden. Aus diesem Grund wird für den Entwurf der Architektur eine Cloud-Infrastruktur als Basis definiert.

## 5.2. Entwurf

In diesem Abschnitt werden für die Erfüllung der Anforderungen aus Abschnitt 5.1.2 das Architekturkonzept und die einzelnen Komponenten und deren Aufgaben vorgestellt. Dabei wird die Architektur eines möglichen Produktivsystems und die Integration des in Abschnitt 5.1.1 vorgestellten Prototypen erläutert.

### 5.2.1. Bereitstellung der Systeme

In diesem Abschnitt werden die Technologien für die Bereitstellung der Systeme vorgestellt. Zunächst wird auf die Container-Technologie *Docker* eingegangen. Aufbauend darauf wird *Kubernetes* als Plattform für Orchestrierung von Container-Hosts und damit als Basis für eine Cloud-Infrastruktur vorgestellt. Anschließend wird, aufbauend auf den Erkenntnissen über *Continuous Deployment*, Gitlab als Plattform für die Projektverwaltung und den Aufbau einer *Continuous Delivery Pipeline* eingeführt. Wichtigstes Kriterium ist, dass alle Komponenten, die verwendet werden, unter einer Open-Source-Lizenz geführt werden, weil dadurch keine Beschaffungskosten anfallen.

#### 5.2.1.1. Docker

Bei Docker handelt es sich um eine Technologie für die Erstellung und Nutzung von *Linux-Containern* (vgl. Red Hat Limited, 2019). Linux-Container sind eine sehr leichtgewichtige Form der Anwendungsvirtualisierung. Mit Hilfe von Kernel-Funktionen, wie *namespaces* und *cgroups*, werden isolierte Instanzen der Systemressourcen, wie CPU, Speicher, Netzwerkadapter und Dateisysteme, erstellt. Prozesse können innerhalb dieser isolierten Ressourcen gestartet und ausgeführt werden. Durch die Isolation erhalten Prozesse ihre eigene Laufzeitumgebung.

Docker erweitert Linux-Container um ein Image-basiertes Bereitstellungsmodell. Bei einem *Docker-Image* handelt es sich um ein persistiertes und Portables Abbild eines Dateisystems. Die Anweisungen für den Aufbau und den Inhalt eines Images werden in einer Text-basierten Konfigurationsdatei, auch *Dockerfile* genannt, spezifiziert. Mit Hilfe von *Docker-Registries* können Images in zentralen Ablagen gespeichert und geteilt werden. Bei dem Start eines Containers wird das Image in die isolierte Umgebung geladen und die in dem Dockerfile hinterlegten Prozessaufrufe gestartet. Docker-Images sind sehr leichtgewichtig, weil sie nur die nötigsten Dateien für die Ausführung einer Anwendung enthalten und somit die Portabilität einer Anwendung erhöhen.

Durch den Gebrauch der Kernel-Funktionen für die Isolation der Anwendungen, verbrauchen Container im Vergleich zu virtuellen Maschinen weniger Systemressourcen. Virtuelle Maschinen benötigen für die Ausführung von Anwendungen ein eigenes Betriebssystem und einen eigenen Kernel, was bei der Ausführung von Containern entfällt. Container ermöglichen durch ihre schlanke Architektur eine viel schnellere und einfachere Anwendungsbereitstellung als virtuelle Maschinen. Für einen Vergleich werden die beiden Konzepte in Abbildung 5.3 dargestellt.

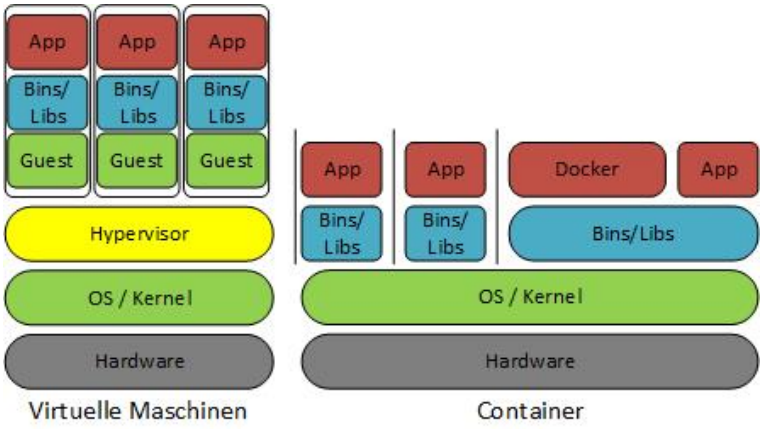


Abbildung 5.3.: Vergleich Virtuelle Maschinen gegenüber Containern

#### 5.2.1.2. Kubernetes

Kubernetes ist eine portable und erweiterbare Plattform für die Verwaltung und Bereitstellung von Container-Applikationen (vgl. Kubernetes.io, 2019b). Dabei nutzt Kubernetes Container-Technologien, typischerweise Docker, für die Ausführung der Anwendungen. Aufgrund der Eigenschaften von Containern, die sie leichtgewichtig und portabel machen, ist Kubernetes in der Lage, Container dynamisch innerhalb eines Clusters schnell bereitzustellen und zu skalieren.

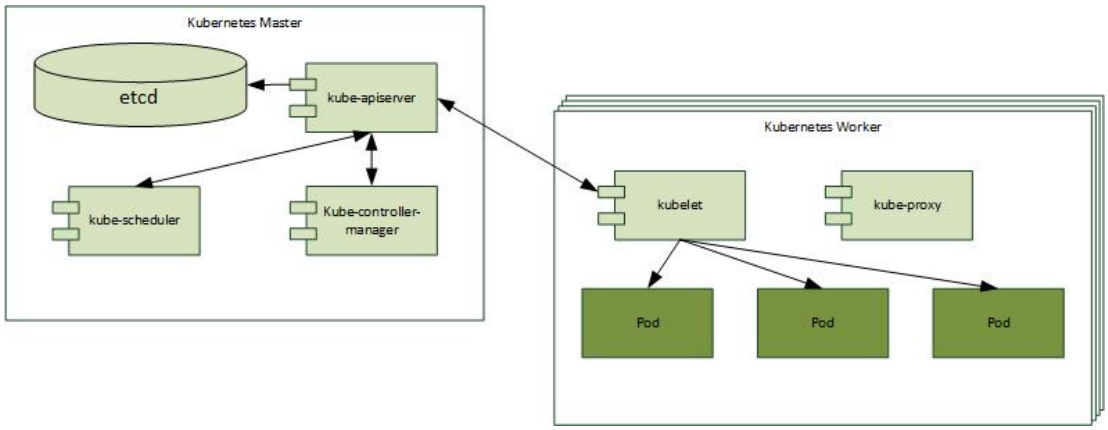


Abbildung 5.4.: Bestandteile eines Kubernetes Clusters
Quelle: Übernommen und abgeändert aus Lukša, 2018, S.21

Ein Kubernetes-Cluster setzt sich aus *Master-* und *Arbeitsknoten* zusammen. Der Master-Knoten hat die Aufgabe, die Cluster-Ressourcen global zu verwalten und die Arbeitsknoten stellen die Hardware für die Ausführung der Anwendungen bereit. Kubernetes arbeitet mit Abstraktionen namens *API-Objects*, mit denen der gewünschte Zustand eines Clusters beschrieben wird. Das ermöglicht, Anwendungssysteme deklarativ zu beschreiben, zu konfigurieren und logisch voneinander zu trennen (vgl. Lukša, 2018, S.21f.).

Der *kubeapi-server* dient dabei als zentrale Schnittstelle, um dem Cluster Änderungen an der Konfiguration mitzuteilen und um den anderen Komponenten alle Cluster-Einstellungen, welche in einem *etcd*-Key-Value-Store gespeichert sind, bereitzustellen. Der *kube-scheduler* sorgt dafür, dass die Anwendungscontainer auf den richtigen Arbeitsknoten gestartet werden. Wenn für einen Anwendungscontainer nicht explizit definiert wird, auf welchem Arbeitsknoten er ausgeführt werden soll, sorgt der kube-scheduler dafür, dass der Anwendungscontainer auf dem Arbeitsknoten mit der geringsten Auslastung gestartet wird. Die Arbeitsknoten kommunizieren mit Hilfe des *kublets* mit dem Master und übertragen ihren Zustand und den der Anwendungscontainer. Darüber hinaus sorgt der *kube-proxy* dafür, dass der Netzwerkverkehr an die richtigen Pods geleitet wird (vgl. Kubernetes.io, 2019b).

Im Kubernetes-Objekt-Modell stellt ein *Pod* den kleinsten Baustein dar. Bei einem Pod handelt es sich um die Kapselung eines oder mehrerer Container. Er stellt eng gekoppelten Anwendungscontainern einen gemeinsamen Kontext, der auch als virtueller Host betrachtet werden

kann, bereit. Jeder Pod erhält im Cluster eine eigene interne IP-Adresse, was den Anwendungscontainern ermöglicht, über Pods hinaus zu kommunizieren. Pods haben einen eigenen Lebenszyklus. Sie werden solange am Laufen erhalten, solange dies der gewünschte Zustand des Clusters vorschreibt. Wenn ein Pod ausfällt, z.B. weil ein Arbeitsknoten seinen Dienst quittiert, wird der Pod auf einem anderen Arbeitsknoten gestartet, um so die Erreichbarkeit des Dienstes aufrechtzuerhalten.

Neben den Pods gibt es noch weitere Abstraktionen. Dazu gehören *Services*, mit denen sich Pods zu logischen Gruppen zusammenfassen, wobei der Service dann eine IP-Adresse erhält, über die alle zugewiesenen Pods der Gruppe zu erreichen sind. Weitere Abstraktionen sind *Deployments*, *StatefulSets* und *DaemonSets*, mit denen Pods, deren Replikanzahl und die zugewiesenen Arbeitsknoten definiert werden. *Namespaces* (nicht zu verwechseln mit den namespaces der Linux-Conatiner) helfen dabei, den Cluster in logische Bereiche aufzuteilen. Pods in unterschiedlichen namespaces können nicht direkt miteinander kommunizieren. Die Kommunikation kann über entsprechende Services ermöglicht werden. Um Anwendungen Persistenzspeicher bereitzustellen, arbeitet Kubernetes mit *Volume*-Objekten, auch *Persistent Volumes* genannt. Mit Hilfe dieser Abstraktion werden Objektspeicher und Netzwerkdateisysteme den Pods zugänglich gemacht (vgl. Kubernetes.io, 2019b).

Kubernetes erfüllt einige der Eigenschaften von *Cloud Computing* und ist damit die technische Grundlage für die Infrastruktur:

**Ressource Pooling:** Durch die Architektur und die Abstraktionen von Kubernetes, werden Ressourcen zu einem gemeinsamen Vorrat zusammengefasst. Diese Abstraktionen erlauben dynamische und bedarfsorientierte Zuweisung von Ressourcen ohne die Hardware zu kennen.

**On-Demand self-service:** Der kube-api-server, als zentrale Schnittstelle des Clusters, ermöglicht mit Hilfe der API-Objekte die Konfiguration der Dienste. Kubernetes stellt dafür ein eigenes Werkzeug zur Verfügung, die *kubectl*. Dabei handelt es sich um ein Werkzeug, mit dem sich Konfigurationen an den kube-api-server übertragen und Zustände abfragen lassen. Es ist außerdem möglich, eigene Client-Applikationen an den kubeapi-server anzubinden.

**Broad network access:** Kubernetes selber ermöglicht keinen breiten Zugriff auf den Cluster. Die Infrastruktur, auf der Kubernetes betrieben wird, bestimmt über die Zugriffsmöglichkeiten. Grundsätzlich können Dienste, die in Pods ausgeführt werden, nicht ohne

Weiteres veröffentlicht werden, jedoch verfügen viele Public Clouds inzwischen über die Funktionalität, auch Pods aus dem Internet schnell und einfach zugänglich zu machen.

**Rapid elasticity:** Die Kubernetes-Architektur ermöglicht die schnelle und dynamische Bereitstellung von Ressourcen. Dabei können Anwendungen beliebig skaliert werden und Regeln für die automatische Skalierung, z.B. bei hoher Auslastung, definiert werden.

**Measured Service:** Kubernetes verfügt über Mechanismen, die den Zustand der Arbeitsknoten und Pods überwachen. Die Stati lassen sich dann mit Hilfe der kubectl oder eines eingebauten Dashboards abrufen. Zusätzlich können Überwachungssysteme, wie z.B. Prometheus (vgl. Prometheus.io, 2019), im Cluster bereitgestellt werden. Daraus ergeben sich weitere Funktionen für die Überwachung der Systeme, wie z.B. der Versand von E-Mails bei zu hoher Auslastung.

Kubernetes bietet sich grundsätzlich für die Bereitstellung des Prototypen an, da sich die gesamte Systemarchitektur der Anwendung in einem Kubernetes Cluster abbilden lässt. Grund dafür ist, dass sich so gut wie jede Applikation in einem Docker-Image kapseln lässt. Zusätzlich gibt es öffentliche Docker-Registries, von denen vorgefertigte Images, z.B. für NoSQL-Datenbanken oder Java-Laufzeitumgebungen, heruntergeladen und zusätzlich angepasst werden können.

- Kubernetes deckt mit seinen Eigenschaften fast alle Anforderungen, die sich aus Big Data ergeben haben, ab:
- Mit Kubernetes lassen sich durch die Volume-Abstraktion beliebig große Speicherressourcen anbinden (BD-1).
- Über eine zentrale Schnittstelle (kube-apiserver) wird die Verwaltung der Systemressourcen ermöglicht (BD-2).
- Die Container-Technologie ermöglicht die schnelle Bereitstellung so gut wie aller Plattformen und Laufzeitumgebungen (BD-4, BD-5).
- Durch höhere Abstraktionen lassen sich die bereitgestellten Systeme beliebig kombinieren und logisch voneinander trennen (BD-6).
- Kubernetes ermöglicht durch Resource Pooling die beliebige Skalierung der bereitgestellten Systeme.(BD-8)

#### 5.2.1.3. Gitlab

Bei *Gitlab* handelt es sich um eine webbasierte Plattform für die verteilte Versionskontrolle. Sie ist interessant für Entwicklerteams, bei denen mehrere Personen gleichzeitig an Quelltexten arbeiten. Neben der Versionskontrolle bietet Gitlab weitere Funktionen, wie z.B. Bugtracking an, um die Kommunikation innerhalb eines Projekts zu vereinfachen. Quelltexte werden dabei in Repositories, also Ablagen für Dateien, abgespeichert. Jede Änderung an einer Datei wird aufgezeichnet.

Gitlab enthält außerdem ein eigenes Continuous Integration-System namens *Gitlab-CI*, welches die Repositories auf Änderungen überwacht und einen automatisierten Build- und Test-Prozess startet, sobald eine Änderung eintritt. Darüber hinaus bietet Gitlab eine Komponente namens *Gitlab-Runner* an, die sich in beliebige Systeme, also auch Container, installieren lässt. Der Runner führt die Schritte für die Bereitstellung aus, die in Gitlab-CI hinterlegt sind. Das ermöglicht zusätzlich Continuous Delivery bzw. Continuous Deployment (vgl. Gitlab, 2019).

Des Weiteren verfügt Gitlab über Funktionen, die dabei helfen, Projekte mit Kubernetes umzusetzen. Dazu gehört vor allem die Kubernetes Integration, die es ermöglicht, über die Gitlab-Weboberfläche Operationen am Kubernetes Cluster vorzunehmen, diesen zu überwachen, und der *Kubernetes Executor* (auch *Kubernetes Runner* genannt), um Build-, Test- und Deploy-Prozesse in einem Kubernetes-Cluster auszuführen. Gitlab bringt zusätzlich eine eigene Docker-Registry mit, wodurch, in Kombination mit den *Code Repositories*, *Configuration Management* ermöglicht wird. Die Configuration-Files, wie z.B. Dockerfiles oder die Zustandsbeschreibungen der Kubernetes-Objekte, können in den *Code Repositories* hinterlegt und zentral verfügbar gemacht werden. Außerdem deckt Gitlab mit seinem Funktionsumfang weitere Anforderungen ab:

- Gitlab verfügt über ein *Issue Board*, mit dem zentrales Projekt-Management betrieben werden kann, und über Funktionalitäten zur Anbindung an Kubernetes (DO-1).
- Mit Hilfe von Gitlab-CI können alle Auslieferungsprozesse automatisiert werden (DO-2).
- Gitlab und Kubernetes verfügen über Funktionalitäten zur Steuerung über die Zugriffe. Damit können Zugriffe nicht nur ermöglicht, sondern auch nutzerbasiert eingestellt werden (DO-3).

- Kubernetes kann durch Überwachungssysteme ergänzt und deren Ergebnisse auf der Oberfläche von Gitlab eingesehen werden (DO-4).
- Gitlab ermöglicht die Verwaltung der Configuration Files, mit denen die Infrastrukturen bereitgestellt werden (DO-5).

### 5.2.2. Systemarchitektur

Die Systemarchitektur Abbildung 5.5 gibt einen Überblick darüber, wie Gitlab und Kubernetes miteinander verbunden und verwendet werden:

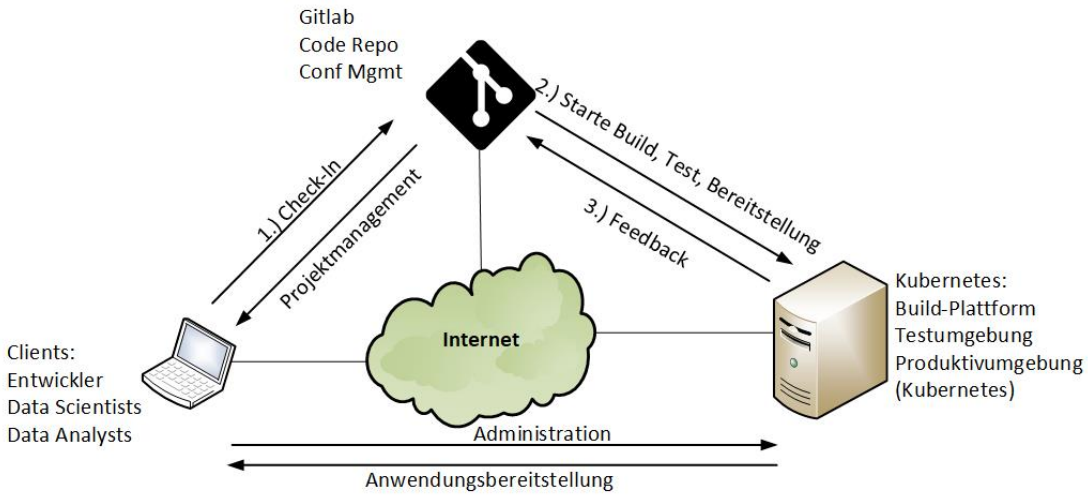


Abbildung 5.5.: Darstellung der Systemarchitektur

Entwickler und Data Scientists arbeiten in den meisten Fällen auf sehr individuell eingerichteten PCs oder Notebooks und betreiben damit ihre Entwicklungsumgebungen. Aus diesem Grund wird häufig eine zentrale Projektverwaltungsplattform verwendet, die auf Webservern basiert und für alle Projektmitarbeiter entweder aus dem Internet oder dem eigenen Netzwerk erreichbar ist. Eine *Versionskontrollplattform* ist dabei oft ein Teil des Projektmanagements und dient als zentrales Code Repository. Das Einchecken von Änderungen am Quelltext findet im Code Repository statt und ist damit der letzte Schritt, bevor das Software-Paket aus den Quelltexten gebaut wird.

Die Versionskontrolle mit ihren Code Repositories stellt in der Systemarchitektur das agile Projektmanagement dar. Durch das Einchecken einer Änderung am Quelltext soll eine An-

wendung automatisch auf der Cloud-Infrastruktur, welche ebenfalls über das Internet oder ein eigenes Netzwerk erreichbar ist, ausgeführt und bereitgestellt werden. Der Zugriff hängt von dem verwendeten Bereitstellungsmodell (vgl. Abschnitt 4.4) ab.

Da Gitlab in Kombination mit Kubernetes verwendet wird, enthält das Code-Repository auch die Funktion des Configuration-Managements. Der Vorteil dabei ist, dass die Vorlagen für die bereitgestellten Systeme außerhalb der Produktivumgebung liegen und im Falle eines Ausfalls trotzdem zentral erreichbar sind, um die Systeme auf einem anderen Cluster bereitstellen zu können.

Die drei nummerierten Pfeile stellen den Ablauf einer Änderung an einer Applikation dar. Ein Entwickler checkt die Änderung im Code-Repository ein (*1. Check-In*). Durch das Einchecken werden Build-, Test- und Deploy-Routinen auf der Cloud-Infrastruktur (Kubernetes) gestartet (*2. Starte Build, Test, Bereitstellung*). Nach der Ausführung gibt die Cloud-Infrastruktur ein Feedback über den Status der ausgeführten Routinen an Gitlab zurück (*3. Feedback*).

### 5.2.3. Big Data-Cloud Infrastruktur

Abbildung 5.6 zeigt eine schematische Darstellung der Big Data-Cloud Infrastruktur:

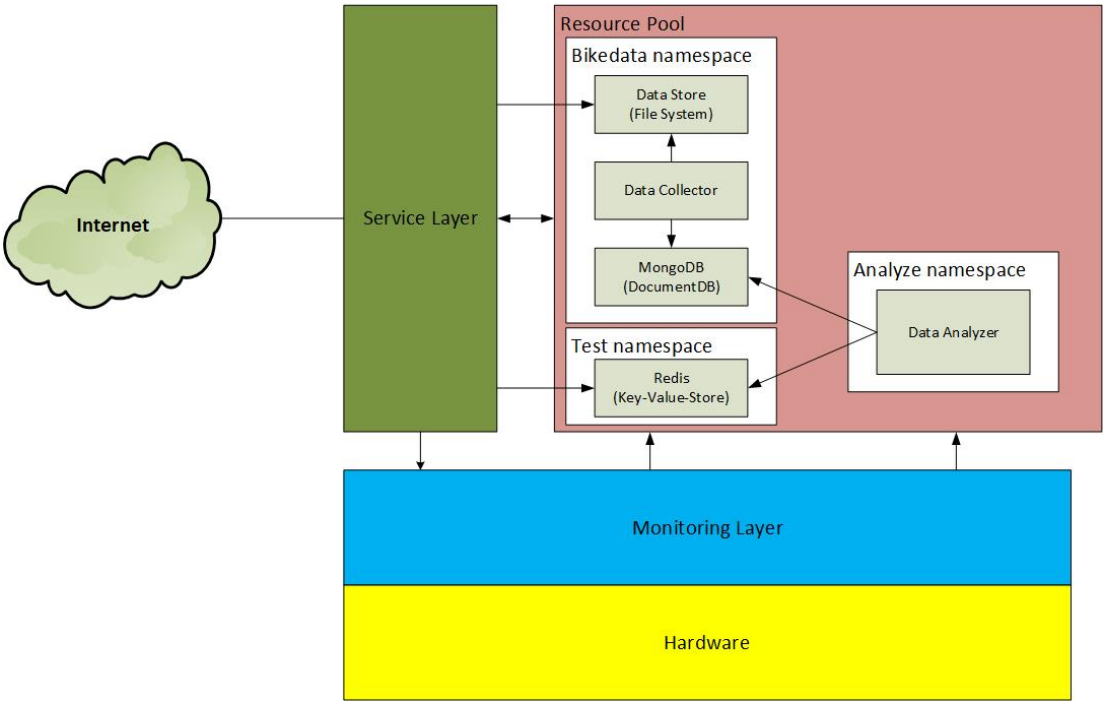


Abbildung 5.6.: Schematische Darstellung Big Data-Cloud-Infrastruktur

Der *Service Layer* ist eine Abstraktion, die alle bereitgestellten Dienste, die über das Internet verfügbar sind, beschreibt. Dazu gehören die *kubeapi*, also die Konfiguration und Verwaltung der Systeme, die Bereitstellung der Ergebnisse aus den Überwachungssystemen (Monitoring Layer) und die Dienste, welche durch die innerhalb des *Resource Pools* betriebenen Anwendungen bereitgestellt werden. Dazu gehört zum einen, die Möglichkeit Dateien auf einem Speicher (*Data Store*) zu sichern, um sie *Data Collector* zugänglich zu machen und zum anderen, die Bereitstellung der Ergebnisse aus der Berechnung des *Data Analyzer*.

Wie schon angedeutet, handelt es sich bei dem *Monitoring Layer* um eine Abstraktion, die alle Systeme zur Überwachung der Cluster-Ressourcen, beinhaltet. Sowohl die Hardware als auch der *Resource Pool* werden überwacht.

Die Abstraktion *Resource Pool* steht für die bereitgestellte Hardware und beinhaltet alle Komponenten, die innerhalb des Clusters für den Betrieb der Big-Data-Anwendung bereitgestellt werden. Außerdem zeigt die Darstellung, wie die Komponenten der Systemarchitektur der

Anwendung integriert werden. Alle Komponenten, die für die Speicherung von Daten zuständig sind, erhalten einen eigenen *namespace* namens *Bikedata*. Um Daten in den *Data Store* übertragen zu können, ist dieser an den Service Layer angebunden. Die Schnittstellen von MongoDB und Redis werden clusterintern bereitgestellt, damit Anwendungen Zugriff erhalten. Zusätzlich wird Redis an den Service Layer angebunden, um die Ergebnisse der Analyse öffentlich zugänglich zu machen und so eine möglichst hohe Verfügbarkeit zu ermöglichen. Der Data Analyzer wird seinen eigenen namespace erhalten. Durch die Trennung der Funktionalitäten innerhalb des Resource Pools, können Verantwortlichkeiten besser definiert, abgebildet und getrennt werden und die einzelnen Komponenten sind austauschbar. Außerdem können um den Data-namespace herum weitere namespaces mit anderen Software-Projekten angelegt werden und falls der Bedarf besteht, können die gespeicherten Daten für andere Anwendungsfälle erhoben werden.

### 5.2.4. Automatisierung

Die Automatisierung des Auslieferungsprozesses ist elementar für die Integration neuer Funktionen einer Big Data-Applikation. Aus diesem Grund wird eine *Deployment Pipeline* verwendet. Durch das *Configuration Management* soll jede Version der bereitgestellten Applikation gespeichert werden, um sie reproduzieren zu können.

#### 5.2.4.1. Deployment Pipeline

Die *Deployment Pipeline* ist die automatisierte Manifestation des Auslieferungsprozesses. Sie ist dafür verantwortlich, die Änderungen an der Applikation, die in der Versionskontrolle eingecheckt werden, für die Benutzer bereitzustellen. Jede Änderung an der Applikation wird die Pipeline durchlaufen, wobei eine Build-Phase, eine Test-Phase und eine Deploy-Phase durchlaufen werden. Wenn eine der Phasen durch einen Fehler unterbrochen wird, sendet die Pipeline automatisch einen entsprechenden Fehler-Code an das Code Repository und ist von da an für alle Projektmitglieder verfügbar, damit möglichst schnell Entsprechende Maßnahmen eingeleitet werden können. Auch bei einem Erfolg wird die Pipeline eine entsprechende Nachricht an das Code Repository senden.
Da es sich bei dem *Data Analyzer* um eine Routine handelt, die keine Benutzerinteraktion anbietet, wird die Deployment Pipeline nicht alle Phasen aus dem *Continuous Delivery*-Konzept (vgl. Abschnitt 3.4.2) abbilden. Es wird nur eine automatisierte Build-, Test- und Deploy-Phase geben.

**Build:** Das Software-Paket wird aus den Quelltexten compiliert und daraus wird ein Image erzeugt, welches das Software-Paket enthält. Dabei führt der Compiler, welcher die Quelltexte in Byte-Code umwandelt, statische Analysen am Quelltext durch, um mögliche Syntaxfehler aufzudecken und es werden Unit-Tests ausgeführt. Nach Abschluss wird das fertige Artefakt gespeichert und für die Test-Phase weiterverwendet.

**Test:** Das erzeugte Artefakt wird in einen Container geladen und es wird eine herunterskalierte Testumgebung für Integrationstests geladen. Für die Tests werden Beispieldaten in eine MongoDB-Datenbank geladen, die Ergebnisse der ausgeführten Funktionen in einer Redis-Datenbank gespeichert und die persistierten Ergebnisse mit erwarteten Werten verglichen. Nach Abschluss der Tests wird das Artefakt gespeichert.

**Deploy:** Das getestete Artefakt wird in die Produktivumgebung geladen und die Applikation wird ausgeführt. Nach Abschluss der Ausführung wird die Applikation beendet und die verwendeten Systemressourcen werden wieder freigegeben.

Abbildung 5.7 zeigt den Ablauf der Deployment-Pipeline in Form eines Aktivitätsdiagramms.

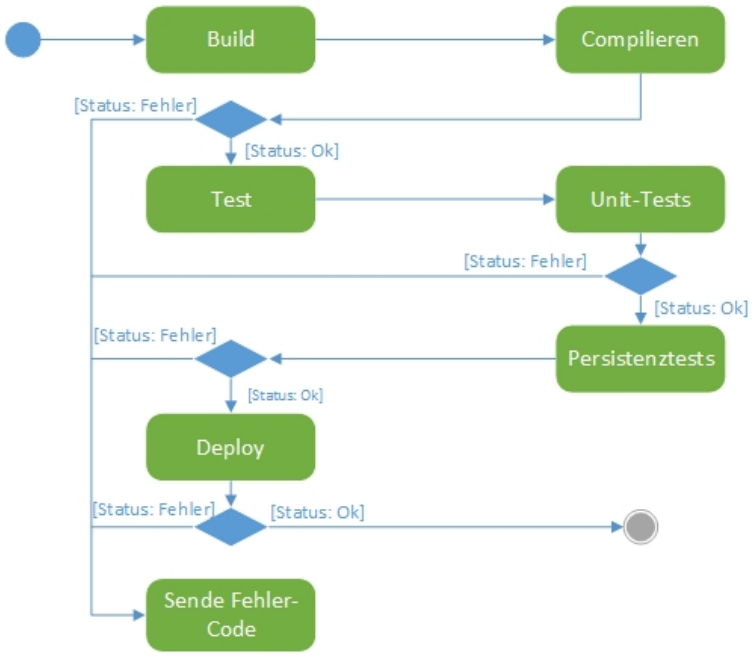


Abbildung 5.7.: Deployment Pipeline

# 6. Realisierung

In diesem Kapitel wird die prototypische Umsetzung des in Abschnitt 5.2 vorgestellten Entwurfs vorgestellt. Dadurch, dass Kubernetes verwendet wird, ist die Architektur von der darunter liegenden Hardware entkoppelt. Das bedeutet, dass die Realisierung auf unterschiedlichen Cloud Plattformen bzw. auf eigenen Systemen möglich ist. Aus diesem Grund werden zwei Möglichkeiten für die Realisierung vorgestellt. Die Realisierung wird nach dem Bottom-Up-Ansatz vorgestellt. Zunächst wird der Aufbau der physischen Infrastruktur der Private Cloud vorgestellt und im direkten Anschluss die Bereitstellung der virtuellen Infrastruktur durch eine Public Cloud, da es sich dabei um die untersten Abstraktionsebenen der beiden Realisierungsmethoden handelt. Da in beiden Fällen Kubernetes verwendet wird, ist die Bereitstellung der Big Data- und DevOps-Systeme annähernd identisch.

## 6.1. Private Cloud

Für diese Bachelorarbeit wurden vom *Big Data Lab* der Hochschule für angewandte Wissenschaften (vgl. Big Data Lab HAW, 2019) insgesamt acht Computersysteme zur Verfügung gestellt. Darunter befinden sich zwei Workstations, die aus Teilen, die gerade zur Verfügung standen, zusammengebaut wurden und sechs baugleiche Workstations. Alle Workstations haben eine SSD, auf der jeweils das Betriebssystem installiert wird. Einen Überblick über die Systemressourcen gibt Tabelle 6.1:

| **Anzahl** | **Name** | **CPU** | **RAM** | **HDD** |
|---|---|---|---|---|
| 1x | HW-Management | 4x3 GHz | 4 GB | 256 GB |
| 1x | SW-Management | 2x2,7 GHz | 4 GB | - |
| 6x | Worker | 4x3,3 GHz | 32 GB | 1 TB |

Tabelle 6.1.: Systemressourcen der Private Cloud

Die beiden *Management*-Workstations helfen dabei, die übrigen sechs *Worker*-Workstations zu verwalten, Kubernetes auf ihnen zu installieren und den Kubernetes-Cluster erreichbar

zu machen. Der Kubernetes-Cluster wird also sechs Prozessoren mit insgesamt 24x3,3 GHz, 192 GB Arbeitsspeicher und 6TB Festplattenspeicher zur Verfügung haben, was für eine Entwicklungsumgebung und die Bereitstellung der Beispielanwendung ausreicht. Der HW-Management-Server hat zusätzlich eine 256GB-Festplatte für System-Backups.

Für den Aufbau des Clusters wurde auch ein VLAN zur Verfügung gestellt. Da es zu Netzwerkproblemen und -konflikten kam, wurden alle acht Workstations jeweils mit einer zusätzlichen Netzwerkkarte ausgestattet und über einen separaten Ethernet-Switch miteinander verbunden, sodass ein abgekapseltes lokales Netzwerk zur Verfügung steht. Die Informationen über die Netzwerkkonfiguration können der Tabelle 6.2 entnommen werden.

| | **141.22.45.0/26** | **192.168.44.0/26** |
|---|---|---|
| Default Gateway | 141.22.45.1 | 192.168.44.1 |
| DNS | 141.22.192.100, 141.22.192.101 | 192.168.44.1 |
| NTP | 141.22.45.1 | 192.168.44.1 |
| DHCP | 141.22.45.4 | 192.168.44.1 |

Tabelle 6.2.: Netzwerkinformationen der Private Cloud

### 6.1.1. Hardware-Management-Server

Der HW-Management-Server übernimmt in diesem Aufbau mehrere Aufgaben. Neben der Provisionierung der übrigen sieben Workstations, übernimmt diese Maschine auch die komplette Netzwerkverwaltung für das lokale Netz. Des Weiteren werden durch diese Maschine auch eine SSH-Verbindung, Remote-Desktop-Sessions mittels xRDP (Remote Desktop Protocol) und eine Portweiterleitung auf den späteren Kubernetes-Master eingerichtet.

Für die Hardware-Verwaltung kommt *Metal as a Service* (MaaS) zum Einsatz (vgl. Canonical, 2019b). Dabei handelt es sich um ein System zur Provisionierung und Verwaltung von Computersystemen und ermöglicht es, diese Systeme wie Cloud-Ressourcen zu verwalten. MaaS ist dazu gedacht, sehr große Infrastrukturen effizient aufbauen und nutzen zu können. Die MaaS-Architektur setzt sich dabei aus *Region-Controllers* und *Rack-Controllers* zusammen. Die Region-Controller stellen dabei die Dienste für die Verwaltung der Konfigurationen bereit. Ein oder mehrere Rack-Controller werden mit dem Region-Controller verbunden. Der Rack-Controller stellt dabei die Dienste für die Provisionierung und Installation der Computersysteme, auch Nodes genannt, bereit. Auf dem HW-Management-Server werden Region- und

Rack-Controller installiert. Damit Computersysteme provisioniert und mit Betriebsysstemen bespielt werden können, müssen sie über PXE (Preboot Execution Environment) gestartet und vom Region-Controller erfasst werden. Ein MaaS-Cluster kann zu jeder Zeit um weitere Computersysteme ergänzt und skaliert werden.

### 6.1.2. Software-Management-Server

Mit dem Software-Management-Server sollen die Anwendungen, mit denen die sechs Worker-Workstations zu einem Cluster zusammengefasst werden, automatisch installiert werden. Dafür wurde *Juju* ausgewählt (vgl. Canonical, 2019a). Dabei handelt es sich um ein System, mit dem Anwendungsgebilde als Graphen modelliert und die einzelnen Anwendungen den einzelnen Computersystemen zugeordnet werden können. Jeder Knoten, auch *Charm* genannt, stellt in dem Graphen eine Anwendung dar und jede Kante eine Beziehung zwischen zwei Anwendungen. Die Modellierung wird dabei einerseits durch eine Web-Oberfläche (vgl. Abbildung 6.1) und andererseits durch Configuration Files und einer CLI ermöglicht.

Juju ist dazu gedacht, Anwendungssysteme auf Cloud-Infrastrukturen zu installieren. Für Juju gibt es Schnittstellen zu vielen der namhaften, großen Public Clouds, aber auch zu MaaS-Infrastrukturen. MaaS wurde vor allem auf Juju abgestimmt (vgl. Canonical, 2019b). Damit Juju genutzt werden kann, wird mit Hilfe von MaaS ein Juju-Controller auf den zweiten Computer installiert. Die Kombination von Juju und Maas ermöglicht zum einen die gewünschten Anwendungen mit Hilfe von Configuration Files zu spezifizieren und zum anderen diese automatisch zusammen mit den Betriebssystemen zu installieren. Dadurch entfällt der Schritt, die Betriebssysteme der Worker-Workstations manuell installieren zu müssen.

### 6.1.3. Aufbau des Clusters

Die sechs übrigen Worker-Workstations sollen zu einem Resource Pool werden. In Abschnitt 5.2.1.2 wurde Kubernetes für die Bereitstellung der Anwendungsarchitektur der Beispielanwendung ausgewählt. Dadurch sind alle Systemressourcen, außer die Festplattenspeicher von jeweils einem Terabyte, gebündelt. Da Kubernetes selber keine Daten persistieren soll, hält es Schnittstellen für viele verschiedene Persistenzprotokolle bereit. Dazu gehören auch NFS (Network File System) und diverse Speicher-Standards der Cloud Anbieter (vgl. Kubernetes.io, 2019a).

Für die Bündelung von Festplatten, die sich in unterschiedlichen Computersystemen befinden, eignet sich ein verteiltes Dateisystem. Aus diesem Grund wurde CephFS ausgewählt. Es handelt sich dabei um ein allgemein kompatibles Dateisystem, welches auf Ceph, einem Block-Device-Cluster, aufsetzt. Bei Ceph handelt es sich um ein hochverfügbares, skalierbares und verteiltes System zur Speicherung von Daten. Kubernetes kann auch mit Ceph-Block-Devices kommunizieren, jedoch lassen sich damit keine Persistent Volumes anlegen, auf denen mehrere Pods gleichzeitig schreiben und lesen dürfen (ReadWriteMany). Diese Art der Zugriffskontrolle ist für die Beispielanwendung nicht erforderlich, wurde jedoch zusätzlich für Testzwecke eingerichtet und mit Hilfe von CephFS umgesetzt. Außerdem ermöglicht CephFS, das verteilte Dateisystem in bestehende Dateisysteme mit einzubinden.

Abbildung 6.1 zeigt die verwendeten Applikationen auf der Juju-Weboberfläche.

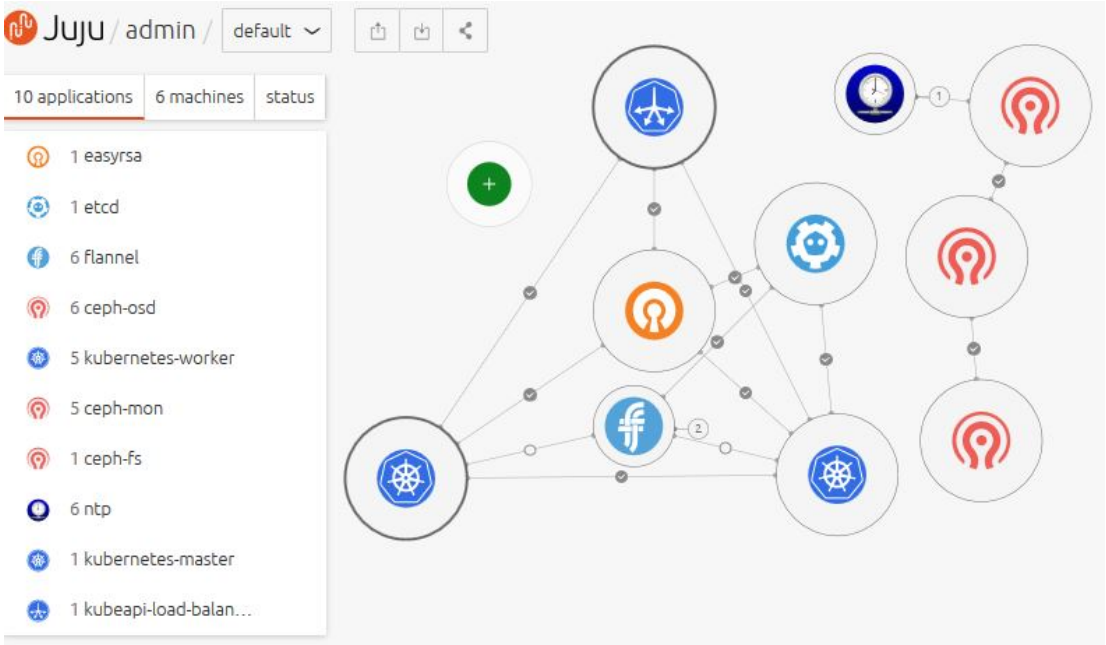


Abbildung 6.1.: Screenshot Juju-Canvas

Auf der linken Seite sind die Kubernetes-Anwendungen angesiedelt und auf der rechten Seite die von CephFS. Alle Anwendungen sind für den reibungslosen Betrieb erforderlich. Des Weiteren ist links von den Kubernetes-Charms zu erkennen, wie die einzelnen Anwendungen skaliert wurden. In Tabelle 6.3 werden die Anwendungen kurz beschrieben.

| Anwendung | Aufgabe |
|---|---|
| easyrsa | Verschlüsselung innerhalb des Kubernetes-Clusters |
| etcd | Speicherung der Kubernetes-API-Objekte |
| flannel | Kommunikation zwischen Arbeitsknoten und Pods |
| kubernetes-master | Koordination der Pods |
| kubernetes worker | Ausführung der Pods |
| kubeapi-load-balancer | Veröffentlichung der Kubernetes-API |
| ntp | Zeitsynchronisierung zwischen *Object Storage Devices* |
| ceph-osd | Physische Speicherung der Daten (*Object Storage Devices*) |
| ceph-mon | Überwachung des Ceph-Clusters (*Monitor*) |
| ceph-fs | Bereitstellung der OSD über ein Dateisystem (*Metadata Server*) |

Tabelle 6.3.: Beschreibung der Anwendungen fürs Resource Pooling

Abbildung 6.2 zeigt die Aufteilung der Applikationen auf die Maschinen. Der Master-Worker, von dem es nur einen gibt, wird als Kubernetes-Master mit allen erforderlichen Anwendungen eingerichtet. Zusätzlich wird ein Ceph-Object Storage Device installiert, um die Festplatte mit denen der anderen Workstations zu bündeln. Außerdem wird der Ceph-Meta Data Server ebenfalls auf dem Master-Worker installiert. Juju unterscheidet dabei zwischen Root Container und LXD-Container. Bei LXD handelt es sich wie bei Docker um eine Container-Technologie. Juju ist also in der Lage, Anwendungen gekapselt in Containern auszuführen.

Auf jedem Node-Worker werden ein Kubernetes-Worker, ein Ceph-Object Storage Device und ein Ceph-Monitor installiert. Diese Infrastruktur ist skalierbar und lässt sich beliebig um Master- und Node-Worker erweitern.

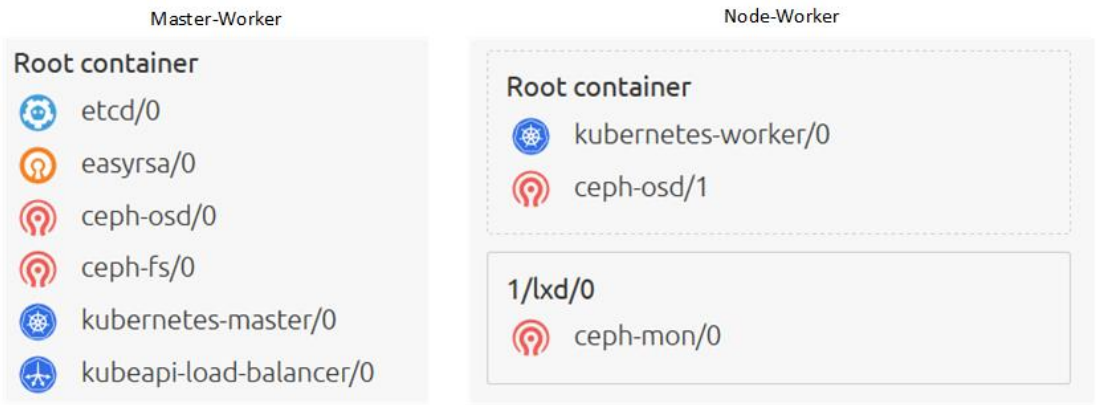


Abbildung 6.2.: Screenshot Juju-Maschinenaufteilung

Nach der Installation der Juju-Charms ist die Einrichtung noch nicht abgeschlossen. CephFS wird nicht nativ von Kubernetes unterstützt und deshalb muss zusätzlich ein CephFS-Provisioner installiert werden, damit Persistenzspeicher in CephFS angelegt und den Pods zugeordnet werden können.

Zusätzlich muss ein Load-Balancer (nicht zu verwechseln mit dem kubeapi-load-balancer) installiert werden, damit Dienste, wie z.B. die Redis-Datenbanken mit den Ergebnissen, über öffentliche, Cluster-externe IP-Adressen zugänglich gemacht werden können. Dies geschieht in Kubernetes mittels eines Service-Objekts der Klasse *LoadBalancer* (nicht zu verwechseln mit dem Load-Balander, der installiert wird, bzw. dem kubeapi-load-balancer). Ein LoadBalancer-Service-Objekt erhält von der jeweiligen Cloud-Plattform automatisch eine öffentliche IP-Adresse. Dies funktioniert aber nur, wenn der jeweilige CSP diese Funktion bereitstellt. Im Falle der Private Cloud ist diese Funktion nicht verfügbar. Für diese Aufgabe wurde *MetalLB* (vgl. Google, 2019) ausgewählt. Es handelt sich dabei um ein System, welches die Load-Balancer-Services in privaten Kubernetes-Clusters ermöglicht. Die Installation des CephFS-Provisioners und MetalLB werden mit Hilfe von Kubernetes-Configuration Files und der kubectl durchgeführt.

## 6.2. Public Cloud

Viele namhafte Unternehmen, wie z.B. Google (vgl. Google Cloud, 2019), Amazon (vgl. Amazon Web Services, 2019) , Microsoft (vgl. Microsoft Azure, 2019) und IBM (vgl. IBM Cloud, 2019) bieten mit Hilfe großer Infrastrukturen Cloud-basierte Dienste an. Dabei sind oft die gängigen Servicemodelle (IaaS, PaaS, SaaS) vertreten. Die Realisierung auf einer Public Cloud ist unproblematisch, da die Verantwortung über die Einsatzbereitschaft und Erreichbarkeit der Systemressourcen beim CSP liegt und die Konfiguration über Web-Oberflächen bzw. CLIs getätigt werden kann.

### 6.2.1. Auswahl der Cloud-Plattform

Da Kubernetes als Basis genutzt wird, muss der CSP *Infrastrucure as a Service* anbieten, um Kubernetes auf virtuellen Hosts installieren zu können. Des Weiteren soll die Umsetzung im Rahmen dieser Arbeit möglichst wenig kosten. Aus diesem Grund fiel die Entscheidung auf Google als CSP. Durch die Anmeldung wird dem Nutzerkonto ein Geldbetrag gutgeschrieben. Der Betrag reicht für die Realisierung aus. Darüber hinaus bietet die Google-Cloud-Plattform nicht nur *Infrastructure as a Service*, sondern auch *Kubernetes as a Service* (auch *Kubernetes-Engine* genannt), an. Kubernetes wurde dabei in einen Service gekapselt und kann über die Konfigurationswerkzeuge der Google-Cloud und denen von Kubernetes verwaltet werden.

### 6.2.2. Kubernetes Engine als Infrastruktur

Die Basis eines Kubernetes-Clusters bilden die zugrundeliegende Hardware- und Netzwerkinfrastruktur. Der Dienst Kubernetes-Engine abstrahiert dabei die zugrundeliegenden Ressourcen und stellt betriebsfähige Kubernetes-Clusters zur Verfügung. Für die Realisierung wird genau ein Kubernetes-Cluster auf Basis von fünf virtuellen Maschinen erstellt. Die virtuellen Maschinen haben Prozessoren mit 4 Kernen und 26 Gigabyte Arbeitsspeicher. Bei der Kubernetes-Engine wird kein zusätzlicher Kubernetes-Master installiert. Sobald ein Kubernetes-Cluster angefordert wird, ist er innerhalb von einer Minute einsatzbereit.

Um Datenspeicher zu erhalten, kann ein NFS-Volume angefordert und mit Kubernetes verbunden werden. Es sind keine weiteren Schritt erforderlich. Ab jetzt kann auch hier mit der kubectl weitergearbeitet werden.

## 6.3. Gitlab und Kubernetes

Gitlab wird hauptsächlich für die zentrale Verwaltung der Quelltexte und Configuration Files verwendet. Darüber hinaus werden die integrierte Docker-Registry und Gitlab-CI in Verbindung mit Kubernetes für die kontinuierliche Bereitstellung der gesamten Anwendungsarchitektur der Beispielanwendung verwendet. Damit diese beiden Systeme miteinander arbeiten können, müssen auf beiden Seiten jeweils ein Schlüssel erzeugt und dem anderen System übertragen werden. Ist die Anbindung der beiden Systeme abgeschlossen, können, mit Hilfe der Gitlab Benutzeroberfläche, weitere Funktionen auf dem Kubernetes-Cluster bereitgestellt werden. Darunter befindet sich auch der *Gitlab-Runner*, der die Prozeduren, die in Gitlab-CI spezifiziert werden, ausführt. Zu diesen Prozeduren gehören alle Kommandos, die durch eine weitere Configuration File spezifiziert wurden. Es entsteht eine Symbiose. Kubernetes stellt die Infrastruktur für leichtgewichtige, gekapselte Container bereit, was den Betrieb der Deployment-Pipeline erst ermöglicht, und Gitlab wiederum stellt mit der Docker-Registry die Container-Images für Kubernetes bereit.

## 6.4. Big Data-Anwendungsumgebung in Kubernetes

Auf Basis der Big Data-Cloud-Infrastruktur (vgl. Unterabschnitt 5.2.3) und mit Hilfe der Abstraktionen von Kubernetes, wird die in Abbildung 6.3 dargestellte Architektur im Kubernetes-Cluster aufgebaut, um die Anwendungsarchitektur der Beispielanwendung abzubilden.

Die Darstellung zeigt, wie die einzelnen Komponenten miteinander verbunden wurden. Die Verbindungen zwischen den Komponenten haben unterschiedliche Bedeutungen. Verbindungen ohne Pfeile stehen für direkte Beziehungen zwischen zwei Abstraktionen. Z.B. ist die Service-Abstraktion *Service MongoDB* die Schnittstelle, mit der andere Pods und Zugriff auf die MongoDB-Instanzen innerhalb der StatefulSet-Abstraktion *StatefulSet MongoDB Cluster* erhalten. Repräsentativ für Zugriffe bzw. die Beanspruchung dieser Services sind die Pfeile. Sie zeigen, welche Komponente, welchen Dienst in Anspruch nimmt.

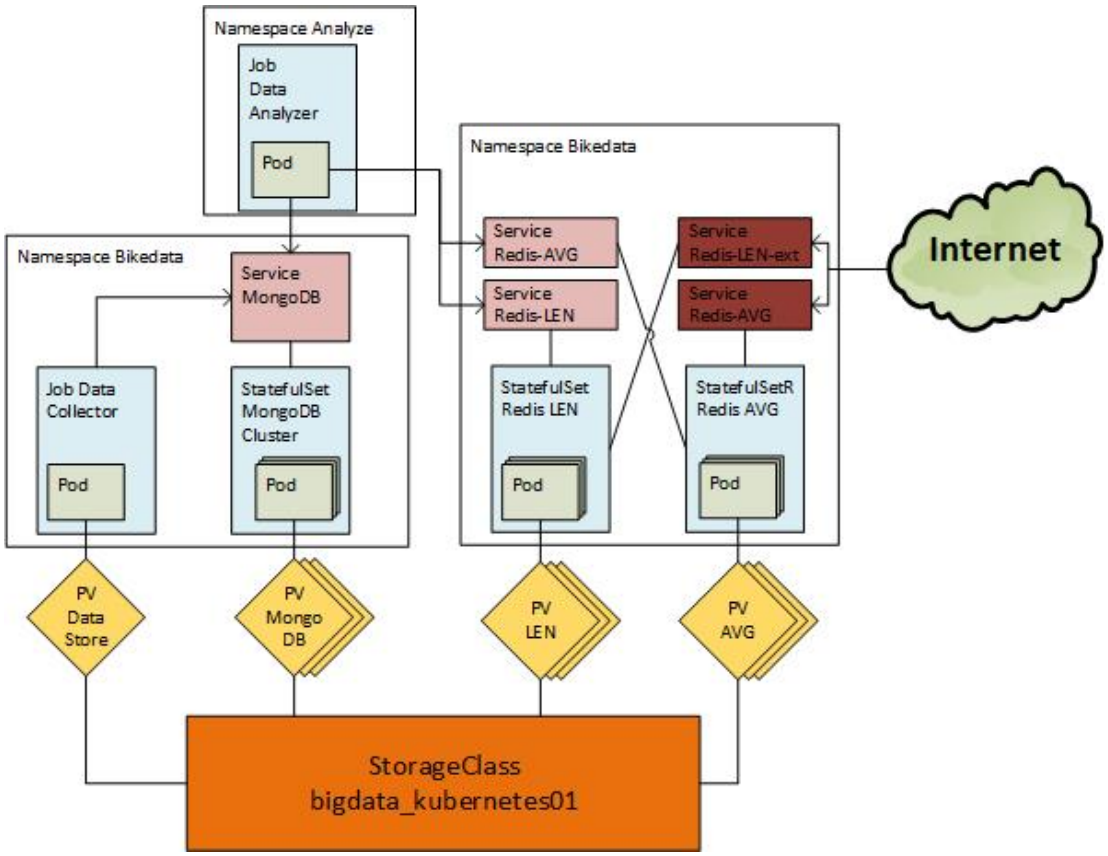


Abbildung 6.3.: Big Data Anwendungsarchitektur in Kubernetes

## 6.5. Deployment Pipeline

Die Deployment-Pipeline ist das wichtigste Werkzeug für die automatische Anwendungsauslieferung. Das Beispielszenario sieht vor, die Auslieferung des *Data Analyzer* zu automatisieren. Eine Pipeline setzt sich in Gitlab-CI aus Stages, die Jobs enthalten, zusammen. Die Stages werden sequenziell und die Jobs innerhalb einer Stage parallel abgearbeitet.
Listing 6.1 zeigt die Stages, die für die Auslieferung des Data Analyzer eingerichtet wurden. In der Build Stage wird das Software-Paket kompiliert und Unit Tests ausgeführt. Nach Abschluss wird das Software-Paket in einem Docker-Image gespeichert, mit der Zusatzbezeichnung *dev* versehen und in der Docker-Registry abgelegt. Im Anschluss daran folgt die Test-Stage, in der eine herunterskalierte und produktivnahe Umgebung aufgebaut wird. Innerhalb dieser Umgebung werden dann Integrationstests ausgeführt. Nach Abschluss dieser Stage wird das

Software-Paket ein weiteres mal in der Docker-Registry abgelegt und anschließend in der Deploy-Stage auf dem Kubernetes-Cluster bereitgestellt und ausgeführt.

```
Stages:
  - build
  - package-dev
  - test
  - package-prod
  - deploy
```

Listing 6.1: Stages der Deployment Pipeline

# 7. Evaluation

In diesem Abschnitt soll das vorgestellte Architekturkonzept im Bezug auf die in Unterabschnitt 5.1.2 genannten Anforderungen für die Bereitstellung einer Big Data-Architektur mit DevOps-Vorgaben evaluiert werden. Dafür werden das Konzept und die beiden Realisierungen herangezogen.

## 7.1. Vergleich mit den Anforderungen

Für die Evaluation wird zunächst überprüft, ob die Konzeption und die beiden Realisierungen den Anforderungen aus Unterabschnitt 5.1.2 gerecht werden.

| ID | Konzept | Realisierung Private Cloud | Realisierung Public Cloud |
|---|---|---|---|
| BD-1 | ✓ | ✓ | ✓ |
| BD-2 | ✓ | ✓ | ✓ |
| BD-3 | ✓ | ✓ | ✓ |
| BD-4 | ✓ | ✓ | ✓ |
| BD-5 | ✓ | ✓ | ✓ |
| BD-6 | ✓ | ✓ | ✓ |
| BD-7 | ✓ | ✗ | ✓ |
| BD-8 | ✓ | (✓) | (✓) |
| DO-1 | ✓ | ✓ | ✓ |
| DO-2 | ✓ | ✓ | ✓ |
| DO-3 | ✓ | ✓ | ✓ |
| DO-4 | ✓ | ✓ | ✓ |
| DO-5 | ✓ | ✓ | ✓ |

Tabelle 7.1.: Evaluationstabelle

Tabelle 7.1 gibt einen genauen Überblick darüber, welche Anforderungen erfüllt werden, welche teilweise erfüllt werden und welche nicht erfüllt werden. (Erfüllt: ✓/ teilweise erfüllt: (✓) /

nicht erfüllt: ✗). Im weiteren Verlauf, wird jede Zeile bewertet und die Begriffe Konzept und Realisierung unter dem Oberbegriff Szenarien weitergeführt.

**BD-1:** Durch die Eigenschaft *Resource Pooling* von Cloud Computing werden dem Nutzer scheinbar unendlich viele Systemressourcen, darunter auch Datenspeicher, zur Verfügung gestellt. Theoretisch wird durch das Konzept unendlich viel Speicher zur Verfügung gestellt. In der Realität sieht das aber anders aus: In der Private Cloud sind die Speicherressourcen durch die Computersysteme begrenzt und in der Public Cloud können Speicherressourcen durch Richtlinien des CSP begrenzt werden. Für dieses Projekt hat der Datenspeicher aber mehr als ausgereicht und deshalb gilt diese Anforderung in jedem Szenario als erfüllt.

**BD-2:** Da in allen Szenarien Kubernetes verwendet wird, ist die Verwaltung von Systemressourcen über eine einheitliche Schnittstelle, der kube-api, in allen Szenarien gesichert. Die Configuration Files sorgen dafür, das Systemzustände in Form von strukturierten Textdateien spezifiziert werden können, wodurch die Systemverwaltung zusätzlich vereinfacht und dokumentiert wird.

**BD-3 & BD-4 & BD-5** Da Kubernetes auf Container-Technologien basiert, ist die schnelle Bereitstellung von Datenbanken und Laufzeitumgebungen in allen Szenarien möglich. Durch den geringen Ressourcenbedarf ist es möglich, innerhalb kürzester Zeit beliebige Systeme in logisch getrennten Bereichen aufzubauen und zu betreiben, was auch ein Erfolgsfaktor für die Effizienz von Deployment-Pipelines ist.

**BD-6:** Die Kubernetes-Abstraktionen erlauben eine beliebige Kombination und Trennung der Anwendungscontainer. Es konnte dabei die gesamte Architektur der Beispielanwendung in den Kubernetes-Clusters abgebildet werden.

**BD-7:** Die hohe Verfügbarkeit der Systeme ist eine der Grundeigenschaften des Cloud Computings, weshalb das Konzept diese Anforderung erfüllt. In der Realität hängt die Verfügbarkeit der Systeme sehr stark von der Hardware-Infrastruktur ab. Im Falle der Realisierung auf der Private Cloud ist dies gescheitert. Die IP-Adressen, die durch das öffentliche Netz vergeben wurden, waren außerhalb der beiden Cluster-Netzwerke nicht zu erreichen, wodurch sie nur für Mitarbeiter des Projekts mit einer entsprechenden SSH-Verbindung verfügbar waren. Aus diesem Grund gilt diese Anforderung für die Realisierung auf der Private Cloud als nicht erfüllt. Außerdem stellt der Hardware-Management-Server einen *Single Point of Failure* dar, weil ein Ausfall dieser Maschine einen kompletten Ausschluss zur Folge hat, was das eine oder andere mal passiert ist. Die Public Cloud

erfüllte diese Anforderung für den Zeitraum des Projekts zu jedem Zeitpunkt. Grund dafür ist eine viel ausgereiftere, größere und richtig konfigurierte Infrastruktur, die mit großer Wahrscheinlichkeit Strategien zur Bewältigung von Ausfällen implementiert bekommen hat. Da Ausfälle nicht spürbar waren, gilt diese Anforderung für die Public Cloud als erfüllt.

**BD-8:** Wie in **BD-1** liefert eine Cloud-Plattform scheinbar unendlich viele Ressourcen, wodurch eine beliebige Skalierung der Systeme möglich ist. In der Realität ist diese Eigenschaft ebenfalls durch Grenzen der Systemressourcen bzw. Begrenzungen durch die CSP eingeschränkt, wodurch diese Anforderung in der Realität nur teilweise erfüllt wird.

**DO-1:** Agile Methoden wurden durch den Einsatz von Gitlab, mit seinen Funktionalitäten, in Kombination mit Kubernetes sehr stark unterstützt. Die gesamte Architektur erlaubt es, eine Änderung sehr schnell auszuliefern. In der Realität hat die Deployment-Pipeline mit ihren fünf Stages bei jedem Durchlauf bei beiden Realisierungen ungefähr sechs Minuten gebraucht, um das Software-Paket nach dem Einchecken komplett bereitzustellen. Recherchen haben gezeigt, dass diese Pipeline jedoch noch um weitere, leistungssteigernde Funktionen erweitert werden könnte, um die Geschwindigkeit zu erhöhen, z.B. durch das Caching der Abhängigkeiten des Software-Pakets. Jedoch wird durch den schnellen automatisierten Auslieferungsprozess diese Anforderung in allen Szenarien erfüllt.

**DO-2:** In allen Szenarien wurde der gesamte Auslieferungsprozess des *Data Analyzer* mit Hilfe von Gitlab-CI automatisiert. Es wurden Phasen für die Erstellung des Software-Pakets, für das Testen und für die Bereitstellung entwickelt und umgesetzt. In der Theorie lies sich das zwar nicht testen, jedoch haben die beiden Realisierungen auf das Konzept aufgebaut, welches die Vorgabe für die Automatisierung war und deshalb gilt diese Anforderung für alle Szenarien als erfüllt.

**DO-3:** Die Bereitstellung aller verwendeten Dienste hat auf genau zwei zentralen Infrastrukturen stattgefunden. Zum einen auf der öffentlichen Gitlab-Plattform und zum anderen auf je einem Kubernetes-Cluster. Die Zentralisierung hätte theoretisch weitergeführt werden können, wenn innerhalb des Kubernetes Clusters eine Gitlab-Instanz bereitgestellt worden wäre. Jedoch hätte dies bedeutet, dass die Projekt-Management-Plattform abhängig vom Produktivsystem gewesen wäre und dies auch mehr Aufwand bei der Wartung bedeutet hätte, da die Verantwortung über den Gitlab-Server dann nicht mehr bei einem externen Dienstleister gewesen wäre. In jedem Fall gilt diese Anforderung insgesamt als erfüllt, weil nur zentralisierte Infrastrukturen zum Einsatz kamen.

**DO-4:** Auf die Überwachung der Systemressourcen wurde im Text nicht weiter eingegangen. Jedoch wird im Konzept die Überwachung durch die Cloud Computing-Eigenschaft *Measurement* gewährleistet. Im Falle der Realisierung auf der Private Cloud hat der Charm des Kubernetes-Masters ein Monitoring System innerhalb des Clusters bereitgestellt, was die Überwachung der Systeme auf Ebenen der Pods und Abreitsknoten ermöglicht hat. Die Public Cloud hat diese Funktionalität direkt angeboten und darüber hinaus noch weitere Funktionen, wie z.B. die Berechnung und Aufstellung der entstandenen Kosten. Zusätzlich bietet Gitlab die Möglichkeit, mit Hilfe der Kubernetes-Integration ein Überwachungssystem in den Cluster zu installieren und die Ergebnisse aus der Überwachung in der eigenen Benutzeroberfläche anzuzeigen. Alle Szenarien sind dieser Anforderung gerecht geworden.

**DO-5:** Mit Hilfe der Configuration Files ist die Reproduzierbarkeit der Systeme sehr unkompliziert. Dadurch, dass deren Zustände klar definiert sind und die Kubernetes-Cluster diese auch immer gleich interpretieren, können die Systeme beliebig oft aufgebaut und wieder zerstört werden. Zusätzlich hat *Juju* bei der Realisierung der Private Cloud ebenfalls mit Configuration Files gearbeitet, wodurch sich die Infrastruktur ebenfalls beliebig oft und in adäquater Zeit aufbauen lies.

## 7.2. Vergleich der beiden Realisierungen

Die beiden Realisierungen basieren auf unterschiedlichen Hardware-Infrastrukturen. Daraus ergeben sich Unterschiede, die im weiteren Verlauf dargestellt werden.

### 7.2.1. Measurement

Die Überwachung der Systeme wurde bei der Public Cloud direkt mitgeliefert. Bei der Private Cloud musste diese Funktionalität nachinstalliert werden, wobei das genutzte Framework diese Funktionalität im Cluster bereitgestellt hat. Die Erreichbarkeit dieser Funktion war etwas komplizierter als bei der Public Cloud. Während die Public Cloud diese Funktion auf ihrer Oberfläche integriert hat und bereitstellt, stellt die Private Cloud diese Funktion auf einer separaten Service-Abstraktion bereit.

### 7.2.2. Speicher

Die Speicherung der Daten war von Grund auf unterschiedlich. Während in der Public Cloud ein Netzwerkspeicher angefordert und direkt bereitgestellt werden kann, musste für die Private

Cloud zunächst eine passende Lösung gefunden und die Realisierung geplant und umgesetzt werden. Obwohl es sich in beiden Fällen um vollkommen unterschiedliche Speicherstandards handelt, konnte mit der StorageClass-Abstraktion in beiden Clustern eine einheitliche Schnittstelle zu den Speicherressourcen bereitgestellt werden.

### 7.2.3. Load Balancer

Durch die unterschiedliche Beschaffenheit der Netzwerk-Infrastrukturen, ist die Bereitstellung der Dienste für die Öffentlichkeit mit unterschiedlichen Herausforderungen verbunden. Während auf der Public Cloud diese Funktionalität mitgeliefert wird, musste auf der Private Cloud, ähnlich wie bei der Speicherlösung, eine Lösung für die Bereitstellung der Dienste gefunden, ausgearbeitet und umgesetzt werden. Dadurch, dass die Dienste aber nicht öffentlich gemacht werden konnten, ist die Private Cloud nicht als Produktivsystem, sondern eher als Entwicklungssystem und Spielraum geeignet.

### 7.2.4. Kosten

Die Kosten für die Private Cloud zu erfassen war nicht möglich. Jedoch werden die Kosten für einen der Worker den Freibetrag der Public Cloud sehr weit übersteigen. Dieser lag bei 300 Dollar. Die Betriebskosten der fünf virtuellen Maschinen lagen bei insgesamt 0,68 Cent pro Stunde. Vom 300 Dollar Freibetrag waren nach Abschluss der Realisierung auf der Public Cloud noch 155,93 Dollar übrig.

## 7.3. Bewertung

Das Projekt hat gezeigt, dass das Architekturkonzept den meisten, in der Theorie sogar allen, Anforderungen gerecht wird. Jedoch hat die Realität gezeigt, dass der Betrieb einer Private Cloud mit einer viel höheren Verantwortung, Komplexität und höheren Kosten verbunden ist, als die Nutzung einer Public Cloud.
Es ist schwer eine Aussage darüber zu treffen, welche der beiden Realisierungen für den Betrieb einer Big Data-Architektur besser geeignet ist. Die negativen Eigenschaften der Private Cloud stehen anderen Faktoren der Public Cloud gegenüber: Ein Nutzer von Public Clouds stellt seine Entwicklungen, Daten, Informationen und Ergebnisse auf externen Infrastrukturen bereit, bei denen nur der CSP Gewissheit darüber hat, was wirklich mit den Erzeugnissen passiert. Es wird ein großer Teil der Kontrolle an den CSP übergeben. Die Wahl der Cloud sollte dabei nicht nur von den Kosten sondern auch vom Anwendungsfall bzw. der Gesamtsituation abhängig gemacht werden.

# 8. Zusammenfassung

Ziel dieser Arbeit war es, ein Architekturkonzept zu entwickeln und einen Big Data-Anwendungsfall mit Hilfe von DevOps-Lösungen abzubilden. Basierend auf den Erkenntnissen und Eigenschaften von Big Data (2) und DevOps (3) wurden zunächst Beziehungen zwischen den beiden Kontexten hergestellt (3.5). Daraus hat sich ergeben, dass Cloud Computing (4) ein zentraler Begriff für beide Kontexte ist.

Im Anschluss daran folgte die Konzeption (5) der Architektur. Dafür wurde zunächst für die Analyse (5.1) ein passendes Beispielszenario vorgestellt (5.1.1). Mit Hilfe des Beispielszenarios und der Eigenschaften von Big Data und DevOps wurde ein Anforderungsprofil (5.1.2) spezifiziert, auf dessen Basis im Anschluss der Entwurf für die Architektur konzeptioniert wurde (5.2). Dafür wurden zunächst die verwendeten Technologien und Systeme und deren Funktionen vorgestellt (5.2.1). Anschließend folgten die Beschreibung der Infrastruktur (5.2.3) und der Deployment-Pipeline (5.2.4), mit dessen Hilfe die Bereitstellung der Big Data-Applikation aus dem Beispielszenario automatisiert wurde.

Auf Grundlage der Konzeption wurde die Anwendungsarchitektur auf zwei unterschiedlichen Cloud-Systemen realisiert (6). Zunächst wurden die beiden zugrundeliegenden Hardware-Infrastrukturen vorgestellt, da sich die Lösungen grundsätzlich nur auf dieser Ebene unterscheiden. Auf höheren, abstrakten Ebenen waren die Lösungen annähernd identisch und es folgte die Umsetzung mit Kubernetes und Gitlab (6.3). Den Schluss bildet die Evaluation (7) des Konzepts und der beiden Realisierungen mit Bezug auf die Anforderungen und das Beispielszenario.

## 8.1. Fazit

Das in dieser Arbeit entwickelte Konzept für die Bereitstellung einer Big Data-Architektur wurde mit Hilfe eines Beispielszenarios vollständig realisiert. Bei dem Beispielszenario handelt es sich jedoch um einen relativ kleinen Anwendungsfall. Jedoch repräsentiert er eine Typische Anforderung aus dem Big Data-Kontext und bildet gleichzeitig in Teilen den Big

Data-Lebenszyklus ab, wodurch sich dieses Konzept auch auf größere Anwendundgsfälle adaptieren lassen sollte, da die wichtigsten Komponenten beachtet worden sind.

Es hat sich gezeigt, dass die Implementation einer Infrastruktur zur automatischen Bereitstellung von Big Data-Anwendungen auf einer privaten Umgebung eine große Herausforderung darstellt und nur unter größter Sorgfalt durchgeführt werden sollte, weil kleine Fehler in den unteren Ebenen der Infrastruktur fatale Folgen für die oberen Ebenen haben kann. Public Clouds helfen dabei, die Komplexität bei der Einrichtung einer Infrastruktur zu verringern und bieten viele kosten-effiziente Lösungen an.

Bei der Ausarbeitung dieser Abschlussarbeit ist Wissen aus vielen unterschiedlichen Bereichen der Informationstechnik zum Einsatz gekommen. Die Realisierung auf unterschiedlichen Ebenen war einerseits anspruchsvoll und andererseits äußerst lehrreich. Nachdem die Deployment-Pipeline für den Data Analyzer fertig war, konnten mit Hilfe der gewonnen Erkenntnisse direkt weitere Auslieferungsprozesse bzw. Prozeduren durch die Architektur automatisiert werden, wodurch die Effizienz des Systems weiter anstieg. Cloud-basierte Systeme sind ein sehr interessanter und vielseitiger Kontext und können durch eine Vielzahl an Service-Modellen auch unerfahrenen Personen einen einfachen Einstieg in die Welt der Informationstechnik ermöglichen.

# A. Inhalt der CD

Dieser Arbeit wurde ein CD mit folgender Verzeichnisstruktur beigelegt:

- **Ausarbeitung:** Diese Abreit als PDF-Dokument

- **Git-Repositories der Komponenten:** Die Komponenten der Beispielanwendung als Zip-Dateien

- **Installationsskipte** Die Installtionsskripte in einer Zip-Datei

# Literaturverzeichnis


[Alt u. a. 2017] Alt, Rainer ; Auth, Gunnar ; Kögler, Christoph: *Innovationsorientiertes IT-Management mit DevOps - IT im Zeitalter von Digitalisierung und Software-defined Business*. Berlin Heidelberg New York : Springer-Verlag, 2017. – Online verfügbar unter: https://link.springer.com/book/10.1007/978-3-658-18704-0 [Abruf: 2018-07-31]. – ISBN 978-3-658-18704-0

[Amazon 2018] Amazon: *Was ist DevOps?* 2018. – Online verfügbar unter: https://aws.amazon.com/de/devops/what-is-devops/ [Abruf: 2018-07-31]

[Amazon Web Services 2019] Amazon Web Services: *Amazon Web Services*. 2019. – Online verfügbar unter: https://aws.amazon.com/de/ [Abruf: 2019-01-01]

[Apache 2014] Apache: *Welcome to Apache Hadoop!* 2014. – Online verfügbar unter: http://hadoop.apache.org/index.pdf [Abruf: 2018-07-31]

[Armbrust u. a. 2009] Armbrust, Michael ; Fox, Armando ; Griffith, Rean ; Joseph, Anthony D. ; Katz, Randy H. ; Konwinski, Andrew ; Lee, Gunho ; Patterson, David A. ; Rabkin, Ariel ; Stoica, Ion ; Zaharia, Matei: Above the clouds: A berkeley view of cloud computing. In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (2009), Nr. 13. – Online verfügbar unter: https://www.researchgate.net/publication/200045935_Above_the_Clouds_A_Berkeley_View_of_Cloud_Computing [Abruf: 2018-09-18]

[Assunção u. a. 2015] Assunção, Marcos D. ; Calheiros, Rodrigo N. ; Bianchi, Silvia ; Netto, Marco A. ; Buyya, Rajkumar: Big Data computing and clouds: Trends and future directions. In: *Journal of Parallel and Distributed Computing* 79-80 (2015), S. 3–15. – Online verfügbar unter: http://www.sciencedirect.com/science/article/pii/S0743731514001452 [Abruf: 2018-09-18]

[Atlassian 2018] Atlassian: *Was ist DevOps?* 2018. – Online verfügbar unter: https://de.atlassian.com/devops [Abruf: 2018-08-17]

[Balalaie u. a. 2016] Balalaie ; Heydarnoori, Abbas ; Jamshidi, Pooyan: Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In: *IEEE Software* 33 (2016), Nr. 3, S. 42–52. – Online verfügbar unter: https://ieeexplore.ieee.org/document/7436659/ [Abruf: 2018-08-17]

[Banothu u. a. 2016] Banothu, Narsimha ; Bhukya, ShankarNayak ; Sharma, K. V.: Big-data: Acid versus base for database transactions. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, S. 3704–3709. – Online verfügbar unter: http://doi.acm.org/10.1145/289.291 [Abruf: 2018-07-31]

[Bass u. a. 2015] Bass, Len ; Weber, Ingo ; Zhu, Liming: *DevOps: A Software Architect's Perspective*. 1. Auflage. Old Tappan, NJ : Addison-Wesley Professional, 2015. – ISBN 978-0-13-404984-7

[Baun u. a. 2011] Baun, Christian ; Kunze, Marcel ; Nimis, Jens ; Tai, Stefan: *Cloud Computing - Web-basierte dynamische IT-Services*. 2. Auflage. Berlin Heidelberg New York : Springer-Verlag, 2011. – Online verfügbar unter: https://link.springer.com/book/10.1007/978-3-642-18436-9 [Abruf: 2018-09-18]. – ISBN 978-3-642-18436-9

[Beck u. a. 2001] Beck, Kent ; Beedle, Mike ; Bennekum, Arie van ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin ; Grenning, James ; Highsmith, Jim ; Hunt, Andrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Martin, Robert C. ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, Dave: *Manifest für Agile Softwareentwicklung*. 2001. – Online verfügbar unter: http://agilemanifesto.org/iso/de/manifesto.html [Abruf: 2018-08-17]

[Big Data Lab HAW 2019] Big Data Lab HAW: *Big Data Lab - Big Data HAW Hamburg University UAS*. 2019. – Online verfügbar unter: https://nosqlzkn.ful.informatik.haw-hamburg.de/bigdatalab [Abruf: 2019-01-01]

[Brewer 2000] Brewer, Eric A.: Towards robust distributed systems. In: *PODC* Bd. 7, 2000, S. 7. – Online verfügbar unter: http://pld.cs.luc.edu/courses/353/spr11/notes/brewer_keynote.pdf [Abruf: 2018-07-31]

[Canonical 2019a] Canonical: *Juju*. 2019. – Online verfügbar unter: https://docs.jujucharms.com/2.5/en/ [Abruf: 2019-01-01]

[Canonical 2019b] Canonical: *Metal as a Service*. 2019. – Online verfügbar unter: https://docs.maas.io/ [Abruf: 2019-01-01]

[Chellappa 1997] Chellappa, Ramnath K.: Intermediaries in cloud-computing: a new computing paradigm. (1997)

[Cockburn 2002] Cockburn, Alistair: *Agile software development.* Boston : Addison-Wesley, 2002. – ISBN 978-0201699692

[Couch und Sun 2004] Couch, Alva ; Sun, Yizhan: On observed reproducibility in network configuration management. In: *Science of Computer Programming* 53 (2004), Nr. 2, S. 215–253. – URL http://www.sciencedirect.com/science/article/pii/S0167642304000796. – Online verfügbar unter: https://www.sciencedirect.com/science/article/pii/S0306437914001288 [Abruf: 2018-09-18]

[Dean und Ghemawat 2008] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *Commun. ACM* 51 (2008), Nr. 1. – Online verfügbar unter: http://doi.acm.org/10.1145/1327452.1327492 [Abruf: 2018-07-31]

[Deutsche Bahn AG 2016] Deutsche Bahn AG: *Buchungen Call a Bike (Stand 07/2016).* 2016. – Online verfügbar unter: https://data.deutschebahn.com/dataset/data-call-a-bike/resource/b51f1366-15a1-4176-bbc0-74c2722faf9c [Abruf: 2019-01-01]

[Deutsche Bahn AG 2019] Deutsche Bahn AG: *Open-Data-Portal - Das Datenportal der Deutschen Bahn AG.* 2019. – Online verfügbar unter: https://www.callabike-interaktiv.de/de [Abruf: 2019-01-01]

[Farroha und Farroha 2014] Farroha, B. S. ; Farroha, D. L.: A Framework for Managing Mission Needs, Compliance, and Trust in the DevOps Environment. In: *2014 IEEE Military Communications Conference*, 2014, S. 288–293. – Online verfügbar unter: https://ieeexplore.ieee.org/document/6956773/ [Abruf: 2018-08-17]

[Fasel und Meier 2016] Fasel, Daniel ; Meier, Andreas: *Big Data - Grundlagen, Systeme und Nutzungspotenziale.* Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978-3-658-11589-0

[Fischer 2015] Fischer, Thomas: *DevOps: Softwarearchitektur an der Schnittstelle zwischen Entwicklung und Betrieb.* 2015. – Online verfügbar unter: https://www.funkschau.de/telekommunikation/artikel/115843/ [Abruf: 2018-07-31]

[Fowler 2013] Fowler, Martin: *ContinuousDelivery*. 2013. – Online verfügbar unter: https://martinfowler.com/bliki/ContinuousDelivery.html [Abruf: 2018-08-17]

[Gartner 2018] Gartner: *DevOps*. 2018. – Online verfügbar unter: https://www.gartner.com/it-glossary/devops [Abruf: 2018-08-17]

[Gitlab 2019] Gitlab: *Gitlab*. 2019. – Online verfügbar unter: https://about.gitlab.com/ [Abruf: 2019-01-01]

[Google 2019] Google: *MetalLB*. 2019. – Online verfügbar unter: https://github.com/google/metallb [Abruf: 2019-01-01]

[Google Cloud 2019] Google Cloud: *Google Cloud*. 2019. – Online verfügbar unter: https://cloud.google.com/ [Abruf: 2019-01-01]

[Grolinger u. a. 2013] Grolinger, Katarina ; Higashino, Wilson A. ; Tiwari, Abhinav ; Capretz, Miriam A.: Data management in cloud environments: NoSQL and NewSQL data stores. In: *Journal of Cloud Computing: Advances, Systems and Applications* 2 (2013), Nr. 1, S. 22. – Online verfügbar unter: https://doi.org/10.1186/2192-113X-2-22 [Abruf: 2018-07-31]

[Gudivada u. a. 2014] Gudivada, Venkat N. ; Rao, Dhana ; Raghavan, Vijay V.: NoSQL Systems for Big Data Management. In: *2014 IEEE World Congress on Services*, 2014, S. 190–197. – Online verfügbar unter: https://ieeexplore.ieee.org/document/6903264/ [Abruf: 2018-07-31]

[Haerder und Reuter 1983] Haerder, Theo ; Reuter, Andreas: Principles of Transaction-oriented Database Recovery. In: *ACM Comput. Surv.* 15 (1983), Nr. 4, S. 287–317. – Online verfügbar unter: http://doi.acm.org/10.1145/289.291 [Abruf: 2018-07-31]

[Hasselbring 2015] Hasselbring, Wilhelm: *DevOps: Softwarearchitektur an der Schnittstelle zwischen Entwicklung und Betrieb*. 2015. – Online verfügbar unter: http://oceanrep.geomar.de/29215/1/2015-07-10Architekturen.pdf [Abruf: 2018-07-31]

[Heer und Shneidermann 2012] Heer, Jeffrey ; Shneidermann, Ben: Interactive Dynamics for Visual Analysis. In: *Queue* 10 (2012), Nr. 2, S. 30:30–30:55. – Online verfügbar unter: http://doi.acm.org/10.1145/2133416.2146416 [Abruf: 2018-07-31]

[Hsieh 2014] Hsieh, Donovan: *NoSQL Data Modeling*. 2014. – Online verfügbar unter: https://www.ebayinc.com/stories/blogs/tech/nosql-data-modeling/ [Abruf: 2018-07-31]

[Humble und Farley 2010] Humble, Jez ; Farley, David: *Continuous Delivery - Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ : Addison-Wesley, 2010. – ISBN 978-0-321-67022-9

[Humble und Molesky 2011] Humble, Jez ; Molesky, Joanne: Why Enterprises Must Adopt Devops to Enable Continuous Delivery. In: *Cutter IT Journal* 24 (2011), Nr. 8, S. 6–12. – Online verfügbar unter: https://www.cutter.com/article/why-enterprises-must-adopt-devops-enable-continuous-delivery-416516 [Abruf: 2018-07-31]

[Hüttermann 2012] Hüttermann, Michael: *DevOps for Developers*. New York : Apress, 2012. – ISBN 978-1-430-24569-8

[IBM Cloud 2019] IBM Cloud: *IBM Cloud*. 2019. – Online verfügbar unter: https://www.ibm.com/cloud/ [Abruf: 2019-01-01]

[James u. a. 2014] James, Doug ; Wilkins-Diehr, Nancy ; Stodden, Victoria ; Colbry, Dirk ; Rosales, Carlos ; Fahey, Mark R. ; Shi, Justin ; Silva, Rafael F. da ; Lee, Kyo ; Roskies, Ralph ; Loewe, Laurence ; Lindsey, Susan ; Kooper, Rob ; Barba, Lorena ; Bailey, David H. ; Borwein, Jonathan M. ; Corcho, Óscar ; Deelman, Ewa ; Dietze, Michael C. ; Gilbert, Benjamin ; Harkes, Jan ; Keele, Seth ; Kumar, Praveen ; Lee, Jong ; Linke, Erika ; Marciano, Richard ; Marini, Luigi ; Mattmann, Chris ; Mattson, Dave ; McHenry, Kenton ; McLay, Robert T. ; Miguez, Sheila ; Minsker, Barbara S. ; Pérez-Hernández, María S. ; Ryan, Dan ; Rynge, Mats ; Pérez, Idafen S. ; Satyanarayanan, Mahadev ; Clair, Gloriana S. ; Webster, Keith ; Hovig, Eivind ; Katz, Daniel S. ; Kay, Sophie ; Sandve, Geir K. ; Skinner, David ; Allen, Gabrielle ; Cazes, John ; Cho, Kym W. ; Fonseca, Jim ; Hwang, Lorraine ; Koesterke, Lars ; Patel, Pragnesh ; Pouchard, Line ; Seidel, Edward ; Suriarachchi, Isuru: Standing Together for Reproducibility in Large-Scale Computing: Report on reproducibility@XSEDE. In: *CoRR* abs/1412.5557 (2014). – Online verfügbar unter: http://arxiv.org/abs/1412.5557 [Abruf: 2018-07-31]

[Kim u. a. 2017] Kim, Gene ; Humble, Jez ; Debois, Patrick ; Willis, John: *Das DevOps-Handbuch - Teams, Tools und Infrastrukturen erfolgreich umgestalten*. 1. Auflage. Heidelberg : Dpunkt.Verlag GmbH, 2017. – ISBN 978-3-960-09047-2

[Klein u. a. 2013] Klein, Dominik ; Tran-Gia, Phuoc ; Hartmann, Matthias: Big Data. In: *Informatik-Spektrum* 36 (2013), Nr. 3, S. 319–323. – Online verfügbar unter: https://doi.org/10.1007/s00287-013-0702-3 [Abruf: 2018-07-31]

[Kubernetes.io 2019a] Kubernetes.io: *Persistent Volumes.* 2019. – Online verfügbar unter: https://kubernetes.io/docs/concepts/storage/persistent-volumes/ [Abruf: 2019-01-01]

[Kubernetes.io 2019b] Kubernetes.io: *What is Kubernetes?* 2019. – Online verfügbar unter: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Abruf: 2019-01-01]

[Labrinidis und Jagadish 2012] Labrinidis, Alexandros ; Jagadish, H. V.: Challenges and Opportunities with Big Data. In: *Proceedings of the VLDB Endowment* 5 (2012), Nr. 12, S. 2032–2033. – Online verfügbar unter: http://dx.doi.org/10.14778/2367502.2367572 [Abruf: 2018-07-31]

[Lukša 2018] Lukša, Marko: *Kubernetes in Action Anwendungen in Kubernets-Clustern bereitstellen und verwalten.* München : Hanser, 2018. – ISBN 978-3446455108

[Meier 2017] Meier, Andreas: *Werkzeuge der digitalen Wirtschaft: Big Data, NoSQL & Co. - Eine Einführung in relationale und nicht-relationale Datenbanken.* Berlin Heidelberg New York : Springer-Verlag, 2017. – ISBN 978-3-658-20337-5

[Meier und Kaufmann 2016] Meier, Andreas ; Kaufmann, Michael: *SQL- & NoSQL-Datenbanken -.* 8. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978-3-662-47664-2

[Mell und Grance 2011] Mell, Peter ; Grance, Timothy: The NIST definition of cloud computing. (2011). – Online verfügbar unter: https://csrc.nist.gov/publications/detail/sp/800-145/final [Abruf: 2018-09-18]

[Merv 2011] Merv, Adrian: It's going mainstream, and it's your next opportunity. In: *Teradata Magazine* 1 (2011), S. 38–42. – Online verfügbar unter: https://archive.is/Lv9aC [Abruf: 2018-07-05]

[Microsoft Azure 2019] Microsoft Azure: *Microsoft Azure.* 2019. – Online verfügbar unter: https://azure.microsoft.com/de-de/ [Abruf: 2019-01-01]

[MongoDB, Inc 2019] MongoDB, Inc: *MongoDB.* 2019. – Online verfügbar unter: https://www.mongodb.com/ [Abruf: 2019-01-01]

[Moniruzzaman und Hossain 2013] Moniruzzaman, A. B. M. ; Hossain, Syed A.: NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. In: *CoRR* abs/1307.0191 (2013). – Online verfügbar unter: http://arxiv.org/abs/1307.0191 [Abruf: 2018-07-31]

[Novkovic 2018] Novkovic, Goran: *Cloud service models and responsibilities*. 2018. – Online verfügbar unter: https://www.controleng.com/single-article/cloud-service-models-and-responsibilities.html?print=1 [Abruf: 2018-09-18]

[Peschlow 2016] Peschlow, Patrick: Die DevOps Bewegung. In: *Java Magazin* (2016), Nr. 1. – Online verfügbar unter: https://public.centerdevice.de/ef7a0a7e-55c7-4757-a5f4-5838d38a826c [Abruf: 2018-07-31]

[Pokorny 2011] Pokorny, Jaroslav: NoSQL Databases: A Step to Database Scalability in Web Environment. In: *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*. New York, NY, USA : ACM, 2011 (iiWAS '11), S. 278–283. – Online verfügbar unter: http://doi.acm.org/10.1145/2095536.2095583 [Abruf: 2018-07-31]. – ISBN 978-1-4503-0784-0

[Pouchard 2015] Pouchard, Line: Revisiting the Data Lifecycle with Big Data Curation. In: *International Journal of Digital Curation* 10 (2015), Nr. 2, S. 176–192. – Online verfügbar unter: http://www.ijdc.net/article/view/10.2.176 [Abruf: 2018-07-24]

[Prometheus.io 2019] Prometheus.io: *Prometheus*. 2019. – Online verfügbar unter: https://prometheus.io/ [Abruf: 2019-01-01]

[Red Hat Limited 2019] Red Hat Limited: *What is Docker?* 2019. – Online verfügbar unter: https://www.redhat.com/en/topics/containers/what-is-docker [Abruf: 2019-01-01]

[Redaktion Digitales Wirtschaftswunder 2017] Redaktion Digitales Wirtschaftswunder: *Big Data: Schlüsseltechnologie für digitale Transformation*. 2017. – Online verfügbar unter: https://blog.qsc.de/2017/04/big-data-schluesseltechnologie-fuer-digitale-transformation/ [Abruf: 2019-01-01]

[Reinheimer 2018] Reinheimer, Stefan: *Cloud Computing - Die Infrastruktur der Digitalisierung*. Berlin Heidelberg New York : Springer-Verlag, 2018. – Online verfügbar un-

ter: https://link.springer.com/book/10.1007/978-3-658-20967-4 [Abruf: 2018-09-18]. – ISBN 978-3-658-20967-4

[Rittinghouse und Ransome 2010] Rittinghouse, John W. ; Ransome, James F.: *Cloud Computing: Implementation, Management, and Security.* Boca Raton : CRC press, 2010. – ISBN 978-1-439-80680-7

[Royce 1987] Royce, Winston W.: Managing the development of large software systems: concepts and techniques. In: *Proceedings of the 9th international conference on Software Engineering* IEEE Computer Society Press (Veranst.), 1987, S. 328–338. – Online verfügbar unter: http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf [Abruf: 2018-08-17]

[Sadalage und Fowler 2013] Sadalage, Pramod J. ; Fowler, Martin: *NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence.* Amsterdam : Addison-Wesley, 2013. – ISBN 978-0-321-82662-6

[Sagiroglu und Sinanc 2013] Sagiroglu, Seref ; Sinanc, Duygu: Big data: A review. In: *2013 International Conference on Collaboration Technologies and Systems (CTS)*, 2013, S. 42–47. – Online verfügbar unter: https://ieeexplore.ieee.org/document/6567202/ [Abruf: 2018-07-31]

[Sharma 2014] Sharma, Sanjeev: *Adopting DevOps for continuous innovation.* 2014. – Online verfügbar unter: https://www.ibm.com/developerworks/library/d-devops-continuous-innovation/index.html [Abruf: 2018-07-31]

[Skourletopoulos u. a. 2016] Skourletopoulos, Georgios ; Mavromoustakis, Constandinos X. ; Chatzimisios, Periklis ; Mastorakis, George ; Pallis, Evangelos ; Batalla, Jordi M.: Towards the evaluation of a big data-as-a-service model: A decision theoretic approach. In: *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, S. 877–883. – Online verfügbar unter: https://ieeexplore.ieee.org/document/7562202/ [Abruf: 2018-09-18]

[Smeds u. a. 2015] Smeds, Jens ; Kristian Nybom, Kristian ; Porres, Ivan: DevOps: A Definition and Perceived Adoption Impediments. In: Lassenius, Casper (Hrsg.) ; Dingsøyr, Torgeir (Hrsg.) ; Paasivaara, Maria (Hrsg.): *Agile Processes in Software Engineering and Extreme Programming.* Cham : Springer International Publishing, 2015, S. 166–177. – ISBN 978-3-319-18612-2

[Soni 2016] Soni, Mitesh: *DevOps for Web Development.* Birmingham : Packt Publishing Ltd, 2016. – Online verfügbar unter: http://file.allitebooks.com/20161028/DevOps%20for%20Web%20Development.pdf [Abruf: 2018-08-17]. – ISBN 978-1-786-46835-2

[Spinellis 2012] Spinellis, Diomidis: *Don't Install Software by Hand.* 2012. – Online verfügbar unter: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6265084 [Abruf: 2018-07-05]

[Stähler 2016] Stähler, Michael: *Auf dem Weg von Continuous Integration zu Continuous Delivery.* 2016. – Online verfügbar unter: https://www.informatik-aktuell.de/entwicklung/methoden/devops-in-der-praxis-von-continuous-integration-zu-continuous-delivery.html [Abruf: 2018-07-31]

[Trepper 2012] Trepper, Tobias: *Agil-systemisches Softwareprojektmanagement* -. Berlin Heidelberg New York : Springer-Verlag, 2012. – ISBN 978-3-834-94202-9

[Turck 2018] Turck, Matt: *Great Power, Great Responsibility: The 2018 Big Data & AI Landscape.* 2018. – Online verfügbar unter: http://mattturck.com/bigdata2018/ [Abruf: 2018-07-31]

[Tvisha 2018] Tvisha: *DevOps-Development.* 2018. – Online verfügbar unter: https://www.tvisha.com/services/devops-software-development-company.html [Abruf: 2018-08-17]

[Unterstein und Matthiessen 2012] Unterstein, Michael ; Matthiessen, Günter: *Relationale Datenbanken und SQL in Theorie und Praxis.* 5.Auflage. Berlin Heidelberg New York : Springer-Verlag, 2012. – ISBN 978-3-642-28986-6

[Virmani 2015] Virmani, Manish: Understanding DevOps amp; bridging the gap from continuous integration to continuous delivery. In: *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015, S. 78–82. – Online verfügbar unter: https://ieeexplore.ieee.org/document/7173368/ [Abruf: 2018-08-17]

[Vossen u. a. 2013] Vossen, Gottfried ; Haselmann, Till ; Hoeren, Thomas: *Cloud-Computing für Unternehmen - Technische, wirtschaftliche, rechtliche und organisatorische Aspekte.* Heidelberg : dpunkt.verlag, 2013. – ISBN 978-3-864-91131-6

[Willis 2010] Willis, John: *What Devops Means to Me.* 2010. – Online verfügbar unter: https://blog.chef.io/2010/07/16/what-devops-means-to-me/ [Abruf: 2018-07-31]

[Wolff 2015] Wolff, Eberhard: *Continuous Delivery - Der pragmatische Einstieg.* 2. Aufl. Heidelberg : dpunkt.verlag, 2015. – ISBN 978-3-864-91931-2

[Xinhua u. a. 2014] Xinhua, E ; Han, Jing ; Wang, Yasong: *Big Data-as-a-Service: Definition and architecture.* 2014. – Online verfügbar unter: https://ieeexplore.ieee.org/document/6820472/ [Abruf: 2018-07-31]

[Zheng u. a. 2013] Zheng, Zibin ; Zhu, Jieming ; Lyu, Michael R.: Service-Generated Big Data and Big Data-as-a-Service: An Overview. In: *2013 IEEE International Congress on Big Data*, 2013, S. 403–410. – Online verfügbar unter: https://ieeexplore.ieee.org/document/6597164/ [Abruf: 2018-09-18]. – ISSN 2379-7703

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 25.01.2019 Björn Freund