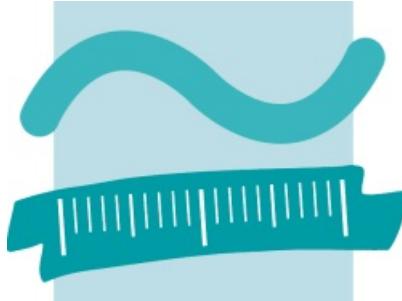


**Beuth Hochschule für Technik Berlin**  
Fachbereich Informatik und Medien  
Studiengang Medieninformatik

# **Serverless Programmierung: State of the Art und Evaluation**

Masterarbeit  
zur Erlangung des Grades Master of Science



eingereicht von: Svenja Haberditzl  
aus: Lübeck  
Matrikel-Nr.: 830791  
Abgabedatum: 29.05.2017  
Erstgutachter: Prof. Dr. Stefan Edlich  
Zweitgutachter: Prof. Christoph Knabe

---

## Inhaltsverzeichnis

1. Einleitung.....	1
1.1 Problemstellung.....	1
1.2 Zielsetzung.....	1
1.3 Methodik.....	2
1.4 Aufbau der Ausarbeitung.....	2
2. Server-basierte Programmierung.....	4
2.1 Anforderungen an Software.....	4
2.2 Architekturarten.....	6
2.2.1 Server/Client.....	6
2.2.2 Web-Architektur.....	9
2.2.3 SOA (Serviceorientierte Architekturen).....	9
2.3 Schichtenmodell.....	10
2.4 Zusammenfassung des Kapitels.....	11
3. Serverless Programmierung.....	12
3.1 Definition.....	12
3.2 Etablierung am Markt.....	13
3.3 Einsatzorte.....	15
3.4 Zusammenfassung des Kapitels.....	18
4. Vorstellung der Möglichkeiten Serverless Programmierung.....	19
4.1 Google Cloud Functions.....	19
4.2 Windows Azure Functions.....	20
4.3 IBM Open Whisk.....	21
4.4 Amazon Webservices.....	23
4.4.1 Allgemeines.....	23
4.4.2 Regionen.....	26
4.4.3 Roles.....	27
4.5 AWS Lambda.....	27
4.6 Zusammenfassung des Kapitels.....	30
5. Erläuterung des Prototyps.....	31
5.1 Anforderungen und Ziele.....	31
5.2 Aufbau.....	31
5.3 Zusammenfassung des Kapitels.....	34
6. Erläuterung Erarbeitung des Prototyps.....	35
6.1 Erarbeitung des Prototyps.....	35
6.2 S3 Buckets.....	37

---

6.3 JavaScript.....	37
6.3.1 Hochladen in Ordner.....	38
6.3.2 Hochladen ohne Ordner.....	41
6.3.3 Größenänderung eines Bildes.....	42
6.4 Zugriffsrechte und Roles.....	45
6.5 Lambda.....	46
6.6 DynamoDB Tabelle.....	47
6.7 Aktueller Stand der nicht funktionalen Anmeldefunktion.....	49
6.8 Verwendete Hard- und Software.....	51
6.9 Übersicht der Funktionen und Elemente.....	51
6.10 Zusammenfassung des Kapitels.....	53
 7. Performance-Messungen.....	54
7.1 Durchführung der Messungen.....	54
7.2 Zusammenfassung des Kapitels.....	56
 8. Ergebnis der Messungen.....	57
8.1 Ergebnisse der durchgeführten Messungen.....	57
8.1.1 Latenz.....	61
8.1.2 Durchsatz.....	62
8.2 Bedeutung der Ergebnisse.....	62
8.3 Zusammenfassung des Kapitels.....	63
 9. Vor - und Nachteile Serverless Programmierung.....	64
9.1 Vorteile.....	64
9.2 Nachteile.....	68
9.3 Checkliste Vor- und Nachteile.....	71
9.4 Zusammenfassung des Kapitels.....	74
 10. Fazit.....	75
10.1 Allgemeines Fazit.....	75
10.2 Persönliches Fazit.....	76
10.3 Probleme und Abweichungen.....	78
10.4 Mögliche Ergänzungen der Anwendungen.....	79
 11. Ausblick in die Zukunft.....	80
 Abbildungsverzeichnis.....	III
Tabellenverzeichnis.....	IV
Literaturverzeichnis.....	V
Anhang.....	X

## 1. Einleitung

### 1.1 Problemstellung

Das wissenschaftliche Problem dieser Ausarbeitung bezieht sich auf die Frage, warum das Programmieren mit Servern Nachteile hat und welche diese im Einzelnen sind.

Seit Jahren ist es im Bereich der Softwareentwicklung üblich, dass Programme mit Hilfe von Servern realisiert werden. Eine relativ neue Möglichkeit ist jedoch dabei, sich in diesem Bereich zu etablieren. Sie wird unter dem Namen “Serverless Programming” gelistet und ermöglicht so eine Server arme Umsetzung von Software.

Warum aber ist diese Variante derzeit nur bei einigen Projekten zu beobachten, wenn sie doch Vorteile gegenüber den bisherigen Methoden bietet? Im Bereich der Softwareentwicklung und damit der Programmierung herrscht ein gewisser Stillstand vor sobald sich ein Standard durchgesetzt hat. Dieser wird auf Grund der Vielzahl an Literatur und Lösungsmöglichkeiten bei auftretenden Problemen standardmäßig eingesetzt ohne über Alternativen nachzudenken. Deshalb setzen sich Neuentwicklungen nur recht langsam und mit viel Überzeugungskraft durch.

Um die Möglichkeiten, die die Serverless Programmierung bietet, verdeutlichen zu können, sollen die Nachteile der Anwendungserstellung mit Servern betrachtet werden. Eine Umstellung weg von Servern kann nur erfolgen, wenn das Verständnis dafür hergestellt wird und ein Umdenken im Kopf stattfindet.

Im Zuge dieser Ausarbeitung soll auf die bisherigen Methoden der Softwareentwicklung ebenso eingegangen werden wie auf die Serverless Programmierung. So wird ein Überblick über beide Seiten geschaffen, der die Stärken und Schwächen aufzeigt und die Funktionsweisen verdeutlicht.

### 1.2 Zielsetzung

Die vorliegende Ausarbeitung soll aufzeigen, wie das bisherige System der Anwendungserstellung aufgebaut ist und welche Nachteile sich daraus ergeben. Weiterhin soll verdeutlicht werden, wieso die Zukunft in der Serverless Programmierung liegt und diese immer mehr an Bedeutung gewinnt.

Durch die Auflistung der bisher verwendeten Verfahren und Möglichkeiten in der Softwareprogrammierung soll gezeigt werden, wo im Einzelnen die Unterschiede zur Serverless Programmierung liegen. Durch die intensive Auseinandersetzung mit den Vor- und Nachteilen der Serverless Programmierung wird zusätzlich ein Überblick über das

---

Potenzial dieser gegeben.

### 1.3 Methodik

Um auf die genannte Problemstellung eingehen zu können, soll der Bereich der Serverless Programmierung beleuchtet und erläutert werden. Damit die daraus resultierenden Erkenntnisse auch praktisch umgesetzt und betrachtet werden können, wird ein Prototyp erstellt, um ein besseres Verständnis zu erzeugen. Damit die ausgangs aufgestellte These, dass Server-basierte Programmierung Nachteile hat, belegt werden kann, werden zusätzlich Performance-Messungen an der im Zuge dieser Ausarbeitung erstellten Anwendung vorgenommen und deren Ergebnisse dahingehend untersucht und ausgewertet. Damit ein schlüssiges Ergebnis zu der Problemstellung aufgestellt werden kann, werden Vergleiche zu den bisherigen Möglichkeiten der Softwareentwicklung gemacht. Diese Fülle an Informationen über bisherige und neue Möglichkeiten, veranschaulicht durch den Prototyp sollte zur Beantwortung des Problems beitragen und dafür ausreichend sein.

### 1.4 Aufbau der Ausarbeitung

Zunächst werden die bisher üblichen Methoden der Softwareentwicklung hinsichtlich der Verwendung von Servern aufgelistet und die am weitesten verbreiteten hinsichtlich der Funktionsmöglichkeiten und dem Aufbau näher beleuchtet. Um ein Verständnis für den Bereich der Serverless Programmierung zu erhalten, wird zunächst eine Definition gegeben mit anschließenden Erläuterungen dazu. Zusätzlich wird auf Einsatzorte eingegangen, in denen bereits Serverless Programmierung angewendet wird oder in denen eine Umsetzung als sinnvoll anzusehen ist. Im Anschluss erfolgt eine Erläuterung zur Serverless Programmierung, ebenfalls mit Erwähnung der Funktionsweisen der Möglichkeiten und dem Aufbau dieser. Eine Vertiefung soll durch das Betrachten von verschiedenen Plattformen, die die Möglichkeit einer „serverless“ Umsetzung bieten, erfolgen. Da die Plattform AWS und explizit der Service Lambda in dieser Ausarbeitung zum Zwecke der Erstellung des Prototypen herangezogen wird, soll hierzu eine detailliertere Betrachtung folgen. Nachdem der theoretische Teil beendet ist, wird der Prototyp näher beleuchtet. Hier folgt zunächst eine Erklärung, wie dieser aussehen soll hinsichtlich der Elemente, die dort verwendet werden und welche Funktionalität er erfüllt. Danach wird die Erstellung des Prototyps beleuchtet und es werden wichtige Codepassagen erklärt, um die Funktionalität besser verdeutlichen zu können. Es wird

dabei auf die unterschiedlichen Umsetzungsarten eingegangen und deren jeweiliger Einsatz im Prototyp. Damit die Ausarbeitung die Problemstellung zusätzlich untermauert, werden Performance-Messungen anhand des Prototyps vorgenommen. Diese beziehen sich auf den Durchsatz und die Latenz und sollen den Unterschied zu den bisherigen Methoden diesbezüglich sichtbar machen und verdeutlichen. Abschließend erfolgt eine Zusammenfassung als Ergebnis für den Prototyp. Dabei wird auf die Umsetzung ebenso eingegangen wie auf die Messergebnisse und deren Bedeutung. Um die These, dass Serverless Programmierung Vorteile bietet zu verdeutlichen, werden nachfolgend die Vor- und Nachteile dieser Methode aufgelistet und näher erläutert. Diese resultieren aus der zu Anfang gemachten Erklärung zur Serverless Programmierung und dessen Funktionsweise sowie ebenfalls aus den Erkenntnissen der Erarbeitung des Prototyps. Am Ende folgt ein Fazit zu der gesamten Ausarbeitung, in der auf die Eingangs gestellte Problemstellung und die Beantwortung dieser eingegangen wird. Ebenso erhält ein persönliches Fazit hinsichtlich der während der Ausarbeitung gemachten Erfahrungen und Probleme Einzug. Abschließend wird ein Ausblick darauf gegeben, wie in Zukunft mit Serverless Programmierung gearbeitet und wie diese Möglichkeit etabliert werden könnte.

## 2. Server-basierte Programmierung

In der Softwareentwicklung existieren unterschiedliche Möglichkeiten und Arten, eine Anwendung zu erstellen. Jeder dieser Bereiche vereint in sich weitere Möglichkeiten, die hier jedoch nicht in Gänze erläutert werden sollen. Vielmehr soll hier der Fokus auf einer Server-basierten Herangehensweise einer Softwareentwicklung liegen. Die vorliegende Arbeit handelt von Serverless Programmierung, deshalb sollen hier nur Möglichkeiten aufgezeigt werden, die nach bisherigen Mitteln mit Servern arbeiten, um im Anschluss zeigen zu können, wo genau im Einzelnen die Vor- und Nachteile der jeweiligen Durchführungsweise liegen. Zunächst wird ein Einblick in die Anforderungen an Software im Allgemeinen gegeben, um aufzuzeigen, was die Umsetzungsmöglichkeiten leisten müssen und auf was ein Programmierer im Einzelnen achten muss. Durch die Auflistung wird nach dem Erläutern der Umsetzungsmöglichkeiten mit Server und „serverless“ das Verständnis für den Aufbau der Aufgaben eines Programmierers und die damit einhergehenden Veränderungen deutlich.

### 2.1 Anforderungen an Software

Um eine gute Software zu erhalten, existieren einige Punkte, die bei der Entwicklung von den Programmierern beachtet werden sollten. Eine Missachtung eines oder mehrere dieser Punkte kann gravierende Auswirkungen auf ein Projekt haben. Schon ein kleiner Fehler kann dazu führen, dass die komplette bisherige Entwicklung gelöscht werden und das Team von vorne anfangen muss. Daher sollte von Anfang an sehr genau auf jedes Detail geachtet werden.

Im Folgenden sollen nun die zu beachtenden Punkte in Form einer Checkliste aufgezählt werden.

- Mit allen Beteiligten des Projekts ausreichend kommunizieren, um Klarheit zu schaffen
- Anforderungen früh und möglichst vollständig definieren
- Nichtfunktionale Anforderungen frühzeitig beachten
- Richtige Architekturart wählen
- Code gut strukturieren
- Code leicht änderbar gestalten
- Qualität nicht aus den Augen verlieren
- Software möglichst gut wartbar gestalten
- Alle Komponenten des Projekts bedenken und einbauen

- Geeignete Namensgebung für Komponenten beachten
- Komponenten nicht über ihre Zuständigkeit hinaus entwickeln
- Zugriffsregeln für Komponenten einhalten
- Richtige Anordnung der Komponenten beachten
- Codeelemente klar voneinander trennen und gruppieren
- Veränderungen im Code zeitnah testen
- Geeignete Patterns verwenden
- Spaghetticode durch Kapselung vermeiden
- Horizontale und vertikale Ebenen beachten
- Code ausreichend und verständlich dokumentieren
- Prototypen realisieren

Zu beachten ist hierbei auch, dass die einzelnen Punkte der Checkliste nicht einer wahllosen Reihenfolge unterworfen sind, sondern diese bewusst so festgelegt wurden, um möglichst von Anfang an auf wichtige Elemente hinzuweisen. So kann die Liste Schritt für Schritt abgearbeitet werden. Es empfiehlt sich dann, die jeweils beachteten oder zur Kenntnis genommenen Punkte zu markieren, damit auch weitere am Projekt beteiligte Personen über den aktuellen Bearbeitungsstand Bescheid wissen und eventuell bei Missachtung eines Eintrags einspringen können.

Weiterhin ist es ratsam den Quellcode mit Hilfe von Bewertungswerkzeugen auf mögliche Fehler oder Unstimmigkeiten hin testen zu lassen. Beispiele für Software dieser Art stellen SonarQube, PMD, FindBugs oder CheckStyle dar. Mit SonarQube lassen sich detailliertere Informationen zu der getesteten Anwendung erzielen, weshalb die Verwendung dieses Programms anzuraten ist. Die Qualität einer Anwendung kann so gemessen und verbessert werden, wenn dem Entwickler aufgezeigt wird, wo eventuell noch Schwachstellen vorhanden sind. Dies ist insbesondere im weiteren Verlauf wichtig, damit die Software später im Betrieb keine Probleme verursacht, die verhindert hätten werden können durch eine eingehende Beschäftigung mit der Qualität des Codes. Vor allem die Reaktionszeiten der Anwendung sind für eine spätere Verwendung im Realbetrieb wichtig, um abschätzen zu können, ob auch eine hohe Zugriffszahl von Anwendern für das Programm tragbar ist. Zwar können bei einem Test kaum Szenarien einer tatsächlichen Nutzung der Anwendung simuliert werden, jedoch bieten sie die Möglichkeit, einen Eindruck zu erhalten wie sich dieser Bereich später wahrscheinlich verhalten wird.

Ein wichtiger Teil der Softwareentwicklung stellt das Testen eben dieser dar. Viele Programmierer unterschätzen diesen Punkt und gefährden so das erfolgreiche Abschließen

eines Projektes. Wird eine Anwendung nicht zeitnah und möglichst von Anfang an getestet, so kann es passieren, dass ein Code schon früh von einem Fehler befallen ist und dieser später nicht mehr ohne weiteres beseitigt werden kann. Nach jeder größeren Änderung bzw. nach dem Einbau wichtiger Funktionen sollte der jeweilige Code umgehend auf seine Funktionalität hin überprüft werden. Weiterhin sollte ebenfalls nicht darauf verzichtet werden, die unterschiedlichen Einzelteile der Anwendung zusammenzusetzen, um die Verbindung zwischen diesen zu testen. Insbesondere, wenn es sich bei der Anwendung um ein Projekt eines Unternehmens handelt, können durch das nicht Beachten regelmäßiger Tests hohe Kosten und Verzögerungen bei der Endabgabe entstehen, was wiederum dem Image schadet. Deshalb sollten die Mitarbeiter dazu angehalten werden, in bestimmten Abständen Tests durchzuführen und diese zu dokumentieren. Die Ergebnisse dieser Tests sollten an alle Beteiligten, die in das jeweilige Projekt involviert sind, weitergeleitet werden, damit diese einen Überblick über die Funktionalität der jeweiligen Aufgabe des anderen und den aktuellen Stand haben. (Heil, 2012, S. 95)

Die Informationen dieses Unterkapitels wurden im Rahmen des Moduls „Paradigmen moderner Softwareentwicklung“ erworben und die dort im Zuge einer Aufgabe erstellte Checkliste sowie die Erkenntnisse aus dem Modul hier verwendet.

## 2.2 Architekturarten

Nachdem auf den vorherigen Seiten die Anforderungen an Software im allgemeinen erläutert wurde, soll im Anschluss nun ein Blick auf die verschiedenen Möglichkeiten der Softwareentwicklung hinsichtlich der Verwendung von Servern betrachtet werden. Dazu wurden die Bereiche Server/Client, Web-Architektur, SOA und das Schichtenmodell ausgewählt, die nun kurz erläutert werden. Die wichtigsten Vor- und Nachteil der Möglichkeiten werden aufgelistet, um die eingangs gestellte Problemstellung nachvollziehen zu können.

### 2.2.1 Server/Client

Diese Variante der Programmierung besteht im wesentlichen daraus, dass die Komponenten aus Servern und Clients bestehen. Dabei können jeweils mehrere Komponenten in einem System verbaut sein. (Abts, 2015, S. 7) Die Funktionalität kann durch das gegenseitige Bearbeiten von Anfragen und Antworten beschrieben werden. Wird in einer Anwendung durch einen Benutzer eine Aktion ausgelöst, so sendet der Client

dabei eine Anfrage an den Server, dieser bearbeitet den Auftrag und sendet das Ergebnis zurück an den Client, der dann dem Anwender entweder die gewünschte Antwort anzeigt oder die Aktion, die aktiviert wurde, auslöst. Ein System mit einem Aufbau dieser Art basiert dabei mindestens auf je einem Client und einen Server. (Abts, 2015, S. 7) (Balzert, 2011, S. 193)

Die Anzahl der Server ist nicht auf einen limitiert, so dass die Performance der Anwendung nicht beschränkt ist. Ein Server kann seinerseits eine gestellte Anfrage eines Clients an einen anderen Server weiterleiten. Der weiterleitende Server fungiert dann selber als Client. (Abts, 2015, S. 7) (Bengel, 2014, S. 21 ff)

Zwischen Server und Client kann die Kommunikation auf zwei verschiedene Arten geschehen. Im Fall der synchronen Kommunikation wartet der Client auf eine Antwort vom Server und erst wenn er diese erhalten hat, wendet er sich anderen Aufgaben zu. In der Zwischenzeit macht der Client zwangsweise eine Pause (Schäfer, 2010, S. 218). Bei der asynchronen Kommunikation hingegen wartet der Client nicht auf die Antwort des Servers, sondern befasst sich umgehend mit anderen Aufgaben. Bei einer Rückmeldung des Servers werden dann Routinen für bestimmte Ereignisse beim Client ausgelöst (Schäfer, 2010, S. 218). Auch wenn die asynchrone Variante durch den Wegfall der Wartezeit für den Client zunächst von Vorteil erscheint, so darf nicht vergessen werden, dass die asynchronen Abläufe mehr Aufmerksamkeit bedürfen und es dort somit ein erhöhtes Fehlerrisiko durch die Anzahl der Prozesse, die zeitgleich ausgeführt werden, gibt. (Abts, 2015, S. 9)

## Tier-Architektur

Ein Server/Client System kann aus mehreren Ebenen bestehen. Ein klassisches System besteht aus zwei Ebenen und wird 2-Tier-Architektur genannt. Es besteht lediglich aus Server und Client. Die 3-Tier-Architektur besteht hingegen aus einer weiteren Komponente, die zum Beispiel aus einem Datenbankserver besteht. Bei mehreren Ebenen wird auch von Multi-Tier-Architekturen gesprochen. Eine Web-Anwendung zeichnet sich hingegen durch einen Aufbau mit vier Schichten aus. Dabei wird zusätzlich mit einem Web-Server gearbeitet. (Abts, 2015, S. 10) (Bengel, 2014, S. 14)

Es sind weiterhin verschiedene Aufbauten von Servern zu unterscheiden. Bei einem parallelen Server werden die eintreffenden Aufträge nicht ihrer Reihenfolge nach bearbeitet. Stattdessen wird für jeden Vorgang ein neuer Prozess gestartet. (Bengel, 2014, S. 29)

Innerhalb eines Client/Server Systems werden die Arbeitsabläufe in unterschiedliche Prozesse aufgeteilt, die dann von der jeweiligen Komponente verarbeitet werden. Dabei ist jedoch eine Unterart von Prozessen zu benennen, die als Threads bezeichnet werden. Um diese zu erhalten, wird ein Prozess in kleine Einzelteile zerlegt. Dies bietet den Vorteil, dass die kleinen Threads schneller verarbeitet werden können. (Bengel, 2014, S. 30)

Es existieren verschiedene Serverarten, die in der folgenden Tabelle kurz aufgezeigt und erläutert werden:

<b>Serverart</b>	<b>Beschreibung</b>
Shared Server	Prozess wird mehreren Services zur Verfügung gestellt
Unshared Server	Jeder Server hat einen anderen Service
Per Request	Erstellung neuer Server bei jeder Anfrage
Persistent Server	Server erhält sofort Anfragen, da er nicht erst aktiviert werden muss (Bengel, 2014, S. 46 f.)

Tabelle 1: Erläuterung von Serverarten

## Vorteile

- Schnellere Problembeseitigung durch Mitverfolgen der Benutzersitzungen
- Reduzierung der Komplexität verschafft übersichtlichen Aufbau
- Umsetzung für große Systeme geeignet
- Geringes Risiko eines Systemausfalls
- Einbau weiterer Clients hat keine Auswirkung auf das System (Balzert, 2011, S. 195) (Uni Ulm)

## Nachteile

- Server/Client Systeme sind nur schlecht skalierbar
- Wechsel der ausführenden Plattform führt zum Umbau der Anwendung und somit Zeit- und Umsatzverlust
- Höhere Kosten durch zusätzliche Hardware
- Nicht geeignet für sensible Daten, da Server anfällig für Angriffe von außen
- Einbau neuer Server stellt Risiko durch nicht kalkulierbares Verhalten dar (Balzert, 2011, S. 195) (Uni Ulm)

### 2.2.2 Web-Architektur

Bei den Web-Architekturen werden Web-Techniken eingesetzt, bei der eine Verteilung der unterschiedlichen Schichten auf die Clients und die Server erfolgt (Balzert, 2011, S. 196 f.). Dabei befinden sich jeweils Teile des GUI's auf verschiedenen Clients, welche im Browser ausgeführt werden. Im Gegensatz zu der Client/Server Architektur werden hier die Anfragen nicht vom Client an den Server, sondern an den Web-Server geschickt. Dieser beantwortet die Anfragen, anstelle des Servers. Weiterhin ist es bei Anwendungen im Web möglich, eine unbegrenzte Anzahl an Anwendern zu verwalten (Schwichtenberg, 2010). Dies ist bei einer Software auf einem Rechner nicht ohne Weiteres möglich, da sich die Kapazitäten anders verteilen. Ebenfalls gibt es im Web keine begrenzte Anzahl an potentiellen Nutzern, während auf einem Rechner nur die Nutzer vorhanden sind, die Zugang zum Rechner und der Software haben. Ein weiterer Unterschied zum Client/Server System ist der Zugriff auf die Laufzeitumgebung, die bei der Web-Architektur für die Entwickler nicht möglich ist. (Balzert, 2011, S. 197)

#### Vorteile

- Einsatz im Internet und Intranet möglich
- Gute Skalierbarkeit
- Umsetzung erlaubt große Anzahl an Anwender (Balzert, 2011, S. 200)  
(Schwichtenberg, 2010)

#### Nachteile

- Engpass beim Server
- Mitunter schlechte Antwortzeiten welche sich auf Image auswirken
- Sperrung von Plug-ins kann Aufrufen von Anwendung oder Teilen verhindern  
(Balzert, 2011, S. 200) (Schwichtenberg, 2010)

### 2.2.3 SOA (Serviceorientierte Architekturen)

Bei den serviceorientierten Architekturen handelt es sich um eine Zusammenfassung zusammengehörender Funktionalitäten (Horn). Durch die Kombination mehrerer Services können komplexe Anwendungen realisiert und verwaltet werden. Dabei ist es das Ziel, dem Nutzer die Möglichkeit zu geben, große Funktionssammlungen zusammenzufassen, um ad-hoc Anwendungen aus fast vollständig vorhandenen Services zu erstellen . (Balzert, 2011, S. 201)

### Vorteile

- Integration verschiedener Anwendungen
- Integration von Diensten und Programmen externer Anbieter
- Kosteneinsparung durch Verwendung von Fremdsoftware (Balzert, 2011, S. 203) (Horn)

### Nachteile

- Hoher Anfangsaufwand bei der Umsetzung
- Komplexe Abläufe führen zu hohem Zeit- und Kostenaufwand
- Entkopplung von Services ist aufwändig (Balzert, 2011, S. 204) (Horn)

## 2.3 Schichtenmodell

Das Schichtenmodell stellt ein häufig eingesetztes Architekturmuster dar (Schäfer, 2010, S. 203). Dabei werden die Daten, die vom Client angefragt werden, durch unterschiedliche Schichten geleitet, bis sie an ihr Ziel gelangen (Starke, 2014, S. 142). Jede Schicht stellt der nächsten Schicht unter sich bestimmte Dienste zur Verfügung, damit diese die Verarbeitung der Daten erfolgreich weiterführen kann (Balzert, 2011, S. 46). Die Struktur der einzelnen Schichten ist nach außen hin nicht sichtbar und somit geschützt und nicht einsehbar. Das Schichtenmodell ist besonders gut geeignet, um große Systeme übersichtlich gestalten zu können. Dadurch, dass die einzelnen Aufgaben auf die verschiedenen Schichten verteilt werden, ist die Belastung der einzelnen Schichten nicht so hoch und Fehler sind nur auf den jeweiligen Bereich beschränkt. Die Komponenten können beliebig untereinander aufeinander zugreifen und bieten so hohe Flexibilität. (Balzert, 2011, S. 46 f.) (Bengel, 2014, S. 12 f.)

Die möglichen Komponenten eines Schichtenmodells bestehen aus den folgenden Punkten:

- Benutzeroberfläche
- GUI
- Anwendung / Verarbeitung
- Datenmanagement
- Persistente Datenspeicherung / Daten (Bengel, 2014, S. 13)

### Vorteile

- Gute Strukturierung der Komponenten erzeugt Übersichtlichkeit
- Unabhängigkeit der Schichten voneinander verhindert Warten aufeinander
- Hohe Flexibilität für Entwickler durch nicht vorhandene Einschränkungen
- Schichten mit gleichen Diensten können bei einem Fehler untereinander ausgetauscht werden (Balzert, 2011, S. 52 f) (Starke, 2014, S. 143)

### Nachteile

- Durchlaufen verschiedener Schichten sorgt für lange Abarbeitungszeiten
- Unordnung in der Struktur durch unabhängige Schichten (Balzert, 2011, S. 53) (Starke, 2014, S. 143)

## 2.4 Zusammenfassung des Kapitels

In diesem Kapitel wurden die bisherigen Methoden in Bezug auf Server, die zum Aufbau von Anwendungen und Programmen seit vielen Jahren verwendet werden, aufgezeigt und kurz erläutert. Dabei fällt auf, dass die jeweiligen Möglichkeiten sich durch die Verwendung mehrerer Komponenten auszeichnen, die jeweils durchlaufen werden müssen, um ein Ergebnis auf eine Anfrage zu erhalten. Ein Ausfall einer Komponente kann unter Umständen zu einem Totalausfall des gesamten Systems führen und erzeugt somit Kosten und führt zu einem Vertrauensverlust bei Kunden. Durch die Vor- und Nachteile der jeweiligen Methoden wurde gezeigt, wie sich die bisherigen Möglichkeiten der Softwareentwicklung in der Praxis bewähren. Dies ist für die Problemstellung der Ausarbeitung als wichtig anzusehen, da so die Unterschiede zum Bereich der Serverless Programmierung im weiteren Verlauf deutlich werden.

### 3. Serverless Programmierung

Die vorliegende Ausarbeitung befasst sich vorrangig mit der Thematik der Serverless Programmierung, welche im Zuge dieses Kapitels aufgegriffen und erläutert werden soll. Dabei wird zunächst auf die Definition eingegangen, um den Begriff der Serverless Programmierung zu erläutern und einen Einstieg in den Bereich zu geben. Da der Bereich in der Softwareentwicklung als relativ neu angesehen werden kann, ist ein Einblick in die derzeitige Etablierung am Markt als nächsten Unterpunkt interessant, um sich einen Überblick verschaffen zu können. Abschließend sollen die Einsatzbereiche für Serverless Programmierung aufgezeigt werden. Dabei wird auf bereits aktuell verwendete Möglichkeiten ebenso eingegangen wie auch auf in Zukunft denkbare Verwendungen.

#### 3.1 Definition

Eine eindeutige Definition, was „serverless“ genau bedeutet, ist derzeit nicht existent (Roberts, 2016). Vielmehr ist es ein Begriff für eine Ansammlung von Gegebenheiten und Elementen. Allgemein wird aber unter Serverless Programmierung eine Möglichkeit der Softwareentwicklung verstanden, bei der es nicht mehr von Nöten ist, einen Server anzumieten (Sbarski, 2017b, S. 9 ff.). Stattdessen werden Kapazitäten bei einem Drittanbieter eingekauft, der sehr große Server besitzt und diesen Platz an Kunden vermietet (Janczuk, 2016). Was auf den ersten Blick vielleicht nicht spektakulär aussieht, bietet den Entwicklern von Software einige Erleichterungen. Dennoch sollte auch Serverless Programmierung genauso wie die bisher gängigen Möglichkeiten nicht wahllos verwendet werden. Was auf der einen Seite Vorteile für bestimmte Projekte aufweist, ist für andere Umsetzungen nicht empfehlenswert. Deshalb ist auch hier eine vorherige Auseinandersetzung mit den genauen Anforderungen an die zu erstellende Software zwingend notwendig. Weiterhin sollte bedacht werden, dass sich die Plattformen der Drittanbieter voneinander abgrenzen und auch diese Entscheidung am Anfang sehr genau getroffen werden sollte, da ein späterer Wechsel der Plattform große Probleme nach sich zieht. (Froehlich, 2016)

Die Art der Berechnung der Kosten für die Serverkapazität verändert sich hierbei grundlegend. Jeder Kunde zahlt dabei einen komplett anderen Preis als die anderen, da jeder nur das zahlt, was er auch tatsächlich an Serverleistung verwendet. (Chan, 2017)

Die meisten der webbasierten Anwendungen haben ein back-end und arbeiten mit Servern. Dies bedeutet die Verwendung diverser front-ends, damit der Anwender sie verwenden kann. Somit wird vorausgesetzt, dass eine Menge Leistung vorhanden sein

muss, um die hohe Anzahl der unterschiedlichen Prozesse bewerkstelligen zu können. Dies ist bei Serverless Programmierung nicht als relevant zu beachten, da so viel Leistung wie nötig erhalten werden kann (Froehlich, 2016).

Bei einer Anwendung werden die Daten durch mehrere Schichten des Systems transportiert, bevor sie in einer Datenbank landen oder von dort geladen werden. Das bedeutet, es müssen diverse Antworten generiert werden, welche wiederum zum Client geschickt werden. Für all diese Prozesse werden Server benötigt. (Balzert, 2011, S. 195) Bei einer „serverless“ Umsetzung ist das Durchlaufen der Schichten nicht mehr vorhanden, was den Nachteil der langen Abarbeitungszeiten verringert (Roberts, 2016).

Die Entwickler verlieren bei der Serverless Programmierung die Verantwortung für den oder die Server. Da ein Drittanbieter die Serverkapazität zur Verfügung stellt, wartet dieser auch seine Server. Demnach fällt dieser Aufgabenbereich hier weg und der Server ist immer auf dem aktuellsten Stand. (Froehlich, 2016) Weiterhin haben die aufgezeigten Möglichkeiten aus dem vorigen Kapitel gezeigt, dass es Probleme bei der Skalierbarkeit der Software geben kann. Dies ist beim Serverless Programmierung nicht mehr vorhanden, da die Skalierung automatisch geschieht. Wird mehr Serverleistung benötigt, so gibt der Drittanbieter automatisch mehr frei. Geeignet ist Serverless Programmierung besonders für Nanoservices. (Roberts, 2016)

Die Funktionsweise bei der Serverless Programmierung besteht aus der Definition von Events, welche die unterschiedlichen Funktionen auslösen. Insgesamt kann festgehalten werden, dass es weniger Komponenten als bei den bisherigen Möglichkeiten gibt und sich daraus eine Vereinfachung der Organisation ergibt. (Poccia, 2016, S. 5 ff.) Weiterhin wird die Möglichkeit geboten, Probleme, die bei verschiedenen Anwendungen auftreten, zeitgleich durch die Funktion des Adressierens an verschiedene Stellen zu lösen.

Das Ziel von Serverless Programmierung lässt sich abschließend wie folgt zusammenfassen: Die Entwickler sollen sich möglichst ausschließlich auf die Entwicklung des Codes und somit der Anwendung widmen und sich nicht mit anderen Bereichen auseinandersetzen, die von ihrem eigentlichen Ziel ablenken und das Risiko bieten, eine schlechtere Qualität oder Fehler zu erhalten. (Roberts, 2016)

## 3.2 Etablierung am Markt

Obwohl das Prinzip der Serverless Programmierung kein gerade erst neu entwickelte Variante der Softwareentwicklung darstellt, ist es doch bisher nahezu unbekannt und wird nur von verhältnismäßig wenigen Unternehmen verwendet. Wer versucht, zu dem Thema

Informationen zu erhalten, der wird auf die Problematik stoßen, dass es nur recht wenig Literatur dazu gibt. Bücher zum Beispiel sind im Prinzip nicht existent, werden allerdings, das sei hier angemerkt, bereits verfasst und bearbeitet und sollen im Jahr 2017 Einzug in den Büchermarkt erhalten. Dem Interessierten und Neugierigen bleiben zahlreiche Artikel, die sich mit der Thematik des „serverless“ befassen und einen Einblick in das geben, was derzeit im Kreise der Entwickler auf dem Vormarsch ist. Die Schwierigkeit, Informationen über diese Thematik zu besorgen, ergibt sich aus einer zumeist englisch-sprachigen Literatur. Auch dies ist der bisher eher nur leichten Verbreitung geschuldet. Es wird noch ein Weile dauern, bis die Welle der Begeisterung die Skeptiker der alten Schule erreicht hat.

Weiterhin ist es ein Problem, in der Welt der Softwareentwicklung neue Systeme und Möglichkeiten integrieren zu wollen. Die Entwickler halten sich zumeist an Altbekanntes und zeigen nur vage Bereitschaft, sich neuen Möglichkeiten zu öffnen. Hat ein Entwicklerteam die Möglichkeit A zu wählen, zu der es eine Unmenge an Literatur und Foren gibt, die bei der Erarbeitung durch zahlreiche Beispiele oder der Beseitigung von Problemen mit hilfreichen Informationen und Tipps helfen, dann ist es wenig verwunderlich, dass Möglichkeit B, die dies nicht aufwarten kann, dabei verliert.

Trotz aller Probleme, die bei der Etablierung am Markt auftreten, existieren bereits einige Anbieter, die die Möglichkeit einer Serverless Programmierung bieten. Zu diesen zählen unter anderem Bustle und der Riese Netflix, welche bereits mit der Integrierung von Serverless Programmierung arbeiten (Chan, 2017). Diese werden zumeist in bereits existente Plattformen wie Cloud-Systeme integriert und bieten so die Möglichkeit, auf die anderen zur Verfügung stehenden Dienste zugreifen zu können. Dadurch kann ein Programmierer bzw. ein Unternehmen eine komplette Anwendung durch die Verwendung einer dieser Plattformen realisieren. Einige der Angebote befinden sich jedoch noch in der Testphase und eignen sich daher nur bedingt für eine kommerzielle Umsetzung, da damit zu rechnen ist, dass sich an der Umsetzung der Möglichkeit der Serverless Programmierung im Zuge der Testphase noch einige Veränderungen ergeben könnten (Novet, 2016). Somit kann ein kompletter Neuanfang von Nöten sein oder bei zu großen Änderungen auch ein Umzug auf eine andere Plattform. Dennoch stellen die derzeit vorhandenen Plattformen eine Möglichkeit dar, sich mit der Serverless Programmierung auseinanderzusetzen und sich damit beschäftigen zu können, um einen Einblick in die Funktionsweise zu erhalten. Durch diesen Vorgang kann es später leichter sein, sich intensiver mit der Erstellung einer Anwendung durch eine Serverless Programmierung zu beschäftigen, da ein gewisses Grundverständnis bereits vorhanden ist.

### 3.3 Einsatzorte

Um zu zeigen, wie vielfältig Serverless Programmierung eingesetzt werden kann, werden in diesem Kapitel die Einsatzorte dieser Methode näher betrachtet.

Zuerst lässt sich allgemein festhalten, dass sich die Serverless Programmierung für jegliche Größe von Projekten eignet. Es ist egal, ob es sich nur um eine kleine Anwendung eines Anfängers der Programmierung handelt oder um ein großes Unternehmen, welches sehr viele Zugriffe auf die Anwendung vorweisen kann. Durch die schnellere Verarbeitung der Anfragen kommt es auch dort zu keinen Ausfallzeiten. (Sbarski, 2017)

Für den Aufbau von Webseiten eignet sich ein „serverless“ Aufbau, da dort viele Anwender zeitgleich auf die Anwendung zugreifen. Gerade für einen Online-Shop, der verschiedene Aktionen auf seiner Seite in Form von einer Suche, den Artikelseiten, den Kundenkonten und dem Bestellvorgang realisieren muss, ist eine geringe Latenz der Anwendung sehr wichtig.

Der Markt für mobile Anwendungen in Form von Apps erfährt seit einigen Jahren einen erheblichen Wachstum, was sich durch die stetige Verbreitung von Smartphones und Tablets begründet. Deshalb zeigen immer mehr Unternehmen Interesse an diesem Markt. Dabei müssen die Apps den Anforderungen der Anwender gerecht werden, die insbesondere Wert auf eine schnelle Reaktionszeit innerhalb der Anwendung legen. Deshalb eignet sich die Serverless Programmierung gut für solch eine Umsetzung.

Eine Real-Time Analyse, die darauf angewiesen ist, dass die jeweilige Anfrage möglichst schnell bearbeitet wird und das Ergebnis zeitnah vorliegt, ist ebenfalls ein Einsatzbereich, der sich für „serverless“ gut eignet, denn auch hier spielt die möglichst geringe Latenz eine große Rolle. (Sbarski, 2017)

Auch wenn dies keine klassische Umsetzung einer Anwendung darstellt, ist die Verwendung eines Aufbaus mit Serverless Programmierung um ein Backup von Daten, Datenbanken oder einer Anwendung erstellen zu können, eine weitere Möglichkeit des Einsatzes. (Mak, 2017)

Die Anzahl der Fotos auf Smartphones nimmt täglich zu. Jedoch haben die Bilder den Nachteil, dass sie eine große Dateigröße haben und ein Hochladen auf eine Plattform oder das Kopieren auf andere Geräte mehr Zeit benötigt. Deshalb gehen immer mehr Anwender dazu über, die Dateigröße der Bilder zu reduzieren. Damit dieser Vorgang schneller umgesetzt werden kann, ist eine „serverless“ Anwendung von Vorteil. (Boyd, 2016)

Für Anwendungen, die in der Cloud entwickelt werden, ist die Serverless Programmierung besonders geeignet, da sie bei den großen Anbietern solch einer Plattform auch erstellt werden kann. Ebenfalls kann die so erstellte Anwendung dann auch dort betrieben werden. Dies vereinfacht die Programmierung, da alles in der gleichen Umgebung vorgenommen werden kann. (Janakiram MSV, 2016)

Eine weitere Art von Anwendung findet im Internet und auch lokal als Software Anklang. Sie dient dazu, eine Datei in ein anderes Format umzuwandeln. Dabei kann es sich sowohl um Bilder handeln, die nicht dem gewünschten Format entsprechen oder um Textdateien, die nicht programmübergreifend geöffnet werden können. Auch hier lohnt sich der Einsatz von Serverless Programmierung, da die Konvertierung damit schneller umgesetzt werden kann. Gerade für die Online-Variante ist dies ein wichtiger Faktor. (Sbarski, 2017a)

Die Erstellung von Microservices erfordert eine schnelle und leicht in fremde Umsetzungen einzubauende Möglichkeit, damit sie leicht zu einer gesamten Anwendung zusammengefügt werden können. Die Event-basierten Funktionen einer Serverless Programmierung bieten sich dafür gut an. (Cuva, 2017)

Der Bereich des Internet of Things hat in letzter Zeit immer mehr an Bedeutung gewonnen. Um dort möglichst kleine Anwendungen konzipieren zu können, die nicht über viel Datengröße verfügen. Durch Serverless Programmierung wird dies ermöglicht.

Für Firmen, die gerade neu gegründet wurden und nun eine Erstellung einer Anwendung oder eines Online-Shops planen, ist es reizvoll, sich mit der Thematik der Serverless Programmierung auseinanderzusetzen, da sie nicht in alten Standards denken. Es bietet sich daher an, sich mit den aktuellen Möglichkeiten der Softwareentwicklung zu beschäftigen, um mit der Umsetzung des geplanten Projekts auf dem neusten Stand der Dinge zu sein und sich auch von anderen abheben zu können. Wenn so zudem die Performance der Anwendung verbessert werden kann, bietet es weitere Wettbewerbsvorteile, welche die Konkurrenz nicht vorweisen kann. Deshalb sind Start-Ups eine interessante Möglichkeit für einen Einsatz der Serverless Programmierung, da die Gründer und Mitarbeiter dort meistens auch mit einer anderen Art und Weise des Denkens an Umsetzungen herangehen als ein Programmierer, der seit 20 Jahren auf die gleiche Art und Weise Anwendungen umsetzt. (Boyd, 2016)

Mitunter gibt es Anwendungen, die sehr viele Anfragen auf das System produzieren, ansonsten aber wenig andere Funktionen haben als dies. Durch die große Anzahl an Events, die auf diese Art und Weise ausgelöst werden, wird die Anwendung extrem

belastet was die Performance angeht. Da sich die Serverless Programmierung unter anderem genau dieser Problematik widmet, ist sie für eine Anwendung, die sehr viele Events verarbeiten muss, als sinnvoll anzusehen. Die Anwendung hat so die Möglichkeit, die Verarbeitung schneller abwickeln zu können und der Anwender selber kann sich einer möglichst schnellen Bearbeitung seines Anliegens erfreuen. (Sbarski, 2017a)

Das Einsatzfeld der Erstellung von Statistiken ist ebenfalls gut geeignet für die Serverless Programmierung. Bei einer solchen Erhebung werden teilweise unzählige Zahlen ins System eingetragen, die dann verarbeitet werden müssen und dies kann mitunter eine Weile dauern. Ein Aufbau ohne Server umgeht das Durchlaufen diverser Schichten und spart deshalb Zeit, so dass die Ergebnisse eher vorliegen. (Janakiram MSV, 2017)

Ein weiterer Bereich, der darauf angewiesen ist, dass die Verarbeitung schnell funktioniert, ist die Datenverarbeitung. Gerade, wenn innerhalb einer Anwendung mit großen Datenbanken gearbeitet wird, sind dort viele Zeilen zu durchsuchen, wenn eine Aktion bzw. eine Suche in der Datenbank vorgenommen wird. Im kommerziellen Bereich ist es unbedingt nötig, dass der Kunde seine Daten schnell vorliegen hat. Hat die Konkurrenz eine schnellere Lösung anzubieten, kann dies einen finanziellen Schaden verursachen. (Janakiram MSV, 2017)

Anwendungen bestehen immer aus einer Reihe von Zeichen, die kodiert werden müssen. Handelt es sich jedoch dabei um sehr viele Zeichen, so kann die Kodierung aufwendig werden. Auch hier kann die Performance einer solchen Anwendung durch eine „serverless“ Umsetzung gesteigert werden.

Verallgemeinert lässt sich festhalten, dass jede Anwendung, bei der eine große Menge an Daten verwaltet und verarbeitet werden muss, sich für den Einsatz der Serverless Programmierung eignet. Sie ermöglicht es, schneller die gewünschten Ergebnisse zu erzielen und durch den einfacheren Aufbau wird die Performance deutlich erhöht. Im Bereich der Online-Anwendungen gibt es immer mehr Anbieter und jeder davon muss eine Möglichkeit finden, sich von der Konkurrenz abzuheben. Das Arbeiten ohne Server stellt daher eine neue Möglichkeit dar, die bisher wenig verbreitet ist und somit einen Vorteil gegenüber anderen Anbietern aufzeigt.

### 3.4 Zusammenfassung des Kapitels

Auf den Seiten des Kapitels 3 wurde das Themenfeld der Serverless Programmierung näher betrachtet. Der Verzicht auf einen Server, der angemietet werden muss, wirkt sich vor allem in der effektiveren Kostenberechnung nach tatsächlich genutzten Ressourcen aus. Die automatische Skalierbarkeit, die je nach Bedarf und Zugriffszahlen auf die Anwendung die Leistung der Anwendung selbstständig regelt, ist ein Gewinn für Unternehmen, die schwankende Anwenderzahlen haben und so Kosten sparen können. Die Erläuterung hat weiterhin einen kleinen Einblick gegeben, welche Vorteile sich aus der Serverless Programmierung ergeben. Dies ist für das Kapitel der Vor- und Nachteile interessant, um die dortigen Argumente besser nachvollziehen zu können. Nachdem nun die Randbedingungen dargestellt wurden, sollen im folgenden Kapitel Möglichkeiten aufgezeigt werden, mit welchen Mitteln diese umgesetzt werden können.

## 4. Vorstellung der Möglichkeiten Serverless Programmierung

Das vorherige Kapitel hat den Sinn der Serverless Programmierung und dessen Funktionsweise sowie der bisherigen Etablierung aufgezeigt, doch wurde bisher die tatsächliche Umsetzung nicht beachtet. Deshalb sollen in diesem Kapitel nun Möglichkeiten in Form von vier Plattformen aufgezeigt werden, die eine Umsetzung für Serverless Programmierung anbieten. Hier soll ein Überblick über die derzeit verfügbaren Funktionen und den Aufbau der Plattformen gegeben werden.

### 4.1 Google Cloud Functions

Google hat seine eigene Möglichkeit für Serverless Programmierung erstellt, die im Jahre 2015 vorgestellt wurde und sich derzeit in der Beta-Phase befindet (Novet, 2016). Sie ist Teil der Google Cloud Plattform und wird dort in Form eines Services zur Verfügung gestellt. Um Zugang zu der Plattform erhalten zu können, müssen Interessierte sich einen Account auf der Plattform erstellen. Google Cloud Functions bietet ein monatliches Freikontingent an, welches aus 2 Millionen Aufrufen besteht. Somit können private Nutzer kleinere Anwendungen mit Hilfe des Services ohne Kostenaufwand erstellen und verwalten (Google, b).

Die Funktionen werden dabei derzeit über eine Konsole eingegeben, da derzeit noch keine ausgereifte Version auf der Plattform selber vorhanden ist, die es ermöglicht, über eine Maske Funktionen zu erstellen (Google, a).

Durch die derzeitige Beta-Phase findet die Plattform im Gegensatz zu anderen Konkurrenzprodukten nur wenig Aufmerksamkeit. Auch Tutorials beziehen sich eher auf andere Anbieter, weshalb Google Cloud Functions es schwer haben dürfte, sich in dem Bereich der Serverless Programmierung zu etablieren bzw. sich durchzusetzen.

Die Nutzer müssen kein passendes System einrichten und keine Hardware oder virtuelle Maschine reservieren (Google Blog, 2017). Deshalb ist es auch gut für Einsteiger geeignet, die im Bereich der Programmierung kaum Erfahrung haben und sich mit Entwicklungsumgebungen nicht gut auskennen. So können sie schnell einen Einblick erhalten und Erfolge erzielen.

Der Service unterstützt derzeit nur JavaScript als Sprache (Casalboni, 2016). Jede der Funktionen lässt sich einzeln umsetzen und wird auch einzeln abgerechnet. Der Dienst der Plattform ist als Open-Source Software verfügbar. Zusätzlich sind Schnittstellen zu IBMs Computersystem Watson vorhanden, wodurch Entwickler Hilfe bei der Auswahl der für sie geeigneten Programmierschnittstellen erhalten können (Neumann, 2016).

Die folgende Grafik zeigt den Aufbau der Google Cloud Plattform. Hier hat der Entwickler die Möglichkeit, seine Dokumente zu verwalten und auszubauen.

The screenshot shows the Google Cloud Platform dashboard for App Engine. The left sidebar has 'Task queues' selected under 'App Engine'. The main content area is titled 'Task queues' with a 'REFRESH' button. Below it, there are tabs for 'Push Queues', 'Pull Queues', and 'Cron Jobs', with 'Cron Jobs' being the active tab. A descriptive text states: 'A cron job is a scheduled task that runs at a specific time or at regular intervals.' A table lists one cron job:

Cron job ^	Description	Frequency	Last run	Status
/dataflow/schedule	Cron job to schedule Dataflow pipeline	every 10 mins (GMT)	Has not run yet	<a href="#">Run now</a>

Abb. 1: Aufbau der Google Cloud Plattform (Quelle: Screen aus Google Cloud Platform)

## 4.2 Windows Azure Functions

Windows Azure Functions ist die Möglichkeit der Serverless Programmierung, die von Windows angeboten wird. Die Plattform wurde im März 2016 vorgestellt (Maiaroto, 2016). Die Tatsache, dass die Plattform noch sehr jung ist, macht sich auch bei der Literatursuche bemerkbar, da sie sich erst langsam in die Riege der „serverless“ Anbieter integriert und bei den Entwicklern verbreitet. Der Nachteil hier ist, dass andere Plattformen, die bereits früher einen Service auf den Markt gebracht haben, bereits von Programmierern verwendet wurden und sich diese nicht in jede Plattform einarbeiten. So haben es neue Möglichkeiten der Serverless Programmierung schwer, die Entwickler davon zu überzeugen, sich ihrer Plattform anzunehmen.

Die Plattform bietet von sich aus die Möglichkeit an Beispiele und Beispielfunktionen anzeigen zu lassen. So können Einsteiger die Funktionsweise besser nachvollziehen und erlernen. Weiterhin können sie diese verändern und dadurch den Aufbau der Funktionen besser verstehen. Die Plattform unterstützt C#, F#, Python und PHP und bietet somit eine gute Vielfalt für Entwickler (Microsoft Azure, 2017). Ein Vorteil der Plattform ist, dass Dateien auf viele Wege bereitgestellt werden können z.B. durch Git, Dropbox oder OneDrive (Microsoft Azure, 2016). Dies ermöglicht eine unabhängige Verwaltung der Dokumente und ist auch bei einer ortsunabhängigen Bearbeitung eines Projekts hilfreich für die Organisation der Mitarbeiter.

Ein Testen des Services zeigte, dass eine Funktion mit wenigen Klicks über eine

Maske erstellt werden kann. Jedoch werden weitere Einstellungen erst nach Erstellung der jeweiligen Funktion vorgenommen. Dazu erscheinen dann im Menü weitere Unterpunkte, die ausgewählt werden können. Allgemein können die einzelnen Services und Daten über ein Menü an der linken Seite der Plattform aufgerufen werden, welches seinerseits weitere verschachtelte Unterpunkte beinhaltet. So verliert ein Anfänger recht schnell den Überblick und wird eher verwirrt, als dass er sich dort zurecht findet, wenn er nicht gezielt weiß, wonach er suchen muss.

Das folgende Bild zeigt den Aufbau der Benutzeroberfläche von Windows Azure Functions. Hier hat der Entwickler die Möglichkeit, diverse Einstellungen vorzunehmen und den Code zu erstellen.

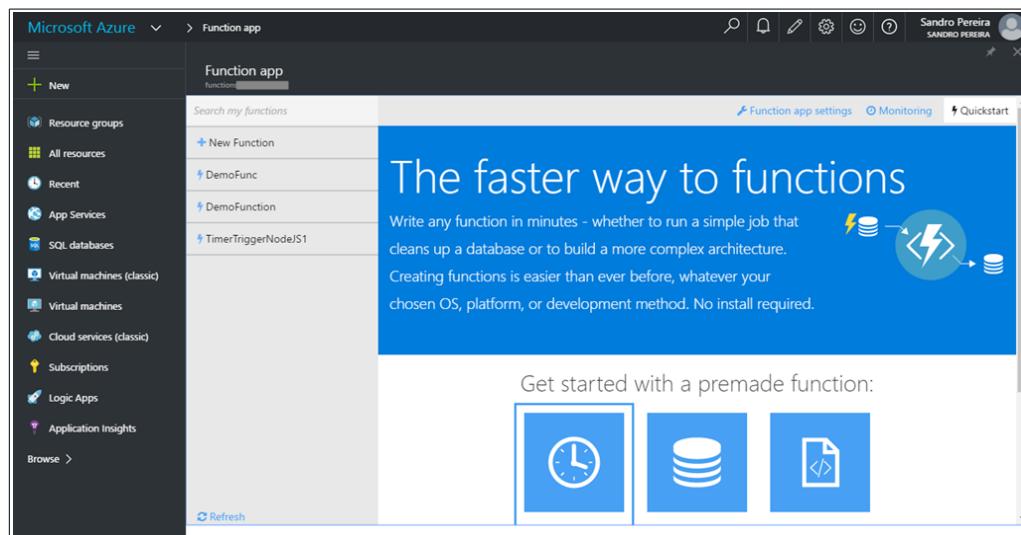


Abb. 2: Aufbau der Plattform von Windows (Quelle: <https://wazcommunity.wordpress.com>)

### 4.3 IBM Open Whisk

Auch IBM hat sich 2016 in die Reihe der Plattformen integriert, die Serverless Programmierung ermöglichen (Janakiram MSV, 2017). Ebenso wie die vorherigen beiden Plattformen ist diese nicht so weit verbreitet wie AWS Lambda, findet jedoch auch immer mehr Verwender. Durch den übersichtlichen Aufbau der Verwaltung können auch Einsteiger leicht die Möglichkeiten überblicken und finden sich dort gut zurecht. IBM Open Whisk ist gut für kleine Teams geeignet und bietet eine hohe Flexibilität. Weiterhin verfügt die Plattform über eine integrierte Containerunterstützung und erleichtert so den Entwicklern ihre Arbeit (Fink, 2016). Durch die Möglichkeit, einer ortsunabhängigen Bearbeitung eines Projekts mehrerer Mitarbeiter zur gleichen Zeit, ist eine flexible Verteilung von Aufgaben an unterschiedlichen Standorten eines Unternehmens möglich,

ohne dass diese alle am gleichen Ort sein müssen. Dies bietet die Möglichkeit, die Erfahrung aus unterschiedlichen Orten miteinander zu vereinen, um das bestmögliche Produkt zu erhalten.

IBM Open Whisk bietet die Möglichkeit, Microservices mit Node.js und Swift zu erstellen. Außerdem können weitere Sprachen integriert werden. Die Entwicklungsoberfläche bietet die Möglichkeit, verschiedene Services, die bereits integriert sind, einzubinden und anzuwenden. Die Verkettung von Microservices garantiert ein schnelles Arbeiten seitens der Entwickler. Weiterhin ist eine bedarfsgesteuerte Ausführung des Codes in einer hoch skalierbaren Server losen Umgebung gewährleistet. (IBM Cloud)

Zunächst wird jedem Anwender ein Testzeitraum von dreißig Tagen gewährt. Ähnlich wie AWS bietet IBM Bluemix die Möglichkeit von freien Kontingenten bei einigen der angebotenen Services, die der Anwender nutzen kann. So können Privatpersonen mit nur kleinen Projekten oder Anfänger kostengünstig arbeiten. Zudem kann durch einen integrierten Kostenrechner ermittelt werden, bei welchen Services welche Kosten anfallen und wie sich diese optimieren lassen. (IBM Cloud)

Die folgende Grafik zeigt den Aufbau der Oberfläche von Bluemix.

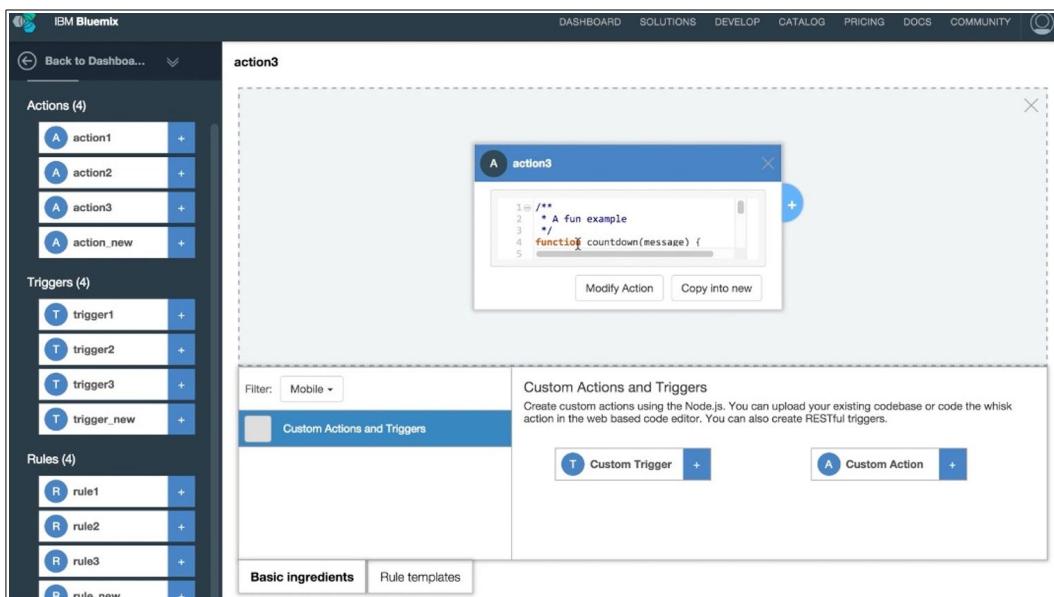


Abb. 3: Aufbau der Plattform von IBM (Quelle: <https://venturebeat.com>)

## 4.4 Amazon Webservices

Für die Umsetzung des Prototyps wird die Plattform AWS von Amazon verwendet, weshalb an dieser Stelle ein genauerer Blick auf eben diese Plattform erfolgen soll. So wird ein Überblick über den Aufbau und die Möglichkeiten, die dort angeboten werden, geschaffen. Weiterhin soll verdeutlicht werden, wieso AWS derzeit die am weitesten verbreitete Plattform seiner Art ist und wie sich diese Tatsache auswirkt.

### 4.4.1 Allgemeines

Die Plattform kann von jeglicher Art von Anwendern genutzt werden. Ein Privatnutzer, der nur kleine Anwendungen erstellen oder testen möchte ist ebenso willkommen wie ein großes Unternehmen, welches einen größeren Umfang an Serverleistung und Diensten nutzen möchte. Gerade Firmen gehen immer mehr dazu über, sich an Drittanbieter solcher Plattformen zu wenden und ihre Anwendungen zu erstellen und zu verwalten. Dort werden viele verschiedene Dienste und Services geboten, die alle vereint sind auf einer Plattform und es so ermöglichen, einen besseren Überblick zu haben und einfacher einzelne Elemente miteinander verbinden zu können.

AWS selber bietet dazu unterschiedliche Möglichkeiten der Preisgestaltung. So sind Kunden, die lediglich wenig Ressourcen nutzen, meistens befreit von monatlichen Kosten. Insbesondere, wenn es sich dabei nur um unregelmäßige Zugriffe handelt, werden monatliche freie Kontingente angeboten, die sich auf Zugriffe auf eine erstellte Anwendung oder auf genutzten Speicherplatz beziehen. (AWS freies Kontingent) Bei Unternehmen werden entweder Paketpreise angeboten oder nach exakt verbrauchten Daten abgerechnet. Dabei ist jedoch zu beachten, dass jeder Dienst eine andere Preisgestaltung hat und auch nicht überall ein freies Kontingent zur Verfügung steht. Deshalb sollte vorher genau recherchiert werden welche Dienste sinnvoll für das eigene Projekt sind, damit es nicht am Ende des Monats zu unerwarteten Kosten kommt. Manche der angebotenen Datenbanken zum Beispiel laufen unendlich weiter und generieren somit immer Kosten, solange sie laufen. Wer sich nur zur Einarbeitung damit auseinandersetzen möchte, sollte auf solche Details achten und regelmäßig die aktuelle Rechnung ansehen, um einen Überblick zu haben und zu behalten. (AWS Preise)

Der Anwender hat die Möglichkeit auf folgende Bereiche in AWS zuzugreifen:

- Compute
- Developer Tools
- Analytics

- Application Services
- Storage
- Management Tools
- Artificial Intelligence
- Messaging
- Database
- Security, Identity and Compliance
- Internet of Things
- Business Productivity
- Desktop und App Streaming
- Networking und Content Delivery
- Migration
- Game Development
- Mobile Services

Die folgende Grafik zeigt eine Übersicht über die jeweils zur Verfügung gestellten Services und Dienste von AWS.

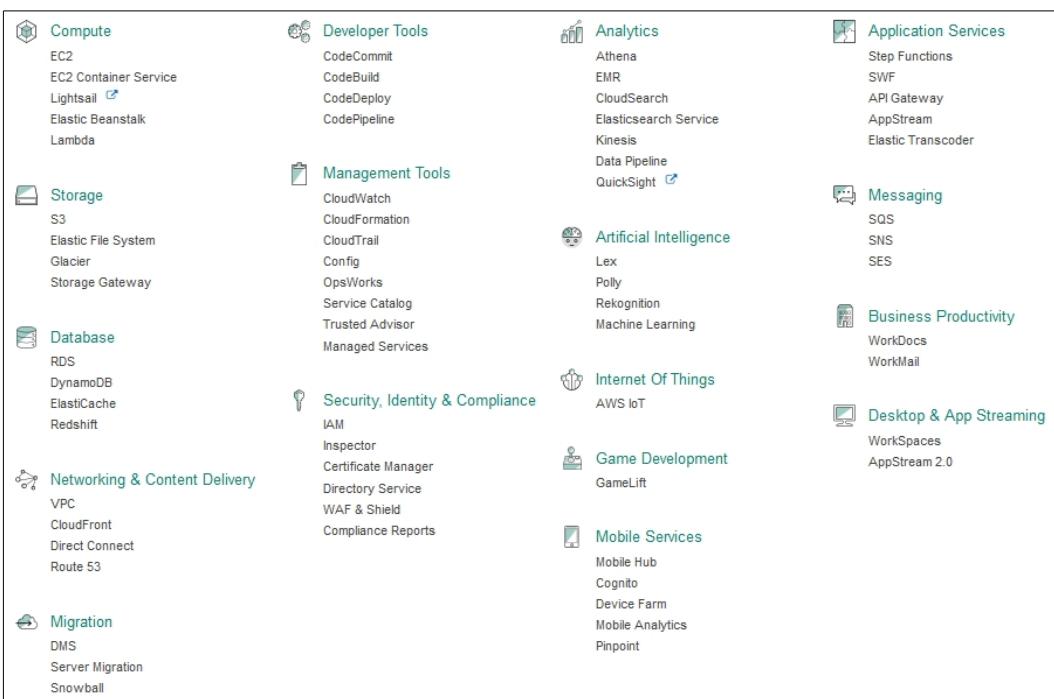


Abb. 4: Übersicht der Services von AWS (Quelle: <https://aws.amazon.com/de/>)

Hier wird deutlich, wie groß die Vielfalt ist, die AWS bietet und wieso so eine Plattform wie diese attraktiv für Programmierer und Unternehmen ist. Durch die Möglichkeit, verschiedene Datenbanken zu nutzen und diese mit anderen Services nutzen zu können, ebenso wie diverse Sicherheitseinstellungen, können hochwertige und aufwendige

Anwendungen konzipiert werden.

Die Dienste können dabei mitunter auf verschiedene Weisen verwendet werden. An sich hat jeder Service, der aus der Übersicht ausgewählt wird, eine Maske, mit deren Hilfe eine Bedienung dessen vorgenommen werden kann. Dies ist für Anfänger gut geeignet, da die Felder, die ausgefüllt werden müssen mit Beispieltexten hinterlegt sind. So können sich Anwender besser an den Einstellungen orientieren. Es gibt auch die Möglichkeit, über die AWS Command Line zu arbeiten. Diese kann für verschiedene Betriebssysteme lokal installiert und über die Eingabeaufforderung des Systems bedient werden. Durch die Eingabe der Zugangsdaten zum AWS Account wird Verbindung mit diesem aufgenommen. Anschließend können die verschiedenen Services von AWS angesteuert und verwendet werden.

Einige Services arbeiten statt einer definierten URL, wie bei einem S3 Bucket, mit so genannten ARN's. Diese werden für Tabellen in DynamoDB, Lambda-Funktion oder Roles bei dessen Erstellung erzeugt. Sollen nun Zugriffsrechte definiert werden, dann wird dies mit Hilfe der Angabe der jeweiligen ARN realisiert. Die ARN selber besteht aus der Information des verwendeten Services, der ID des AWS Accounts, welcher für die Erstellung des Elements verwendet wird und aus dem Namen des Elements. (AWS ARN) Die folgende Grafik zeigt dazu ein Beispiel:

**ARN - arn:aws:lambda:eu-central-1:934888181710:function:imgae-index**

Abb. 5: Beispiel einer ARN einer Lambda-Funktion (Quelle: Eigene Lambda-Funktion auf <https://aws.amazon.com/de/>)

Um mit AWS arbeiten zu können, muss der Interessent erst einen Account dort erstellen. Für Kunden vom Onlinehändler Amazon ist diese neue Registrierung nicht nötig, da sie sich mit den schon vorhandenen Anmeldedaten auch bei AWS einloggen können. (AWS Konto)

Wird ein Service das erste Mal verwendet, dann erscheint bei vielen die Auswahlmöglichkeit „getting started“. Unter dieser Option verbirgt sich eine Einführung in die Verwendung und Erstellung des jeweiligen Services. So können Anfänger recht schnell ein Verständnis für den Aufbau erlernen und zeitnah erste Erfolge erzielen. Gerade bei einer so komplexen Plattform, wie AWS sie darstellt, ist dies als sehr hilfreich und sinnvoll anzusehen.

Werden für eine Anwendung, die entwickelt werden soll, zusätzliche Bibliotheken

und Module benötigt, so können diese einfach nachinstalliert und eingebunden werden. Dazu muss auf dem Rechner „npm“ installiert werden. Dies kann dann in der Eingabeaufforderung des jeweiligen Systems verwendet werden. Die so installierten Dateien können dann in den Zip-Ordner mit den Funktionen und Codedateien eingefügt und hochgeladen werden. Der Aufruf der Bibliotheken erfolgt dann in der Code-Datei, in der sie verwendet werden soll. (NPM)

Die Sprache auf der Plattform AWS ist in Englisch gehalten und kann lediglich in Japanisch verändert werden. Damit bietet sie eine einheitlich international gültige Sprache, die in sämtlichen Ländern der Welt verwendet wird. So können auch Programmierer verschiedener Länder oder Unternehmen mit Standorten auf der ganzen Welt miteinander arbeiten und die Daten gemeinsam nutzen. (AWS Sprache, 2016)

#### 4.4.2 Regionen

Allgemein kann in AWS die Region ausgewählt werden, in der sich der Anwender befindet und die Anwendung laufen soll. Dies dient dazu, damit möglichst die am nächst liegende Region vom Programmierer ausgewählt werden kann, um so die Latenz zu reduzieren und schnellere Zugriffszeiten erhalten zu können. Dies ist gerade für größere Anwendungen, die eine Vielzahl an Zugriffen haben, wichtig zu berücksichtigen, da das System sonst zu langsam arbeitet und somit mögliche Kunden abschreckt. Zur Verfügung stehen dabei folgende Hauptregionen: US-west, US-east, Canada, EU, Asia Pacific und South America. Jede dieser übergeordneten Regionen hat Unterregionen, die sich auf bestimmte Städte beziehen. Insgesamt gibt es 14 verschiedene dieser Unterteilungen. (AWS Regionen)

Dabei ist unbedingt zu beachten, dass nicht jeder Service, den AWS zur Verfügung stellt, in jeder der Regionen verfügbar ist. Deshalb sollte sich ein Programmierer oder Unternehmen von vornherein darüber bewusst sein, welche Ressourcen und Möglichkeiten für das anvisierte Projekt benötigt werden. Wird dabei etwas übersehen, kann es vorkommen, dass ein Teil nicht umgesetzt werden kann, weil der benötigte Service nicht in der am nächsten gelegenen Region verfügbar ist. Natürlich kann in dem Fall einfach eine Region ausgewählt werden, die den gewünschten Service anbietet. Dies ist jedoch nicht ohne Probleme durchführbar, da es einen Mehraufwand für den Programmierer bedeutet und sollte unbedingt vermieden werden. Da AWS an sich über eine sehr gute Dokumentation über ihre Plattform und deren Services verfügt, sollten diese Informationen im Vorfeld auch genutzt und hinreichend studiert werden. (AWS Regionen) Aus einem Wechsel der Region ergibt sich zudem die Problematik einer geteilten

Rechnungsstellung, da jede Region ihre eigene Rechnung generiert. Dies birgt die Gefahr, dass eine Rechnung übersehen wird und der AWS Account als Folge gesperrt werden kann. (Eigene Erfahrung bei AWS)

Um eine Anwendung für den Browser zu erstellen, können HTML Dateien verwendet werden, die dann mit einer ausführenden JavaScript Datei und den gewünschten AWS Services verbunden werden. Verschiedene Elemente, die nicht in AWS integriert sind, können zum Aufbau der gewünschten Anwendung in unterschiedlichen Bereichen erstellt werden ohne dabei auf Flexibilität in den Gestaltungsmöglichkeiten verzichten zu müssen.

#### 4.4.3 Roles

Bei den so genannten Roles handelt es sich um Rollen, denen bestimmte Zugriffsrechte zugewiesen werden können. Dabei beziehen sich die Einstellungsmöglichkeiten maßgeblich auf die verschiedenen Services, die AWS bietet. Eine Rolle kann demnach zum Beispiel Zugriffsrechte für ein S3 Bucket beinhalten und erlauben, dass get und put Vorgänge vorgenommen werden dürfen. (Poccia, 2016, S. 7 ff.)

AWS bietet von sich aus bereits eine große Anzahl vordefinierter Roles. Es besteht die Möglichkeit, diese mit weiteren Rechten und Einstellungen zu ergänzen oder mit eigenen Rollen je nach Anwendung. Sie sind somit effizienter als allgemein vorgegebene Einstellungen. Die Rollen lassen sich dabei bei verschiedenen Services verwenden und können auch bei mehreren Lambda Funktionen zeitgleich eingesetzt werden. (AWS Guide)

### 4.5 AWS Lambda

AWS Lambda gehört zum Anbieter Amazon und wurde 2014 vorgestellt (Janakiram, 2016). Eine nähere Suche im Internet zum Thema Serverless Programmierung zeigt, dass AWS Lambda häufig im Zuge von Fachartikeln verwendet wird. Ebenfalls existieren viele Tutorials dazu, so dass hier eine große Verbreitung von Lambda zu erkennen ist. Zusätzlich ist eine ausgiebige Dokumentation vorhanden, welche die Möglichkeiten der Verwendung aufzeigt und näherbringt (AWS). Da AWS Lambda durch seine Einführung 2014 eine der ersten Plattformen war, die Serverless Programmierung ermöglicht hat, ist dementsprechend auch verhältnismäßig viel Literatur in Form von Anleitungen und Foren vorhanden. So wird es Einsteigern erleichtert sich damit zurecht zu finden (Janakiram, 2016).

AWS Lambda unterstützt standardmäßig Node.js und Python und bietet den

Verwenden der Entwicklungsumgebung Eclipse die Möglichkeit, zusätzliche Plug-ins einbinden zu können, welche den Umgang mit den Funktionen erleichtert. (Janakiram, 2016) Weiterhin können auch andere Programmiersprachen wie Java verwendet werden, diese sind jedoch nicht direkt in AWS verwendbar, sondern müssen erst extern eingebunden werden (Zub; Sack, 2016). Dennoch werden so viele Anwendungsmöglichkeiten und vor allem Flexibilität für Programmierer und Unternehmen angeboten.

Eine neue Funktion kann recht einfach und übersichtlich erstellt werden, was im Zuge der Einarbeitung in die Thematik der Serverless Programmierung erlernt wurde. Dazu wählt der Anwender aus der Übersicht der Services unter dem Punkt „Compute“ den Eintrag Lambda aus. Danach öffnet sich eine Oberfläche, auf der alle bisher erstellten Funktionen in der gewählten Region aufgelistet sind. Dort kann der Programmierer die vorhandenen Funktionen aufrufen, verändern oder neu erstellen. Als nächstes erscheint eine Auswahl an sogenannten Blueprints. Diese stellen Beispiele von verschiedenen Einsatzmöglichkeiten dar, die verwendet werden können. Die erste Auswahl in der Übersicht zeigt einen blanko-Blueprint, bei dem der Anwender alle Einstellungen selber vornehmen muss. Nachdem eine Auswahl getroffen wurde, erscheinen die Einstellungsmöglichkeiten. Dabei wird zunächst der Name der Funktion nebst einer Beschreibung dieser festgelegt. Anschließend folgt ein Codefeld, in welches der Code für die Lambda-Funktion geschrieben werden kann. Darüber befindet sich eine Drop-Down Liste, aus der die Art, wie der Code integriert werden soll, ausgewählt werden kann. Standardmäßig ist die Eingabe des Codes direkt in der Maske voreingestellt. Die anderen beiden Optionen ermöglichen es entweder den Code per Zip-Datei oder per Integration aus S3 zur Verfügung zu stellen. Weiterhin muss ausgewählt werden, welche Sprache zur Ausführung verwendet werden soll. Dabei steht Node.js, Java, C#, Python und Edge Node.js zur Verfügung. Andere Sprachen können über Umwege verwendet werden, wenn gewünscht. Auch hier gibt es eine Voreinstellung, die in diesem Fall Node.js ist. Nachdem der Code erstellt oder integriert wurde, muss der gewünschte Handler sowie die Rolle ausgewählt werden, die für diese Funktion verwendet werden soll. Dabei kann jede zuvor erstellte Rolle verwendet oder aber eine neue erstellt werden. Ein Handler ist eine Funktion im Code, die von Lambda ausgeführt werden kann, wenn die entsprechende Lambda-Funktion abgerufen wird. Dabei muss der Name des Handlers dem Dateinamen der hochgeladenen Datei, welche den Code beinhaltet, entsprechen. Wurde Inline-Code verwendet, dann heißt der Eintrag immer „index.handler“. Wenn ein ZIP-Ordner mit einer Funktion mit Namen „Test“ hochgeladen wurde, dann lautet es „Test.handler“.

Bei der Auswahl der Angaben zur Größe des Arbeitsspeichers der Funktion kann

zwischen 128 MB und 1,5 GB gewählt werden. Standardmäßig ist hier 128MB voreingestellt. Bei größeren Funktionen sollte unbedingt darauf geachtet werden, eine entsprechende Größe des Arbeitsspeichers zu wählen.

Mit der Einstellung „timeout“ kann festgelegt werden, wie lange die Lambda Funktion höchstens ausgeführt werden soll. Dabei ist ein maximaler Wert von 300 möglich. Standardmäßig steht der Timeout auf drei Sekunden, aber kann jederzeit wieder geändert werden in den Einstellungen der Funktion, auch nachdem diese bereits gespeichert und verwendet wurde.

Nachdem die weiteren Einstellungen vorgenommen wurden, erhält der Anwender eine Zusammenfassung. Anschließend kann die Funktion erstellt und verwendet werden. Diese kann nun getestet oder verändert werden. Dabei können Änderungen an dem Code sowie an den Einstellungen erfolgen. (Poccia, 2016) (Sbarski, 2017) (Lambda Documentation, S. 5 ff.)

Weiterhin sorgt Amazon für eine regelmäßige Aktualisierung der Server, so dass eventuell auftretende Komplikationen minimiert werden. Mit AWS Lambda können neue Backend-Services für Anwendungen erstellt werden, die On-Demand ausgelöst werden und dazu die Lambda API oder mit Amazon API Gateway entwickelte benutzerdefinierte API-Endpunkte verwenden. Außerdem kann mit AWS Lambda eine benutzerdefinierte Logik zu AWS-Ressourcen wie Amazon S3-Buckets und Amazon Dynamo DB-Tabellen hinzugefügt werden. Durch die nutzungsabhängige Zahlung zahlt der Kunde nur das, was er auch wirklich an Serverleistung verbraucht hat ohne fixe Kosten eingehen zu müssen. (n.A., 2016b) (Lambda Documentation, S. 126 ff.) ()

## Trigger

Ein Trigger wird dazu verwendet, um einen Vorgang auszulösen. Bei einer Lambda Funktion kann ein Trigger verwendet werden, um bei einem bestimmten Ereignis eben diese Funktion auszulösen. Wird festgelegt, dass der Trigger ein S3 Bucket ist und das Event dahinter bei erstellten Objekten Verwendung finden soll, dann wird die betroffenen Lambda Funktion immer ausgelöst, sobald in dem betreffenden Bucket ein Objekt erstellt wird. So lässt sich das Auslösen von Funktionen gezielt steuern, insbesondere in Hinsicht auf das durchgeführte Event. So wird die Funktion nicht bei jeder Art von Vorgang ausgelöst. (Poccia, 2016, S. 24) (Lambda Documentation, S. 1 ff.)

Die folgende Grafik zeigt mögliche Kombinationen der AWS Komponenten für Serverless Programmierung.

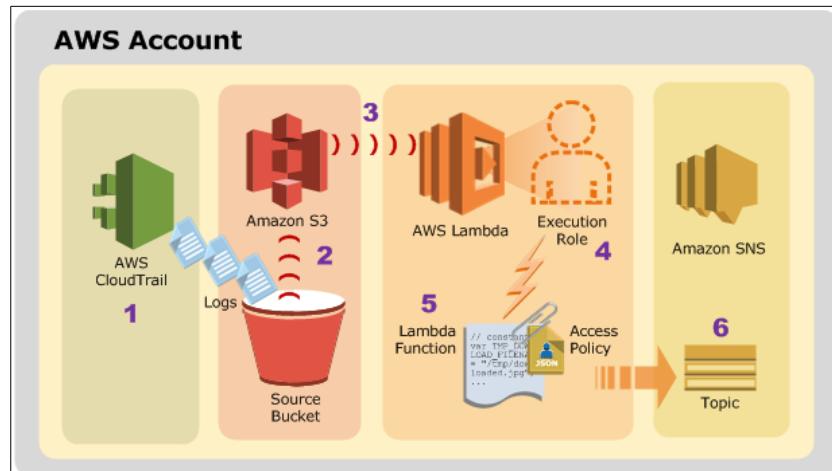


Abb. 6: Möglichkeiten der Verknüpfung von Services in AWS (Quelle: <http://docs.aws.amazon.com>)

Wenn es um das Thema Literatur geht, steht AWS Lambda im Fokus der Publikationen und in Artikeln im Internet, was die Recherche zu der Ausarbeitung gezeigt hat. Dies ist mitunter der Tatsache geschuldet, dass andere vergleichbare Plattformen, die sich mit der Thematik des Serverless Programmierung beschäftigen, erst später auf dem Markt erschienen sind und sich deshalb diverse Autoren und IT-Fachleute mit AWS Lambda auseinandersetzt haben. Da jede Plattform einen eigenen Aufbau hat und mit unterschiedlichen Services und Diensten arbeitet, ist es nicht möglich, diese miteinander zu kombinieren. Wer sich mit einer Plattform beschäftigt hat, der wird sich aller Wahrscheinlichkeit nach nicht noch mit einer weiteren auseinandersetzen.

#### 4.6 Zusammenfassung des Kapitels

In diesem Kapitel wurden die Möglichkeiten aufgezeigt, die derzeit bestehen, um Serverless Programmierung verwenden zu können. Dabei wurden die vier größten Plattformen vorgestellt, die bereits auch vorher schon existent waren, aber Services integriert haben, um den Bereich der Serverless Programmierung aufzugreifen. Teilweise befinden sich diese Services noch in der Testphase, weshalb dort mit stetigen Veränderungen zu rechnen ist. Die Plattform AWS mit dem Service Lambda wird in der Literatur im Internet am häufigsten genannt und findet viel Verwendung in Tutorials und gilt derzeit als am weitesten verbreitete Plattform diesbezüglich.

## 5. Erläuterung des Prototyps

Im Zuge dieser Ausarbeitung soll ein Prototyp erstellt werden, der die auf den vorherigen Seiten vorgestellten Möglichkeiten der Serverless Programmierung aufgreift und verdeutlicht. Durch die praktische Vorführung einer gezielten Anwendung lassen sich die Vorteile zeigen und es findet eine Verknüpfung mit dem theoretischen Teil des Dokuments statt. Zunächst werden daher die Anforderungen und Ziele, die an den Prototyp gestellt werden, erläutert. Anschließend wird der Aufbau der Anwendung betrachtet und die einzelnen Elemente, die dabei Verwendung finden, erläutert.

### 5.1 Anforderungen und Ziele

Die Anforderungen an diesen Prototyp beziehen sich maßgeblich darauf verschiedene Elemente, die AWS bietet, miteinander zu vereinen und so die Flexibilität und die Komplexität der Plattform in Verbindung mit Serverless Programmierung zu zeigen.

Weiterhin soll der Prototyp so aufgebaut sein, dass es die Möglichkeit gibt, für die nachfolgende Performance-Messung Daten sammeln und vergleichen zu können für einzelne Aktionen der Anwendung und der Anwendung an sich. Dies setzt voraus, dass auch wissenschaftliche Mittel genutzt werden, die eine Messung ermöglichen.

Geplant ist dabei, eine Anbindung an eine Datenbank zu ermöglichen, in der Daten gespeichert werden, die Anwender durch Aktionen in der Anwendung abrufen können. Weiterhin soll auch eine Speicherung von Daten in Form von Bildern und Dateien der Anwender erfolgen können. Die Anwendung an sich soll mit Hilfe von HTML erstellt werden.

### 5.2 Aufbau

Der konkrete Aufbau der Anwendung soll dabei die folgenden Teile und Elemente beinhalten:

- HTML Gerüst
  - Begrüßungsseite / Startseite
  - Bild hochladen ohne Ordner mit Link
  - Bild hochladen in einem Ordner mit Link
  - Datei hochladen ohne Ordner mit Link
  - Datei hochladen in einem Ordner mit Link
  - Bildgröße ändern

- Datenbank
  - speichern der hochgeladenen Dateien
  - speichern des Ordnernamens
- AWS
  - Lambda Funktionen
    - Bild in Datenbank speichern
    - Bildgröße ändern
  - Datei in Datenbank speichern
  - Roles
  - Policies
  - S3 Bucket
    - HTML Dateien
    - JacaScript Dateien
    - hochgeladene Bilder/Dateien
    - hochgeladene Bilder/Dateien in Ordnern

Die folgende Grafik zeigt das UseCase Diagramm für den Prototyp, welches die Auflistung der Funktionen veranschaulichen soll:

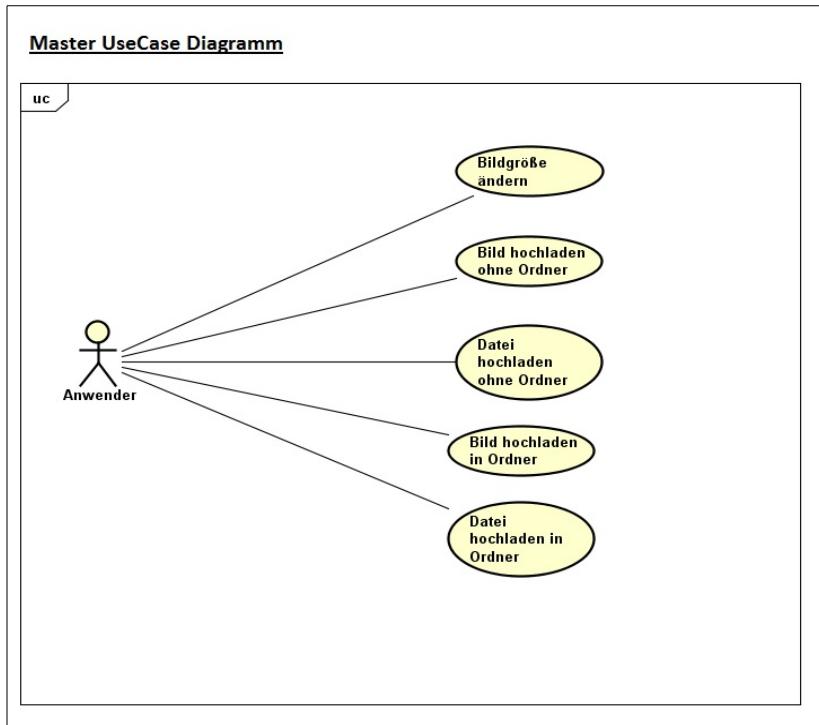


Abb. 7: UseCase Diagramm zur Übersicht des Funktionsaufbaus der Anwendung

Um auch die verwendeten Elemente der Anwendung grafisch darzustellen, wurde ein Komponentendiagramm erstellt, welches die Services von AWS und dessen Verbindung untereinander darstellt:

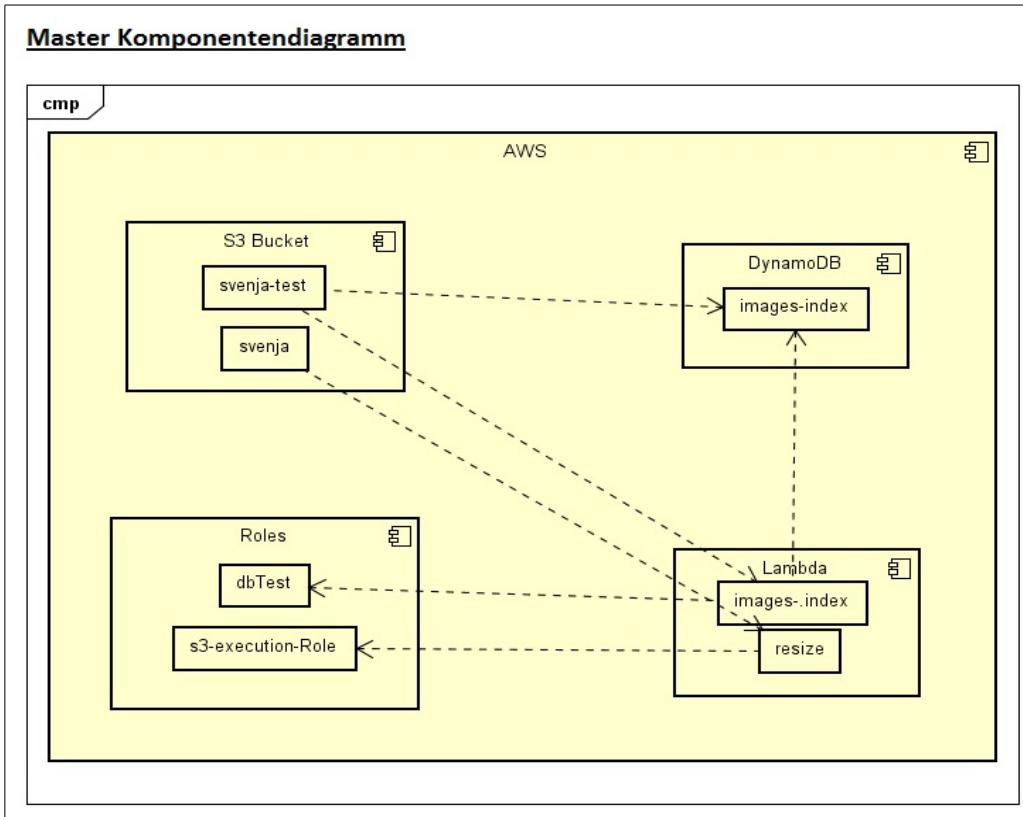


Abb. 8: Komponentendiagramm zur Übersicht der verwendeten Elemente in AWS

Die Anwendung hat demnach eine HTML Homepage als Anwendungsoberfläche, auf der die einzelnen Aktionen angewählt und ausgeführt werden können. Zunächst sehen alle Anwender eine Begrüßungsseite, die kurz die Anwendung erläutert und zeitgleich als Startseite fungiert. An der oberen Leiste der Anwendung erscheint ein Menü, über welches der Anwender zu einer neuen Unterseite gelangt.

Die Anwender können über einen Button die Datei von ihrem Rechner auswählen, die hochgeladen werden soll. Durch einen Klick auf den Hochladen-Button wird der Vorgang dann gestartet.

Bei der Funktion zum Hochladen in einen Ordner muss der Anwender einen Ordnernamen angeben. Ist ein Ordner bereits existent, können weitere Bilder in diesen durch das Eintragen des Ordnernamens hochgeladen werden. So können Bilder thematisch sortiert und verwaltet werden. Weiterhin können durch Eingabe des Ordnernamens alle bisher hochgeladenen Dateien eingesehen werden. Diese werden dann untereinander

aufgezeigt. Die Bilder sind verlinkt, so dass sie durch Anwählen in einem neuen Tab des Browserfenster geöffnet werden.

Beim Verwendung der Funktion ohne Ordner kann nur die jeweils gerade hochgeladene Datei in Form des dazugehörigen Links einsehen. Dieser wird mit der Vorschau erzeugt und kann, nachdem die Datei hochgeladen wurde, aufgerufen werden.

Die Anwendung ist so konzipiert, dass sie sowohl für den privaten als auch für einen geschäftlichen Gebrauch verwendet werden kann. Eine Privatperson kann seine Bilder und Dateien archivieren und Familie oder Freunde durch die Weitergabe der Anmeldedaten integrieren. In einem Unternehmen kann die Anwendung für eine Projektgruppe genutzt werden. Insbesondere für einen Betrieb mit verschiedenen Standorten ist dies praktisch. Jeder Mitarbeiter des Projekts kann einen eigenen Ordner erstellen, in den er seine Dokumente hochlädt. Weiterhin werden verschiedene Arten der Umsetzung angewendet. So können Bilder und Dateien über die gleiche Lambda-Funktion hochgeladen werden. Dies verdeutlicht die Wiederverwendbarkeit von Lambda-Funktionen.

### 5.3 Zusammenfassung des Kapitels

Das Kapitel hat den Aufbau des Prototyp hinsichtlich der Funktionen und verwendeten Elementen aufgezeigt. Die Komponenten bestehen dabei aus der Verwendung von den Services S3, DynamoDB und Lambda sowie verschiedenen Roles bei AWS. Die Oberfläche wird durch HTML und CSS aufgebaut und mit JavaScript-Dateien verbunden, welche die Funktionen der ausführbaren Aktionen definieren. Weiterhin wurde das Konzept samt Hintergrund erläutert und die Funktion der unterschiedlichen Aktionen, die in der Anwendung ausgelöst werden können, beleuchtet. Durch die UML Diagramme wurde der Aufbau der Anwendung visualisiert. Es wurde ein Verständnis über den Aufbau der Anwendung geschaffen, welcher für das nächste Kapitel unabdingbar ist.

## 6. Erläuterung Erarbeitung des Prototyps

Nachdem im vorherigen Kapitel der Aufbau des Prototyps hinsichtlich der einzelnen Elementen näher betrachtet und erläutert wurde, soll nun die Erarbeitung Gegenstand dieses Kapitels sein. Dazu werden die unterschiedlichen Einzelteile, die für die Anwendung verwendet werden, aufgezeigt und wichtige Funktionen und Bestandteile des Codes näher erläutert.

### 6.1 Erarbeitung des Prototyps

Die folgende Grafik zeigt die Startseite der Anwendung, um einen ersten Eindruck zu erhalten.

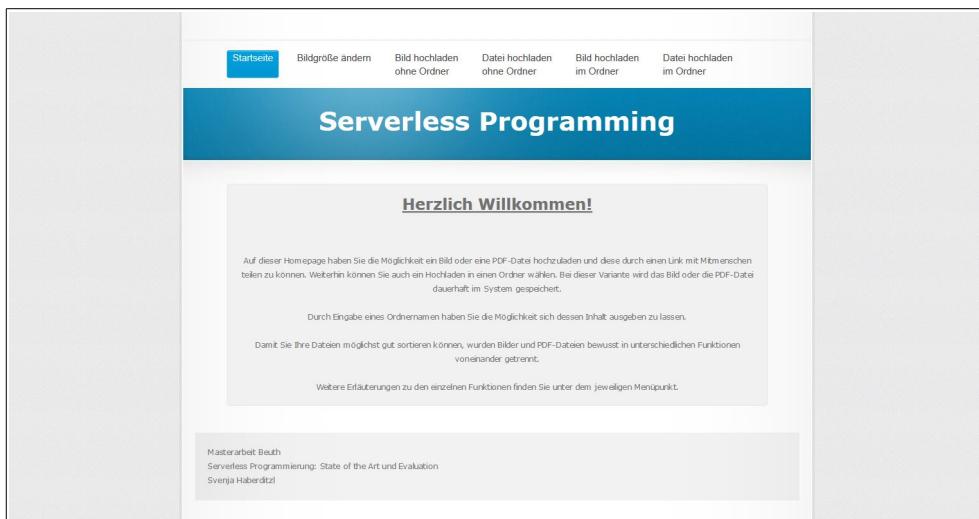


Abb. 9: Startseite der Anwendung

Zuerst wurde dabei das HTML-Gerüst erstellt, um eine Oberfläche für die Anwendung zu erhalten. Dazu wurde eine index.html sowie fünf weitere Unterseiten, die von der Hauptseite verlinkt werden, erstellt. Auf jeder dieser Seiten befindet sich am oberen Rand ein Menü, über welches die einzelnen Funktionen der Anwendung erreicht werden können. Dieses taucht in jeder der HTML-Dateien auf, damit eine übersichtlichere Möglichkeit der Navigation sowie der Funktionen an sich entsteht und es eine übersichtliche Einbindung der JavaScript Dateien ermöglicht.

Da es bei der Anwendung vorrangig um die Umsetzung einer „serverless“ Möglichkeit zum Verdeutlichen der auf den vorherigen Seiten erläuterten Informationen geht, wurde dem Design nicht das Hauptaugenmerk geschenkt, weshalb es nicht mit kommerziellen Umsetzungen mithalten kann. Jedoch wurde darauf geachtet, dass es dennoch ansprechend aussieht und vorzeigbar ist.

Um die Funktionen zum Laufen bringen zu können, müssen die später erläuterten JavaScript-Dateien in den HTML-Dateien in Form einer Einbindung miteinander verknüpft werden.

Die Gestaltung der Anwendung wurde mit Hilfe von CSS realisiert, dabei wird eine externe Datei erstellt, die jeweils in jede HTML-Datei eingebunden wurde. Die folgende Grafik zeigt eine Unterseite der Anwendung auf. Dabei wurde die Funktion „Bild hochladen im Ordner“ ausgewählt, um die Anzeige des Ordnerinhalts zeigen zu können.

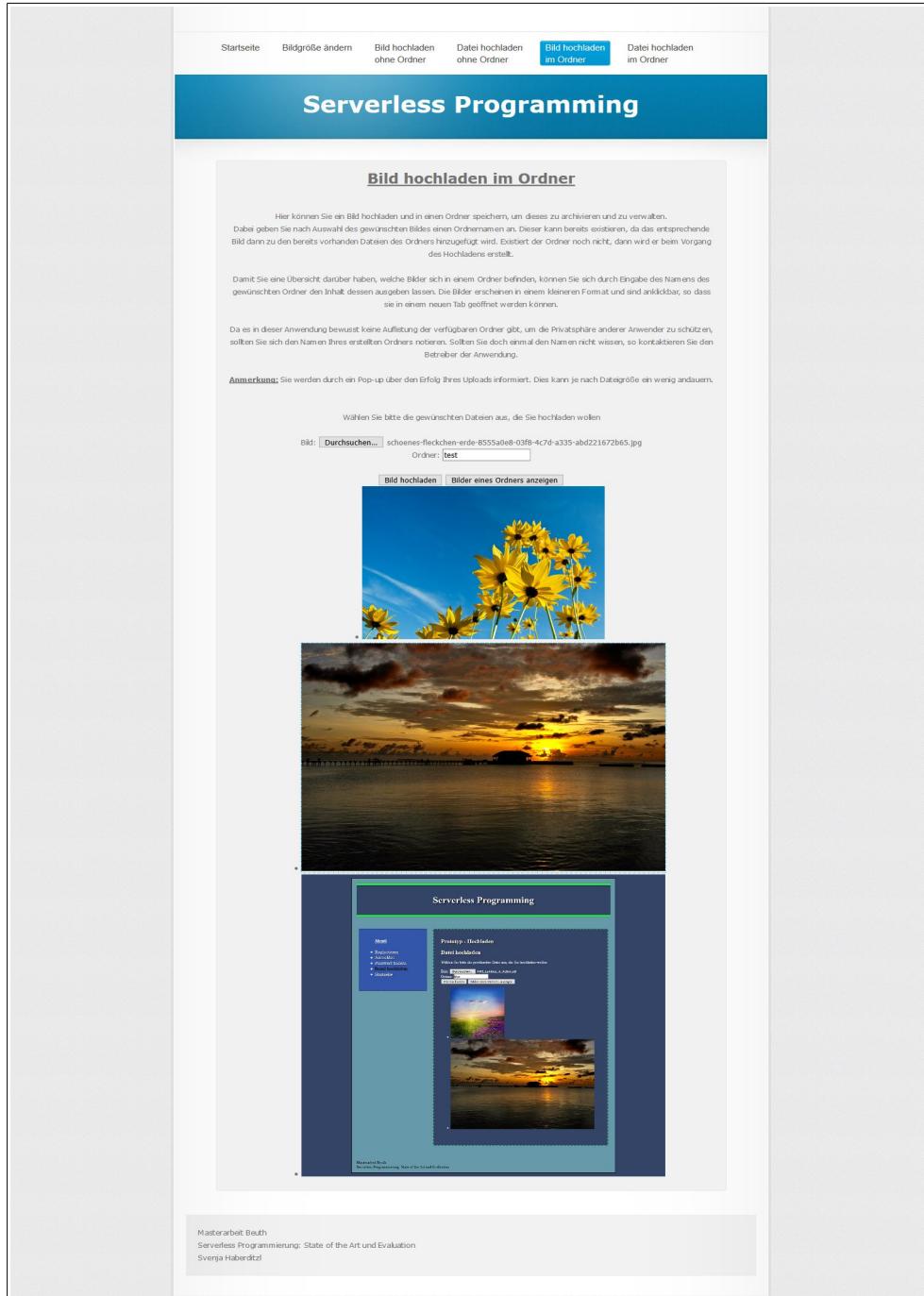


Abb. 10: Abrufen des Inhalts eines Ordners der Anwendung

## 6.2 S3 Buckets

Die gesamte Anwendung wurde mit Hilfe des Services S3 erstellt. Dort befinden sich alle Dateien, die zur Erzeugung der Anwendung benötigt werden. Dazu zählen jene, die sich auf die Oberfläche und das Design beziehen und die Dateien, mit deren Hilfe die einzelnen Funktionen umgesetzt und ausgeführt werden. Die Lambda Funktionen befinden sich hier nicht, da sie vom S3 Bucket mittels eines Triggers aufgerufen werden. Insgesamt arbeitet die Anwendung mit zwei unterschiedlichen Buckets, wobei einer lediglich zum Ablegen von Dateien dient, was im weiteren Verlauf des Kapitels noch näher erläutert wird. Die folgende Grafik zeigt den Inhalt des hauptsächlich genutzten Buckets „svenja-test“:

<input type="checkbox"/>	Name	Last modified	Size	Storage class
<input type="checkbox"/>	images	--	--	--
<input type="checkbox"/>	images2	--	--	--
<input type="checkbox"/>	js	--	--	--
<input type="checkbox"/>	tiere	--	--	--
<input type="checkbox"/>	xyz	--	--	--
<input type="checkbox"/>	bildHochladen.html	May 11, 2017 1:59:10 PM	1.7 KB	Standard
<input type="checkbox"/>	bildHochladenOrdner.html	May 13, 2017 9:08:27 PM	2.4 KB	Standard
<input type="checkbox"/>	config.json	Mar 27, 2017 6:48:46 AM	420.0 B	Standard

Abb. 11: Übersicht der im S3 Bucket befindlichen Ordner und Dateien

Der Bucket wurde in der Region Frankfurt erstellt, da dies die am nächsten liegende Region, die zur Verfügung steht, darstellt. Ebenso wurden alle Lambda Funktionen und sonstigen Elemente, die einer Auswahl einer Region bedürfen, in dieser erstellt.

## 6.3 JavaScript

Die JavaScript Dateien bilden das Bindeglied zwischen den HTML Dateien und den Lambda Funktionen. Sie sorgen dafür, dass die hinterlegten Funktionen bei AWS aufgerufen und ausgeführt werden. Dabei besitzt jede Funktion in der Anwendung seine eigene JavaScript Datei, in der sich die jeweiligen benötigten Codeteile zum Ausführen der gewünschten Aktion befinden. Nachfolgend soll auf die Dateien eingegangen und die wichtigsten Passagen und Funktionen näher erläutert werden.

### 6.3.1 Hochladen in Ordner

Die Anwendung bietet zwei Optionen, eine Datei bzw. ein Bild in einen Ordner hochladen zu können. Weiterhin ist es möglich sich den Inhalt eines Ordners nach Angabe des entsprechenden Namens ausgeben zu lassen.

Die folgende Grafik zeigt den Anfangscode der JavaScript-Datei, mit der ein Bild oder eine Datei hochgeladen werden. Dabei wird zunächst durch Eingabe der Zugangsdaten zum AWS-Account die Verbindungsaufnahme zu eben diesem ermöglicht. Weiterhin wird je eine Verbindung zum S3 Bucket, in welchem die Dateien zur Erstellung der Anwendung und die Ordner der Uploads liegen, und der Dynamo DB Tabelle, welche die Ordner samt Inhalt in sich trägt, hergestellt. Die Zugangsdaten lassen sich auch in einer allgemeinen Datei ausgliedern, damit diese nicht in jeder Datei, welche auf den AWS-Account zugreifen muss, vermerkt werden müssen. Jedoch wurde hier auf die Auslagerung der Daten verzichtet, da so eine bessere Übersicht über die Namensgebungen der Elemente geschaffen wird. Gerade bei mehreren Elementen kann es so schnell zu einer falschen Verknüpfung und daraus resultierenden Fehlern kommen. Die Zugangsdaten wurden hier aus Sicherheitsgründen geschrägt, nachdem bereits schon einmal der Account wegen eines solchen Vorfalls seitens von AWS stillgelegt wurde.

```
AWS.config.update({
  accessKeyId: '██████████',
  secretAccessKey: '████████████████████████████████',
  region: 'eu-central-1'
});

var S3 = new AWS.S3({params: {Bucket: 'svenja-test'}});
var Dynamo = new AWS.DynamoDB.DocumentClient({region: 'eu-central-1'});
```

Abb. 12: Verbindung mit AWS Konto, S3 Bucket und DynamoDB Tabelle herstellen

Um die Anwendung nicht nur lokal, sondern auch online und von außen nutzen zu können, wurde in AWS S3 ein Bucket mit dem Namen „svenja-test“ erstellt. Dort befinden sich sämtliche HTML, CSS und JavaScript Dateien, die für die Anwendung verwendet werden. Zusätzlich werden dort die Ordner gespeichert, die beim Hochladen eines Bildes oder ein PDF-Datei in der Anwendung erstellt bzw., wenn sie schon existent sind, aktualisiert werden. Die jeweiligen Bilder und Dateien der Ordner werden automatisch in diese hochgeladen und gespeichert. Die folgende Grafik zeigt, wie die Eingabe des Feldes des Ordnernamens ausgelesen und für die Funktion verwendet wird. Weiterhin erhält der Button zum Hochladen der jeweils ausgewählten Datei einen EventListener, so dass er auf das Klicken reagiert und dabei die ausgewählte Datei hinsichtlich ihrer Einstellungen ausliest. Dabei wird der Zähler auf 0 gesetzt, was einer gewählten Datei entspricht.

```
// Der eingegebene Ordnername wird aus dem Feld ausgelesen
function getOrdnername() {
    return document.getElementById('ordnername').value;
}

// Verknüpfung des Hochladebuttons mit einer Variablen herstellen
var hochladen = document.getElementById('hochladen');

// Der Hochladebutton reagiert durch den EventListener auf Klicks
hochladen.addEventListener('click', function() {
    var auswahl = document.getElementById('auswahl');
    var file = auswahl.files[0];
});
```

Abb. 13: Auslesen des Ordnernamens und Hinzufügen eines Event  
Listeners zum Upload-Button

Anschließend erfolgt die Kontrolle der Felder hinsichtlich der an diese gestellten Anforderungen, wie auf der nachfolgenden Grafik zu sehen ist. Dabei muss der Anwender zwingend ein Bild zum Hochladen auswählen, ansonsten erscheint eine entsprechende Fehlermeldung. Die ausgewählte Datei wird hinsichtlich des Formats überprüft. Nur wenn dieses mit der Erlaubten eines Bildes übereinstimmt, kann der Anwender mit dem Vorgang fortfahren. Ebenfalls muss ein Ordnername angegeben werden, der erstellt wird beim Hochladen der Datei, wenn er noch nicht existiert. Sollte der Ordner bereits existent sein, so wird die Datei in diesen geladen.

```
// Überprüfung ob eine Datei ausgewählt wurde
if (!file) {
    alert("Bitte eine Datei auswählen");
    return;
}

// Überprüfung ob Auswahl eine PD-Datei ist
if (file.type.indexOf("application/pdf") == -1) {
    alert("Falsches Format, bitte eine PDF-Datei auswählen");
    return;
}

// Get the gallery name and check that it isn't empty
var ordnername = getOrdnername();
if (!ordnername) {
    alert("Bitte Ordnernamen eingeben");
    return;
}
```

Abb. 13.1: Überprüfung auf leere Felder und Dateiformat vor  
Hochladen einer Datei

Das eigentliche Hochladen der Dateien geschieht durch das Einbinden der Informationen in S3. Dabei ist auf der folgenden Grafik zu sehen, dass zunächst die zu übergebenden Daten definiert werden. Neben dem Namen des Ordners, den der Anwender in das entsprechende Feld in der Anwendung eingegeben hat, wird auch der Dateityp und der Dateiname übergeben. Durch die Upload-Funktion werden die Informationen samt der Datei in den S3 Bucket der Anwendung übertragen. Dem Anwender wird der erfolgreiche Abschluss des Vorgangs mit einem Pop-up bekanntgegeben.

```
// Parameter für das Hochladen in den S3 Bucket werden definiert
var params = {
  Key: ordnername + '/' + file.name,
  ContentType: file.type,
  Body: file,
  ACL: 'public-read'
};

// Die Datei wird in den anfang definierten Bucket hochgeladen
S3.upload(params, function(err, data) {
  if (err) {
    alert(err);
  } else {
    alert('Die Datei wurde hochgeladen');
  }
});
```

Abb. 14: Festlegung des Aufbaus der hochzuladenden Elemente sowie Definition eigentliches Hochladens

Der Inhalt eines Ordners kann durch Aktivierung des entsprechenden Buttons in der Anwendung ausgegeben werden. Die Grafik zeigt den Code, mit der die Ausgabe der Dateien aus den bereits erstellten Ordnern aufgebaut ist. Dabei werden zunächst die Bilder verkleinert, um sie einheitlich übersichtlich darstellen zu können. Es wird eine Liste mit Bildern erstellt, die so lange ausgegeben werden, wie sich Elemente in dieser befinden. Der Inhalt des Ordners wird aus S3 ausgelesen. Für jeden Eintrag der Liste wird ein Element des Typs image erstellt. Die Größe wurde dabei auf 600 Pixeln in der Breite gesetzt. So kann eine gleichbleibende Breite in der Auflistung gewährleistet werden und größere Bilder erzeugen keine Scrollmöglichkeit auf der rechten Seite. Für jedes Element des Typs image wird ein Listenelement in der Anwendung zur Ausgabe erzeugt, welches wiederum in der Liste landet. Sobald ein Bild aus der ausgelesenen Liste des Ordner in S3 in die Liste der Anwendung ausgegeben wurde, wird dieses gelöscht, so dass der nächste Eintrag verwendet wird, bis die Bilderliste leer ist und der Vorgang somit abgeschlossen wurde. Die nun aufgelisteten Dateien bzw. Bilder eines Ordners sind verlinkt worden, so dass sie vom Anwender in einem neuen Tab des Browserfensters geöffnet werden können. Die Adresse der Datei kann dann an andere Mitmenschen gesendet werden, damit diese sich die Datei oder das Bild ansehen können.

```
// Es wird eine Liste in HTML erstellt, bei der für jedes Element der Dateiliste ein Link mit Dateiinformationen erstellt wird
function dateiHinzufügen(imagePath) {
  var ul = document.getElementById('image-list');
  var li = document.createElement('li');
  var a = document.createElement('a');
  var text = document.createTextNode(imagePath);

  a.href = 'https://s3.eu-central-1.amazonaws.com/svenja-test/' + imagePath;
  a.target = '_blank';

  li.appendChild(a);
  a.appendChild(text);
  ul.appendChild(li);
}
```

Abb. 14.1: Erstellung der Auflistung des Inhalts eines Ordners

### 6.3.2 Hochladen ohne Ordner

Bei der Funktion, ein Bild oder eine Datei ohne Verwendung eines Ordners hochzuladen, wird eine Vorschau des ausgewählten Bilds bzw. der Name der Datei in Form einer verkleinerten Variante bzw. des Textes gegeben. Dies dient zur Überprüfung der ausgewählten Datei seitens des Anwenders. Die folgende Grafik zeigt einen Codeausschnitt, mit dem die Vorschau beim Hochladen eines Bildes erstellt wird. Dabei wird das ausgewählte Bild ausgelesen. Es erfolgt eine Abfrage der Dateiart und nur, wenn eine Bild-Datei ausgewählt wurde, wird der Vorgang fortgesetzt. Für jede ausgewählte Datei wird ein Element des Typs image erstellt, welches in eine Liste ausgegeben wird. Dies wird so lange wiederholt, bis die Anzahl der ausgewählten Dateien abgearbeitet wurde und alle Bilder in einer kleinen Vorschau erscheinen. Dabei sind diese ankliebar, damit der Anwender sie wie bei der Ausgabe des Ordnerinhalts der anderen Funktion der Anwendung aufrufen und mit anderen teilen kann. Dabei ist darauf zu achten, dass der Link erst ein Resultat in Form der hochgeladenen Datei erzeugt, wenn diese hochgeladen wurde. Wird der Link vorher aktiviert, so wird die Startseite geladen, da diese in den Einstellungen des Buckets als Ausgabe bei einem Fehler hinterlegt wurde. Bewegt sich der Anwender innerhalb der Anwendung bzw. von der Seite weg oder lädt diese neu, so verschwindet die Vorschau und damit auch der Link und die Möglichkeit die Datei so aufzurufen. Sie kann jedoch durch Eingabe des korrekten Links in das Browserfenster aufgerufen werden. Die Kurzlebigkeit der Funktion ohne Ordner wird hierdurch bewusst aufgezeigt. Diese Funktion eignet sich demnach auch für einen Mehrfachupload, der in dieser Anwendung jedoch nicht umsetzbar war auf Grund der Regelungen von S3.

```
var reader = new FileReader();
reader.onload = (function (datei) {
    return function (e) {
        // Festlegung der Eigenschaften der Vorschau und Erzeugung dieser
        var vorschau = document.createElement('img');
        var a = document.createElement('a');
        vorschau.className = 'vorschau';
        vorschau.src = e.target.result;
        vorschau.title = datei.name;
        vorschau.style.maxWidth = "300px";
        a.href = 'https://s3.eu-central-1.amazonaws.com/svenja-test/ohneOrdner/' + datei.name;
        a.target = '_blank';
        a.appendChild(vorschau);
        // Die Liste der Bilder wird ausgegeben und kann angeklickt werden zum Aufrufen des Bildes
        document.getElementById('list')
            .insertBefore(a, null);
    };
});
```

Abb. 15: Erstellung einer Vorschau des zum Upload ausgewählten Bildes

Nachdem die Vorschau des Bildes erscheint, kann dieses durch Aktivieren des Buttons hochgeladen werden. Es wird hierbei nur in den S3 Bucket hochgeladen, aber nicht in die Tabelle der Datenbank übernommen. Dies bietet die Option, die Bilder und Dateien, die ohne Ordner hochgeladen werden, nach einer definierten Zeitspanne zu löschen. Nur die

Dateien, die sich in Ordnern befinden, bleiben dabei erhalten. Alle ohne Ordner hochgeladenen Dateien und Bilder werden automatisch in den Ordner „ohneOrdner“ gespeichert. Dieser kann dann nach belieben gelöscht werden, um die kurzlebigen Dateien zu entfernen und so weniger Inhalt im Bucket zu haben.

Da der Vorgang des Hochladens in den S3 Bucket an sich gleich ist mit der Funktion mit Ordner, wird er an dieser Stelle nicht nochmal aufgegriffen.

### 6.3.3 Größenänderung eines Bildes

Möchte ein User ein Bild hochladen, welches sich durch eine nicht unerhebliche Größe auszeichnet, so kann diese verändert werden. Dabei ist sowohl die Dateigröße als auch die Größe in Maßen gemeint. Die entsprechenden Bilder werden dazu zunächst in einen eigens dazu erstellen S3 Bucket hochgeladen. Dies geschieht, damit die Dateien der regulären Hochladefunktion und die der Größenänderung für große Bilder nicht durcheinander geraten. Beim Hochladen allerdings werden neue Bilder erstellt, die dann automatisch in den Ordner „images2“ des eigentlichen S3 Buckets der Anwendung geladen werden. Hier wurde die Bezeichnung „images2“ verwendet, da sich bereits ein Ordner „images“ im Bucket befindet, in welchem sich die Grafiken für die Verwendung des CSS der Anwendung befinden. Die umgewandelten Bilder weisen eine Größe von 800 Pixeln in der Breite auf. Wählt ein Anwender ein Bild zum Ändern auf, so erscheint zunächst eine Vorschau, die genauso aufgebaut ist wie beim Hochladen ohne Ordner und deshalb an dieser Stelle nicht nochmals näher erläutert wird. Der Aufbau der Funktion zur Größenänderung ist auf der folgenden Grafik zu sehen. Zunächst werden am Anfang verschiedene Module zum Laden angeführt. Die benötigten Module befinden sich in dem Ordner, der im Zuge der Erstellung der zugehörigen Lambda-Funktion als Ressource hochgeladen wurde.

```
// benötigte Module
var async = require('async');
var path = require('path');
var AWS = require('aws-sdk');
var gm = require('gm').subClass({
  imageMagick: true
});
var util = require('util');
```

Abb. 16: Einbindung von Modulen zur Änderung der Bildgröße eines Bildes

Die folgende Grafik zeigt den Anfang der dazugehörigen JavaScript-Datei:

```
var srcKey = decodeURIComponent(event.Records[0].s3.object.key.replace(
    '/\+/g, " "));
    // Festlegung des Zielbuckets durch Anhängen einer Endung an den ausgelesenen aktuellen Bucket
    var zielBucket = ausgangBucket + "-test";
// Festlegen der Eigenschaften für das zu verkleinernde Bild
var _800px = {
    width: 800,
    dstnKey: srcKey,
    destinationPath: "images2"
};
```

Abb. 17: Festlegung des Ausgabe-Buckets und der Eigenschaften des verkleinerten Bildes

Dabei wird der Bucket für die Ausgabe der in der Größe geänderten Bilder festgelegt. In diesem Fall ist dies der Bucket, in dem sich die eigentliche Anwendung befindet. Der Name des Buckets, in den die Bilder, die in ihrer Größe geändert werden sollen, hochgeladen werden, wird ausgelesen und die Endung „-test“ angehängt, um den anderen Bucket erreichen zu können. So ergibt sich „svenja“ und „svenja-test“ als Bucket Namen. Als Anmerkung sei an dieser Stelle erwähnt, dass diese Namen zu Anfang der Bearbeitungszeit nur willkürlich zu Testzwecken so gewählt wurden, bei dem Versuch die Anwendung und dessen Einstellungen auf neu erstellte Buckets mit einem passenden Namen zu übertragen, kam es zu Fehlermeldungen, da die Anwendung und ihre Dateien inzwischen sehr weit miteinander verflochten waren und die Schwierigkeit alle Verbindungen untereinander, in denen der Bucket Name auftaucht, ausfindig zu machen. Um auf Nummer sicher zu gehen blieb die Anwendung deshalb in den ursprünglichen Buckets. Auf der nächsten Grafik ist zu sehen, wie die Eigenschaften für die geänderten Bilder aussehen sollen. Es wurde, wie bereits erwähnt, eine Größe von 800 Pixeln in der Breite festgelegt. Damit kein Anwender Bilder oder Dateien in den Ordner „images2“ lädt, wurde dieser für ein Hochladen blockiert. Gibt ein Anwender als Ordnernamen „images2“ oder „images“ ein, dann erscheint die Meldung, dass der Ordner für Systemzwecke verwendet wird und ein anderer Name eingegeben werden muss. Im Ordner „images“ befinden sich die Bilder, die durch die CSS-Datei verwendet werden. Alle weiteren Bilder werden automatisch in diesen Ordner eingefügt. So kann nachvollzogen werden, welche Bilder genau in ihrer Größe verändert wurde, da sie sich alle in einem gesonderten Ordner befinden und dieser aufgerufen werden kann durch Verwendung einer Funktion der Anwendung, die mit Ordnern arbeitet.. Die Bilder des Buckets, in dem sich die Originalbilder befinden, findet in der Anwendung keine weitere Beachtung. Die Bilder in der Originalgröße werden nicht weiter verwendet, da sie genau ihrer Größe wegen verändert wurden und somit nicht zum Einsatz kommen. Deshalb bietet sich hier die Möglichkeit an, den Inhalt des Buckets nach einer gewissen Zeitspanne automatisch löschen zu lassen, damit sich dort nicht zu viele Daten ansammeln. Im Zuge dieser Ausarbeitung wurde darauf jedoch verzichtet, da hier nicht mit Unmengen an Dateien

gearbeitet wird und es sich dem Konzept nach um eine Umsetzung für einen bewusst kleinen Kreis an Anwendern handelt. Das nachfolgende Bild zeigt die Übertragung der veränderten Datei in den Zielbucket:

```
s3.putObject({
  Bucket: zielBucket,
  Key: _sizesArray[
    index].destinationPath +
  "/" + fileName.slice(0, -4) +
  ".jpg",
  Body: data,
  ContentType: 'JPG'
```

Abb. 18: Speichern des verkleinerten Bildes in Zielordner des Ausgabe-Bucket

Dabei wird ein Objekt in S3 durch put hinzugefügt. Als Bucket wird der am Anfang definierte Zielbucket mit der Endung „-test“ gewählt. Dazu verwendet der Code den ausgelesenen aktuellen Bucket inklusive der definierten zusätzlichen Endung. Alle in ihrer Größe geänderten Bilder werden im Format jpeg abgespeichert, egal welches Format sie vorher hatten. Dies garantiert eine gewisse Einheitlichkeit der Dateien und der Anwender weiß genau in welchem Format die Bilder vorliegen. Damit der Link für das Vorschaubild auch korrekt bei einer Datei mit der Endung png auf jpg gesetzt wird, wurden die letzten 4 Zeichen des Links entfernt und durch „.jpg“ ersetzt.

```
vorschau.className = 'vorschau';
vorschau.src = e.target.result;
vorschau.title = datei.name;
vorschau.style maxWidth = "300px";
var test = 'https://s3.eu-central-1.amazonaws.com/svenia-test/images2/large/' + datei.name;
a.href = test.slice(0, -4) + '.jpg';
```

Abb. 19: Änderung des Links auf die Endung ".jpg"

Somit können auch sehr große Bilder übersichtlich eingebaut und verwendet werden. Soll ein Bild für eine Homepage verwendet werden, kann dies bei großen Abmessungen zu einem Übertritt über das Fenster des Sichtbaren hinaus bedeuten und somit nicht für den eigentlichen Zweck verwendbar. Bei Dateien tritt dieses Problem nicht auf, weshalb sich die Funktion der Größenänderung auch nur auf Bilder bezieht.

Der Aufbau der Funktionen zum Hochladen von Bildern und Dateien mit und ohne Ordner basieren auf dem gleichen Aufbau. Sie unterscheiden sich maßgeblich durch eine geänderte Abfrage der gültigen Dateiformate und unterschiedliche Funktionen. Beim Hochladen von Bildern ohne Ordner gibt es eine Vorschau der ausgewählten Datei, damit der Anwender eine bessere Übersicht hat, welche Datei er genau ausgewählt hat, bevor diese hochgeladen werden. Bei den Dateien werden an der Stelle die Namen der ausgewählten Dokumente angezeigt. So kann eine Kontrolle seitens des Anwenders erfolgen, ob auch die richtige Auswahl der Datei getroffen wurde. Bei den Funktionen mit

Ordner kommt die Eingabe des Ordnernamens und das Hochladen der Dateien in diese hinzu. Die Funktion zum Ändern der Bildgröße hingegen verwendet einen eigenen Aufbau mit anderen Funktionen. Dort können nur Bilder in ihrer Größe verändert werden, nicht aber Dateien.

## 6.4 Zugriffsrechte und Roles

Damit die Anwendung auch von außen erreichbar ist, wurde die Startseite auf die index2.html gesetzt. Das heißt, sobald die Anwendung durch den automatisch erstellten Link aufgerufen wird, erscheint der Inhalt der index2.html. In einem Fehlerfall wird ebenfalls diese Seite aufgerufen. Sollte also eine Unterseite oder ein Link nicht funktionieren, erscheint keine Fehlermeldung und somit ein Ausfall der Anwendung, sondern die Startseite, um weiterhin in der Anwendung zu verbleiben. Damit jeder beliebige Nutzer die Anwendung auch sehen kann, wurden die Permissions so gesetzt, dass Zugriff für jeden besteht, der die Seite aufruft. Jedoch bestehen dabei nur Leserechte. Die folgende Grafik zeigt die Permissions, die für den Bucket festgelegt wurden.

```

1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Sid": "PublicReadGetObject",
6              "Effect": "Allow",
7              "Principal": "*",
8              "Action": "s3:GetObject",
9              "Resource": "arn:aws:s3:::svenja-test/*"
10         }
11     ]
12 }
```

Abb. 20: Zugriffsrechte für den Bucket der Anwendung

Die weiteren Zugriffsrechte der einzelnen Funktionen werden jeweils bei eben diesen festgelegt, um einen übersichtlichen Aufbau zu haben.

Für die Lambda Funktion zum Speichern der Daten in die Datenbank nach Hochladen von Dateien, wurde die Role „dbTest“ verwendet, welche vorher erstellt wurde. Dabei wurde dieser Rolle eine Policy zugewiesen, die vollen Zugriff auf DynamoDB gewährt. Für die Funktion der Größenänderung eines Bildes wurde die Rolle „s3-execution-Role“ erstellt und verwendet. Diese enthält eine Policy, welche vollen Zugriff auf S3 gewährt.

## 6.5 Lambda

Die Serverless Programmierung wird durch die Verwendung von AWS Lambda maßgeblich realisiert. Im Zuge dessen werden die Funktionen für die jeweiligen Aktionen, die in der Anwendung verwendet werden können und jeweils als HTML-Unterseite verfügbar sind, einzeln in Lambda erstellt und später mit den dazugehörigen Dateien verknüpft. Die Werte für Memory und Timeout wurden dabei bei den Voreinstellungen belassen, um eine möglichst gute Performance erreichen zu können und keine unnötigen Ladezeiten zu produzieren. Um die einzelnen Funktionen besser nachvollziehen zu können und aus Gründen der Übersichtlichkeit, befinden sich diese zusätzlich als Dateien abgespeichert in den Ordner der Anwendung der Endabgabe.

Die folgende Grafik zeigt die Lambda-Funktion, die bei dem Hochladen sämtlicher Dateien verwendet wird. Dabei wird zunächst auf die Region zugegriffen, in der auch die anderen Elemente in AWS erstellt wurden. Die Funktion speichert die hochgeladenen Dateien des S3-Buckets bei Ausgabe eines Ordnerinhalts in die Datenbank. Wichtig dabei ist, dass die Speicherung in die Datenbank ausgeführt wird, wenn der Inhalt eines Ordners aufgerufen wird. Sobald dies einmal geschehen ist, verbleiben die Daten in der DynamoDB Tabelle.

```

1 var AWS = require('aws-sdk');
2 var dynamo = new AWS.DynamoDB.DocumentClient({region: 'eu-central-1'});
3
4 exports.handler = function(event, context) {
5     var file_path = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));
6     var gallery = file_path.split("/")[0];
7
8     var params = {
9         TableName: 'images-index',
10        Item: {
11            gallery: gallery,
12            file_path: file_path
13        }
14    };
15
16    dynamo.put(params, context.done);
17 }

```

Abb. 21: Lambda Funktion zum Speichern des Ordnerinhalts in die DynamoDB Tabelle

Für die Lambda Funktionen des Hochladens von Dateien und der Größenänderung eines Bildes wurde jeweils ein Trigger verwendet. Dabei wurde S3 als Auslöser für die Funktionen ausgewählt, der aktiv wird, sobald ein neues Objekt innerhalb eines der beiden verwendeten S3 Buckets erstellt wird. Das heißt, bei jedem Hochladen eines Bildes in den Bucket „svenja“ in den Ordner images2, wird die Funktion zur Größenänderung ausgelöst. Ebenso beim Hochladen einer Datei in den Bucket „svenja-test“. Die folgende Grafik zeigt den Trigger für die Größenänderung und somit für den zweiten S3 Bucket:



Abb. 22: Trigger für die Lambda Funktion der Größenänderung

Die Lambda Funktion bezüglich der Größenänderung eines Bildes wurde bereits im Unterbereich JavaScript dieses Kapitels erläutert, da die Funktion nicht durch direkte Codeeingabe in Lambda selber, sondern durch das Hochladen eines ZIP-Ordners realisiert wurde. Für die erfolgreiche Ausführung der Funktion werden Module benötigt, die lokal mittels Verwendung von npm und Node.js in der Eingabeaufforderung des Rechners erzeugt wurden. Die erstellten Ordner zu den Modulen befinden sich in der ZIP-Datei, die in der Funktion hochgeladen wurde. Der Handler innerhalb der Einrichtungsseite der Lambda-Funktion steht dabei auf dem Dateinamen der JavaScript-Datei, da dieser dort gesucht und ausgeführt wird. Normalerweise steht dieser standardmäßig auf index.handler, da die Codeeingabe innerhalb des dafür vorgesehenen Feldes zur Inline-Eingabe einer Datei mit Namen Index gleichgesetzt wird.

## 6.6 DynamoDB Tabelle

Um möglichst viele Elemente zentral bei AWS zu verwalten, wurde auf eine der dort zur Verfügung stehenden Datenbanken zurückgegriffen. In diesem Fall ist dies DynamoDB. Dort wurde eine Tabelle mit dem Namen „images-index“ angelegt, in der die Daten, die in Ordner hochgeladen wurden, gespeichert werden. Die Wahl der Datenbank fiel auf DynamoDB, da der Aufbau dort übersichtlich ist und im Gegensatz zu anderen zur Verfügung gestellten Datenbanken diese nur Kosten abrechnet, wenn sie auch wirklich aktiv genutzt wird. Die Grafik zeigt die Einstellungen und Informationen zu der erstellten Tabelle. Dabei fungiert „gallery“ als Primary Key und „file\_path“ als Sortierschlüssel.

Table details	
Table name	galleryTest
Primary partition key	gallery (String)
Primary sort key	file_path (String)
Time to live attribute	DISABLED <a href="#">Manage TTL</a>
Table status	Active
Creation date	April 2, 2017 at 2:28:28 PM UTC+2
Provisioned read capacity units	5
Provisioned write capacity units	5
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0
Region	EU (Frankfurt)
Amazon Resource Name (ARN)	arn:aws:dynamodb:eu-central-1:934888181710:table/galleryTest

Abb. 23: Informationen über erstellte DynamoDB Tabelle mit Primary Key für Ordner

Die Einträge der Tabelle werden als Items aufgelistet. Dabei müssen nicht zwangsweise nur die beiden Schlüssel verwendet werden, die bei der Erstellung angegeben wurde. Es können auch weitere Attribute hinzugefügt werden innerhalb der Funktion zum Hochladen, welche dann in Form einer neuen Spalte in die Tabelle automatisch übernommen werden. Die nachfolgende Grafik zeigt den Inhalt der Tabelle, nachdem bereits einige Dateien in Ordnern hochgeladen wurde.

images-index Close	
<a href="#">Overview</a> <a href="#">Items</a> <a href="#">Metrics</a> <a href="#">Alarms</a> <a href="#">Capacity</a> <a href="#">Indexes</a> <a href="#">Triggers</a> <a href="#">Access control</a> <a href="#">Tags</a>	
<a href="#">Create item</a> <a href="#">Actions</a>	
Scan: [Table] images-index: gallery, file_path ▾	
Scan	[Table] images-index: gallery, file_path
+ Add filter	
Start search	
<input type="checkbox"/>	gallery file_path
<input type="checkbox"/>	gtg gtg/C_Users_...
<input type="checkbox"/>	gtg gtg/C_Users_...
<input type="checkbox"/>	xyz xyz/bild1.jpg
<input type="checkbox"/>	xyz xyz/blumen.jpg
<input type="checkbox"/>	tier tier/bild1.jpg
<input type="checkbox"/>	tier tier/blumen - ...
<input type="checkbox"/>	tier tier/blumen - ...
<input type="checkbox"/>	tier tier/schoenes...

Abb. 24: Übersicht über bereits in DynamoDB eingetragene Dateien

An dieser Stelle sei angemerkt, dass die Anwendung bewusst über keine Löschfunktion für die Dateien und Bilder verfügt. Diese Entscheidung beruht darauf, dass nur der Hauptverantwortliche der Anwendung die Daten verwalten soll. Die bereits genannten Einsatzmöglichkeiten der im Zuge der Ausarbeitung erstellten Anwendung bestehen in erster Linie aus einer privaten Anwendung zum Verwalten von Bildern und Dateien.

Weiterhin ist der Einsatz innerhalb einer Firma denkbar, bei der eine Abteilung oder ein Projekt über eine solche Anwendung verfügt. Existiert eine Löschfunktion, so kann es passieren, dass unbemerkt Dateien verschwinden, die der eigentliche Verantwortliche der Anwendung aber behalten möchte. In einem Betrieb könnten Mitarbeiter so Zwischenergebnisse entfernen, die für die Historie des Projektverlaufs jedoch wichtig sind. Damit dies nicht passieren kann, sollte nur der Betreiber der Anwendung, der Zugang zum internen Bereich in AWS hat, die Möglichkeit haben Dateien im S3 Bucket oder der DynamoDB Tabelle direkt löschen zu können.

## 6.7 Aktueller Stand der nicht funktionalen Anmeldefunktion

Ursprünglich war es geplant eine Anmeldefunktion in die Anwendung einzubauen, welche auf ein fixes Paar aus Nutzernamen und Passwort besteht. Jeder, der die richtige Kombination der beiden Komponenten kennt, kann dann auf die Funktionen des Hochladens von Bild und Datei in einen Ordner und das Ändern der Größe eines Bildes zugreifen. Das Hochladen ohne Ordner soll eine Gastfunktion sein.

Diese Anmeldefunktion ist in ihrem derzeitigen Zustand nicht ausführbar. Da sie dennoch nicht vernachlässigt und die bisherigen Ergebnisse diesbezüglich aufgezeigt werden sollen, wird sie an dieser Stelle näher betrachtet.

Es wurde eine HTML-Unterseite erstellt, auf der sich ein Eingabefeld für Nutzernamen und Passwort befindet. Die Felder werden über eine JavaScript-Datei hinsichtlich des Inhalts abgefragt. Sind sie leer, dann erhält der Anwender einen Hinweis darüber. Bestehen die Daten aus „admin“ und „123456“, dann soll der Anwender eingeloggt werden. Dies wird über das Aufrufen einer Lambda-Funktion aus der JavaScript-Datei heraus realisiert. Die folgende Grafik zeigt die Prüfung der beiden Felder hinsichtlich des definierten Nutzers und ruft die Lambda-Funktion auf, die für den Login erstellte wurde.

```

} else if (user.value == 'admin' && password.value == '123456') {

var params = {
    FunctionName:'loginTest',
    LogType: 'Tail',
    InvocationType: 'RequestResponse'
};

var lambda = new AWS.Lambda();
lambda.invoke(params);
}
}

```

Abb. 25: Prüfung der definierten Nutzerdaten und Aufrufen der Lambda-Funktion zum Einloggen

Zum authentifizieren des Anwenders wurde in AWS Cognito ein User Pool erstellt, in dem die Anmeldedaten eines fixen Nutzers hinterlegt sind. Weiterhin wurde ein Identity Pool angelegt, in dem die Zugriffsrechte für unautorisierte und autorisierte User festgelegt werden können. Nur wenn ein Anwender den Status eines autorisierten Nutzers erhält, kann er die Lambda-Funktionen zum Hochladen von Dateien in Ordner oder die Größenänderung der Bilder verwenden. Die folgende Grafik zeigt die Einstellungen für die Rolle zur authentifizierte Anwender:

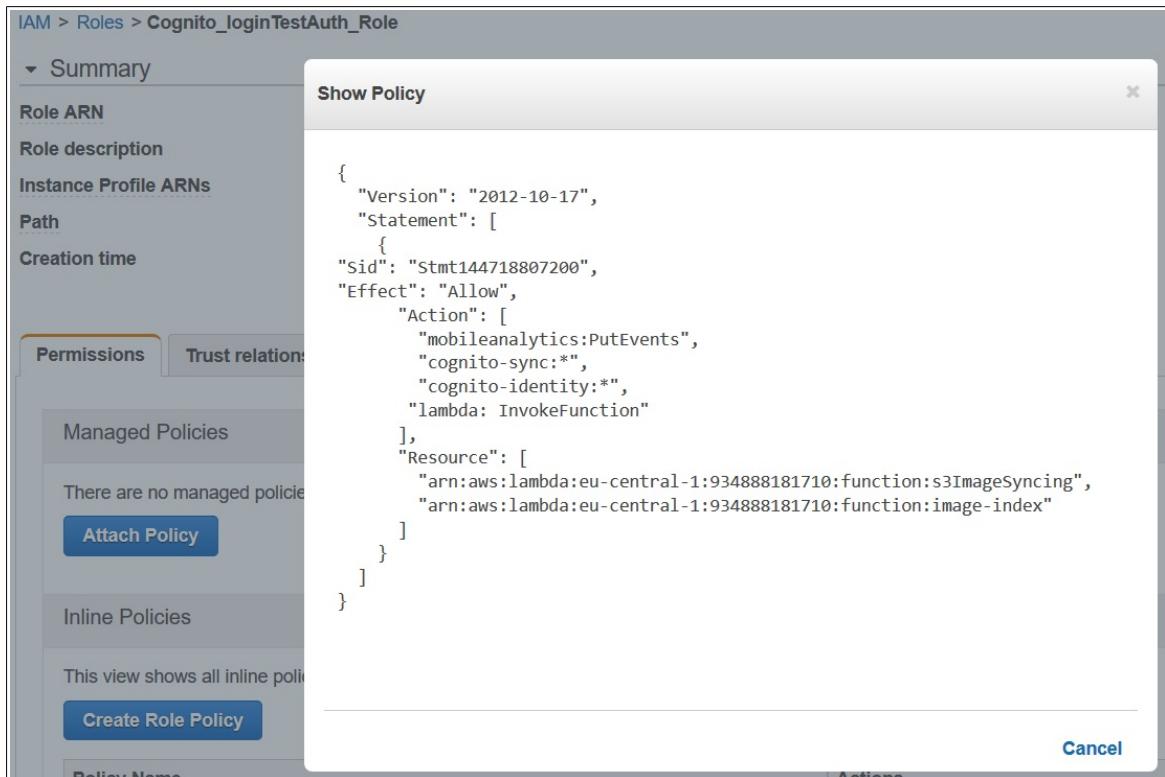


Abb. 26: Zugriffsrechte für autorisierte Anwender für den Login

Der Schritt des Autorisierens jedoch funktioniert nicht. Auch nach einigen verschiedenen Ansätzen, die getestet und ausgebaut wurden, konnte die richtige Verknüpfung der einzelnen Komponenten nicht hergestellt werden. Die ausführlichen Codedateien befinden sich im Abgabeordner auf Github.

## 6.8 Verwendete Hard- und Software

An dieser Stelle soll ein Einblick in die verwendeten Komponenten, die während der Ausarbeitung verwendet wurde, gegeben werden. Dabei wird auch auf die in den nächsten Kapiteln folgenden Messungen Bezug genommen, damit eine einheitliche Übersicht über alle Komponenten erfolgen kann und diese nicht auf verschiedene Kapitel aufgeteilt wird.

### Komponenten Anwendung

#### **Hardware:**

- Windows 10 Education Edition
- Acer Laptop

#### **Software:**

- AWS Plattform
- Netbeans als Entwicklungsumgebung
- Firefox als Browser zum Aufrufen der Anwendung
- PDF Creator zum Sichern der Quellen
- Astah für die UML Diagramme

### Komponenten Messungen

#### **Hardware:**

- Windows 10 Education Edition
- Acer Laptop

#### **Software:**

- Artillery.io
- JMeter
- Java SDK
- Eingabeaufforderung von Windows

## 6.9 Übersicht der Funktionen und Elemente

Die nachfolgenden Tabellen zeigen eine Auflistung der verwendeten Elemente der Anwendung ebenso wie eine Liste der Funktion, die hinsichtlich der jeweils verwendeten Umsetzungen betrachtet werden. Dies soll zum Abschluss des Kapitels eine Übersicht über die Anwendung zeigen.

<u>Funktion</u>	<u>HTML</u>	<u>JavaScript</u>	<u>Lambda</u>	<u>Role</u>	<u>Policy</u>
Startseite	X				X
Datei hochladen	X	X			
Datei im Ordner hochladen	X	X	X	X	X
Bild hochladen	X	x			
Bild hochladen Ordner	X	X	X	X	X
Größenänderung Bild	X	X	X	X	X

Tabelle 2: Übersicht der verwendeten Elemente der einzelnen Funktionen des Prototyps

<u>Element</u>	<u>Name</u>	<u>Inhalt</u>	<u>Funktion</u>
S3 Bucket	Svenja-test	- HTML-Dateien - CSS-Datei - JavaScript-Dateien - Dateien/Bilder - Dateien/Bilder in Ordnern - Verkleinert Bilder	- Startseite - Datei hochladen - Bild hochladen - Datei in Ordner hochladen - Bild in Ordner hochladen - Größenänderung von Bildern
S3 Bucket	svenja	- Originalbilder bei Änderung der Bildgröße	- Speicherung aller Originalbilder, die in ihrer Größe verändert werden sollen
DynamoDB Tabelle	images-index	- Dateien in Ordnern - Bilder in Ordnern	- Aufbewahrung der in Ordner hochgeladenen Bilder und Dateien zur dauerhaften Archivierung
Lambda		- image-index - Syncing	- Speicherung der Dateien, die in Ordner hochgeladen wurde - Größenänderung von Bildern mit Speicherung in Haupt-Bucket
Role	dbTest	- Zugriff auf DynamoDB	- Ermöglicht die Speicherung der Daten in die Datenbank
Role	s3-execution-Role	- Zugriff auf S3	- Ermöglicht das Speichern des Bildes in Bucket „svenja-test“

Tabelle 3: Auflistung der verwendeten Elemente innerhalb der Plattform AWS mit dessen Inhalt

## 6.10 Zusammenfassung des Kapitels

In diesem Kapitel wurde der genaue Aufbau der Anwendung inklusive der Erstellung der einzelnen Funktionen erläutert. Dabei wurde auf die Erstellung des Gerüsts durch Verwendung von HTML in Verbindung mit CSS zur Ausformung des Designs der Anwendung eingegangen. Die Funktion zum Hochladen einer PDF-Datei oder eines Bildes ohne oder mit Ordner wurden mit Hilfe von JavaScript aufgebaut.. Diese wurden mittels Einbindung in den HTML Dateien in die Anwendung integriert. Die Funktion des Hochladens der Dateien in einen Ordner wurde mit einer Lambda Funktion realisiert, welche direkt aus der JavaScript Datei aufgerufen wird. Dabei werden die definierten Parameter einer Datei in die Tabelle der Datenbank gespeichert, um eine dauerhafte Archivierung zu erreichen. Die Funktion für die Größenänderung von Bildern wurde mittels einer JavaScript Datei verwirklicht, die innerhalb der dazugehörigen Lambda Funktion als ZIP-Ordner hochgeladen wurde. Dabei werden die Originalbilder in einen eigenen S3 Bucket geladen und nach der Größenänderung in den eigentlichen Haupt-Bucket gespeichert. So wurden verschiedene Möglichkeiten der Umsetzung bezüglich Lambda dargestellt. Weiterhin verwenden mehrere Funktionen der Anwendung die selbe Lambda Funktion, um Dateien in Ordner laden zu können. Um die Anwendung überhaupt sichtbar machen zu können, wurden entsprechende Zugriffsrechte gesetzt. Alle Daten, durch welche die Anwendung erstellt wird und die hochgeladenen Dateien und Bilder werden in zwei S3 Buckets gespeichert und verwaltet. Alle Elemente und Dateien werden direkt auf der Plattform gespeichert, so dass nur diese verwendet wird.

## 7. Performance-Messungen

In diesem Kapitel wird der Vorgang der Messungen näher erläutert. Dabei wird auf die verwendeten Programme sowie den Aufbau der Messungen eingegangen. Die verwendeten Szenarien bezüglich der Messungen werden aufgezählt, damit die anschließenden Ergebnisse besser nachvollzogen werden können.

### 7.1 Durchführung der Messungen

Ein Ziel der Serverless Programmierung ist es, dass die Anwendungen bessere Reaktionszeiten erzielen können. Durch die geänderte und vereinfachte Struktur, bei der nicht unnötig viele Schichten durchlaufen werden müssen, können Aktionen in einer Anwendung schneller ausgeführt werden. Um zu verdeutlichen wie dies konkret zu bewerten ist, sollen im Zuge dieser Ausarbeitung anhand des vorgestellten Prototyps Performance-Messungen durchgeführt werden. Diese beziehen sich auf die Latenz und den Durchschlag. Dazu werden bei Aufruf der Anwendung und der Durchführung von Aktionen innerhalb dieser die Daten aufgezeichnet und analysiert.

Um möglichst verschiedene Messergebnisse zu erhalten, die dann miteinander verglichen werden können, werden Zugriffe auf die Anwendung simuliert. Dabei werden verschiedene Abstufungen an fiktiven Usern, die auf die Anwendung zugreifen, festgelegt. So wird die Effizienz von der Serverless Programmierung aufgezeigt. Weiterhin werden verschiedene Aktionen innerhalb der Anwendungen aufgerufen, um die Performance-Messungen anschließend vergleichen zu können. Durch die große Anzahl an Messergebnissen, die aus den getroffenen Anforderungen für die Messungen entstehen, kann die Effizienz der Anwendung gut beurteilt werden.

Um die fiktiven Anwender zu simulieren wird auf Software zurückgegriffen, welche sich auf Performance Tests für Anwendungen spezialisiert hat und auch mit der Plattform AWS verknüpft werden kann. Dabei stehen unterschiedliche Anbieter einer solchen Software zur Verfügung, die jeweils ihre eigene Umgebung mit sich bringen, um die nötigen Einstellungen und Ergebnisse präsentieren zu können. Es werden verschiedene Ziele verfolgt, die unterschiedliche Zielgruppen ansprechen. Einige Programme geben nur einen kurzen Überblick über die Latenz, mit anderen wiederum lässt sich eine Vielzahl an Werten erheben. Für die Messungen, die im Zuge dieser Ausarbeitung vorgenommen werden, wurden zwei verschiedene Programme verwendet. Zum einen Artillery.io, mit dessen Hilfe es möglich ist, auf die Anwendung zuzugreifen und zeitgleich virtuelle User zu erzeugen, die einen Zugriff unternehmen. Die Dauer und die Anzahl der Zugriffe pro

User können ebenfalls selbst bestimmt werden. Dieses Programm kann nach Installation in der Eingabeaufforderung verwendet werden. Dabei können die Messungen direkt dort eingegeben werden. Oder aber es werden Dateien der Endung yml erstellt, die die Einstellungen der durchzuführenden Messung enthält. Die Anwendung steht zusätzlich für jeden zum Download bereit und kann frei verwendet werden.

Die weitere Möglichkeit der Messung stellt JMeter dar. Dieses Programm weist wesentlich mehr Möglichkeiten der Durchführung und Einstellungen auf als Artillery.io. Um damit arbeiten zu können, muss Java samt Entwicklungsumgebung auf dem Rechner installiert sein. Dabei kann sowohl in der Eingabeaufforderung als auch in der Nutzeroberfläche des Programms selber gearbeitet werden. Der Anwender hat die Möglichkeit, ein Szenario für die Messung festzulegen, welches dann ausgeführt wird. Dabei kann der Vorgang der Messung über die gesamte Dauer der Abarbeitung des Szenarios aufgenommen und ausgewertet werden. Dieser Vorgang wurde mit den Funktionen der Anwendung durchgeführt. Dabei wurden jeweils Dateien hochgeladen, um die Performance für diesen Vorgang messen zu können. Als Referenz wurde Dropbox als vergleichbare Plattform, auf der ebenso Dateien hochgeladen werden können, herangezogen, um einen Eindruck der Unterschiede der Messergebnisse erhalten zu können. Es wurden zwei verschiedene Programme gewählt, da so eine Absicherung der Ergebnisse vorgenommen werden kann und mit Artillery.io nicht alles umsetzbar ist, wie angedacht. Die genauen Messergebnisse befinden sich im Anhang der Ausarbeitung.

Da die Möglichkeiten von JMeter denen von Artillery.io weit überlegen sind und dort auch die Einbindung von AWS in den Messvorgang nicht zufriedenstellend umgesetzt werden konnte, beziehen sich die Ergebnisse maßgeblich auf die Verwendung von JMeter. Allerdings konnte durch Artillery.io festgestellt werden, dass die Verhältnismäßigkeit der Messwerte gegeben ist.

Nachfolgend sind die unterschiedlichen Szenarien, die bei der Messung verwendet wurden, aufgelistet:

#### Szenarien innerhalb der Anwendung

- Startseite
- Gast Bild hochladen
- Gast Datei hochladen
- Nutzer Bild hochladen Ordner
- Nutzer Datei hochladen Ordner
- Bildgröße ändern

### Szenarien mit Dropbox

- Startseite
- Hochladen Datei
- Hochladen Bild

### Anzahl der simulierten User

- 1. Stufe: 1 User
- 2. Stufe: 25 User
- 3. Stufe: 100 User

### Ausführungszeiten

- 1 Sekunde
- 300 Sekunden

Das angedachte Testszenario der Änderung der Bildgröße konnte nicht als Messung umgesetzt werden. Dies ist damit begründet, dass die Funktion sich über zwei S3 Buckets erstreckt und deshalb einige Schritte zum Ausführen der Funktion nötig sind, die in JMeter nicht nachempfunden werden konnten. Weiterhin wurde bei der Funktion des Hochladens einer Datei bzw. eines Bildes in einen Ordner der Anwendung ein bereits existenter gewählt, damit dieser nicht in JMeter erstellt werden muss.

## 7.2 Zusammenfassung des Kapitels

Das Kapitel hat die verwendete Software zur Messung der Performance, die innerhalb der Ausarbeitung verwendet wurde, aufgezeigt und hinsichtlich der Möglichkeiten näher beleuchtet. Dabei fiel die Wahl auf Artillery.io, welches nur als Bestätigung der Werte herangezogen wurde. Die verwendeten Messergebnisse, welche im Zuge dieses Kapitels und im Anhang aufgezeigt wurde, stammen von JMeter. Weiterhin wurden die Gründe für die Entscheidung für genau die genannten Programme dargelegt und ein Einblick in die Testszenarien der Messungen gegeben.

## 8. Ergebnis der Messungen

Nachdem die Performance-Messungen anhand der Anwendung durchgeführt wurden, sollen nun die Ergebnisse aufgezeigt und analysiert werden. Dazu werden diese in einer Tabelle festgehalten, um eine übersichtliche Aufschlüsselung der Werte erhalten zu können. Anschließend werden die Werte hinsichtlich ihrer Bedeutung für die Serverless Programmierung beurteilt.

### 8.1 Ergebnisse der durchgeführten Messungen

Um eine Möglichkeit zu haben, die Ergebnisse der Messungen bezüglich der Anwendung bewerten zu können, wurde eine andere Plattform als Referenz gewählt. Diese besteht aus der Plattform Dropbox, welche sich durch einen ähnlichen Aufbau wie dem des Prototyps auszeichnet. Dort können Dateien jeglicher Art hochgeladen werden. Dropbox stellt eine weit verbreitete Möglichkeit der Speicherung von Daten dar, weshalb sie aus diesem Gesichtspunkt nicht mit dem Prototyp vergleichbar ist. Jedoch können die Ergebnisse der Messungen so verglichen und visualisiert werden hinsichtlich der Latenz und des Durchsatzes.

Die Ergebnisse sind in der nachfolgenden Tabelle zu sehen und sind dabei nach Anzahl der simulierten User sowie des Anwendungsfalls untergliedert. Dabei wurden bei jeder Funktion drei verschiedene Userzahlen simuliert, die in den Abständen 1, 25 und 100 definiert wurden. Da die entwickelte Anwendung als Zielgruppe private Anwender oder eine Abteilung bzw. eine Projektgruppe hat, wurden bewusst kleinere Abstufungen der Anzahl der User gewählt. Es ist nicht anzunehmen, dass hunderte Anwender auf die Anwendung zugreifen werden. Weiterhin wurden zusätzlich zu den drei Messdurchgängen je Funktion weitere drei hinzugefügt, was sich durch unterschiedliche Ausführungszeiten erklärt. Zum einen wurde mit einer Ausführung innerhalb einer Sekunde und zum anderen mit 300 Sekunden. Demnach wird unterschieden zwischen einem zeitgleichen Zugriff aller jeweils erstellten User und einem zeitversetzten Zugriff.

Die Einstellungen wurden für alle Messungen genau so übernommen, um die Daten vergleichen zu können. Weiterhin wurde die gleiche PDF-Datei und das gleiche Bild in Dropbox und in den Prototyp hochgeladen.

<b>Beschreibung</b>	<b>Latenz Durchschnitt</b>	<b>Latenz min</b>	<b>Latenz max</b>	<b>Durchsatz</b>	<b>Anzahl User</b>
<b>Startseite Prototyp 1 Sekunde</b>	1.780	1.780	1.780	33,7/min (0,56/sec)	1
	9.207	3.170	11.945	02,0/sec	25
	34.412	19.080	72.854	01,4/sec	100
<b>Startseite Prototyp 300 Sekunden</b>	1.818	1.818	1.818	33,0/min (0,55/sec)	1
	3.256	1.351	7.779	05,1/min (0,085/sec)	25
	2.564	1.579	4.185	20,1/min 0,34	100
<b>Startseite Dropbox 1 Sekunde</b>	459	459	459	02,2/sec	1
	487	385	1.132	16,8/sec	25
	2.714	450	9.419	09,6/sec	100
<b>Startseite Dropbox 300 Sekunden</b>	417	417	417	02,4/sec	1
	439	357	1.004	05,2/min (0,087/sec)	25
	452	347	3.009	20,2/min (0,34/sec)	100
<b>Dropbox Datei hochladen 1 Sekunde</b>	8.904	444	44.727	06,7/min (0,11/sec)	1
	20.405	447	104.149	52,7/min (0,88/sec)	25
	27.403	1.163	138.742	02,6/sec	100
<b>Dropbox Datei hochladen 300 Sekunden</b>	6.682	453	30.533	09,0/min (0,15/sec)	1
	8.935	334	54.497	26,6/min (0,44/sec)	25
	26.384	333	221.879	01,4/sec	100

<b>Prototyp Datei Hochladen 1 Sekunde</b>	777	79	1.341	01,3/sec	1
	4.700	155	3.0511	03,5/sec	25
	13.449	109	76.936	05,2/sec	100
<b>Prototyp Datei Hochladen 300 Sekunden</b>					
	1.039	151	2.469	57,7/min (0,96/sec)	1
	1.019	69	3.703	25,4/min (0,42/sec)	25
<b>Prototyp Datei hochladen Ordner 1 Sekunde</b>	763	67	2.146	01,7/sec	100
	565	71	1.229	01,8/sec	1
<b>Prototyp Datei hochladen Ordner 300 Sekunden</b>	3.484	71	27.426	05,0/sec	25
	9.496	70	85.710	07,4/sec	100
<b>Dropbox Bild hochladen 1 Sekunde</b>	544	88	1.227	01,8/sec	1
	537	68	1.298	36,0/min (0,6/sec)	25
	617	66	5.798	02,3/sec	100
<b>Dropbox Bild hochladen 300 Sekunden</b>					
	9.168	452	41.400	06,5/min (0,11/sec)	1
	27.514	446	146.657	42,4/min (0,71/sec)	25
	35.705	1.279	238.505	01,7/sec	100
<b>Dropbox Bild hochladen 300 Sekunden</b>	9.084	454	42.166	06,6/min (0,11/sec)	1
	9.459	333	53.096	25,8/min (0,43/sec)	25
	42.501	325	401.966	01,1/sec	100

<b>Prototyp Bild hochladen 1 Sekunde</b>	502	77	1.227	02,0/sec	1
	4.722	110	3.1615	03,3/sec	25
	11.962	196	6.7359	35,7/sec	100
<b>Prototyp Bild hochladen 300 Sekunden</b>	1.101	131	2.590	54,5/min (0,91/sec)	1
	994	95	31.615	26,6/min (0,44/sec)	25
	981	71	4.492	01,7/sec	100
<b>Prototyp Bild hochladen Ordner 1 Sekunde</b>	511	82	1.226	02,0/sec	1
	3.308	78	25.870	04,4/sec	25
	9.625	83	69.450	07,2/sec	100
<b>Prototyp Bild hochladen Ordner 300 Sekunden</b>	492	81	1.225	02,0/sec	1
	490	66	1.270	26,0/min (0,43/sec)	25
	506	67	1.254	02,3/sec	100

Tabelle 4: Zusammenfassung der Ergebnisse der durchgeföhrten Messungen

### 8.1.1 Latenz

Bei der Messung des Zugriffs auf die Startseiten von Dropbox und dem Prototyp ist bei der Ausführung innerhalb einer Sekunde ein großer Unterschied zu sehen. Die Werte für die Dropbox hinsichtlich der Latenz sind mehr als drei Mal so hoch und steigern sich bei 100 Simulierten Anwendern auf mehr als 30.000 ms. Bei der Anwendung steigen die Werte zwar auch, jedoch nicht ansatzweise so stark, wie bei der Dropbox. Bei der Ausführungszeit von 300 Sekunden ist die Latenz bei einem User gleichwertig wie bei einer Sekunde, jedoch steigert sich der Wert bei den beiden Steigerungen der Userzahl nur minimal. Bei der Anwendung ist dies ähnlich, jedoch ist die Latenz um zwei/drittel geringer als bei der Dropbox. Es lässt sich demnach festhalten, dass bei einer Ausführung innerhalb von 300 Sekunden der Verlauf der Latenz wesentlich ausgeglichener ist und somit bessere Werte erzielt werden als bei einer Sekunde. Die Werte für die Dropbox sind aber insgesamt bei der Messung der Startseiten höher als beim Prototypen.

Beim Hochladen einer PDF-Datei und eines Bildes in die Anwendung mit und ohne Verwendung eines Ordners ist festzuhalten, dass die Latenz bei der Datei geringer ist als bei dem Bild. Dies ist auf Grund der Dateigröße und des damit verbundenen längeren Vorgangs des Hochladens bei einem Bild ein zu erwartendes Ergebnis. Es ist aber auch zu beobachten, dass bei der Speicherung in einen Ordner sowohl bei einer Datei als auch bei einem Bild die Latenz geringer ist als ohne Verwendung eines Ordners. Dies ist auch bei beiden Ausführungszeiten zu beobachten. Wie auch bereits auf den Startseiten, sind die Werte bei 300 Sekunden Durchführungszeit wesentlich geringer hinsichtlich der Latenz, da nicht alle User zeitgleich auf die Anwendung zugreifen.

Die Latenz bei 100 simulierten Usern ohne Verwendung eines Ordners und einer Ausführungszeit von 300 Sekunden ist geringer als bei nur eine Anwender. Dies ist sowohl bei einer Datei als auch bei einem Bild der Fall. Verglichen mit den Werten der ersten Messung der Startseite sind diese für die Latenz nun bei der Ausführung mehrerer Vorgänge hintereinander höher.

Insgesamt ist die Latenz als recht gering anzusehen. Die Unterseiten der Anwendung werden schnell geladen und sind sofort verfügbar. Beim ersten Laden der Anwendung ist die Latenz höher als bei nochmaligem Zugriff. Bewegt sich ein Anwender also auf mehreren Seiten der Anwendung, so hat er insgesamt gesehen eine geringe Latenz zu erwarten. Die Auflistung des Inhalts eines Ordners erfolgt schnell, womit zeitgleich das Schreiben in die Datenbank zügig umgesetzt wird.

Bei dem Hochladen von Dateien in die Dropbox ist die Latenz bei der Ausführung mit 300 Sekunden wieder geringer als in einer Sekunde. Weiterhin ist beim Hochladen eines

Bildet die Latenz höher als bei einer PDF-Datei. Somit stimmt dies mit den Ergebnissen der Messungen bezüglich der Anwendung überein. Allerdings ist die Latenz deutlich höher, als bei der Anwendung. Bei einer Ausführung von 300 Sekunden und 100 simulierten Usern liegt diese bei 763, während sie bei der Dropbox bei 26.384 ist. Die Abstände der Werte der unterschiedlichen Userzahlen sind bei der Dropbox wesentlich größer, was zeigt, dass die Verarbeitung von steigenden Userzahlen schlechter umgesetzt wird, als bei der „serverless“ Variante der Anwendung.

### 8.1.2 Durchsatz

Bei den Werten des Durchsatzes ist zu beobachten, dass bei einer Ausführung von 1 Sekunde bei 25 Anwendern der Durchsatz ansteigt und bei 100 simulierten Usern wieder absinkt. Bei der Ausführung von 300 Sekunden verhält es sich genau andersherum. Dies gilt für alle Funktionen der Anwendung sowie auch für die Plattform Dropbox. Insgesamt variieren die Wert für den Durchsatz recht stark je nach Userzahl und Ausführungsart, was sich allerdings bei beiden Anwendungen bemerkbar macht. Bei einer Durchführungszeit von einer Sekunde sind die Werte für den Durchsatz bei den Messungen höher, als bei 300 Sekunden. Insgesamt ist der Durchsatz als recht gering zu bewerten. Je mehr Aktionen zeitgleich ausgeführt werden, desto weniger Durchsatz wird erreicht. Dies ist aber bei der Anwendung und der Dropbox so zu beobachten. Dennoch kann ein Upload von Dateien schnell durchgeführt werden. Nur bei sehr großen Dateien muss der Anwender mit ein wenig Wartezeit rechnen.

## 8.2 Bedeutung der Ergebnisse

Insgesamt lässt sich festhalten, dass die Anwendung eine teilweise erheblich geringere Latenz hat und sie somit schnell ausführbar ist, was letztendlich das oberste Ziel einer Anwendung ist. Der Anwender erwartet eine schnell ladende Ausführung der Anwendung und wandert bei längeren Ladezeiten oder Sprüngen zu der Konkurrenz ab. Auch wenn bei der ersten Verwendung einer Lambda Funktion die Latenz etwas höher ist, ist sie insgesamt jedoch gering genug, um auch für mobile Anwendungen eine gute Alternative zu bisherigen Möglichkeiten zu bieten. Auch wenn die Latenz der Plattform Dropbox höher ist, da diese nicht wie die Anwendung über die Region Frankfurt läuft, ist die Verhältnismäßigkeit der Unterschiede auf die „serverless“ Umsetzung zurückzuführen. Die Ergebnisse der einzelnen Schritte innerhalb einer Messung, bei der innerhalb der Anwendung unterschiedliche Seiten aufgerufen werden, zeigen, dass die Latenz sinkt was damit zu begründen ist, dass der erste Aufruf einer Unterseite eine höhere Latenz aufweist,

da die Anwendung erst einmal geladen werden muss. Insgesamt ist festzuhalten, dass eine Auswirkung zum Positiven hin bei der Verwendung einer „serverless“ Umsetzung zu erkennen ist.

### 8.3 Zusammenfassung des Kapitels

In diesem Kapitel wurden die Ergebnisse der Performance Messungen aufgezeigt und tabellarisch festgehalten. Sie wurden hinsichtlich ihrer Bedeutung betrachtet und die Auswirkung auf eine „serverless“ Umsetzung in Bezug gesetzt. Die Ergebnisse haben gezeigt, dass die Latenz beim Prototyp sich auch bei mehreren Anwendern nicht großartig verändert und trotzdem ein schneller Zugriff auf die Anwendung möglich ist. Weiterhin ist der Durchsatz als recht gut anzusehen, auch wenn dieser mit Zunahme der zu durchlaufenden Teilfunktionen bei der Ausführung einer Aktion der Anwendung abnimmt. Dennoch werden die Dateien zügig hochgeladen und auch die Anwendung an sich weist keine größeren Ladezeiten auf. Die Anwendung zeigt, dass mit wenigen Mitteln und ohne Verwendung eines Servers passable Ergebnisse erzielt werden können, die die bisherigen Standards durchaus ablösen und verdrängen können. Weiterhin kann durch die Plattform Dropbox als Referenz einer ähnlich aufgebauten Anwendung verdeutlichen, dass die Latenz des Prototyps erheblich geringer ist, als bei Dropbox und somit ein eindeutiger Effekt der „serverless“ Umsetzung zu sehen ist.

## 9. Vor - und Nachteile Serverless Programmierung

Um auf die eigentliche Problemstellung dieser Ausarbeitung einzugehen, sollen im Folgenden nun die Vor- und Nachteile der Serverless Programmierung näher betrachtet werden. Diese sind nun, nachdem die derzeit existenten Methoden sowie die Serverless Programmierung hinsichtlich ihres Aufbaus erläutert wurden, sowie nach Umsetzung des Prototyps als Praxisbeispiel in vollem Umfang durch eigene Schlussfolgerungen und Recherche definierbar. Zunächst wird dabei auf die Vorteile Bezug genommen, um abschließend die Nachteile aufzulisten und zu erläutern. Anschließend werden alle Argumente in einer Checkliste wiedergegeben, um einen Überblick über diese geben zu können.

### 9.1 Vorteile

Das Hauptargument ist die Kostenminimierung. Es gibt keine fixen Kosten mehr, wenn auf eine „serverless“ Methode zurückgegriffen wird. Dies ergibt sich daraus, dass kein Server mehr angemietet werden muss. Diese werden in fixen Größen zum Mieten angeboten und es ist egal, ob die Servergröße für das jeweils vorliegende Projekt überhaupt benötigt wird. Der Entwickler bzw. das Unternehmen wird derzeit gezwungen mehr zu mieten, als er eigentlich braucht und hat deshalb keinen Einfluss auf die entstehenden Kosten. Bei der Serverless Programmierung wird jedoch kein Server mehr angemietet, sondern Platz auf dem Server eines Drittanbieters verwendet. Es wird nur die benötigte Serverleistung am Ende des Monats bezahlt. Somit resultieren im Umkehrschluss daraus monatliche variable Kosten, da die Zugriffe auf die Anwendung und die damit ausgelösten Events immer unterschiedlich sind. (Froehlich, 2016)

Ein weiterer Hauptgrund, der für die Verwendung von Serverless Programmierung spricht, ist die Skalierbarkeit. Diese läuft hierbei völlig automatisch ab und der Entwickler muss sich dieses Problems nicht mehr annehmen. Dabei wird erkannt wie viele User auf die Anwendung zugreifen und bei einer größeren Belastung wird mehr Leistung freigegeben. In der Nacht bei weniger Zugriffen wird die Leistung heruntergefahren, ohne dass ein Außenstehender eingreifen muss. (Roberts, 2016)

Ein weitere Vorteil ist die Tatsache, dass weniger Personal benötigt wird. Zur Wartung der Server werden keine zusätzlichen Mitarbeiter eingestellt. Die Server müssen auf dem aktuellsten Stand gehalten und korrekt eingebunden werden. Da der Drittanbieter jedoch die Wartung seiner Server selbst übernimmt, wird das Personal für diese Aufgabe nicht benötigt. Weiterhin müssen sich die Programmierer auch nicht in die Thematik der Server einarbeiten, um dies in ihrer Planung zu berücksichtigen oder sich selber in diese Einzuarbeiten, um im Notfall reagieren zu können. (Froehlich, 2016)

Durch die Verwendung von Serverless Programmierung wird weniger Code benötigt. So müssen die Komponenten nicht wie vorher alle miteinander verbunden und verknüpft

werden. Außerdem werden zunehmend nur noch Events programmiert, die dann je nach Aktion in der Anwendung ausgelöst werden und schon definierten Code enthalten. Dieser muss vom Programmierer nicht mehr erstellt werden. Er selbst erstellt dann nur die benötigten Events. Dies bedeutet eine erhebliche Ersparnis an Code. Es entsteht zum einen eine enorme Zeitersparnis, aber auch ein übersichtlicherer Code, in den sich andere Programmierer besser und schneller einarbeiten können. Im Notfall kann relativ schnell jemand Anderes an dem System arbeiten und benötigt keine lange Einarbeitungszeit in den Code. Zudem wird die Anzahl der möglichen Fehler reduziert, da durch den übersichtlicheren Code Fehlerquellen besser und schneller identifiziert und ausgebessert werden können. (Looi, 2016)

Die Erstellung einer Anwendung basierend auf Serverless Programmierung bietet eine hohe Flexibilität. Es können Änderungen am System unkompliziert vorgenommen werden ohne dass zeitgleich auch eine Änderung anderer Komponenten wie dem Server erfolgen muss. Weiterhin können verwendete Elemente, die zur Erstellung einer Anwendung benötigt werden, durch andere ausgetauscht werden, ohne eine Neuerstellung dieser. Eine Optimierung kann demnach jederzeit erfolgen.

Ein Vorteil, der die Qualität der Anwendung beeinflusst, ist in der Arbeitsweise der Programmierer zu finden. Diese waren bisher dazu gezwungen, sich auf mehrere Bereiche konzentrieren zu müssen. So gehörte nicht nur die Erstellung des Codes der Anwendung zu ihren Aufgaben, sondern auch die Organisation bezüglich des Managements der Anwendung und mitunter auch der Server. Durch die Verwendung von Serverless Programmierung können sich die Entwickler vollkommen auf den zu erstellenden Code konzentrieren. Dies hat zur Folge, dass die Qualität des Code steigt, da die Konzentration nur auf diesem liegt und die Programmierer nicht durch andere Umstände abgelenkt werden und ihre Arbeit unterbrechen müssen.(Froehlich, 2016)

Ein weiterer Punkt betrifft die Größe und den Umfang von Projekten, die nun keine Limits kennen. Serverless Programmierung kann von jedem Entwickler verwendet werden. Dabei ist es unerheblich, ob es sich um einen Privatanwender handelt, der in seiner Freizeit an einer Anwendung arbeitet oder ob es ein großes Unternehmen ist, welches die Server für seine Zwecke nutzen möchte. Gerade Privatpersonen haben so die Möglichkeit, sich auszuprobieren ohne einen Server anmieten zu müssen. Da sie verhältnismäßig wenig Serverauslastung haben, ist die Methode, nur das zu zahlen, was auch verwendet wird, für sie ideal, um günstig und schnell Anwendungen testen zu können. Zwar sind die Drittanbieter potentiell an Unternehmen interessiert und nicht an kleinen Privatanwendern, aber auch diese werden dort bedient.

Serverless Programmierung bietet den Vorteil, dass unterschiedliche Programmiersprachen miteinander kombiniert werden können. So ist es möglich, dass Mitarbeiter an einem Projekt arbeiten und dabei ihren Code in verschiedenen Sprachen schreiben. Diese Dateien können dann trotzdem miteinander verbunden und in die Anwendung eingebaut werden, ohne dass Probleme dabei entstehen und Teile nicht funktionieren.

Eine Anwendung, die auf einer „serverless“ Umsetzung aufgebaut wurde, kann schneller ausgeführt werden. Die definierten Funktionen, die bei Aufruf einer Aktion in der Anwendung ausgelöst werden, werden umgehend aktiv und erzielen eine bessere Latenz, als viele Umsetzungen, die mit einem Server arbeiten. Insbesondere wenn auf die Anwendung eine Vielzahl an Usern zugreift, kann diese auch zu Stoßzeiten mit einer niedrigen Ausführungszeit aufwarten. (O'Reilly, 2016)

Durch Serverless Programmierung können mehrere Anwendungen zeitgleich angesprochen werden. Auftretende Probleme können in mehreren Anwendungen zur selben Zeit gelöst werden, anstatt jede Anwendung einzeln durchgehen zu müssen, um eine Lösung zu finden. So können Funktionen in verschiedenen Anwendungen verwendet und aufgerufen werden. Dies spart Zeit und macht Änderungen in unterschiedlichen Anwendungen einfacher und minimiert die Möglichkeit von Fehlern.

Die Programmierung in einem „serverless“ Projekt bezieht sich auf die Definition von Events. Dadurch werden lange komplizierte Funktionen vermieden und die Möglichkeit von Fehler wird minimiert. Die Events lösen dann hinterlegte Funktionen aus und handeln selbstständig. Die Anwendung erkennt, je nachdem, welche Aktion vom Anwender ausgeführt wird, welches Funktion ausgelöst werden muss. (Poccia, 2016, S. 5 ff.)

Im Bereich „serverless“ sind viele Möglichkeiten einer Umsetzung denkbar. Während derzeitig verbreitete Methoden eher nur in eine bestimmte Richtung laufen und daher nicht viele Wahlmöglichkeiten bieten, kann in der Serverless Programmierung mit verschiedenen Methode gearbeitet werden. Die unterschiedlichsten Projekte können umgesetzt und gestaltet werden, je nach Wunsch des Unternehmens oder des Kunden.

Ein weiterer Vorteil ist in der Verwendung von einzelnen Funktionen zu sehen. Anstatt wie bisher einen kompletten Code zu haben, in dem sämtliche Funktionen integriert sind, werden diese jeweils einzeln erstellt. Dieser Vorgang bedeutet, dass sich der Programmierer immer nur auf eine Funktion zur Zeit fokussiert und er keine Rücksicht auf den Einfluss dieser auf den Rest des Codes nehmen muss. Dadurch können bessere Ergebnisse erzielt werden, weil die Konzentration auf nur einem Teil der Anwendung liegt.

Die Verwendung von einzelnen Funktionen ermöglicht eine einfache Wiederverwendung dieser. Wird eine Funktion erstellt, so kann diese beliebigen Anwendungen zugeordnet werden und ist nicht auf nur eine limitiert. Dies reduziert die Programmierzeit auf Dauer und ermöglicht eine gleichbleibende Qualität der Ergebnisse. Die Programmierer werden durch die Wiederverwendung von Funktionen entlastet und können sich wiederum anderen Aufgaben mit mehr Zeit widmen.

Die Anwendungen können früher auf den Markt gebracht werden. Die Serverless Programmierung ermöglicht eine Zeitersparnis und somit eine frühere Etablierung auf dem Markt. Die Kundenzufriedenheit ist insbesondere für Unternehmen schneller zu erreichen und bindet diese besser an sich. Weiterhin können eventuelle Konkurrenten unterboten werden, die mit einer Anwendung ihrerseits erst später aufwarten können. (Roberts, 2016)

Ebenfalls ist es als Vorteil anzusehen, dass Microservices als Funktionen umgesetzt werden können. So kann eine Kombination aus regulären Funktionen und Microservices erfolgen und diese können wie alle anderen Funktionen auch eingebaut und angesteuert werden. Dies bietet eine Einheitlichkeit in der Umsetzung für die jeweilige Anwendung. (Roberts, 2016)

Bei einer Anwendung, die mit Hilfe der Serverless Programmierung erstellt wurde, ist ein Systemadministrator nicht zwingend notwendig. Die Anwendung verwaltet sich im Prinzip selbst. Durch den Wegfall der eigenen Server und dem damit verbundenen Arbeitsaufwand ist kein zusätzliches Personal für diesen Zweck von Nöten. Sollte es sich allerdings um eine größere Anwendung mit vielen Dateien und Funktionen handeln, so ist ein gänzlicher Verzicht auf einen Administrator eventuell zu überdenken. Auch hier gilt, dass sich darüber im Vorwege Gedanken gemacht werden sollte, bevor die Wahl einer Umsetzungsart getroffen wird. (Roberts, 2016)

Durch die Möglichkeit der Kombinationen verschiedener Sprachen und Technologien wird eine hohe Flexibilität geboten. So haben Programmierer die Freiheit, bei der

Erstellung einer Funktion genau die Sprache zu wählen, mit der sie am besten zurecht kommen oder welche das beste Ergebnis für den jeweiligen Fall bietet. Die Erstellung von Funktionen kann gut auf verschiedene Mitarbeiter verteilt werden, die nicht warten müssen bis das Ergebnis eines anderen fertig ist. Jeder kann unabhängig voneinander programmieren und die Einzelteile am Ende zu einer gesamten Anwendung zusammenfügen. Weiterhin ist dieses Konzept auch gut dazu geeignet über mehrere Standorte an verschiedenen Orten hinweg eine Anwendung zu erstellen. Es gibt keine speziellen Vorgaben, welche Sprache oder Technologie verwendet werden darf.

## 9.2 Nachteile

Einer der größeren Nachteile der Serverless Programmierung ist die Auslagerung bei Drittanbietern. Dies ist zwar aus dem Gesichtspunkt der Nutzung des Servers des Drittanbieters und den damit verbundenen Vorteilen als sinnvoll anzusehen, jedoch bedeutet dies auch eine Kontrolle durch eben diesen Drittanbieter. Kommerzielle Anwendungen mit sensiblen Daten sind für den Anbieter transparent. Das setzt ein hohes Maß an Vertrauen seitens des AWS-Nutzers in die Betreiber voraus, dass vertrauliche Daten nicht an Konkurrenzunternehmen weitergegeben oder für eigenen Zwecke verwendet werden. (Froehlich, 2016)

Die Auslagerung auf die Server eines Drittanbieters hat eine Unsicherheit durch Zugriffe von Außen zur Folge, die für manche Anwendungen nicht akzeptabel ist. So können die Daten eines Unternehmens in falsche Hände geraten, was gerade bei sensiblen Daten wie bei Banken eine Katastrophe wäre. Zwar sind die Drittanbieter natürlich bemüht eine hohe Sicherheit für ihre Kunden zu gewährleisten, jedoch kann dies nicht zu 100% garantiert werden. Dies führt dazu, dass die Methode des Serverless Programmierung schlicht zu unsicher ist für Anwendungszwecke und Unternehmen, die mit sehr sensiblen Daten arbeiten. Deshalb muss ein möglicher Hackerangriff auf den Drittanbieter einkalkuliert und das Risiko mit dem Nutzen abgewägt werden.

Es ist zwar ein Vorteil, dass keine zusätzlichen Mitarbeiter für die Server benötigt werden, jedoch geht damit auch der Verlust über Serveroptimierungen einher. So müssen sich Entwickler darauf verlassen, dass der Drittanbieter sich auch zeitnah um Updates bemüht und der Server so zur Verfügung steht wie er für die Anwendung benötigt wird.

Ein weiterer Nachteil ist die Einarbeitung in die jeweilige Funktionalität der Drittanbieter. Jede der Plattformen hat ein eigens entwickeltes System, welches einen jeweils unterschiedlichen Aufbau hat. So kann ein Programmierer, der sich in System A

eingearbeitet hat, nicht auch mit System B oder C arbeiten. Dies ist jedoch hinderlich in der Hinsicht, dass die Plattformen stetig weiterentwickelt werden und sich dementsprechend die Wahl der Drittanbieter ändert und sich die Entwickler in den Aufbau immer neu einarbeiten müssen. Weiterhin können unterschiedliche Anwendungen verschiedene Sprachen benötigen und nicht jede Sprache wird überall unterstützt. Auch da ist ein Wechsel der Plattform als hinderlich anzusehen. Ebenfalls kann es auch vorkommen, dass ein Kunde eines Unternehmens auf die Verwendung einer bestimmten Plattform besteht, da er dort schon andere Anwendungen integriert hat oder er dem anderen Drittanbieter nicht vertraut. Auch hier müssen sich die Entwickler erst einarbeiten, wenn ihnen das System unbekannt ist oder es sind weitere Mitarbeiter nötig, die sich jeweils auf eine Plattform spezialisiert haben.

Für Entwickler ergibt sich ein Nachteil aus der Verfügbarkeit von Literatur zum Thema Serverless Programmierung. Auch wenn dieser Bereich immer mehr an Bedeutung gewinnt und die Zahl der Anwendungen, die mit Hilfe der Möglichkeiten erstellt werden stetig ansteigt, so ist dies doch ein relativ neues Gebiet. Einhergehend damit ist auch nur wenig Literatur dazu vorhanden. Teilweise sind derzeit Bücher in der Vorbereitung, die sich näher gehend mit der Thematik des Serverless Programmierung beschäftigen. Wer sich allerdings damit einarbeiten will, hat zumeist nur die Dokumentationen der Drittanbieter selbst zur Verfügung. Dies ist insofern problematisch, als dass die Entwicklung einer Anwendung so unter Umständen mehr Zeit benötigt. Trifft ein Entwickler auf ein Problem, kann er bei den bisher integrierten Systemen und deren Möglichkeiten sehr leicht Hilfe finden, indem er entweder Bücher dazu findet, andere Programmierer um Hilfe bitten oder er sich in einem der zahlreichen Foren informieren kann. Er hat demnach recht viele Möglichkeiten, eine schnelle und effiziente Lösung für sein Problem zu erhalten. Durch das Fehlen dieser Möglichkeiten bei der Serverless Programmierung ist die Fehlerbehebung auf Grund fehlender Erfahrung recht schwierig und zeitaufwändiger. Auch die nur langsam anlaufende Verbreitung der Methoden führt dazu, dass sich eine Community ebenso langsam entwickelt. Gerade Programmierer sind eher dazu geneigt, Altbekanntes zu verfolgen und sich nicht in Neues einarbeiten zu wollen. Die schlechte Verbreitung von Literatur hilft dabei nicht wirklich.

Die Methode des Serverless Programmierung ist nicht für jegliche Art von Anwendung gedacht. Es gibt durchaus Anwendungsfälle, wo es ratsam ist einen eigenen Server angemietet zu haben und nicht auf einen Drittanbieter auszuweichen. Dieser Fall tritt zum Beispiel bei einer Anwendung auf, die auf mehrere Datenbanken zugreift oder der Code eine nicht endende Schleife erzeugt, was für eine lange Ausführungszeit des Codes sorgt. Die Ausführung eines Browser-Games, welches über Stunden am Stück von

Anwenden gespielt werden kann, erzeugt bei einer „serverless“ Umsetzung auf Dauer hohe Kosten und würde dem Sinn der Serverless Programmierung widersprechen. Deshalb sollte vor der Wahl der Methode das Projekt und dessen Funktionalität überdacht werden. Wenn dieser Punkt nicht genau ausgearbeitet wird, ist es im Verlauf der Erstellung der Anwendung unter Umständen zu spät, um die Methode zu wechseln und das Projekt kann scheitern. (O'Reilly, 2016) (Chan, 2017)

Erschwerend für die Entwicklung von Anwendungen ist es, dass die Drittanbieter nicht jede Programmiersprache unterstützen und sich Entwickler so nach den Möglichkeiten der Anbieter richten müssen. Es ist möglich, dass eine Sprache verwendet werden muss, die für die jeweils vorliegende Anwendung nicht die Geeignete ist, da sie sich nicht optimal dafür anbietet. Das volle Potenzial einer freien Wahl kann hier demnach Einbußen bei der Qualität mit sich bringen. Weiterhin bieten die Plattformen jeweils unterschiedliche Sprachen an, die sie unterstützen. Dies ist gerade bei einem möglichen Wechsel der Anbieter hinderlich, wenn das favorisierte System die angedachte Sprache nicht unterstützt. (Froehlich, 2016)

Generell birgt es einen gewissen Nachteil mit den Plattformen der Drittanbieter zu arbeiten, da diese stetig weiterentwickelt werden und so ein Wechsel zu einem anderen Anbieter als sinnvoll erachtet werden kann. Es notwendig, dass die Entwickler sich stets und ständig über Neuerungen informieren und vor allem auf diese auch reagieren müssen. Sie müssen sich demnach darauf einstellen, dass sie mit einem sich ändernden System arbeiten und eine Änderung der Funktionen erfolgen kann, auf die sie in der Lage sein müssen zu reagieren. (Froehlich, 2016)

Tritt ein Problem bei einem Drittanbieter auf, sind mitunter sämtliche Anwendungen, die dort gespeichert sind, nicht aufrufbar. Sollte es zu einem Ausfall des Servers kommen, so ist das ein „Worst-Case“ für betroffene Unternehmen und bedeutet einen enormen finanziellen Verlust. Zwar ist ein Szenario dieses Ausmaßes relativ unwahrscheinlich, jedoch sind kleinere Teilausfälle möglich. Bei einem eigenen Server ist ein Unternehmen auch nicht vor Ausfällen geschützt, jedoch ist es dann möglich direkt vor Ort und relativ schnell nach einer Lösung suchen zu können. Ist eine Firma auf den Drittanbieter angewiesen, muss sie mit diesem die schnelle Kommunikation suchen und ist darauf muss sich darauf verlassen, dass dieser schnell kooperiert und in der Lage ist den Fehler zeitnah zu finden und zu beheben.

Ein Wechsel zu einer Serverless Programmierung ist schwer umzusetzen. Entscheidet sich ein Unternehmen auf Grund der Vielzahl an Vorteilen für den Weg einer Serverless Programmierung, kann dies nicht ohne Weiteres umgesetzt werden. Sämtliche bisherige

Anwendungen müssen komplett oder in Teilen umgeschrieben werden und auch die Systeme, die für die Programmierung notwendig sind, bedürfen einer Überarbeitung. Das Personal muss erst für einen Umbau geschult werden und sich mit der Thematik intensiv auseinandersetzen, was unter Umständen eine Aufstockung des Personals als Folge hat und somit höhere Kosten verursacht. Die Tatsache des aufwändigen Umbaus sollte daher bei der Abwägung eines möglichen Wechsel einbezogen werden in die Entscheidung. (Looi, 2016)

Als weiterer Nachteil einer „serverless“ Umsetzung ist die daraus resultierende Komplexität der Anwendung zu nennen. Durch die Verteilung der einzelnen Aktionen auf eigene Funktionen kommt bei einer größeren Anwendung eine Vielzahl dieser zu Stande welche eine enorme Komplexität nach sich zieht. Dies ist nicht zu unterschätzen und kann einige der zuvor genannten Vorteile wieder relativieren und ist deshalb im Vorfeld abzuwägen.

Es existieren derzeit nur wenige Werkzeuge, um die Dateien und Funktionen einer „serverless“ Anwendung zu verwalten. Insbesondere im Hinblick auf größere Anwendung stellt dies ein Problem dar, da die große Anzahl an einzelnen Funktionen so nicht angemessen verwaltet werden kann. Die Gefahr, dass schnell der Überblick verloren geht, ist sehr hoch. Weiterhin sieht die Serverless Programmierung nicht vor, dass zusätzliche Programme benötigt werden, um die Funktionen vernünftig verwalten zu können. Dieser Nachteil ist allerdings eher zu vernachlässigen, da sich die Thematik der Serverless Programmierung derzeit verbreitet und daher in absehbarer Zukunft entsprechende Programme ihren Einzug in die Softwareentwicklung finden werden. (Looi, 2016)

### 9.3 Checkliste Vor- und Nachteile

Um eine Übersicht über die eben genannten und erläuterten Vor- und Nachteile erhalten zu können folgt an dieser Stelle eine Checkliste, in welcher sich alle genannten Punkte mitsamt ihrer Gewichtung und damit der Wichtigkeit für ein Unternehmen befinden. Für diese wurde eine Skala von 1 bis 10 definiert, wobei 1 die geringstes Gewichtung darstellt. Weiterhin wurde die Dauer des jeweiligen Arguments eingebunden, da einige nicht als dauerhaft gültig anzusehen sind.

<b><u>Art</u></b>	<b><u>Beschreibung</u></b>	<b><u>Gewichtung</u></b>	<b><u>Dauer</u></b>
Nachteil	Auslagerung bei Drittanbietern	10	Dauerhaft
Vorteil	Skalierbarkeit	10	Dauerhaft
Vorteil	Definition von Events	9	Dauerhaft
Nachteil	Verfügbarkeit von Literatur	9	kurzzeitig
Nachteil	Ausfall bei Problem Anbieter	8	Dauerhaft
Vorteil	Schnellere Ausführungszeit	8	Dauerhaft
Vorteil	Kombination von Sprachen	8	Dauerhaft
Nachteil	Unterstützt nicht jede Sprache	8	kurzzeitig
Vorteil	Viele Umsetzungsmöglichkeiten	8	Dauerhaft
Vorteil	Weniger Kosten	8	Dauerhaft
Vorteil	Weniger Personal	8	Dauerhaft
Nachteil	Einarbeitung in Plattform	7	Dauerhaft
Vorteil	Größe/Umfang Projekt	7	Dauerhaft
Nachteil	Nicht für jede Anwendung geeignet	7	Dauerhaft
Nachteil	Unsicherheit wegen Drittanbieter	7	Dauerhaft
Vorteil	Weniger Code	7	Dauerhaft
Vorteil	Hohe Flexibilität	6	Dauerhaft
Vorteil	Mehrere Anwendungen ansprechen	6	Dauerhaft
Nachteil	Weniger Personal	6	Dauerhaft

Vorteil	Arbeitsweise Programmierer	5	Dauerhaft
Nachteil	Weiterentwicklung der Plattformen	5	kurzzeitig
Vorteil	Verwendung einzelner Funktionen	8	Dauerhaft
Vorteil	Einfache Wiederverwendung	8	Dauerhaft
Vorteil	Anwendungen früher auf Markt	9	Dauerhaft
Vorteil	Microservices als Funktionen	7	Dauerhaft
Vorteil	Kein Administrator nötig	5	Dauerhaft
Vorteil	Kombinationsmöglichkeiten	8	Dauerhaft
Nachteil	Wechsel zu „serverless“ schwierig	7	kurzzeitig
Nachteil	Komplexität großer Anwendungen	6	Dauerhaft
Nachteil	Wenige Tools auf dem Markt	4	kurzzeitig

Tabelle 5: Zusammenfassung der Vor- und Nachteile der Serverless Programmierung

#### 9.4 Zusammenfassung des Kapitels

Die Auflistung und anschließende Erläuterung der Argumente bezüglich der Serverless Programmierung hat gezeigt, dass die Vor- und Nachteile je nach Betrachtung als wichtig oder zu vernachlässigen eingestuft werden können. Die Gewichtung der Vorteile ist auf Grund der Skalierbarkeit, der hohen Flexibilität, der Kostenminimierung, der geringen Personaldecke und der Kombinationsmöglichkeiten als stärker als die Nachteile anzusehen. Weiterhin muss in Betracht gezogen werden, dass einige der Nachteile ihre Begründung in der derzeitigen Situation der Verbreitung der Serverless Programmierung ihren Ursprung finden. Wenn sich der Bereich fest im Kreise der Softwareentwicklung etabliert hat, dann sind diverse Nachteile hinfällig, weshalb sie nun zwar anzubringen, jedoch als nicht ausschlaggebend für eine allgemeine Bewertung der Thematik anzusehen sind.

## 10. Fazit

Im folgenden wird ein ausführliches Fazit zu der vorliegenden Ausarbeitung gegeben. Dabei wird auf die Ausarbeitung im allgemeinen ebenso eingegangen wie auf ein persönliches Fazit. Als Abschluss werden Probleme, die während der Bearbeitung aufgetaucht sind, beleuchtet und Abweichungen vom eigentlichen Umsetzungsplan aufgezeigt.

### 10.1 Allgemeines Fazit

Die Ausarbeitung hat gezeigt, dass es zwar einige Varianten der regulären und bisher umgesetzten Möglichkeiten in der Softwareentwicklung gibt, jedoch sind diese auf Grund ihres Alters, der Dauer und den daraus resultierenden Nachteilen mitunter nicht mehr zeitgemäß. Die Problemstellung der Nachteile der Verwendung von Servern konnte im Zuge der Ausarbeitung analysiert und bestätigt werden. Die Technik hat sich in den letzten Jahren extrem verändert und deshalb ist es nötig sich auch im Bereich der Entwicklung von Anwendungen auf diese Neuerungen einzurichten und sich an sie anzupassen. Die Anforderungen an Software haben sich zudem verändert. Die Anwendungen sollen immer kleiner werden und auch auf mobilen Geräten aufrufbar sein. Viele existierende Programme werden nach und nach auch auf mobile Plattformen ausgeweitet. Dort sind bisherige Verfahren nicht ohne Weiteres übertragbar. Deshalb werden heute Methoden benötigt, die darauf basieren, die Anfragen in einer Anwendung möglichst schnell und präzise abarbeiten zu können und schnelle Ergebnisse zu präsentieren. Heute ist insbesondere der mobile Anwendungsmarkt hart umkämpft, jeden Tag erscheinen unzählige neue Apps in eine der verschiedenen App-Stores der unterschiedlichen Betriebssysteme. Wer sich dort dauerhaft etablieren möchte, muss sich regelmäßig mit der Kritik der Anwender auseinandersetzen. Auch wenn es immer mehr Mobilfunkanbieter gibt, die Flatrates für die mobile Datennutzung zur Verfügung stellen, so haben diese durch das tägliche Verwenden von diversen Apps meist keine allzu lange Lebensdauer und die anschließende Drosselung der Geschwindigkeit führt dazu, dass einige Anwendungen für den User nicht mehr verwendbar sind. Bei wichtigen Apps, die auch unterwegs ohne WLAN jederzeit aufrufbar sein müssen, ist dieser Zustand unvorteilhaft.

Die Vorteile der Serverless Programmierung haben deutlich gemacht welchen Nutzen die neue Möglichkeit der Softwareumsetzung bietet. Daher ist es als sinnvoll anzusehen, wenn Programmierer und Unternehmen sich nicht nur an alt bekannte Varianten einer Umsetzung halten, sondern sich an die aktuellen Gegebenheiten des Markts anpassen und sich nach der passendsten Methode für das jeweilige Projekt richten und vorher die

Möglichkeiten gegeneinander abwägen, um eine möglichst effiziente Anwendung zu erhalten.

Zwar gibt es auch hier Nachteile, die angeführt wurden, jedoch sind diese maßgeblich der derzeitigen geringen Verbreitung der Serverless Programmierung geschuldet und werden sich somit in absehbarer Zeit von selbst erledigen bzw. ihr Effekt wird geringer werden. Mit einer größeren Verbreitung der Thematik steigt auch die Bereitschaft seitens von Programmierern, sich mit den neuen Möglichkeiten auseinanderzusetzen. Die Einarbeitungszeit und die Zeit der Umstellung wird somit verkürzt. Insbesondere die Veröffentlichung von Literatur wird die Bereitschaft zur Einarbeitung in die Thematik der Serverless Programmierung steigern.

Die Messungen haben gezeigt, dass die Latenz recht gering ist und die Anwendung sofort geladen wird, auch bei der ersten Verwendung, bei der es laut AWS selbst zu einer verzögerten Ladezeit kommen kann. Auch beim Zugriff von mehreren Anwendern zur gleichen Zeit hält die Anwendung stand und kann regulär verwendet werden. Selbst bei einem zeitgleichen Zugriff von mehreren Anwendern ist die Latenz verhältnismäßig gering. Sie erhöht sich zwar insgesamt, wenn mehr User auf die Anwendung zugreifen, jedoch nicht proportional sondern abgeschwächt. Der Vergleich mit der Plattform Dropbox zeigte, dass schon das Laden der Startseite eine wesentlich höhere Latenz verursacht als bei der „serverless“ Umsetzung der Ausarbeitung. Hier ist demnach eindeutig ein Potential für die Zukunft zu sehen. Der Vergleich der Latenz innerhalb der Anwendung zeigt eine Erhöhung dieser an, je mehr Aktionen für die Durchführung einer Funktion durchlaufen werden müssen. Jedoch kommt es dennoch zu keinen Defiziten in der Benutzung der Anwendung.

Der Durchsatz verringert sich mit Verwendung von Funktionen, die die Durchführung mehrerer Aktionen intern erfordern. Dennoch werden die Dateien schnell hochgeladen, auch wenn mehrere User auf die Anwendung zugreifen. Nur wenn zeitgleich eine große Anzahl an Anwender aktiv ist, kann es zu Verzögerungen bei der Ausführung des Hochladens kommen.

## 10.2 Persönliches Fazit

Zunächst musste ich mich mit JavaScript auseinandersetzen, welches ich im Zuge der Arbeit mit der Plattform AWS zum Aufbau der Funktionen verwendet habe. Da ich weder in meinem Bachelor noch in meinem Master mit JavaScript gearbeitet habe, ging der Einarbeitung mit der eigentlichen Thematik Serverless Programmierung erst ein intensives

Training mit JavaScript voraus.

Unabhängig von der Tatsache, dass ich in der vorliegenden Ausarbeitung mit AWS als Plattform gearbeitet habe, weil diese recht weit verbreitet ist und sie gute Einarbeitungsmöglichkeiten für Anfänger bietet, empfinde ich AWS als sehr sinnvoll aufgebaut. Ich habe mich im Zuge von Kapitel 4 mit anderen Anbietern solcher Plattformen auseinandergesetzt und mir diese näher betrachtet und der Aufbau dort war teilweise sehr unübersichtlich und durcheinander, so dass es oft nicht gut ersichtlich war, wo sich welcher Service verbirgt und wie sich mit diesen arbeiten lässt. Zwar gab es oftmals Einführungen in Form von Dokumentationen dazu, jedoch sollte der Aufbau eigentlich nahezu selbsterklärend sein, um unnötig viel Zeit zu vermeiden und somit auch das Fehlerrisiko zu verringern. Bei AWS werden alle Services übersichtlich auf einen Blick dargestellt und sind schnell zu finden. Weiterhin werden die meisten der Services über eine vorgefertigte Maske erstellt, auf der alle relevanten Einstellungen und Codeeingabemöglichkeiten vorhanden sind, so dass der Anwender genau sehen kann, was von ihm erwartet wird. AWS arbeitet auch viel mit Beispieleintragungen in den Textfeldern, so dass der Anwender schnell eine Vorstellung von dem bekommt, was er dort eintragen soll bzw. in welchem Format dies erfolgen soll. Auch die Tatsache, dass ich bei AWS nicht mit den Worten „Sie können 30 Tage testen“ begrüßt werde, empfand ich als angenehm, da ich so nicht das Gefühl hatte, ich muss mich gezwungener Maßen nun irgendwo anmelden und darauf achten, etwas zu kündigen. Da ich beim Versandhandel Amazon bereits seit vielen Jahren einen Account habe, konnte ich mich mit diesen Daten auch bei AWS einloggen und musste so kein zusätzliches Konto erstellen.

Es war sehr interessant, sich in das Themengebiet der Serverless Programmierung einzuarbeiten und sich damit zu befassen. Zeigt es doch wie wichtig es ist sich neuen Entwicklungen im Bereich der Softwareentwicklung zu öffnen, anstatt an Altbekanntem festzuhalten. Ich bin mir sicher, ohne die vorliegende Ausarbeitung hätte ich in naher Zukunft nichts von dem Begriff der Serverless Programmierung gehört und sie nicht wahrgenommen. Es ist zu hoffen, dass durch die bevorstehende Veröffentlichung einiger Bücher etwas Leben in die Thematik kommt und sich die Gemeinde der Anwender stetig weiterentwickelt. Allerdings war der Effekt der insgesamt geringen Literatur und somit der Recherchemöglichkeiten bei auftretenden Problemen über die Dauer der Ausarbeitung gesehen recht hoch. Immer wieder musste viel mehr Zeit in die Vollendung eines Abschnitts der Anwendung investiert werden, weshalb am Ende nicht alles so umgesetzt werden konnte, wie ich es gerne gehabt hätte.

### 10.3 Probleme und Abweichungen

Während der Erstellung der Ausarbeitung und des Prototyps entstanden Probleme und Abweichungen vom eigentlichen Plan der Umsetzung, welche an dieser Stelle kurz aufgezeigt und erläutert werden sollen.

Zunächst war es geplant, eine umfangreiche Anmeldefunktion in die Anwendung zu integrieren, bei der sich verschiedene Anwender einen eigenen Account innerhalb der Anwendung erstellen können. Dies entwickelte sich jedoch zu einem größeren Problem, da sich die geplante Umsetzung nicht so einbauen ließ wie erhofft. Nachdem auch nach einigen Wochen keine Lösung erzielt werden konnte, um das umzubauen, was gewünscht war, wurde das Projekt Anmeldung beiseite gelegt und sich den anderen Funktionen der Anwendung gewidmet.

Auch der Mehrfachupload von Dateien konnte nicht umgesetzt werden, da die verwendeten Komponenten diese Funktion nicht unterstützen und dies eines Umbaus der kompletten Anwendung bedarf, welcher auf Grund des zeitlichen Aufwands zum Zeitpunkt der Erkenntnis über diesen Zustand nicht umsetzbar war. Zudem war es das Ziel die Anwendung so einfach wie möglich in ihrer Umsetzung zu gestalten, dass Privatanwender mit wenigen Schritten diese bei sich integrieren können. Die Verwendung von Services, die tiefgreifendes Wissen benötigen war daher nicht zweckmäßig für das Konzept.

Die aufgetretenen Probleme sind maßgeblich dem Umstand geschuldet, dass die Einarbeitung in die Thematik der Serverless Programmierung und in die dazu benötigten Komponenten länger als gedacht gedauert haben und ein Verständnis der Zusammenhänge der einzelnen Elemente nur nach und nach entstand.

Es kam zu einer kurzzeitigen Sperrung des AWS Accounts, da sich in einigen der JavaScript-Dateien die Keys zum Aufbau der Verbindung zum Account befinden und diese bei Github hochgeladen wurden. Sofort danach kam eine Email von AWS, die dazu aufforderte dies sofort zu unterbinden und neue Keys zu generieren, was nicht ohne Weiteres möglich war und einige Zeit mit Hilfe des Supports benötigte. Seitdem habe ich auf das Hochladen bei Github verzichtet, um keine endgültige Sperrung zu riskieren.

Bei der Umsetzung der Funktion war es erforderlich JavaScript und den Zugriff auf AWS-Komponenten in einer Datei miteinander zu verknüpfen. Die Unkenntnis über beide Bereiche hat deshalb relativ viel zusätzliche Zeit in Anspruch genommen, um die Verbindung herstellen zu können.

## 10.4 Mögliche Ergänzungen der Anwendungen

Die im Zuge dieser Ausarbeitung erstellte Anwendung wurde für den privaten Gebrauch oder die Verwendung einer Abteilung oder Projektgruppe eines Unternehmens entwickelt. Dabei sind durchaus auch andere Möglichkeiten einer Einsetzung denkbar, die weitere Ergänzungen benötigen würden.

Eine Möglichkeit des Löschens von Dateien bei einem falschen Upload oder bei veralteten Dateien ist dabei eine sinnvolle Funktion. Bei der Verwendung für einen weitreichenderen Zwecks wäre auch das Arbeiten mit einem Passwort für bestimmte Ordner denkbar, wenn nicht alle Anwender jeden Ordner sehen können dürfen. Eine Ausweitung auf einer Anwendung, bei der generell jede Art von Nutzer Bilder hochladen können, würde ein ausgereiftes System zur Erstellung von Nutzerkonten benötigen.

Die Einbindung der Möglichkeit eines Mehrfachuploads von Dateien wäre eine sinnvolle Erweiterung, wenn viele Bilder und Dateien regelmäßig in die Anwendung integriert werden sollen. Da dies mit dem Service S3 nicht ohne weiteres möglich ist, da dies in den Richtlinien nicht erlaubt ist, würde dies einen Umbau der Anwendung mit Ausweichen auf andere Services bedeuten.

Eine Begrenzung der Dateigröße bei einer größeren Anwendung, auf die viele Anwender pro Tag zugreifen, ist eine Möglichkeit die ansonsten sehr schnell steigende Gesamtgröße des S3 Buckets unter Kontrolle zu halten.

Eine Erweiterung, bei der die Anwender nur Zugriff auf die von ihnen erstellten Ordner haben ist ebenfalls für eine größere Umsetzung denkbar. Bei der Anwendung der vorliegenden Ausarbeitung soll jeder Anwender Zugang zu allen Ordner haben, da es um eine Archivierung im Familien- und Freundeskreis oder einer Projektgruppe geht und die Dateien thematisch sortiert werden.

Die Anwendung kann demnach für verschiedene Zwecke verwendet werden und vor allem für jeden dieser Anwendungsfälle individuell erweitert oder verändert werden. Das Grundgerüst bietet dabei eine Basis, um die gewünschte Richtung der Verwendung gestalten zu können.

## 11. Ausblick in die Zukunft

Wie bereits auf den vorherigen Seiten erwähnt, ist der Bereich der Serverless Programmierung keine Erfindung, die erst seit kurzem existiert. Bereits seit Jahren entwickelt er sich stetig weiter. Doch die nur langsam vorangehende Verbreitung dieser Möglichkeit sorgt für einen langwierigen Weg.

Es sind derzeit einige Bücher in Planung und Bearbeitung, die sich explizit mit der Thematik der Serverless Programmierung beschäftigen und die dafür sorgen werden, dass dieses Thema mehr Aufmerksamkeit erhalten wird. Allerdings ist dabei festzuhalten, dass es sich dabei um englischsprachige Literatur handelt und es wohl noch einige Jahren dauern wird, bis diese auch in den jeweiligen Landessprachen zur Verfügung stehen wird.

Weiterhin ist davon auszugehen, dass angesichts des stetig ansteigenden Interesses an der Thematik der Serverless Programmierung die Zahl der dafür verwendbaren Werkzeuge steigen wird. Insbesondere die Verwaltung der einzelnen Elemente ist bei größeren Projekten bezüglich der Übersichtlichkeit wichtig.

Weitere Anbieter von Services für einer „serverless“ Umsetzung werden Einzug auf den Markt erhalten und es wird sich nach und nach zeigen, welcher dieser Anbieter sich an die Spitze setzen wird. Auch wenn AWS auf Grund des frühen Veröffentlichungsdatums derzeit am weitesten verbreitet ist, kann sich dies durch Veränderungen bereits bestehender Plattformen ändern. Es ist mit einer stetigen Veränderung der Anbieter bezüglich der angebotenen Möglichkeiten „serverless“ zu Arbeiten zu rechnen. Diese geht zwar mit Verbesserungen einher, aber auch mit einer regelmäßigen Auseinandersetzung seitens der Entwickler mit den jeweiligen Umständen. Es wird einige Zeit dauern, ehe sich ein Standard bzw. Regeln in dem Bereich durchgesetzt haben und alle Entwickler nach dem gleichen roten Faden ihre „serverless“ Anwendungen konzipieren.

Generell ist es als sinnvoll anzusehen, dass die Anbieter der Plattformen, die Serverless Programmierung bisher anbieten, möglichst große und bekannte Unternehmen für sich gewinnen, um die Verbreitung der Plattformen voranzutreiben. Die Hoffnung besteht, dass die Aufmerksamkeit auf die Serverless Programmierung gelenkt wird. Es werden Programmierer darauf aufmerksam, wenn in der Öffentlichkeit mehr Anwendungen „serverless“ erstellt werden und so ihr Interesse geweckt wird.

Das Potential der Serverless Programmierung ist groß und sie wird die Zukunft erst einmal eine Weile bestimmen. Der Weg zu einem ausgewogenen etablierten Gebiet der Softwareentwicklung bedarf jedoch Zeit und Geduld.

## Abbildungsverzeichnis

Abb. 1: Aufbau der Google Cloud Plattform (Quelle: Screen aus Google Cloud Platform).....	20
Abb. 2: Aufbau der Plattform von Windows (Quelle: <a href="https://wazcommunity.wordpress.com">https://wazcommunity.wordpress.com</a> ).....	21
Abb. 3: Aufbau der Plattform von IBM (Quelle: <a href="https://venturebeat.com">https://venturebeat.com</a> ).....	22
Abb. 4: Übersicht der Services von AWS (Quelle: <a href="https://aws.amazon.com/de/">https://aws.amazon.com/de/</a> ).....	24
Abb. 5: Beispiel einer ARN einer Lambda-Funktion (Quelle: Eigene Lambda-Funktion auf <a href="https://aws.amazon.com/de/">https://aws.amazon.com/de/</a> ).....	25
Abb. 6: Möglichkeiten der Verknüpfung von Services in AWS (Quelle: <a href="http://docs.aws.amazon.com/lambda/latest/dg/with-cloudtrail.html">http://docs.aws.amazon.com/lambda/latest/dg/with-cloudtrail.html</a> ).....	30
Abb. 7: UseCase Diagramm zur Übersicht des Funktionsaufbaus der Anwendung.....	32
Abb. 8: Komponentendiagramm zur Übersicht der verwendeten Elemente in AWS.....	33
Abb. 9: Startseite der Anwendung.....	35
Abb. 10: Abrufen des Inhalts eines Ordners der Anwendung.....	36
Abb. 11: Übersicht der im S3 Bucket befindlichen Ordner und Dateien.....	37
Abb. 12: Verbindung mit AWS Konto, S3 Bucket und DynamoDB Tabelle herstellen.....	38
Abb. 13: Auslesen des Ordnernamens und Hinzufügen eines Event Listeners zum Upload-Button.....	39
Abb. 13.1: Überprüfung auf leere Felder und Dateiformat vor Hochladen einer Datei .....	39
Abb. 14: Festlegung des Aufbaus der hochzuladenden Elemente sowie Definition eigentliches Hochladens.....	40
Abb. 14.1: Erstellung der Auflistung des Inhalts eines Ordners.....	40
Abb. 15: Erstellung einer Vorschau des zum Upload ausgewählten Bildes.....	41
Abb. 16: Einbindung von Modulen zur Änderung der.....	42
Abb. 17: Festlegung des Ausgabe-Buckets und der Eigenschaften des verkleinerten Bildes.....	43
Abb. 18: Speichern des verkleinerten Bildes in Zielordner des Ausgabe-Bucket.....	44
Abb. 19: Änderung des Links auf die Endung ".jpg".....	44
Abb. 20: Zugriffsrechte für den Bucket der Anwendung.....	45
Abb. 21: Lambda Funktion zum Speichern des Ordnerinhalts in die DynamoDB Tabelle.....	46
Abb. 22: Trigger für die Lambda Funktion der Größenänderung.....	47
Abb. 23: Informationen über erstellte DynamoDB Tabelle mit Primary Key für Ordner...	48
Abb. 24: Übersicht über bereits in DynamoDB eingetragene Dateien.....	48
Abb. 25: Prüfung der definierten Nutzerdaten und Aufrufen der Lambda-Funktion zum Einloggen.....	49
Abb. 26: Zugriffsrechte für autorisierte Anwender für den Login.....	50

## Tabellenverzeichnis

Tabelle 1: Erläuterung von Serverarten.....	8
Tabelle 2: Übersicht der verwendeten Elemente der einzelnen Funktionen des Prototyps.....	52
Tabelle 3: Auflistung der verwendeten Elemente innerhalb der Plattform AWS mit dessen Inhalt.....	52
Tabelle 4: Zusammenfassung der Ergebnisse der durchgeföhrten Messungen.....	58
Tabelle 5: Zusammenfassung der Vor- und Nachteile der Serverless Programmierung.....	72

## Literaturverzeichnis

Abts, Dietmar (2015): Masterkurs Client/Server-Programmierung mit Java - Anwendungen entwickeln mit Standard-Technologien, Wiesbaden: Springer Fachmedien.

AWS ARN: Amazon Resource Names (ARNs) and AWS Service Namespaces, <http://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html> (Abgerufen: 13.11.2016)

AWS freies Kontingent: Kostenloses Kontingent für AWS, <https://aws.amazon.com/de/free/> (Abgerufen: 20.12.2016)

AWS Guide: AWS Identity and Access Management, <http://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html> (Abgerufen: 11.12.2016)

AWS Konto: Creating an AWS Account, <http://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/AboutAWSAccounts.html> (Abgerufen: 03.11.2016)

AWS Preise: Preise für Cloud-Services, <https://aws.amazon.com/de/pricing/services/> (Abgerufen. 20.12.2016)

AWS Regionen: Tabelle der Regionen, <https://aws.amazon.com/de/about-aws/global-infrastructure/regional-product-services/> (Abgerufen: 11.12.2016)

AWS Sprache (2016): Does AWS provide support in languages other than English?, <https://aws.amazon.com/de/premiumsupport/knowledge-center/aws-support-languages/> (Abgerufen: 17.02.2017)

Balzert, Helmut (2011): Lehrbuch der Softwaretechnik - Entwurf, Implementierung, Installation und Betrieb, Heidelberg: Spektrum Akademischer Verlag.

Bengel, Günther (2014): Grundkurs Verteilte Systeme, Wiesbaden: Springer Vieweg.

Boyd, Mark (2016): Serverless Computing Use Cases: Image Processing, Social and Cognition, <https://thenewstack.io/serverless-computing-use-cases-image-processing-social-cognition/> (Abgerufen: 11.12.2016)

Casalboni, Alex (2016): Google Cloud Functions VS AWS Lambda: fight for serverless cloud domination begins, <http://cloudacademy.com/blog/google-cloud-functions-serverless/> (Abgerufen: 12.02.2017)

- Chan, Mike (2017): Serverless Architectures: Everything You Need to Know,  
<https://www.thorntech.com/2017/03/serverless-architectures-everything-need-know/>  
(Abgerufen: 15.04.2017)
- Cuva, Marc (2017): Writing a cron job microservice with Serverless and AWS Lambda,  
<https://blog.readme.io/writing-a-cron-job-microservice-with-serverless-and-aws-lambda/> (Abgerufen: 12.02.2017)
- Fink, Stephen (2016): Build a Serverless Spell Checker with OpenWhisk and Docker,  
<https://developer.ibm.com/openwhisk/2016/06/17/spellcheck-openwhisk-docker/>  
(Abgerufen: 13.12.2016)
- Froehlich, Andrew (2016): Serverless Architecture Pros And Cons,  
<http://www.networkcomputing.com/data-centers/serverless-architecture-pros-and-cons/2090515944> (26.10.2016)
- Google (a): Google Cloud Functions Documentation , <https://cloud.google.com/functions/docs/> (Abgerufen: 17.11.2017)
- Google (b): Cloud Functions Pricing, <https://cloud.google.com/functions/pricing>  
(Abgerufen: 07.02.2017)
- Google Blog (2017): Google Cloud Functions: a serverless environment to build and connect cloud services ,[https://cloudplatform.googleblog.com/2017/03/Google-Cloud- Functions-a-serverless-environment-to-build-and-connect-cloud-services\\_13.html](https://cloudplatform.googleblog.com/2017/03/Google-Cloud- Functions-a-serverless-environment-to-build-and-connect-cloud-services_13.html) (Abgerufen: 13.04.2017)
- Heil, Andreas (2012): Anwendungsentwicklung für Intelligente Umgebungen im Web Engineering, Wiesbaden: Springer Vieweg.
- Horn, Thorsten: SOA (Service Oriented Architecture), <http://www.torsten-horn.de/techdocs/soa.htm> (Abgerufen: 13.11.2016)
- IBM Cloud: IBM Bluemix OpenWhisk, <https://www.ibm.com/cloud-computing/bluemix/de/openwhisk> (Abgerufen: 09.03.2017)
- Janakiram (2016): Five Serverless Computing Frameworks To Watch Out For,  
<https://www.janakiram.com/posts/blog/five-serverless-computing-frameworks-to-watch-out-for> (07.10.2016)
- Janakiram MSV (2017): An Introduction to OpenWhisk, IBM's Open Source Serverless Computing Platform, <https://thenewstack.io/openwhisk-open-source-serverless-computing-platform/> (Abgerufen: 17.02.2017)

- Janakiram MSV (2016): Why Enterprises Should Care About Serverless Computing,  
<https://www.forbes.com/sites/janakirammsv/2016/10/12/why-enterprises-should-care-about-serverless-computing/#789eb2365583> (Abgerufen: 09.03.2017)
- Lambda Documentation: AWS Lambda – Developer Guide, <http://docs.aws.amazon.com/lambda/latest/dg/lambda-dg.pdf> (Abgerufen: 13.11.2016)
- Looi, Mark (2016): AWS Lambda and Serverless Computing, <https://www.linkedin.com/pulse/aws-lambda-serverless-computing-mark-looi> (26.10.2016)
- Maiaroto, Tom (2016): Azure Cloud Functions vs. AWS Lambda,  
<https://serifandsemaphore.io/azure-cloud-functions-vs-aws-lambda-caf8a90605dd> (Abgerufen: 07.12.2016)
- Mak, Mick (2017): Save 80% of development time with Serverless architecture,  
<https://blog.magicsketch.io/save-80-of-development-time-with-serverless-architecture-6a3767032274> (Abgerufen: 15.04.2017)
- Microsoft Azure (2017): Einführung in Azure Functions, <https://docs.microsoft.com/de-de/azure/azure-functions/functions-overview> (Abgerufen: 15.04.2017)
- Microsoft Azure (2016): Continuous Deployment für Azure Functions,  
<https://docs.microsoft.com/de-de/azure/azure-functions/functions-continuous-deployment> (Abgerufen: 26.11.2016)
- n.A. (2016): What is Amazon Lambda, <https://www.iopipe.com/what-is-amazon-lambda/> (04.09.2016)
- Neumann, Alexander (2016): Bluemix OpenWhisk: IBM stellt Konkurrenten zu AWS Lambda und Google Cloud Functions vor, <https://www.heise.de/developer/meldung/Bluemix-OpenWhisk-IBM-stellt-Konkurrenten-zu-AWS-Lambda-und-Google-Cloud-Functions-vor-3115411.html> (Abgerufen: 21.11.2016)
- Novet, Jordan (2016): Google has quietly launched its answer to AWS Lambda,  
<https://venturebeat.com/2016/02/09/google-has-quietly-launched-its-answer-to-aws-lambda/> (Abgerufen: 23.03.2017)
- NPM: packages with keyword ‘aws’, <https://www.npmjs.com/browse/keyword/aws> (Abgerufen: 3.12.2016)
- O'Reilly, Dennis (2016): The Benefits of Serverless Applications in a Data-Rich World,  
<https://dzone.com/articles/the-benefits-of-serverless-applications-in-a-data> (26.10.2016)

Roberts, Mike (2016): Serverless Architectures, <https://martinfowler.com/articles/serverless.html> (04.09.2016)

Sbarski, Peter (2017a): The essential guide to serverless technologies and architectures, <https://techbeacon.com/essential-guide-serverless-technologies-architectures> (Abgerufen: 23.04.2017)

Schäfer, Werner (2010): Softwareentwicklung, München: Pearson Studium.

Schwichtenberg, Holger (2010): Sinn und Unsinn der Windows Communication Foundation – Vor- und Nachteile eines Application Server, <https://www.heise.de/developer/artikel/Vor-und-Nachteile-eines-Application-Server-934674.html> (Abgerufen: 13.11.2016)

Starke, Gernot (2014): Effektive Softwarearchitekturen - Ein praktischer Leitpfaden, München: Carl Hanser Verlag.

Uni Ulm: Vor- und Nachteile von Client-Server Netzen, <http://www.mathematik.uni-ulm.de/~melzer/md29/network-10.html> (Abgerufen: 13.11.2016)

Zub, Dimitrij; Sack, Raphael (2016): Scripting Languages for AWS Lambda: Running PHP, Ruby, and Go, <https://aws.amazon.com/de/blogs/compute/scripting-languages-for-aws-lambda-running-php-ruby-and-go/> (Abgerufen: 17.02.2017)

## Quellen der Einarbeitung

### AWS:

Poccia, Danilo (2016): AWS in Action - Event-driven serverless applications, Shelter Island: Manning Publications Co.

Sbarski, Peter (2017b): Serverless Architectures on AWS, Shelter Island: Manning Publications Co.

### JavaScript:

Ackermann, Philip (2016): JavaScript – Das umfassende Handbuch, Bonn: Rheinwerk Verlag GmbH.

Koch, Stefan (2011): JavaScript – Einführung, Programmierung und Referenz,  
Heidelberg: d.punkt.verlag GmbH.

Theis, Thomas (2016): Einstieg in JavaScript, Bonn: Rheinwerk Verlag.

Wenz, Christian (2014): JavaScript – Grundlagen, Programmierung, Praxis, Bonn: Galileo Press.

# **ANHANG**

## Anhang 1 – Ergebnisse der Messungen

### Messungen für die Startseite von Dropbox mit einer Ausführungszeit von 1 Sekunde

Messung Startseite Dropbox – 1 User

Label	Anz. der Pro...	Durchsch... ▾	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	1	1780	1780	1780	1780	1780	1780	1780	0,00%	33,7/min	78,03	0,07
Gesamt	1	1780	1780	1780	1780	1780	1780	1780	0,00%	33,7/min	78,03	0,07

Messung Startseite Dropbox – 25 User

Label	Anz. der Pro...	Durchsch... ▾	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	25	9207	10118	11044	11354	11945	3170	11945	0,00%	2,0/sec	274,52	0,29
Gesamt	25	9207	10118	11044	11354	11945	3170	11945	0,00%	2,0/sec	274,52	0,29

Messung Startseite Dropbox – 100 User

Label	Anz. der Pro...	Durchsch... ▾	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	100	34412	32956	47048	48120	70538	19080	72854	0,00%	1,4/sec	190,36	0,21
Gesamt	100	34412	32956	47048	48120	70538	19080	72854	0,00%	1,4/sec	190,36	0,21

## Messungen für die Startseite von Dropbox mit einer Ausführungszeit von 300 Sekunden

Messung Startseite Dropbox – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	1	1818	1818	1818	1818	1818	1818	1818	0,00%	33,0/min	76,32	0,06
Gesamt	1	1818	1818	1818	1818	1818	1818	1818	0,00%	33,0/min	76,32	0,06

Messung Startseite Dropbox – 25 User

Label	Anz. der Pro...	Durchsch... ▾	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	25	3256	2491	5418	6816	7779	1351	7779	0,00%	5,1/min	11,86	0,01
Gesamt	25	3256	2491	5418	6816	7779	1351	7779	0,00%	5,1/min	11,86	0,01

Messung Startseite Dropbox – 100 User

Label	Anz. der Pr...	Durchsch... ▾	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	100	2564	2209	3823	3989	4125	1579	4185	0,00%	20,1/min	46,62	0,05
Gesamt	100	2564	2209	3823	3989	4125	1579	4185	0,00%	20,1/min	46,62	0,05

## **Messungen für die Startseite vom Prototyp mit einer Ausführungszeit von 1 Sekunde**

**Messung Startseite Prototyp – 1 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	1	459	459	459	459	459	459	459	0,00%	2,2/sec	3,88	0,33
Gesamt	1	459	459	459	459	459	459	459	0,00%	2,2/sec	3,88	0,33

**Messung Startseite Prototyp – 25 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	25	487	466	507	522	1132	385	1132	0,00%	16,8/sec	30,02	2,57
Gesamt	25	487	466	507	522	1132	385	1132	0,00%	16,8/sec	30,02	2,57

**Messung Startseite Prototyp – 100 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	100	2714	2114	4219	5013	9408	450	9419	0,00%	9,6/sec	17,05	1,46
Gesamt	100	2714	2114	4219	5013	9408	450	9419	0,00%	9,6/sec	17,05	1,46

## **Messungen für die Startseite vom Prototyp mit einer Ausführungszeit von 300 Sekunden**

**Messung Startseite Prototyp – 1 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	1	417	417	417	417	417	417	417	0,00%	2,4/sec	4,27	0,37
Gesamt	1	417	417	417	417	417	417	417	0,00%	2,4/sec	4,27	0,37

**Messung Startseite Prototyp – 25 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	25	439	412	490	568	1004	357	1004	0,00%	5,2/min	0,15	0,01
Gesamt	25	439	412	490	568	1004	357	1004	0,00%	5,2/min	0,15	0,01

**Messung Startseite Prototyp – 100 User**

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
startseite	100	452	391	451	502	3008	347	3009	0,00%	20,2/min	0,60	0,05
Gesamt	100	452	391	451	502	3008	347	3009	0,00%	20,2/min	0,60	0,05

## Messungen für das Hochladen einer Datei in die Dropbox mit einer Ausführungszeit von 1 Sekunde

Messung Dropbox Datei hochladen – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	1	1279	1279	1279	1279	1279	1279	1279	0,00%	46,9/min	32,61	0,32
/ajax_login	1	830	830	830	830	830	830	830	0,00%	1,2/sec	0,93	22,43
post token	1	444	444	444	444	444	444	444	0,00%	2,3/sec	1,69	10,47
Subscribe	1	44727	44727	44727	44727	44727	44727	44727	0,00%	1,3/min	0,01	0,01
Dropbox Home	1	1569	1569	1569	1569	1569	1569	1569	0,00%	38,2/min	55,41	0,18
post hochzul...	1	4576	4576	4576	4576	4576	4576	4576	0,00%	13,1/min	0,23	30,45
Gesamt	6	8904	1279	4576	44727	44727	444	44727	0,00%	6,7/min	2,46	3,06

Messung Dropbox Datei hochladen – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	25	2822	2747	3626	4228	5163	1719	5163	0,00%	4,8/sec	200,85	2,22
/ajax_login	25	9761	9757	14380	14414	14474	2499	14474	0,00%	1,5/sec	1,19	28,71
post token	25	3685	3599	7138	7218	9598	447	9598	0,00%	1,6/sec	1,16	7,21
Subscribe	25	41440	37791	53051	54947	59191	31136	59191	0,00%	23,3/min	0,16	0,18
Dropbox Home	25	2936	2047	5946	8832	11203	1064	11203	0,00%	51,5/min	74,66	0,24
post hochzul...	25	61785	62116	77148	92232	104149	36136	104149	0,00%	12,2/min	0,43	28,26
Gesamt	150	20405	7138	58456	65166	92232	447	104149	0,00%	52,7/min	19,44	23,96

Messung Dropbox Datei hochladen – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	100	11840	8599	21003	23748	24699	2063	27214	0,00%	3,6/sec	104,26	1,29
/ajax_login	100	16842	21001	21003	21004	23963	2805	28174	0,00%	2,5/sec	3,89	21,96
post token	100	6830	5228	14941	21002	21002	1163	21003	0,00%	2,4/sec	2,14	10,17
Subscribe	100	44133	44870	55142	55911	61569	20999	66899	0,00%	1,1/sec	0,56	0,49
Dropbox Home	100	14120	12480	26378	31077	32388	1395	53812	0,00%	58,8/min	55,79	0,19
post hochzul...	100	70652	77783	121687	130550	136000	12193	138742	0,00%	30,8/min	1,37	30,06
Gesamt	600	27403	21001	57689	103642	125016	1163	138742	0,00%	2,6/sec	39,56	31,28

## **Messungen für das Hochladen einer Datei in die Dropbox mit einer Ausführungszeit von 300 Sekunden**

Messung Dropbox Datei hochladen – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	1	1211	1211	1211	1211	1211	1211	1211	0,00%	49,5/min	34,40	0,34
/ajax_login	1	851	851	851	851	851	851	851	0,00%	1,2/sec	0,91	21,87
post token	1	453	453	453	453	453	453	453	0,00%	2,2/sec	1,65	10,26
Subscribe	1	30533	30533	30533	30533	30533	30533	30533	0,00%	2,0/min	0,01	0,01
Dropbox Home	1	1174	1174	1174	1174	1174	1174	1174	0,00%	51,1/min	74,13	0,24
post hochzul...	1	5875	5875	5875	5875	5875	5875	5875	0,00%	10,2/min	0,18	23,71
Gesamt	6	6682	1174	5875	30533	30533	453	30533	0,00%	9,0/min	3,28	4,08

Messung Dropbox Datei hochladen – 25 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	25	1503	1439	1942	2159	2214	899	2214	0,00%	5,2/min	3,61	0,04
/ajax_login	25	866	829	964	1195	1236	676	1236	0,00%	5,2/min	0,07	1,60
post token	25	477	452	574	666	813	334	813	0,00%	5,2/min	0,06	0,40
Subscribe	25	43813	44159	53266	54412	54497	30616	54497	0,00%	4,5/min	0,03	0,03
Dropbox Home	25	1302	1147	1456	2096	3631	959	3631	0,00%	5,2/min	7,57	0,02
post hochzul...	25	5649	4838	6951	10153	12436	3523	12436	0,00%	5,2/min	0,09	12,01
Gesamt	150	8935	1266	41622	49052	54412	334	54497	0,00%	26,6/min	9,74	12,09

Messung Dropbox Datei hochladen – 100 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	100	1841	1656	2641	2854	4656	1130	4936	0,00%	20,1/min	14,06	0,17
/ajax_login	100	1089	944	1586	1912	2951	683	2976	0,00%	20,1/min	0,26	6,24
post token	100	647	565	1062	1331	1901	333	2044	0,00%	20,1/min	0,25	1,56
Subscribe	100	43711	43608	51891	53058	54285	30397	54370	0,00%	17,6/min	0,12	0,13
Dropbox Home	100	1654	1395	2462	3307	4380	1157	5278	0,00%	19,9/min	28,92	0,09
post hochzul...	100	109363	112087	171504	184178	218655	7828	221879	0,00%	15,5/min	0,53	29,87
Gesamt	600	26384	1586	97078	125817	183034	333	221879	0,00%	1,4/sec	30,58	32,23

## **Messungen für das Hochladen einer Datei im Prototyp mit einer Ausführungszeit von 1 Sekunde**

**Messung Datei hochladen Prototyp – 1 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	484	484	484	484	484	484	484	0,00%	2,1/sec	3,68	0,31
/dateiHochlad...	1	79	79	79	79	79	79	79	0,00%	12,7/sec	27,71	2,03
post hochzula...	1	1229	1229	1229	1229	1229	1229	1229	0,00%	48,8/min	2,56	1,54
put AccessKe...	1	756	756	756	756	756	756	756	0,00%	1,3/sec	0,91	0,87
put der Datei i...	1	1341	1341	1341	1341	1341	1341	1341	0,00%	44,7/min	0,91	14,08
Gesamt	5	777	756	1229	1341	1341	79	1341	0,00%	1,3/sec	2,32	5,10

**Messung Datei hochladen Prototyp – 25 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	4546	1548	8171	21021	30511	1034	30511	0,00%	48,3/min	1,47	0,11
/dateiHochlad...	25	1820	296	1979	11697	12139	155	12139	0,00%	49,2/min	1,80	0,13
post hochzula...	25	9002	6653	16430	21021	21021	1344	21021	0,00%	47,4/min	2,41	1,06
put AccessKe...	25	2822	792	4114	21026	21032	450	21032	0,00%	53,5/min	0,73	0,54
put der Datei i...	25	5309	5005	6608	12261	12346	1455	12346	0,00%	56,5/min	1,15	17,78
Gesamt	125	4700	1997	12261	21021	21032	155	30511	0,00%	3,5/sec	6,37	13,83

**Messung Datei hochladen Prototyp – 100 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	16337	10418	37906	39098	39943	443	43444	0,00%	2,3/sec	4,18	0,31
/dateiHochlad...	100	11656	10464	21018	21021	21028	109	21030	0,00%	1,7/sec	3,84	0,14
post hochzula...	100	18173	15516	32783	37615	74897	1503	76936	0,00%	1,1/sec	3,34	1,82
put AccessKe...	100	8390	4859	21002	21009	21020	456	21024	0,00%	1,2/sec	1,19	0,61
put der Datei i...	100	12689	10841	21003	21108	22737	1331	25999	0,00%	1,2/sec	1,77	16,35
Gesamt	500	13449	10457	23607	32309	39943	109	76936	0,00%	5,2/sec	10,05	15,42

## **Messungen für das Hochladen einer Datei im Prototyp mit einer Ausführungszeit von 300 Sekunden**

**Messung Datei hochladen Prototyp – 1 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	726	726	726	726	726	726	726	0,00%	1,4/sec	2,45	0,21
/dateiHochlad...	1	151	151	151	151	151	151	151	0,00%	6,6/sec	14,50	1,06
post hochzula...	1	1229	1229	1229	1229	1229	1229	1229	0,00%	48,8/min	2,56	0,22
put AccessKe...	1	624	624	624	624	624	624	624	0,00%	1,6/sec	1,10	1,06
put der Datei i...	1	2469	2469	2469	2469	2469	2469	2469	0,00%	24,3/min	0,50	7,65
Gesamt	5	1039	726	1229	2469	2469	151	2469	0,00%	57,7/min	1,74	3,82

**Messung Datei hochladen Prototyp – 25 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	753	675	1008	1239	1679	569	1679	0,00%	5,2/min	0,15	0,01
/dateiHochlad...	25	143	132	162	286	294	69	294	0,00%	5,2/min	0,19	0,01
post hochzula...	25	1239	1228	1283	1300	1322	1206	1322	0,00%	5,2/min	0,27	0,59
put AccessKe...	25	898	896	1032	1133	1292	530	1292	0,00%	5,2/min	0,06	0,06
put der Datei i...	25	2063	1947	2483	2557	3703	1443	3703	0,00%	5,1/min	0,10	1,62
Gesamt	125	1019	971	1931	2183	2557	69	3703	0,00%	25,4/min	0,77	1,68

**Messung Datei hochladen Prototyp – 100 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	557	559	630	673	781	328	1555	0,00%	20,2/min	0,60	0,05
/dateiHochladen.html	100	101	87	162	175	197	67	197	0,00%	20,2/min	0,74	0,05
post hochzuladende Datei	100	1242	1232	1291	1326	1363	1142	1416	0,00%	20,1/min	1,05	0,66
put AccessKey S3	100	615	606	700	846	988	505	1162	0,00%	20,2/min	0,23	0,22
put der Datei in S3	100	1297	1244	1471	1652	1831	1187	2146	0,00%	20,1/min	0,41	6,33
Gesamt	500	763	629	1270	1316	1652	67	2146	0,00%	1,7/sec	3,00	6,60

## Messungen für das Hochladen einer Datei im Prototyp in einen Ordner mit einer Ausführungszeit von 1 Sekunde

Messung Datei hochladen Ordner Prototyp – 1 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	389	389	389	389	389	389	389	0,00%	2,6/sec	4,58	0,39
/dateiHochladenOrdner.h...	1	71	71	71	71	71	71	71	0,00%	14,1/sec	31,75	2,34
post hochzuladene Datei	1	1229	1229	1229	1229	1229	1229	1229	0,00%	48,8/min	2,56	0,78
put AccessKey S3	1	539	539	539	539	539	539	539	0,00%	1,9/sec	1,28	1,22
put der Datei in Ordner in...	1	94	94	94	94	94	94	94	0,00%	10,6/sec	7,53	6,95
put DynamoDB Dateien ...	1	817	817	817	817	817	817	817	0,00%	1,2/sec	0,87	23,11
get Ordnerinhalt	1	816	816	816	816	816	816	816	0,00%	1,2/sec	3,01	0,82
Gesamt	7	565	539	817	1229	1229	71	1229	0,00%	1,8/sec	2,97	5,35

Messung Datei hochladen Ordner Prototyp – 25 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	5246	9477	9548	9576	9723	352	9723	0,00%	2,3/sec	4,17	0,36
/dateiHochladenOrdner.h...	25	1380	569	1152	1257	21001	71	21001	0,00%	47,9/min	1,80	0,13
post hochzuladene Datei	25	11444	6440	24893	25220	27426	1494	27426	0,00%	45,9/min	2,41	1,22
put AccessKey S3	25	1094	830	1939	2443	2547	414	2547	0,00%	52,1/min	0,60	0,57
put der Datei in Ordner in...	25	205	112	284	658	668	73	668	0,00%	55,3/min	0,65	0,60
put DynamoDB Dateien ...	25	3806	2967	6053	6132	21002	763	21002	0,00%	54,0/min	0,70	16,30
get Ordnerinhalt	25	1214	659	1239	1667	12272	353	12272	0,00%	57,7/min	2,36	0,64
Gesamt	175	3484	947	9527	16650	25099	71	27426	0,00%	5,0/sec	8,38	14,48

Messung Datei hochladen Ordner Prototyp – 100 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	11039	10259	21015	21030	21030	453	21030	0,00%	4,5/sec	8,76	0,50
/dateiHochladenOrdner.h...	100	12886	4812	33789	38749	38936	119	39139	0,00%	1,6/sec	3,75	0,21
post hochzuladene Datei	100	18980	21011	35824	47441	62944	1240	85710	0,00%	1,1/sec	3,10	0,87
put AccessKey S3	100	5376	4237	10664	21016	21033	333	21038	0,00%	1,1/sec	0,92	0,70
put der Datei in Ordner in...	100	3085	1424	10007	10187	10579	70	21027	0,00%	1,2/sec	0,88	0,79
put DynamoDB Dateien ...	100	11378	11044	21031	21034	21043	741	21049	0,00%	1,2/sec	1,29	18,33
get Ordnerinhalt	100	3727	1527	7235	21020	21037	354	21039	0,00%	1,3/sec	3,17	0,80
Gesamt	700	9496	5567	21030	22856	41397	70	85710	0,00%	7,4/sec	12,83	18,09

## Messungen für das Hochladen einer Datei im Prototyp in einen Ordner mit einer Ausführungszeit von 300 Sekunden

### Messung Datei hochladen Ordner Prototyp – 1 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	432	432	432	432	432	432	432	0,00%	2,3/sec	4,13	0,35
/dateiHochladenOrdner.h...	1	88	88	88	88	88	88	88	0,00%	11,4/sec	25,61	1,89
post hochzuladene Datei	1	1227	1227	1227	1227	1227	1227	1227	0,00%	48,9/min	2,56	0,57
put AccessKey S3	1	841	841	841	841	841	841	841	0,00%	1,2/sec	0,82	0,78
put der Datei in Ordner in...	1	99	99	99	99	99	99	99	0,00%	10,1/sec	7,15	6,60
put DynamoDB Dateien ...	1	764	764	764	764	764	764	764	0,00%	1,3/sec	0,93	24,71
get Ordnerinhalt	1	358	358	358	358	358	358	358	0,00%	2,8/sec	6,86	1,86
Gesamt	7	544	432	841	1227	1227	88	1227	0,00%	1,8/sec	3,08	5,56

### Messung Datei hochladen Ordner Prototyp – 25 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	373	340	397	455	707	327	707	0,00%	5,2/min	0,15	0,01
/dateiHochladenOrdner.h...	25	74	70	85	95	110	68	110	0,00%	5,2/min	0,20	0,01
post hochzuladene Datei	25	1226	1226	1228	1231	1298	1176	1298	0,00%	5,2/min	0,27	0,79
put AccessKey S3	25	845	865	876	926	928	539	928	0,00%	5,2/min	0,06	0,06
put der Datei in Ordner in...	25	93	98	103	128	137	77	137	0,00%	5,2/min	0,06	0,06
put DynamoDB Dateien ...	25	789	773	796	887	1060	743	1060	0,00%	5,2/min	0,06	1,63
get Ordnerinhalt	25	362	359	368	389	478	342	478	0,00%	5,2/min	0,21	0,06
Gesamt	175	537	376	1225	1226	1229	68	1298	0,00%	36,0/min	1,01	1,81

### Messung Datei hochladen Ordner Prototyp – 100 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	658	634	799	838	1565	341	1965	0,00%	20,2/min	0,60	0,05
/dateiHochladenOrdner.h...	100	86	71	149	168	181	66	194	0,00%	20,2/min	0,76	0,06
post hochzuladene Datei	100	1294	1231	1318	1354	1628	1167	5798	0,00%	20,1/min	1,05	0,47
put AccessKey S3	100	615	564	858	937	953	449	964	0,00%	20,2/min	0,23	0,22
put der Datei in Ordner in...	100	111	108	135	149	186	74	221	0,00%	20,2/min	0,24	0,22
put DynamoDB Dateien ...	100	950	828	1063	1141	3101	762	5141	0,00%	20,1/min	0,24	6,34
get Ordnerinhalt	100	609	381	1305	1456	1756	341	2007	0,00%	20,2/min	0,83	0,22
Gesamt	700	617	560	1231	1293	1756	66	5798	0,00%	2,3/sec	3,89	7,50

## Messungen für das Hochladen eines Bildes in die Dropbox mit einer Ausführungszeit von 1 Sekunde

### Messung Bild hochladen Dropbox – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	1	1669	1669	1669	1669	1669	1669	1669	0,00%	35,9/min	24,97	0,25
/ajax_login	1	809	809	809	809	809	809	809	0,00%	1,2/sec	0,95	23,01
post token	1	452	452	452	452	452	452	452	0,00%	2,2/sec	1,66	10,28
Subscribe	1	41400	41400	41400	41400	41400	41400	41400	0,00%	1,4/min	0,01	0,01
Dropbox Home	1	1190	1190	1190	1190	1190	1190	1190	0,00%	50,4/min	73,15	0,24
post hochzul...	1	9490	9490	9490	9490	9490	9490	9490	0,00%	6,3/min	0,11	32,34
Gesamt	6	9168	1190	9490	41400	41400	452	41400	0,00%	6,5/min	2,39	6,02

### Messung Bild hochladen Dropbox – 25 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	25	2852	3037	3767	3791	4396	950	4396	0,00%	5,3/sec	222,95	2,90
/ajax_login	25	9608	9246	13399	15043	15532	2038	15532	0,00%	1,4/sec	1,06	25,62
post token	25	2433	1510	2758	10432	10542	446	10542	0,00%	1,5/sec	1,13	7,01
Subscribe	25	42667	43755	49396	55367	55567	31054	55567	0,00%	23,1/min	0,16	0,17
Dropbox Home	25	2989	1946	6193	8484	8779	1106	8779	0,00%	52,4/min	76,01	0,25
post hochzul...	25	104538	115905	127171	133170	146657	24778	146657	0,00%	9,2/min	0,43	32,06
Gesamt	150	27514	7636	103583	123629	133170	446	146657	0,00%	42,4/min	15,76	27,47

### Messung Bild hochladen Dropbox – 100 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	100	11135	9021	21002	21004	28558	2058	29142	0,00%	3,4/sec	121,37	1,52
/ajax_login	100	15721	16779	21002	21003	21004	2441	21005	0,00%	2,1/sec	3,05	22,40
post token	100	12001	11200	21001	21002	21002	1279	21003	0,00%	1,8/sec	2,18	6,28
Subscribe	100	40177	41646	54928	57027	60857	20999	65468	0,00%	1,0/sec	0,94	0,35
Dropbox Home	100	14554	13231	30961	31602	34029	1947	34041	0,00%	1,0/sec	51,94	0,20
post hochzul...	100	120641	115264	220064	229683	235984	21000	238505	0,00%	19,1/min	0,77	27,35
Gesamt	600	35705	21000	86747	169780	225448	1279	238505	0,00%	1,7/sec	25,92	28,53

## **Messungen für das Hochladen eines Bildes in die Dropbox mit einer Ausführungszeit von 300 Sekunden**

### Messung Bild hochladen Dropbox – 1 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	1	1774	1774	1774	1774	1774	1774	1774	0,00%	33,8/min	24,12	0,47
/ajax_login	1	697	697	697	697	697	697	697	0,00%	1,4/sec	1,11	26,71
post token	1	454	454	454	454	454	454	454	0,00%	2,2/sec	1,65	10,24
Subscribe	1	42166	42166	42166	42166	42166	42166	42166	0,00%	1,4/min	0,01	0,01
Dropbox Home	1	1181	1181	1181	1181	1181	1181	1181	0,00%	50,8/min	73,67	0,24
post hochzul...	1	8236	8236	8236	8236	8236	8236	8236	0,00%	7,3/min	0,13	37,27
Gesamt	6	9084	1181	8236	42166	42166	454	42166	0,00%	6,6/min	2,44	6,09

### Messung Bild hochladen Dropbox – 25 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	25	1637	1454	1932	2451	3249	1123	3249	0,00%	5,2/min	3,64	0,05
/ajax_login	25	884	825	1130	1266	1354	678	1354	0,00%	5,2/min	0,07	1,61
post token	25	496	452	584	696	972	333	972	0,00%	5,2/min	0,06	0,40
Subscribe	25	40790	39701	51194	52313	53096	30256	53096	0,00%	4,5/min	0,03	0,03
Dropbox Home	25	1534	1166	1611	3877	6046	952	6046	0,00%	4,9/min	7,14	0,02
post hochzul...	25	11414	10304	15372	17674	17735	7574	17735	0,00%	4,8/min	0,08	24,57
Gesamt	150	9459	1298	37279	46662	52313	333	53096	0,00%	25,8/min	9,46	23,73

### Messung Bild hochladen Dropbox – 100 User

Label	Anz. der Pro...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Dropbox	100	1892	1676	2610	3744	4639	1075	5096	0,00%	20,1/min	14,05	0,17
/ajax_login	100	1187	1069	1830	2047	2709	674	4626	0,00%	20,1/min	0,26	6,23
post token	100	743	581	1150	1861	3575	325	3711	0,00%	20,1/min	0,25	1,56
Subscribe	100	43106	43256	52864	54001	54493	30603	54808	0,00%	17,0/min	0,12	0,13
Dropbox Home	100	1907	1585	3034	3614	5165	1092	10954	0,00%	19,3/min	27,94	0,09
post hochzul...	100	206171	194839	338647	349594	388633	33192	401966	0,00%	11,8/min	0,51	23,53
Gesamt	600	42501	1752	173915	237528	349532	325	401966	0,00%	1,1/sec	24,02	25,95

## **Messungen für das Hochladen eines Bildes im Prototyp mit einer Ausführungszeit von 1 Sekunde**

**Messung Bild hochladen Prototyp – 1 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	415	415	415	415	415	415	415	0,00%	2,4/sec	4,29	0,37
/bildHochla...	1	77	77	77	77	77	77	77	0,00%	13,0/sec	26,68	2,07
post hochzu...	1	1227	1227	1227	1227	1227	1227	1227	0,00%	48,9/min	2,56	1,64
put Access...	1	690	690	690	690	690	690	690	0,00%	1,4/sec	1,00	0,95
put des Bild...	1	102	102	102	102	102	102	102	0,00%	9,8/sec	6,94	6,41
Gesamt	5	502	415	690	1227	1227	77	1227	0,00%	2,0/sec	3,34	0,65

**Messung Bild hochladen Prototyp – 25 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	3931	954	3840	31538	31815	831	31615	0,00%	46,0/min	1,37	0,12
/bildHochladen.html	25	730	279	898	957	10084	110	10084	0,00%	46,0/min	1,58	0,12
post hochzuladendes Bild	25	6756	6805	8853	9271	19951	1277	19951	0,00%	44,1/min	2,31	0,57
put AccessKey S3	25	6588	4124	21012	21017	21018	903	21018	0,00%	49,6/min	0,79	0,46
put des Bildes in S3	25	5607	4185	9902	13270	21016	1812	21016	0,00%	52,5/min	1,11	15,88
Gesamt	125	4722	2163	10027	21012	31538	110	31615	0,00%	3,3/sec	6,04	12,46

**Messung Bild hochladen Prototyp – 100 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	10364	9657	21018	33510	34014	914	34105	0,00%	2,9/sec	5,49	0,33
/bildHochladen.html	100	8757	4297	21026	21029	21037	196	21039	0,00%	2,4/sec	5,10	0,26
post hochzuladendes Bild	100	19836	21012	36722	59341	61722	4324	67359	0,00%	1,3/sec	3,67	1,54
put AccessKey S3	100	9515	9945	21021	21028	21031	350	21123	0,00%	1,4/sec	1,51	0,66
put des Bildes in S3	100	11339	9211	21021	21032	21223	1387	21638	0,00%	1,3/sec	1,85	20,04
Gesamt	500	11962	9903	21029	21039	59341	196	67359	0,00%	5,7/sec	10,74	18,51

## **Messungen für das Hochladen eines Bildes im Prototyp mit einer Ausführungszeit von 300 Sekunden**

**Messung Bild hochladen Prototyp – 1 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	797	797	797	797	797	797	797	0,00%	1,3/sec	2,24	0,19
/bildHochlade...	1	131	131	131	131	131	131	131	0,00%	7,6/sec	15,68	1,22
post hochzula...	1	1249	1249	1249	1249	1249	1249	1249	0,00%	48,0/min	2,52	0,78
put AccessKe...	1	738	738	738	738	738	738	738	0,00%	1,4/sec	0,93	0,89
put des Bilde...	1	2590	2590	2590	2590	2590	2590	2590	0,00%	23,2/min	0,47	7,29
Gesamt	5	1101	797	1249	2590	2590	131	2590	0,00%	54,5/min	1,62	3,60

**Messung Bild hochladen Prototyp – 25 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	701	704	779	882	1017	571	1017	0,00%	5,2/min	0,15	0,01
/bildHochlade...	25	153	137	198	261	262	95	262	0,00%	5,2/min	0,18	0,01
post hochzula...	25	1239	1227	1304	1305	1316	1135	1316	0,00%	5,2/min	0,27	0,78
put AccessKe...	25	851	902	1004	1052	1091	554	1091	0,00%	5,2/min	0,06	0,06
put des Bilde...	25	2026	1984	2292	2431	2564	1679	2564	0,00%	5,2/min	0,11	1,63
Gesamt	125	994	910	1977	2237	2431	95	2564	0,00%	25,6/min	0,76	1,69

**Messung Bild hochladen Prototyp – 100 User**

Label	Anz. der Proben	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	633	485	1004	1152	2231	347	4429	0,00%	20,2/min	0,60	0,05
/bildHochladen.html	100	142	111	179	262	851	71	1068	0,00%	20,2/min	0,69	0,05
post hochzuladendes Bild	100	1267	1243	1355	1472	1661	1170	1732	0,00%	20,1/min	1,05	0,28
put AccessKey S3	100	1060	1027	1149	1289	3555	475	4272	0,00%	20,2/min	0,23	0,22
put des Bildes in S3	100	1801	1780	2219	2370	2626	1340	2794	0,00%	20,1/min	0,41	6,33
Gesamt	500	981	1065	1798	1915	2594	71	4429	0,00%	1,7/sec	2,95	6,59

## Messungen für das Hochladen eines Bildes im Prototyp in einen Ordner mit einer Ausführungszeit von 1 Sekunde

### Messung Bild hochladen Ordner Prototyp – 1 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	382	382	382	382	382	382	382	0,00%	2,6/sec	4,67	0,40
/bildHochladenOrdner.ht...	1	82	82	82	82	82	82	82	0,00%	12,2/sec	27,49	2,02
post hochzuladenen Bild	1	1226	1226	1226	1226	1226	1226	1226	0,00%	48,9/min	2,56	0,39
put AccessKey S3	1	933	933	933	933	933	933	933	0,00%	1,1/sec	0,74	0,71
put des Bildes in Ordner ...	1	500	500	500	500	500	500	500	0,00%	2,0/sec	1,42	37,88
put DynamoDB Bilder sp...	1	367	367	367	367	367	367	367	0,00%	2,7/sec	5,35	1,77
get Ordnerinhalt	1	92	92	92	92	92	92	92	0,00%	10,9/sec	31,81	7,24
Gesamt	7	511	382	933	1226	1226	82	1226	0,00%	2,0/sec	3,76	5,93

### Messung Bild hochladen Ordner Prototyp – 25 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	5811	10359	10683	10815	10916	345	10916	0,00%	2,1/sec	3,81	0,33
/bildHochladenOrdner.ht...	25	2208	1347	9842	9856	9978	78	9978	0,00%	1,2/sec	2,70	0,20
post hochzuladenen Bild	25	9103	6188	18305	24901	25870	4252	25870	0,00%	39,4/min	2,07	2,02
put AccessKey S3	25	1448	659	3489	3495	3508	334	3508	0,00%	44,5/min	0,51	0,49
put des Bildes in Ordner ...	25	3456	2690	7396	12108	12317	477	12317	0,00%	47,4/min	0,56	14,97
put DynamoDB Bilder sp...	25	945	780	1443	1740	3659	340	3659	0,00%	50,8/min	1,66	0,55
get Ordnerinhalt	25	186	189	275	330	441	81	441	0,00%	52,9/min	2,58	0,59
Gesamt	175	3308	1073	10436	10815	19080	78	25870	0,00%	4,4/sec	8,47	13,36

### Messung Bild hochladen Ordner Prototyp – 100 User

Label	Anz. der Pr... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	11492	10591	21020	21035	21035	437	21036	0,00%	4,6/sec	8,86	0,49
/bildHochladenOrdner.ht...	100	9116	6675	21025	21025	21040	151	21045	0,00%	2,4/sec	5,35	0,28
post hochzuladenen Bild	100	23256	21014	53997	60703	69434	3240	69450	0,00%	1,2/sec	3,50	0,87
put AccessKey S3	100	8645	5073	21018	21029	21030	398	21030	0,00%	1,3/sec	1,36	0,71
put des Bildes in Ordner ...	100	9659	8514	21010	21024	21030	651	21033	0,00%	1,2/sec	1,05	19,44
put DynamoDB Bilder sp...	100	3024	1575	5656	11999	21021	345	21029	0,00%	1,2/sec	2,41	0,77
get Ordnerinhalt	100	2180	312	6099	10124	21024	83	21037	0,00%	1,3/sec	3,85	0,85
Gesamt	700	9625	6223	21025	21035	54221	83	69450	0,00%	7,2/sec	14,21	19,17

## **Messungen für das Hochladen eines Bildes im Prototyp in einen Ordner mit einer Ausführungszeit von 300 Sekunden**

### Messung Bild hochladen Ordner Prototyp – 1 User

Label	Anz. der Pr...	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	1	383	383	383	383	383	383	383	0,00%	2,6/sec	4,65	0,40
/bildHochladenOrdner.ht...	1	91	91	91	91	91	91	91	0,00%	11,0/sec	24,77	1,82
post hochzuladenen Bild	1	1225	1225	1225	1225	1225	1225	1225	0,00%	49,0/min	2,57	2,03
put AccessKey S3	1	764	764	764	764	764	764	764	0,00%	1,3/sec	0,90	0,87
put des Bildes in Ordner ...	1	478	478	478	478	478	478	478	0,00%	2,1/sec	1,48	39,62
put DynamoDB Bilder sp...	1	424	424	424	424	424	424	424	0,00%	2,4/sec	4,63	1,54
get Ordnerinhalt	1	81	81	81	81	81	81	81	0,00%	12,3/sec	36,13	8,22
Gesamt	7	492	424	764	1225	1225	81	1225	0,00%	2,0/sec	3,91	6,16

### Messung Bild hochladen Ordner Prototyp – 25 User

Label	Anz. der ... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	25	365	344	420	453	461	321	461	0,00%	5,2/min	0,15	0,01
/bildHochladenOrdner.h...	25	83	70	101	136	263	66	263	0,00%	5,2/min	0,20	0,01
post hochzuladenen Bild	25	1226	1226	1227	1256	1270	1206	1270	0,00%	5,2/min	0,27	0,22
put AccessKey S3	25	772	780	857	869	915	595	915	0,00%	5,2/min	0,06	0,06
put des Bildes in Ordn...	25	506	502	510	533	580	481	580	0,00%	5,2/min	0,06	1,64
put DynamoDB Bilder ...	25	397	380	416	443	655	351	655	0,00%	5,2/min	0,17	0,06
get Ordnerinhalt	25	84	82	85	90	137	79	137	0,00%	5,2/min	0,25	0,06
Gesamt	175	490	396	1225	1226	1238	66	1270	0,00%	36,0/min	1,16	1,82

### Messung Bild hochladen Ordner Prototyp – 100 User

Label	Anz. der ... ▾	Durchschnitt	Mittel	90% Line	95% Line	99% Line	Min	Max	% Fehler	Durchsatz	KB/sek	Sent KB/sec
Startseite	100	414	399	472	530	828	324	1029	0,00%	20,2/min	0,60	0,05
/bildHochladenOrdner.h...	100	74	71	83	89	116	67	117	0,00%	20,2/min	0,76	0,06
post hochzuladenen Bild	100	1220	1229	1249	1252	1148	1254	0,00%	20,1/min	1,05	1,64	
put AccessKey S3	100	781	866	938	958	975	501	1025	0,00%	20,2/min	0,23	0,22
put des Bildes in Ordn...	100	509	500	541	556	678	474	805	0,00%	20,2/min	0,24	6,37
put DynamoDB Bilder ...	100	417	393	461	507	854	341	1222	0,00%	20,2/min	0,66	0,22
get Ordnerinhalt	100	128	114	179	186	250	78	280	0,00%	20,2/min	0,99	0,22
Gesamt	700	506	424	1225	1237	1250	67	1254	0,00%	2,3/sec	4,49	7,07