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 Webanwendungen 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 Unterstützung 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 gegenübergestellt, sodass Kriterien für den sinnvollen Einsatz von 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 und 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 Anwendungsstack 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 aus dem Verantwortlichkeitsbereich des Entwicklers abstrahiert werden. Im ersten Schritt werden hierzu häufig Infrastructure as a Service (IaaS) Plattformen verwendet. Diese bewirken, dass einzelne Infrastrukturkomponenten wie Speicher, Netzwerkleistungen und Hardware durch virtuelle Maschinen verwaltet werden. Skalierung kann so zum Beispiel einfach durch das Allokieren weiterer Ressourcen in der virtuellen Maschinen erreicht werden. 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 Kap-

selung 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]

In den frühen 2000er Jahren fingen die ersten Anbieter damit an 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. 2014 tat sich Amazon dann als Vorreiter für die neueste Evolutionsstufe, dem 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]

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 entlassen, sodass die Anwendungsentwicklung mehr in den Fokus gerückt wird. Das National Institute of Standards and Technology (NIST) definiert Cloud-Computing folgendermaßen.

> *„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.“*

[MG11]

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 und 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. 1). Dies wären zum einen das IaaS Modell, bei dem die Hardware durch virtuelle Einheiten abstrahiert wird und 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.

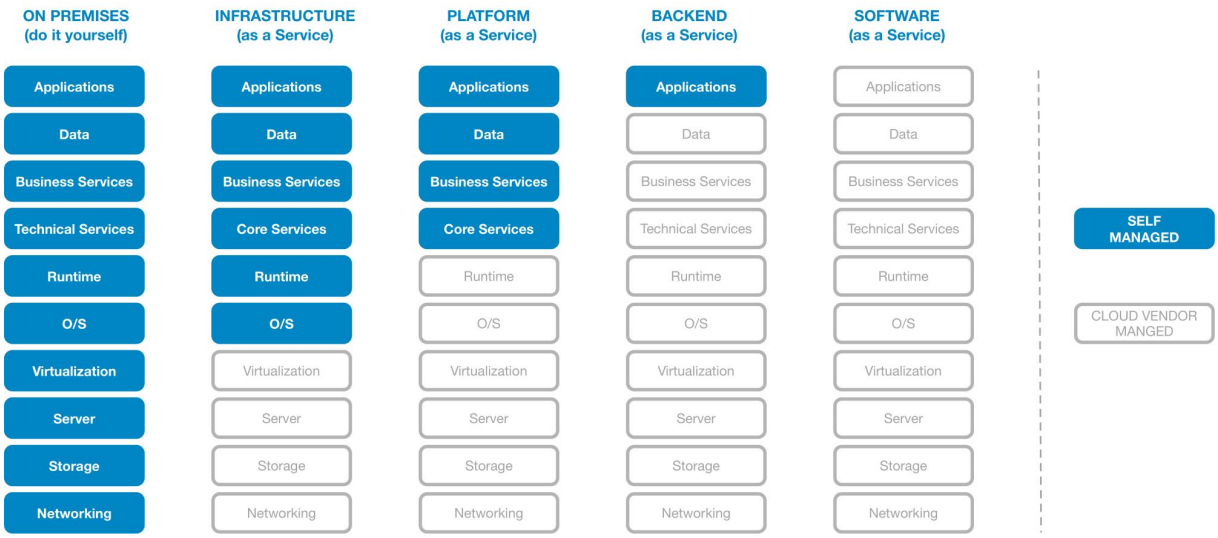

Abbildung 1: Verantwortlichkeiten des Entwicklers [Rö17]

Oftmals nutzen PaaS Anbieter ein IaaS Angebot und zahlen dafür. Nach dem gleichen Prinzip bauen SaaS Anbieter oft auf einem PaaS Angebot auf. Letztendlich ist alles darauf ausgelegt, dass sich im Entwicklungs- und operationalen Aufwand so viel wie möglich einsparen lässt. Ebenfalls von Bedeutung ist, dass die Anwendung automatisch skaliert und sich so an eine wechselnde Beanspruchung anpassen kann. Außerdem sollen hohe Initialkosten für eine entsprechende Serverlandschaft bei einem Entwicklungsprojekt für den Nutzer vermieden werden und auch die Betriebskosten sollen 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, hat dieser nun nicht mehr die volle Verantwortung über den kompletten Entwicklungsprozess. Dies führt allerdings auch dazu, dass ein gewisses Maß an Kontrolle verloren geht.

2.1.2 Abgrenzung zu PaaS

2.1.3 Abgrenzung zu Microservices

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.

Einen großen Teil der Gegenüberstellung nimmt der Entwicklungsprozess ein. Hierbei wird der Implementierungsaufwand betrachtet. 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. Von großer Wichtigkeit ist auch der Deploymentprozess. Dabei 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. Ein weiteres Kriterium ist die Testbarkeit. Zum einen ist ebenfalls wieder der Implementierungsaufwand relevant und zum anderen sollte die Durchführung der Tests den Entwicklungsprozess nicht unverhältnismäßig lange aufhalten. Hier ist dann auch eine effektive Einbindung der Tests in den Deploymentprozess gefragt. Im Speziellen werden mit den beiden Anwendungen Komponenten- und Integrationstests betrachtet. Außerdem wird untersucht inwieweit die Umsetzung 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.

Auf der anderen Seite der Gegenüberstellung wird beispielsweise die Performance der Anwendungen analysiert. Das Augenmerk liegt hierbei auf der Messung von Antwortzeiten einzelner Requests sowie der Reaktion des Systems auf große Last. Ebenfalls im Fokus liegt die Erweiterbarkeit der Applikation. 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 eingesetzt werden könnten. Des Weiteren können in einer theoretischen Betrachtung die Betriebskosten für die jeweiligen Anwendungen gegenübergestellt werden. So können anhand einer Hochrechnung für die Menge der benötigten Ressourcen die Kosten berechnet werden. Zuletzt wird noch die Sicherheit der beiden Anwendungen analysiert. An dieser Stelle ist zum Beispiel die Unterstützung zum Anlegen einer Nutzerverwaltung von Interesse.

6 Quellenverzeichnis

[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

[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

[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