Fakultät für Informatik und Mathematik 07

Bachelorarbeit

über das Thema

Sinnvolle Einsatzmöglichkeiten und Umsetzungsstrategien für serverless Webanwendungen

Meaningful Capabilities and Implementation Strategies for Serverless Web Applications

Autor: Thomas Großbeck
grossbec@hm.edu

Prüfer: Prof. Dr. Ulrike Hammerschall

Abgabedatum: 09.03.19

I Kurzfassung

Das Ziel der Arbeit ist es, Unterschiede in der Entwicklung von Serverless und klassischen Webanwendungen zu betrachten. Es soll ein Leitfaden entstehen, der Entwicklern und IT-Unternehmen die Entscheidung zwischen klassischen und Serverless Anwendungen erleichtert. Dazu wird zuerst eine Einführung in die Entwicklung des Cloud-Computing und insbesondere in das Themenfeld des Serverless Computing gegeben. Im nächsten Schritt werden zwei beispielhafte Anwendungen entwickelt. Zum einen eine klassische Webanwendung mit der Verwendung des Spring Frameworks im Backend und einem Javascript basiertem Frontend und zum anderen eine Serverless Webanwendung. Hierbei werden die Besonderheiten im Entwicklungsprozess von Serverless-Applikationen hervorgehoben. Abschließend werden die beiden Vorgehensweisen mittels vorher festgelegter Kriterien gegenübergestellt, sodass sinnvolle Einsatzmöglichkeiten für Serverless Anwendungen abgeleitet werden können.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Listing-Verzeichnis

VI Abkürzungsverzeichnis

FaaS Function as a Service

PaaS Platform as a Service

SaaS Software as a Service

BaaS Backend as a Service

IaaS Infrastructure as a Service

AWS Amazon Web Services

NIST National Institute of Standards and Technology

1 Einführung und Motivation

2 Grundlagen der Serverless-Architektur

2.1 Historische Entwicklung des Cloud-Computing

Die Evolution des Cloud-Computings begann in den sechziger Jahren. Es wurde das Konzept entwickelt Rechenleistung über das Internet anzubieten. John McCarthy beschrieb das Ganze im Jahr 1961 folgendermaßen. [Gar99, S. 1]

> *„If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as a telephone system is a public utility. [...] The computer utility could become the basis of a new and important industry.“*

McCarthy hatte also die Vision Computerkapazitäten als öffentliche Dienstleistung, wie beispielsweise das Telefon, anzubieten. Der Nutzer soll sich dabei nicht mehr selber um die Bereitstellung der Rechenleistung kümmern müssen, sondern die Ressourcen sind über das Internet verfügbar. Es wird je nach Nutzung verbrauchsorientiert abgerechnet.

Vor allen Dingen durch das Wachstum des Internets in den 1990er Jahren bekam die Entwicklung von Webtechnologien noch einmal einen Schub. Anfangs übernahmen traditionelle Rechenzentren das Hosting der Webseiten und Anwendungen. Hiermit einhergehend war allerdings eine limitierte Elastizität der Systeme. Skalierbarkeit konnte beispielsweise nur durch das Hinzufügen neuer Hardware erlangt werden. Neben der Hardware und dem Application Stack war der Entwickler außerdem für das Betriebssystem, die Daten, den Speicher und die Vernetzung seiner Applikation verantwortlich.

Durch das Voranschreiten der Cloud-Technologien konnten immer mehr Teile des Entwicklungsprozesses abstrahiert werden, sodass sich der Verantwortlichkeitsbereich des Entwicklers verschoben hat (siehe Abb. **??**). Im ersten Schritt werden hierzu häufig Infrastructure as a Service (IaaS) Plattformen verwendet. Diese wurden für eine breite Masse verfügbar, als die ersten Anbieter in den frühen 2000er Jahren damit anfingen Software und Infrastruktur für Kunden bereitzustellen. Amazon beispielsweise veröffentlichte seine eigene Infrastruktur, die darauf ausgelegt war die Anforderungen an Skalierbarkeit, Verfügbarkeit und Performance abzudecken, und machte sie so 2006 als Amazon Web Services (AWS) für seine Kunden verfügbar. [RPMP17]

Ein weiterer Schritt in der Abstrahierung konnte durch die Einführung von Platform as a Service (PaaS) vollzogen werden. PaaS sorgt dafür, dass der Entwickler sich nur noch um die Anwendung und die Daten kümmern muss. Damit einhergehend kann eine hohe Skalierbarkeit und Verfügbarkeit der Anwendung erreicht werden.

Auf der Virtualisierungsebene aufsetzend kamen schließlich noch Container hinzu. Diese sorgen beispielsweise für einen geringeren Ressourcenverbrauch und schnellere Bootzeiten. Bei PaaS werden Container zur Verwaltung und Orchestrierung der Anwendung verwendet. Es wird also auf die Kapselung einzelner wiederverwendbarer Funktionalitäten als Service geachtet. Dieses Schema erinnert stark an Microservices. Die genauere Abgrenzung zu Microservices wird im weiteren Verlauf der Arbeit behandelt. [Inc18, S. 6-7]

Als bisher letzter Schritt dieser Evolution entstand das Serverless Computing. Dabei werden zustandslose Funktionen in kurzlebigen Containern ausgeführt. Dies führt dazu, dass der Entwickler letztendlich nur noch für den Anwendungscode zuständig ist. Er unterteilt die Logik anhand des Function as a Service (FaaS) Paradigmas in kleine für sich selbstständige Funktionen. [Inc18, S. 7]

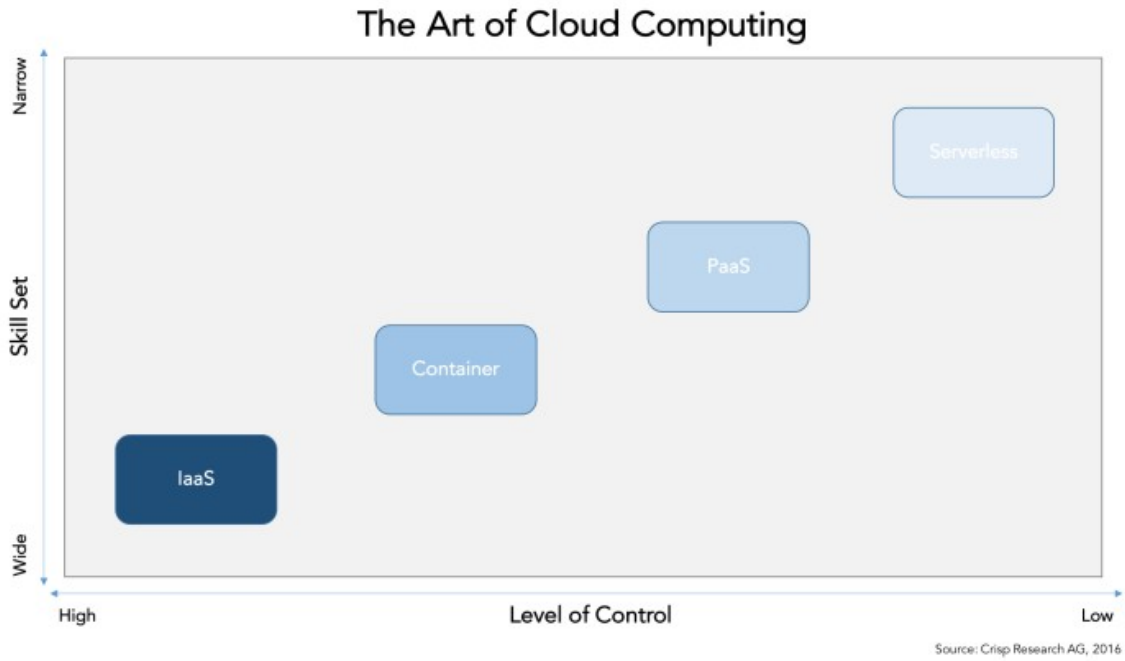

Abbildung 1: Zusammenhang Kenntnisstand und Kontroll-Level [Bü17]

2014 tat sich Amazon dann als Vorreiter für das Serverless Computing hervor und brachte AWS Lambda auf den Markt. Diese Plattform ermöglicht dem Nutzer Serverless Anwendungen zu betreiben. 2016 zogen Microsoft mit *Azure Function* und Google mit *Cloud Function* nach. [RPMP17]

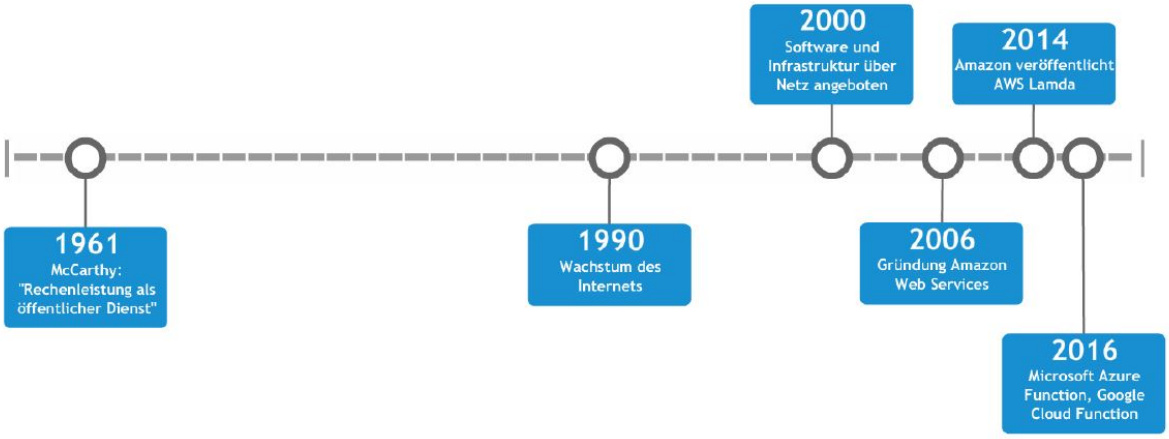

Abbildung 2: Historische Entwicklung des Cloud-Computings

2.1.1 Grundlagen des Cloud-Computing

„Run code, not Server [Rö17]“

Dies kann als eine der Leitlinien des Cloud-Computings angesehen werden. Cloud-Angebote sollen den Entwickler entlasten, sodass die Anwendungsentwicklung mehr in den Fokus gerückt wird. Das National Institute of Standards and Technology (NIST) definiert Cloud-Computing folgendermaßen. [MG11]

> *„Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“*

Der Anwender kann also über das Internet selbstständig Ressourcen anfordern, ohne dass beim Anbieter hierfür ein Mitarbeiter eingesetzt werden muss. Der Kunde hat dabei allerdings keinen Einfluss auf die Zuordnung der Kapazitäten. Freie Ressourcen werden auch nicht für einen bestimmten Kunden vorgehalten. Dadurch kann der Anbieter schnell auf einen geänderten Bedarf reagieren und für den Anwender scheint es, als ob er unbegrenzte Kapazitäten zur Verfügung hat.

Zur Verwendung dieses Angebots stehen dem Nutzer verschieden Out-of-the-Box Dienste in unterschiedlichen Abstufungen zur Verfügung (siehe Abb. 3). Dies wären zum einen das IaaS Modell, bei dem einzelne Infrastrukturkomponenten wie Speicher, Netzwerkleistungen und Hardware durch virtuelle Maschinen verwaltet werden. Skalierung kann so zum Beispiel einfach durch allokieren weiterer Ressourcen in der virtuellen Maschinen erreicht werden.

Zum anderen das PaaS Modell. Dabei wird dem Entwickler der Softwarestack bereitgestellt und ihm werden Aufgaben wie Monitoring, Skalierung, Load Balancing und Server Restarts abgenommen. Ein typisches Beispiel hierfür ist Heroku. Ein Webservice bei dem der Nutzer seine Anwendung deployen und konfigurieren kann.

Ebenfalls zu den Diensten gehört Backend as a Service (BaaS). Dieses Modell bietet modulare Services, die bereits eine standardisierte Geschäftslogik mitbringen, sodass lediglich anwendungsspezifische Logik vom Entwickler implementiert werden muss. Die einzelnen Services können dann zu einer komplexen Softwareanwendung zusammengefügt werden.

Die größte Abstraktion bietet SaaS. Hierbei wird dem Kunden eine konkrete Software zur Verfügung gestellt, sodass dieser nur noch als Anwender agiert. Beispiele dafür sind Dropbox und GitHub.

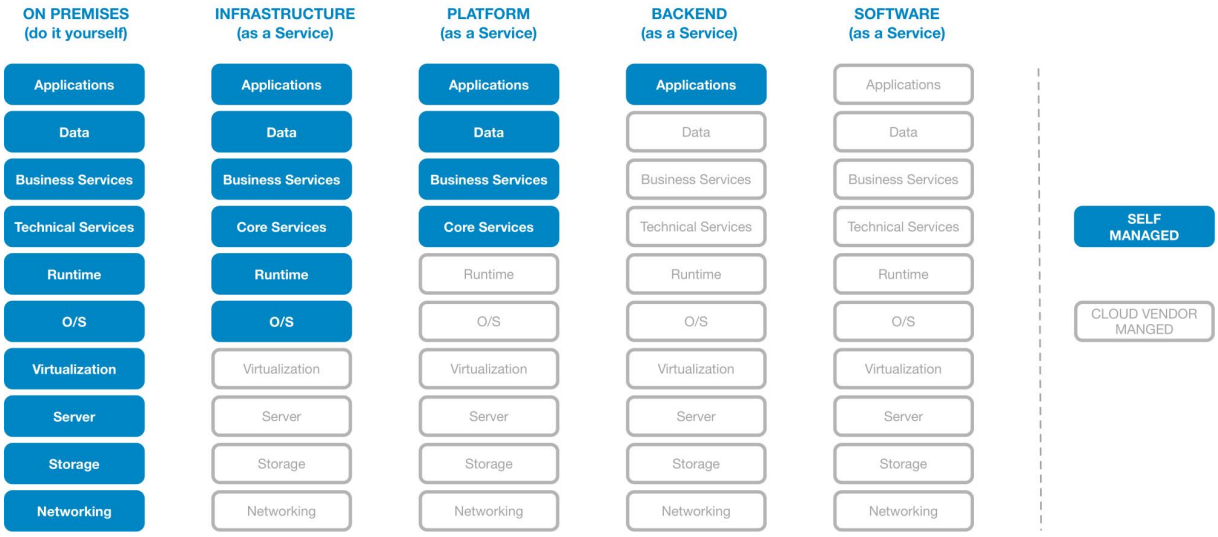

Abbildung 3: Verantwortlichkeiten der Organisation [Rö17]

Oftmals nutzen PaaS Anbieter ein IaaS Angebot und zahlen dafür. Nach dem gleichen Prinzip bauen SaaS Anbieter oft auf einem PaaS Angebot auf. So betreibt Heroku zum Beispiel seine Services auf Amazon Cloud Plattformen. [Her18] Ebenso ist es möglich eine Infrastruktur durch einen Mix der verschiedenen Modelle zusammenzustellen.

Letztendlich ist alles darauf ausgelegt, dass sich im Entwicklungs- und operationalen Aufwand so viel wie möglich einsparen lässt. Diese Weiterentwicklung wurde zum Beispiel in der Automobilindustrie bereits vollzogen. Dabei war es das Ziel die Fertigungstiefe, das heißt die Anzahl der eigenständig erbrachten Teilleistungen, zu reduzieren. [Dja02, S. 8] Nun findet diese Entwicklung auch Einzug in den Informatiksektor.

Ebenfalls von Bedeutung ist, dass die Anwendung automatisch skaliert und sich so an eine

wechselnde Beanspruchung anpassen kann. Außerdem werden hohe Initialkosten für eine entsprechende Serverlandschaft bei einem Entwicklungsprojekt für den Nutzer vermieden und auch die Betriebskosten können gesenkt werden. Dem liegt das Pay-per-use-Modell zugrunde. Der Kunde zahlt aufwandsbasiert. Das heißt, er zahlt nur für die verbrauchte Rechenzeit. Leerlaufzeiten werden nicht mit einberechnet. [Rö17]

Da Cloud-Dienste dem Entwickler viele Aufgaben abnehmen und erleichtern, sodass sich die Verantwortlichkeiten für den Entwickler verschieben, ist dieser nun beispielsweise nicht mehr für den Betrieb sowie die Bereitstellung der Serverinfrastruktur zuständig. Dies führt allerdings auch dazu, dass ein gewisses Maß an Kontrolle und Entscheidungsfreiheit verloren geht.

2.1.2 Abgrenzung zu PaaS

Prinzipiell klingen PaaS und Serverless Computing aufgrund des übereinstimmenden Abstrahierungsgrades sehr ähnlich. Der Entwickler muss sich nicht mehr direkt mit der Hardware auseinandersetzen. Dies übernimmt der Cloud-Service in Form einer Blackbox, sodass lediglich der Code hochgeladen werden muss.

Jedoch gibt es auch einige grundlegenden Unterschiede. So muss der Entwickler bei einer PaaS Anwendung durch Interaktion mit der API des Anbieters eigenständig für Skalierbarkeit und Ausfallsicherheit sorgen. Bei der Serverless Infrastruktur übernimmt das Kapazitätsmanagement der Cloud-Service (siehe Abb. 4). Es gibt zwar auch PaaS Plattformen, die bereits Funktionen für das Konfigurationsmanagement bereitstellen. Oft sind diese jedoch Anbieter-spezifisch, sodass der Programmierer auf weitere externe Tools zurückgreifen muss. [Bü17]

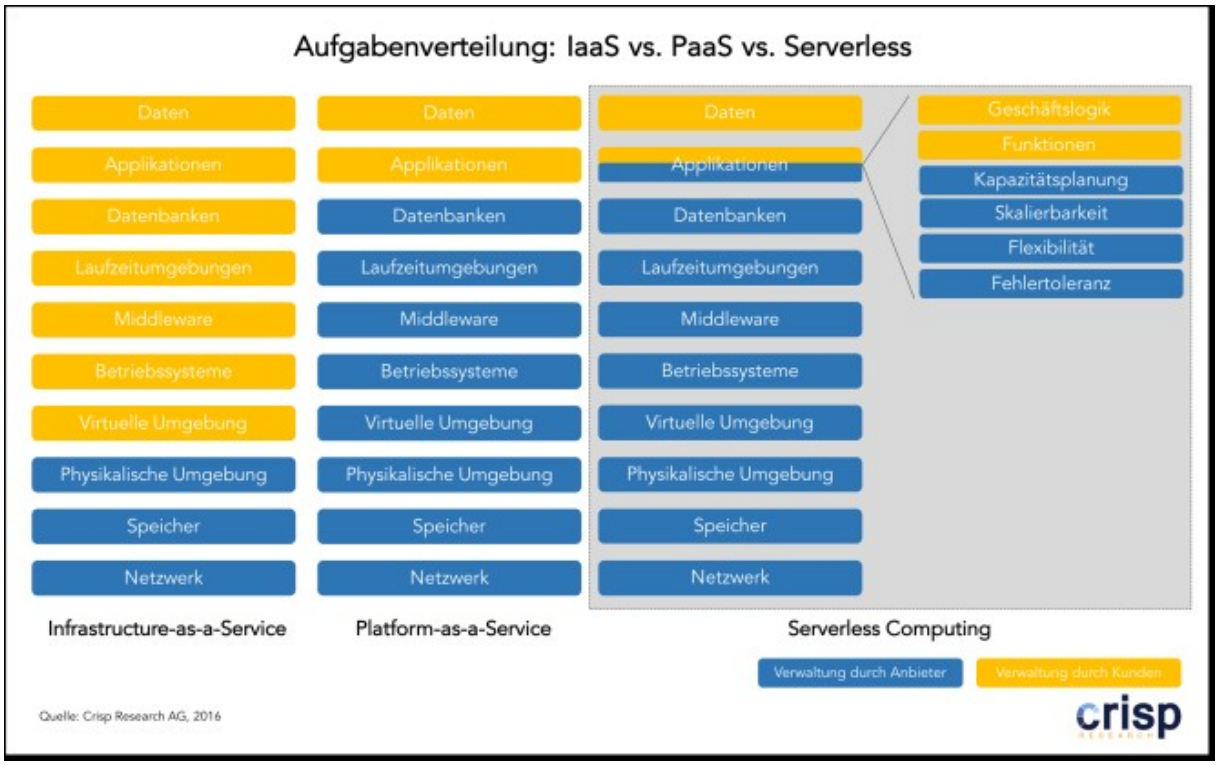

Abbildung 4: Aufgabenverteilung: IaaS vs. PaaS vs. Serverless [Bü17]

Ein weiterer Unterschied ist, dass PaaS für lange Laufzeiten konstruiert ist. Das heißt die PaaS Anwendung läuft immer. Bei Serverless hingegen wird die ganze Applikation als Reaktion auf ein Event gestartet und wieder beendet, sodass keine Ressourcen mehr verbraucht werden, wenn kein Request eintrifft. [Ash17]

Aktuell wird PaaS hauptsächlich wegen der sehr guten Toolunterstützung genutzt. Hier hat Serverless Computing den Nachteil, dass es durch den geringen Zeitraum seit der Entstehung noch nicht so ausgereift ist.

Final kann als einer der Schlüsselunterschiede, wie oben bereits erwähnt, die Skalierbarkeit festgehalten werden. Skalierbarkeit ist zwar auch bei PaaS Applikationen erreichbar, allerdings bei weitem nicht so hochwertig und kosteneffizient. Adrian Cockcroft von AWS bringt das folgendermaßen auf den Punkt. [Rob18]

> *„If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless.“*

2.1.3 Abgrenzung zu Microservices

Bei der Entwicklung einer Anwendung kann diese in verschieden große Komponenten aufgeteilt werden. Das genaue Vorgehen wird dazu im Voraus festgelegt. Entscheidet sich das Entwicklerteam für eine große Einheit, wird von einer Monolithischen Architektur

gesprochen. Hierbei wird die komplette Applikation als ein Paket ausgeliefert. Dies hat den Nachteil, dass bei einem Problem die ganze Anwendung ausgetauscht werden muss. Auch die Einführung neuer Funktionalitäten braucht eine lange Planungsphase. [Inc18, S. 9]

Auf der anderen Seite steht die Microservice Architektur. Die Anwendung wird in kleine Services, die für sich eigenständige Funktionalitäten abbilden, aufgeteilt. Teams können nun unabhängig voneinander an einzelnen Services arbeiten. Auch der Austausch oder die Erweiterung einzelner Module erfolgt wesentlich reibungsloser. Durch die Aufteilung in verschiedene Komponenten erreichen Microservice Anwendungen eine hohe Skalierbarkeit. [Bac18]

Das Konzept die Funktionalität in kleine Einheiten aufzuteilen, findet sich auch im Serverless Computing wieder. Im Gegensatz zur Microservices ist Serverless viel feingranularer. Bei Microservices wird oft das Domain-Driven Design herangezogen, um eine komplexe Domäne in sogenannte *Bounded Contexts* zu unterteilen. Diese Kontextgrenzen werden dann genutzt, damit die fachlichen Aspekte in verschiedene individuellen Services aufgeteilt werden können. [FL14] In diesem Zusammenhang wird auch oft von serviceorientierter Architektur gesprochen. Dahingegen stellt eine Serverless Funktion nicht einen kompletten Service dar, sondern eine einzelne Funktionalität. So eine Funktion kann beispielsweise gleichermaßen auch einen Event Handler darstellen. Daher handelt es sich hierbei um eine ereignisgesteuerte Architektur. [Tur18]

Ebenso ist es bei Serverless Anwendungen nicht notwendig die unterliegende Infrastruktur zur verwalten. Das heißt, dass lediglich die Geschäftslogik als Funktion implementiert werden muss. Weitere Komponenten wie beispielsweise ein Controller müssen nicht selbstständig entwickelt werden. Außerdem bietet der Cloud-Provider bereits eine automatische Skalierung als Reaktion auf sich ändernde Last an. Also auch hier werden dem Entwickler Aufgaben abgenommen. [Inc18, S. 9]

> *„The focus of application development changed from being infrastructure-centric to being code-centric. [Inc18, S. 10]“*

Im Vergleich zu Microservices rückt bei der Implementierung von Serverless Anwendungen die Funktionalität der Anwendung in den Fokus. Es muss keine Rücksicht auf die Infrastruktur, die bei Microservices einen großen Anteil einnimmt, genommen werden.

2.2 Eigenschaften von Function-as-a-Service

2.3 Allgemeine Patterns für Serverless-Umsetzungen

3 Entwicklung einer prototypischen Anwendung

3.1 Vorgehensweise beim Vergleich der beiden Anwendungen

Zum Vergleich der beiden Anwendungen werden einige Kriterien, die dabei helfen eine Aussage über die Qualität der jeweiligen Applikation zu treffen, abgearbeitet. Diese Kriterien werden nun im Folgenden genauer erläutert.

Implementierungsaufwand Es wird auf den zeitlichen Aufwand sowie auf die Codekomplexität geachtet. Das heißt, es wird untersucht, mit wie viel Einsatz einzelne Anwendungsfälle umgesetzt werden können und wie viel Overhead bei der Umsetzung möglicherweise entsteht.

Frameworkunterstützung Dabei wird analysiert inwieweit die Entwicklung durch Frameworks unterstützt werden kann. Dies gilt nicht nur für die Abbildung der Funktionalitäten, sondern auch für andere anfallende Aufgaben im Entwicklungsprozess wie zum Beispiel dem Testen und dem Deployment.

Deployment Beim Deploymentprozess sollen Änderungen an der Anwendung möglichst schnell zur produktiven Applikation hinzugefügt werden können, damit sie dem Kunden zeitnah zur Verfügung stehen. An dieser Stelle sind eine angemessene Toolunterstützung sowie die Komplexität der Prozesse ein großer Faktor. Optimal wäre in diesem Punkt eine automatische Softwareauslieferung.

Testbarkeit Hier ist zum einen ebenfalls der Implementierungsaufwand relevant und zum anderen sollte die Durchführung der Tests den Entwicklungsprozess nicht unverhältnismäßig lange aufhalten. Es ist dann auch eine effektive Einbindung der Tests in den Deploymentprozess gefragt. Im Speziellen werden mit den beiden Anwendungen Komponenten- und Integrationstests betrachtet.

Erweiterbarkeit Das Hinzufügen neuer Funktionalitäten oder Komponenten wird dabei im Besonderen überprüft. Damit einhergehend ist auch die Wiederverwendbarkeit einzelner Komponenten. Dies bedeutet, dass beleuchtet wird, ob einzelne Teile losgelöst vom restlichen System in anderen Projekten erneut einsetzbar sind.

Betriebskosten In einer theoretischen Betrachtung werden die Betriebskosten für die jeweiligen Anwendungen gegenübergestellt. So können anhand einer Hochrechnung für die Menge der benötigten Ressourcen die Kosten berechnet werden.

268 **Performance** Das Augenmerk liegt hierbei auf der Messung von Antwortzeiten einzelner
269 Requests sowie der Reaktion des Systems auf große Last.

270 **Sicherheit** An dieser Stelle ist zum Beispiel die Unterstützung zum Anlegen einer Nut-
271 zerverwaltung von Interesse.

6 Quellenverzeichnis

[Ash17] Ashwini, Amit: Everything You Need To Know About Serverless Architecture. (2017). https://medium.com/swlh/everything-you-need-to-know-about-serverless-architecture-5cdc97e48c09. – Zuletzt Abgerufen am 28.08.2018

[Bü17] Büst, René: Serverless Infrastructure erleichtert die Cloud-Nutzung. (2017). https://www.computerwoche.de/a/serverless-infrastructure-erleichtert-die-cloud-nutzung,3314756. – Zuletzt Abgerufen am 28.08.2018

[Bac18] Bachmann, Andreas: Wie Serverless Infrastructures mit Microservices zusammenspielen. (2018). https://blog.adacor.com/serverless-infrastructures-in-cloud_4606.html. – Zuletzt Abgerufen 09.11.2018

[Dja02] Djabarian, Ebrahim: *Die strategische Gestaltung der Fertigungstiefe*. Deutscher Universitätsverlag, 2002. – ISBN 9783824476602

[FL14] Fowler, Martin ; Lewis, James: Microservices. (2014). https://martinfowler.com/articles/microservices.html. – Zuletzt Abgerufen 19.11.2018

[Gar99] Garfinkel, Simson L.: *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. The MIT Press, 1999. – ISBN 9780262071963

[Her18] Heroku: Heroku Security. (2018). https://www.heroku.com/policy/security. – Zuletzt Abgerufen 08.11.2018

[Inc18] Inc., Serverless: Serverless Guide. (2018). https://github.com/serverless/guide. – Zuletzt Abgerufen am 06.09.2018

[MG11] Mell, Peter ; Grance, Tim: The NIST Definition of Cloud Computing. (2011). https://csrc.nist.gov/publications/detail/sp/800-145/final. – Zuletzt Abgerufen am 03.11.2018

[Rö17] Röwekamp, Lars: Serverless Computing, Teil 1: Theorie und Praxis. (2017). https://www.heise.de/developer/artikel/Serverless-Computing-Teil-1-Theorie-und-Praxis-3756877.html?seite=all. – Zuletzt Abgerufen am 30.08.2018

[Rob18] Roberts, Mike: Serverless Architectures. (2018). https://martinfowler.com/articles/serverless.html. – Zuletzt Abgerufen am 30.08.2018

[RPMP17] Rai, Gyanendra ; Pasricha, Prashant ; Malhotra, Rakesh ; Pandey, Santosh: Serverless Architecture: Evolution of a new paradigm. (2017). https://www.globallogic.com/gl_news/serverless-architecture-evolution-of-a-new-paradigm/. – Zuletzt Abgerufen am 30.08.2018

[Tur18] Turvin, Neil: Serverless vs. Microservices: What you need to know for cloud. (2018). https://www.computerweekly.com/blog/Ahead-in-the-Clouds/Serverless-vs-Microservices-What-you-need-to-know-for-cloud. – Zuletzt Abgerufen 15.11.2018