Dokumentation der betrieblichen Projektarbeit

Migration eines über einen Dienstleister gehosteten Microservice in die Amazon Web Services-Umgebung

Ausbildungsberuf

Fachinformatiker Fachrichtung Systemintegration

Prüfungsbewerberin

Antje Radkiewicz

Loher Straße 15b

22149 Hamburg

Ausbildungsbetrieb

OTTO (GmbH & Co. KG)

Werner-Otto-Straße 1-7

22179 Hamburg

14. Mai 2018

Vorwort

Diese Dokumentation richtet sich an Personen die in der Fachrichtung Informatik Vorwissen besitzen. In dieser Dokumentation werden fachbezogene Begriffe verwendet, die im Glossar näher erläutert werden. Fachbegriffe, die in dem Glossar enthalten sind, werden unterstrichen dargestellt.

Im Textverlauf werden Abkürzungen verwendet. Die ausgeschriebene Variante wird einmalig genutzt und hinter dieser wird in Klammern die Abkürzung dargestellt. Im Abkürzungsverzeichnis können die Abkürzungen und ihre ausgeschriebene Variante eingesehen werden.

Des Weiteren werden kurze Beispiele genutzt bei denen ein Attribut gesetzt werden muss. Hier wird der Name des Attributs mit eckigen Klammern umschlossen, um eine solche Stelle zu kennzeichnen. Beispiel: www.otto.de/p/[Name des Attributs]

```
Code-Beispiele werden in kleinen umrahmten Boxen dargestellt.
```

Der Name von projektbeiteiligten Person kann in *Tabelle d.* eingesehen werden. Zum Schutz der Personen wurden die Nachnamen gekürzt.

Inhaltsverzeichnis

Dokumentation der betrieblichen Projektarbeit

Migration eines über einen Dienstleister gehosteten Microservice in die Amazon Web Services-Umgebung

Einleitung

Dieses Dokument ist im Rahmen der betrieblichen Projektarbeit für den Ausbildungsgang des Fachinformatikers in der Fachrichtung Systemintegration entstanden. Das Projekt wurde im genehmigten Projektzeitraum vom 27. März 2018 bis zum 14. Mai 2018 durchgeführt. Weiterhin wurde im Rahmen des Durchführungszeitraumes diese Dokumentation erstellt.

1. Projektbeschreibung

Im Rahmen der praktischen Abschlussarbeit wurde die Migration eines Teil-Microservices in die Amazon Web Services (AWS) realisiert. Der Microservice bestand aus vier Kernfunktionalitäten. Hieraus wurde ein Teil-Service ausgewählt, freigeschnitten und lauffähig in die AWS überführt. Diese Migrationsaufgabe wurde im Rahmen eines teamübergreifenden strategischen Entscheides durchgeführt und einige vordefinierte Rahmenbedingungen mussten eingehalten werden.

2. Projektumfeld

2.1. OTTO (GmbH & Co. KG)

Die OTTO (GmbH & Co. KG) ist ein Versandhandelsunternehmen mit Sitz in Hamburg, welches zur Otto Group gehört. Diese agiert mit über 49.000 Mitarbeitern weltweit. Seit 1995 betreibt das Unternehmen die E-Commerce-Plattform www.otto.de.

2.2. E-Commerce

Der Bereich E-Commerce der OTTO Einzelhandelsgesellschaft betreibt und entwickelt die Seite www.otto.de. Die Plattform ist in mehrere Vertikalen aufgeteilt. Für jede Vertikale ist ein

Entwicklerteam zuständig, welches die Funktionalität der entsprechenden Shop-Segmente verantwortet. Die Teams selber sind interdisziplinär aufgestellt, mit dem Ziel eine möglichst autarke und zielführende Arbeitsweise zu ermöglichen.

2.3. Externer Dienstleister

Der Bereich E-Commerce wurde bisher durch einen externen Dienstleister bei der Bereitstellung von www.otto.de unterstützt. Als Lösungspartner für IT-Infrastruktur stellt dieser dem OTTO E-Commerce derzeit nicht nur die notwendige Hardware in einem Rechenzentrum, diverse Applikationen und Schnittstellen, sondern auch personelle Ressourcen bereit.

2.4. Team Social

Team Social stellt mehrere Produkte auf der Shop-Seite bereit, die im Reiter Inspiration auf www.otto.de zu finden sind. Darunter sind beispielsweise der Feed (www.otto.de/feed), über den man von Anwendern zusammengestellte Inhalte abonnieren und betrachten kann, Tags (www.otto.de/tags/[TAGNAME]), welches derzeit die im Feed genutzten Tags aggregiert und eine Liste der relevantesten Ergebnisse darstellt und der StyleCreator (www.otto.de/stylecreator). Das Team betreut mehrere Microservices, die diese Produkte bereitstellen.

Alle Services liegen auf Systemen, die von dem externen Dienstleister gehosted werden. Durch eine abgesprochene Vorgehensweise werden Produkte, die im Hause OTTO für die Webseite www.otto.de entwickelt werden, auf die Maschinen des Dienstleisters übertragen und lauffähig gemacht. Der Dienstleister stellt zusätzlich diverse Infrastrukturressourcen für die gesamte Plattform zur Verfügung.

2.4.1. StyleCreator

Der StyleCreator erlaubt es Artikelbilder von www.otto.de auszuwählen, sie zu einer Collage zusammenzustellen und diese anschließend zu veröffentlichen. Der Bearbeiter kann zwischen diversen Design-Vorlagen wählen oder im freien Design die Collage erstellen. Die Freiform-Vorlage bietet dem Ersteller (anders als bei den Design-Vorlagen) die Möglichkeit, Einzelbilder zu rotieren, zu vergrößern, zu verkleinern und diese freizustellen. Des Weiteren können die Einzelbilder frei auf der Collage versetzt und übereinandergelegt werden.

Abbildung a.: StyleCreator im Freiform-Modus mit zwei sich überlappten Artikelbildern

3. Projektbegründung

3.1. Strategische Entscheidung

Im Rahmen einer strategischen Entscheidung wurde entschieden, dass die Plattform www.otto.de nicht weiter über den externen Dienstleister gehostet, sondern über die AWS bereitgestellt werden wird. Durch die autarke Aufstellung der Teams kann jedes Team zu großen Teilen selber entscheiden, wie die Migration ausgeführt wird und welche Entitäten priorisiert werden.

3.2. Zeitlicher Rahmen

Aufgrund der Entscheidung, den Hoster für die Plattform zu wechseln, wurde der bestehende Hosting-Vertrag nicht verlängert. Bevor dieser Vertrag ausläuft müssen alle Teams, einschließlich der operativen, die Lauffähigkeit aller ihrer Kernservices in der AWS-Umgebung sichergestellt haben. Die Migration verfolgt nicht primär das Ziel eine Kostenersparnis zu erlangen. In der Migrationsphase sollen zunächst die Kernfunktionalitäten in der neuen Umgebung lauffähig gemacht werden, ohne technische Schulden aufzunehmen.

4. IST-Analyse

4.1. Team K2

Um den Migrationsprozess für alle Teams erfüllbar zu machen wurde das Team K2 gegründet. In der Vorbereitungsphase auf das Projekt wurden hier erste Erfahrungen gesammelt, die

Möglichkeiten der technischen Umsetzung und sicherheitsrelevante Aspekte überprüft. Das Team hat wichtige Infrastrukturressourcen erzeugt und stellt diese allen anderen Teams zur Verfügung. Das gesammelte Wissen dieses Teams wird im Paten-Prinzip auf die restlichen Teams verteilt. Bei technischen Problemen stehen die Kollegen von K2 als Ansprechpartner bereit.

4.1.1. Accountmanagement in der AWS-Umgebung

Unter dem AWS Root-Account von OTTO (GmbH & Co. KG) wurden, in Absprache mit K2, weitere AWS Accounts angelegt. Jedes Team erhielt jeweils zwei dieser Accounts. Diese werden in Zukunft jeweils für die Live- und die Nonlive-Umgebung genutzt. Ressourcen, die in einem Account angelegt werden, können nicht automatisch auf Ressourcen der anderen Accounts zugreifen.

4.1.2. Rollen- und Rechtemanagement in der AWS-Umgebung

Das K2 Team hat in der AWS bereits ein Rechte- und Rollenkonzept aufgebaut, welches es allen Teams erlaubt in der AWS generell unabhängig voneinander zu arbeiten. Jeder Benutzer benötigt einen ihm zugewiesenen Benutzeraccount. Dieser wird zusammen mit den Kollegen von K2 im AWS Identity and Access Management (IAM) angelegt.

Ein Benutzeraccount kann verschiedene Rollen annehmen. Es gibt vorgefertigte Rollen, wie zum Beispiel die Manager-, die Read-Only- oder die Developer-Rolle. Rollen werden von Benutzeraccounts explizit für einen Zeitraum angenommen. Mit dem Annehmen der Managerrolle in einem Account können Rechte auf Rollen innerhalb dieses Accounts gemanaged werden. Jedes Team erhält für eine oder mehrere Personen die Berechtigung die Managerrolle anzunehmen. Die Benutzer mit dem Recht, diese Rolle anzunehmen, verteilen für ihr Team die nötigen Rechte im Selbstmanagement.

4.2. Abgeschlossene Migrationsaufgaben von Team Social

4.2.1. Account-, Rollen- und Rechtemanagement

Das K2 Team hat, zusammen mit den Teammitgliedern von Social, für alle Benutzer einen persönlichen Benutzeraccount in der AWS angelegt und sichergestellt, dass die Zwei-Faktor-Authentifizierung eingerichtet ist. Die Managerrolle für die Accounts von Team Social, wurde durch K2 an den technischen Entwickler vergeben. Alle Entwickler aus dem Team wurden für die Read-Only- und Developer-Rolle freigeschaltet.

4.2.2. Region

Die AWS-Umgebung besteht weltweit aus Rechenzentren, die sich in verschiedenen Regionen befinden. Jede Region ist ein separater geografischer Bereich. Dieser verfügt über mehrere isolierte Standorte, die als Availability Zones oder Verfügbarkeitszonen bezeichnet werden. Da die Plattform www.otto.de in Deutschland genutzt wird, hat sich das Team Social für die Region eu-central-1 entschieden. Diese Region wird in Frankfurt gehostet und besteht aus drei Verfügbarkeitszonen, wie in *Abbildung c.* dargestellt ist. Diese Zonen sind physisch voneinander getrennt, aber über Verbindungen mit geringer Latenz verbunden.

4.2.3. Tools

Die Arbeiten für die Implementierung der grundsätzlichen Infrastruktur in der AWS-Umgebung wurden im August 2017 begonnen. Das Team hat sich entschieden AWS CodeCommit zur verteilten Versionsverwaltung und die AWS CodePipeline als Pipeline-Tool zu nutzen. Alle Ressourcen sollen einheitlich via CloudFormation-Templates erzeugt werden. Außerdem hat sich das Team in Absprache mit K2 für das CloudFormation-YAML-Format entschieden.

4.2.4. Infrastruktur

Um die Infrastruktur zu generieren wurde ein Bootstrap Repository angelegt. Hierüber werden alle grundsätzlichen Infrastrukturressourcen erzeugt, die für alle Services des Teams benötigt werden. Via CloudFormation wurde ein Template erstellt, welches pro Account eine Virtual Private Cloud (VPC) erzeugt. Auf jede VPC wurde für jede Verfügbarkeitszone ein öffentliches und ein privates Subnetz erzeugt, welches jeweils über ein NAT-Gateway verbunden ist. Die Nonlive-Umgebung wird immer mit denselben Templates erstellt wie die Live-Umgebung und wird sich daher technisch nicht von dieser unterscheiden.

4.2.5. Services

Vor dem Start dieser Projektarbeit hatte das Team zwei Microservices in die AWS-Umgebung migriert. Die bereits migrierten Microservices werden mit CloudFormation-Templates erzeugt und werden in Elastic Compute Cloud-Containern (EC2) ausgeführt.

4.3. Bevorstehende Migrationsausgaben von Team Social

4.3.1. Imageprocessing

Die Reihenfolge der zu migrierenden Microservices wurde von Team Social vor der Migration festgelegt. Alle Microservice wurden anhand ihrer Schnittstellen und Komplexität in eine sinnvolle Reihenfolge gesetzt. Der Microservice, der als nächstes migriert werden sollte, war

Imageprocessing.

Der Imageprocessing Microservice bestand aus vier Kernfunktionalitäten. Für die Mirgation innerhalb dieser Projektarbeit sollte nur ein Teil-Service aus dem Service gelöst und in die AWS migriert werden. Der Teil-Service mit der geringsten Komplexität und den wenigsten Abhängigkeiten war setFree. SetFree wurde von anderen Microservices aufgerufen, um im StyleCreator Bilder freizustellen.

4.3.2. SetFree-Teil-Service

Der setFree-Teil-Service wurde über einen speziellen Endpunkt einer REST-Schnittstelle aufgerufen. Die POST-Methode des Endpunktes erwartete die Parameter „ImageId“, „width“ und „height“, wobei die ImageId als Pfadparameter und die Breite und Höhe als Queryparameter übergeben wurden. Die angegebene ImageId muss einer eindeutigen ID auf ein Bild auf dem Otto-Image-Server entsprechen. Die Breite und die Höhe sind frei wählbar.

Der Service nutzt das von Node.js bereitgestellte Package namens gm (eine Node.js Implementierung von GraphicsMagick). Mit diesem können Funktionen von GraphicsMagick und ImageMagick ausgeführt werden.

Das Bild, welches anhand der ImageId identifizierbar ist, wird per GraphicsMagick mit den Werten: setFormat('png') borderColor('white') border(1, 1) fuzz(5, true) fill('rgba(0, 0, 0, 0)') setDraw('color', '0', '0', 'floodfill') quality(100) freigeschnitten und nach der Freistellung direkt im Browser sichtbar gemacht.

Abbildung b.: StyleCreator im Freiform-Modus mit zwei sich überlappten Artikelbildern, freigestellt

5. Projektziel

5.1. Migrationsziel Imageprocessing

Im Rahmen dieser Projektarbeit sollte nur ein Teil des Imageprocessing-Services freigeschnitten und migriert werden. Nach dieser ersten Teilmigration sollte das Vorgehen bewertet werden. Zusätzlich sollte überprüft werden ob der Teil-Service wie gewünscht in die AWS integriert wurde. Bei einer erfolgreichen Migration würden im Anschluss an das Projekt die anderen Teile des Services auf die selbe Weise in die AWS migriert werden.

6. Projektdurchführung

6.1. Umsetzungsstrategien

Um den setFree-Teil-Service aus der alten Hosting-Infrastruktur des Dienstleisters zu trennen und in die AWS zu integrieren gab es zwei Lösungsansätze.

Der Teil-Servie hätte, wie bei Tags, in einem EC2-Container lauffähig gemacht werden können. Es hätten keine Anpassungen am Service selber vorgenommen werden müssen. Mittels einer Amazon EC2 Auto Scaling Group hätten die Instanzen je nach Last hoch- und heruntergefahren werden gekonnt.

Als Alternative hätte der Teil-Service als AWS Lambda-Funktion implementiert werden gekonnt. Hierbei würde ein Service nicht in einem EC2-Container sonder ohne jeglichen Overhead ausführbar gemacht werden. Es gäbe viele Lambda-Trigger über die der Service später gestartet werden könnte. Nachdem eine Lambda-Funktion getriggered werden würde, würde die Instanz beendet werden und keine weiteren Kosten erzeugen.

6.2. Entscheidungskriterien

6.2.1. Kosten

Der setFree-Teil-Service wurde im Schnitt einhundert Mal am Tag aufgerufen und hatte eine durchschnittliche Ausführungsdauer von 2 Sekunden.

Eine EC2 t2.nano Instanz ist die kleinste EC2 Option, sie stellt einen vCpu und 0,5 GiB RAM zur Verfügung. Sie kostet 0,0058 US-Dollar pro Stunde. Eine über EC2 bereitgestellte Instanz läuft ununterbrochen. Es ist generell möglich, via Amazon Auto Scaling Group, eine Instanz erst zu starten, wenn der Service benötigt wird. Die Erstellungszeit ist aber vergleichsweise hoch. (Tägliche Kosten pro Umgebung: 0,0058 $ * 24)

Lambda-Funktionen werden nach der Häufigkeit der Anforderung gezahlt, Testaufrufe der Funktion werden mitberechnet. Der einmalige Aufruf einer Lambda-Funktion kostet 0,0000002 US-Dollar. Zusätzlich werden auf Basis der Ausführungsdauer Kosten erzeugt. Je nachdem wieviel Arbeitsspeicher einer Funktion zugewiesen wird kostet zum Beispiel die kleinste Arbeitsspeichervariante alle 100 ms 0,000000208 US-Dollar. (Tägliche Kosten pro Umgebung: 0,0000002 + 0,0000208 $ * 2 Sekunden * 100 Ausführungen am Tag)

6.2.2. Aufwand

Das team hatte einen erhöhten Erfahrungsschatz, wie ein Service als EC2-Container in die AWS zu deployen wäre. Um den Container bereit zu stellen würden viele Arbeitsstunden in die Ressourcenerzeugung und deren Design erzeugt werden. Die lokalen Tests hätten wie gewohnt ausführbar gemacht werden gekonnt. Die Tests in der AWS hätten angepasst werden gemusst.

Bei der Umsetzung als Lambda-Funktion hätte Testumgebung erstellt werden müssen, die sowohl in MAC-Umgebungen als auch unter Linux funktionstüchtig gewesen wären. Eine Lambda-Funktion zu erstellen war laut Dokumentation mit wenig Aufwand behaftet.

6.2.3. Performance

Bei der Ausführung von setFree wird ein ImageMagick Befehl ausgeführt. Bei einer Arbeitsspeicherkapazität von einem GB dauerte die Umwandlung der Bilder etwa 2 Sekunden. Die kleinste Instanz bei dem externen Dienstleister war eine virtuelle Maschine mit einem GB RAM. Es war noch nicht absehbar in welcher Weise sich die Durchführungsdauer verändern wird, wenn dem Teil-Servie weniger als diese RAM-Kapazität zur Verfügung gestellt wird. Der kleinste EC2-Container würde vierfach (0,5 GiB) so viel Kapazität besitzten wie die kleinste Lambda-Ausführungsvariante (0,12 GiB).

6.2.4. Nutzwertanalyse

Auswahlkriterien	
Kosten	45 %
Aufwand	30 %
Performance	25 %
Summe	100 %

Tabelle a.: Gewichtungsanalyse Auswahlkriterien

Punktewertung					
1	2	3	4	5	6
sehr gut	gut	befriedigend	ausreichend	mangelhaft	ungenügend

Tabelle b.: Wertungsskala

	Gewichtung	EC2	Wert	Nutzwert	Lambda	Wert	Nutzwert
Kosten	45 %	0,14 $/Tag	2	90	0,004 $/Tag	1	45
Aufwand	30 %	hoch	4	120	gering	2	60
Performance	25 %	0,5 GiB	2	50	0,12 GiB	4	100
Summe	100 %		8	260		7	205

Tabelle c.: Nutzwertanalyse

6.3. Umsetzungsstrategieentscheidung

Aufrgund des Ergebnisses der Nutzwertanalyse wurde die Migration von setFree via AWS Lambda durchgeführt.

6.4. Arbeitsschritte

6.4.1. Repository

Um ein neues Repository in der AWS anzulegen wurde ein Cloudformation-Template erstellt. Dieses Template wurde einmalig mit Autostacker24 in die AWS deployed. Im Anschluss wurde das leere Repository auf eine lokale Festplatte geklont. Alle Dateien, die zur Funktionalität von setFree nötig sind, wurden aus dem derzeitigem Imageprocessing Repository herausgelöst und in das Neue übertragen. Des Weiteren wurde die README.md angepasst und die neuen Dateien in das AWS Repository hochgeladen. (repository.yaml siehe *Abbildung d.*)

6.4.2. AWS Lambda-Funktion Handler

Der setFree-Teil-Service wurde in der Vergangenheit über das Express.js Framework ausgeführt. Die Quelldateien des setFree-Teil-Services mussten so angepasst werden, dass der Service als Lambda-Funktion aufgerufen werden konnte. In der Dokumentation von AWS ist beschrieben, dass ein Node.js Projekt über ein sogenanntes Handler-Objekt aufrufbar gemacht wird. Zusätzlich muss in dem Projekt eine buildspec Datei angelegt werden, in die verschiedenen Ausführungsphasen der Funktion definiert sind.

```
exports.myHandler = function(event, context) {}
```

Hierfür wurde die setFreeService.js Datei angepasst und in das Root-Verzeichnis des Repositories verschoben. Der setFreeRouter wurde durch die Umstellung oboslet und gelöscht. Die Funktionalitäten das setFreeController wurde in den Service eingefügt. (Auszug aus der neuen imagprocessing.js siehe *Abbildung e.*, buildspec.yaml siehe *Abbildung f.*)

6.4.3. Pipeline

Durch das Anlegen eines neuen CloudFormation-Templates für die CodePipeline wurde der Deployment-Prozess automatisiert. Ein CloudFormation-Pipeline-Template besteht aus Pipeline-Schritten (Stages), die jeweils einen Namen und verschiedene Aktionen besitzen. Für diesen Teil-Service wurden fünf Schritte erstellt, checkout, common, build, deploy-nonlive und deploy-live. Die Schritte werden nacheinander ausgeführt und werden den Service erst in non-live und bei erfolgreicher Durchführung anschließend automatisch in die live-Umgebung deployen. Dieses Template wurde im Anschluss auch via Autostacker24 manuell in die AWS hochgeladen.

```
autostacker24 update --template cloudformation/ci/pipeline.yaml --stack imageprocessing-pipeline --profile [Profilname] --region eu-central-1
```

6.4.4. API-Gateway

Der set-Free-Teil-Service sollte, wie gehabt, über einen Endpunkt erreichbar gemacht werden. Die http-Schnittstelle hatte sich bewährt. In der AWS können über CloudFormation-Templates API-Gateways erstellt werden. Hier musste eine Struktur definiert werden. Nach der Erstellung des APIs wurde die Schnittstelle /setFree erzeugt. Diese erwartete weiterhin die Parameter ImageId, height und width. Allerdings wurden diese alle als Queryparameter implementiert. (Auszug aus dem api.yaml siehe *Abbildung g.*)

6.4.5. Binary Media Types

Nach der Implementierung des API-Gateway-Templates in die Pipeline war ein Gateway vorhanden, die Test-Rückgaben konnten jedoch nicht mehr von Browsern dargestellt werden. Der Rückgabewert des setFree-Teil-Services war ein base64 entschlüsseltes Bild und in der Lambda-Testumgebung von AWS wurde eine korrekte Ausgabe erzielt. Laut der Dokumentation sollten base64 entschlüsselte Bilder durch das API-Gateway definiert werden, indem man den Binary Media Type auf das erwartete Image-Format setzt. (image/png)

Nach weiteren Anläufen die Bilder korrekt darzustellen wurde das Problem bei K2 platziert und ein Ticket bei dem AWS-Service eröffnet. Nach diversen Problemlösungsansätzen durch den AWS-Support wurde entdeckt, dass Binary Media Types in API-Gateways nicht mit einem explizitem Typ gesetzt werden können. Die Binary Media Types Option wurde auf */* gesetzt und die Bilder wurden korrekt dargestellt. (Codezeile für die Implementierung von Binary Media Types siehe *Abbildung h.*)

6.4.6. Alarming, Monitoring und Logging

In der alten Hosting-Landschaft wurde das Alarming durch den Dienstleiter festgelegt. Die Datenbanken und die Server wurden von ihnen erstellt und gemanaged. Durch die Umstellung in die AWS-Umgebung musste jedes Fachteam eine eigene Alarmierungsumgebung aufbauen. Für den setFree-Teil-Service waren wichtige Alarmierungsoptionen die, die die Gesundheit des Services anzeigen konnten. Da eine Lambda-Funktion keinen expliziten Status hat, wurden Stati des API-Gateway gewählt und als CloudWatch Alarm implementiert. Teammitglieder würden bei http-Fehlern per E-Mail benachrichtigt werden.

Durch den Dienstleisterwechsel würde in Zukunft kein zentrales Monitoring mehr zur verfügung stehen. Daher musste jedes Fachteam auch eine eigene Monitoringlösung implementieren. Hier bot es sich an CloudWatch auszuprobieren und es wurde eine CloudFormation-Template angelegt, welches alle Standard-Werte in ein Dashboard überführt, welches in der AWS eingesehen werden kann.

Das bestehende Logging über ein Logging Tool wurde angepasst. Alle Logausgaben wurde auf Konsolen-Logging umgestellt. Dadurch würden in der AWS automatisch CloudWatch Logs vom Service erzeugt werden.

6.4.7. Testumgebung

Die Tests waren ohne weitere Anpassungen direkt in der AWS-Umgebung lauffähig. Um eine höhere Effizienz zu erreichen wurden Anpassungen am Lambda-File vorgenommen, damit nach dem Pipeline-Schritt des Testlaufs, nur die Quelldateien des Programms in die Lambda-Funktion implementiert wurden. Das lokale Testen war vorerst nicht möglich. Hierfür bietet Amazon ein Tool, welches sich noch in der Beta-Version befand, AWS Serverless Application Model CLI (SAM CLI) auch SAM lokal genannt. SAM lokal kann für Lambda-Funktionen eine lokale Umgebung erzeugen, Tests darin ausführen und die Umgebung im Anschluss beenden.

7. Projektlösung

7.1. Durchführungszeit

Die für das Projekt genehmigte Arbeitszeit konnte genau eingehalten werden. Für jede Teilaufgabe wurde im Vorfeld eine Zeitschätzung erstellt (siehe *Tabelle d.*). Es wurden im Verlauf des Projekts Teilaufgaben schneller als geplant und andere langsamer als geplant fertiggestellt werden. Das größte Defizit zwischen Planung und Ausführung hatte die Teilaufgabe den Service lauffähig zu machen (siehe *Tabelle e.*).

7.2. Performancetest

Nach der Migration sollte ein Performancetest durchgeführt werden. Dieser sollte vorderrangig nachweisen, dass der Service ohne Unterschiede zu der vorherigen Migration läuft oder performanter ist. Da der Service nicht als EC2-Container implementiert wurde, hätte der Performancetest Hinweise auf einnachbessern der Arbeitsspeicherkapazität geben können. Der Test wurde mittels „Gatling", einem Performancest-Tool, durchgeführt. Die Test wurden in Scala geschrieben.

Die neue Landschaft in der AWS wurde im Test über das Internet aufgerufen, die Tests an der alten Landschaft werden automatisch über das Haus geroutet. Zusätzlich wurden in der alten Umgebung bereits intelligente Caching Optionen gesetzt, die in der AWS-Umgebung noch nicht implementiert waren. Bei der Betrachtung der Ergebnisse musste dieser Umstand mit einberechnet werden. Anhand der Testergebnisse war zu erkennen dass sich die Performance nicht signifikant verändert. Die Lambda-Funktion hat eine etwas langsamere Rückgabezeit gehabt, als der Service in der derzeitigen Umgebung (siehe *Abbildung i. und j.*).

7.3. Kostenunterschied

Das Projekt beinhaltete die Messung des Kostenunterschieds. Der zuständigen Analysten hat errechnet, dass der gesamte Imageprocessing Service ohne Pipeline und anderer Kompenenten etwa 28 Euro im Monat kostete (siehe *Abbildung k.*) Der neuimplementierte Teil-Service, welcher nur ein viertel des Gesamtservices ausmachte hat bis dahin keine Kosten erzeugt (siehe *Abbildung l.*). In der AWS gibt es diverse kostenfreie Kontingent. Eine Million Lambda-Aufrufe sind im Monat kostenfrei. Zusätzlich sind 3,2 Millionen Lambda Verarbeitungssekunden im Monat kostenfrei. Wenn dieses Kontingent erschöpft wäre, würde der Service in der AWS Kosten erzeugen.

Fazit

Der Teil-Service wurde erfolgreich in die AWS mirgirert. Nach der Übergabe an den technischen Entwickler des Teams, wurde entschieden, dass der gesamte Service als Lambda-Funktionen in die AWS überführt werden sollen. An dem Aufbau des API-Gateways müssten dann noch Änderungen vorgenommen werden, damit auch die anderen Teil-Services aufrufbar gemacht werden können. Für die Verbesserung der Performance würde man sich überlegen, ob man den verfügbaren Arbeistspeicher der Lambda-Funktion vergrößert.

Anhang

A. Grafiken

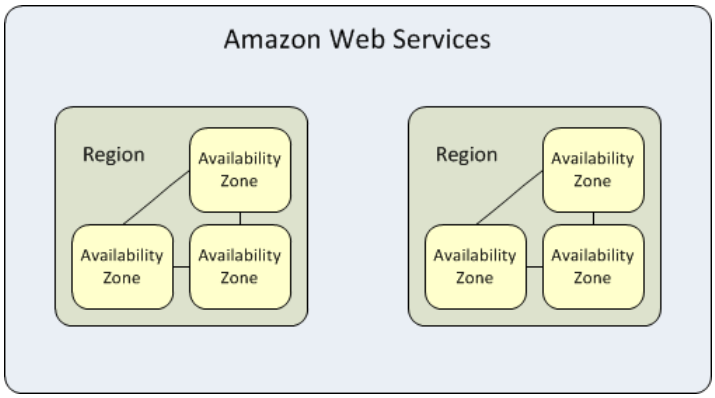

Abbildung c.: AWS Region mit drei Verfügbarkeitszonen (Quelle: https://docs.aws.amazon.com/de_de/AWSEC2/latest/UserGuide/using-regions-availability-zones.html, 2018)

```yaml
---
AWSTemplateFormatVersion: "2010-09-09"

# manually deploy stack, if repository needs to re-initialized

Resources:
  ImageprocessingRepository:
    Type: "AWS::CodeCommit::Repository"
    Properties:
      RepositoryName: imageprocessing
      RepositoryDescription: Pipeline for imageprocessing service
```

Abbildung d.: repository.yaml

```javascript
exports.setFree = (event, context, respond) => {
  if (!event.queryStringParameters || !event.queryStringParameters.imageId ||
    !event.queryStringParameters.width || !event.queryStringParameters.height) {
    console.error(`${ERROR_PREFIX} missing required imageId, width or height`);
    respond(null, {
      statusCode: HttpStatus.NOT_ACCEPTABLE,
      body: `Request not acceptable`
    });
    return;
  }
```

Abbildung e.: imageprocessing.js mit implementierten Lambda Handler

```yaml
version: 0.1
phases:
  install:
    commands:
      - npm install
  pre_build:
    commands:
      - npm test
  build:
    commands:
      - rm -rf 'node_modules'
      - NODE_ENV=production npm install
artifacts:
  files:
    - 'node_modules/**/*'
    - 'cloudformation/**/*'
    - 'src/**/*'
    - 'imageprocessing.js'
  type: zip
```

Abbildung f.: buildspec.yaml

```yaml
SetfreeApiResource:
  Type: "AWS::ApiGateway::Resource"
  Properties:
    PathPart: setfree
    ParentId: !GetAtt Api.RootResourceId
    RestApiId: !Ref Api

ApiStage:
  Type: "AWS::ApiGateway::Stage"
  Properties:
    DeploymentId: !Ref ApiDeployment
    MethodSettings:
      - DataTraceEnabled: true
        HttpMethod: "*"
        # https://docs.aws.amazon.com/apigate
        LoggingLevel: INFO
        ResourcePath: "/*"
    RestApiId: !Ref Api
    StageName: !Ref DeploymentStage
```

Abbildung g.: api.yaml

```yaml
Resources:
  Api:
    Type: "AWS::ApiGateway::RestApi"
    Properties:
      Name: !Sub "${ServiceName}-api-${Environment}"
      BinaryMediaTypes:
        - "*/*"
      FailOnWarnings: true
```

Abbildung h.: Alle binary media types erlauben

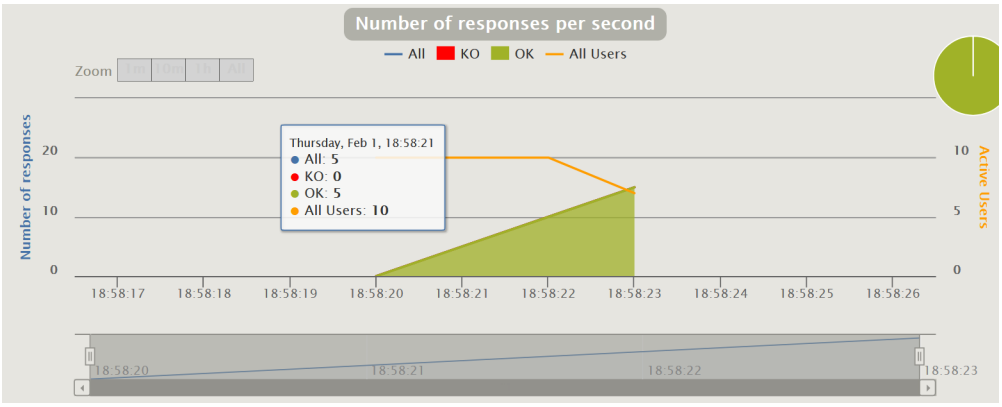

Abbildung i.: Gatling Test alte Landschaft

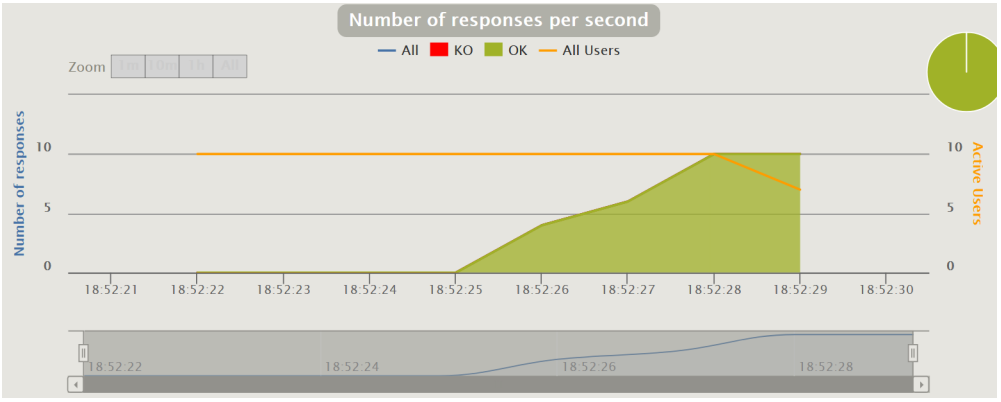

Abbildung j.: Gatling Test neue Landschaft

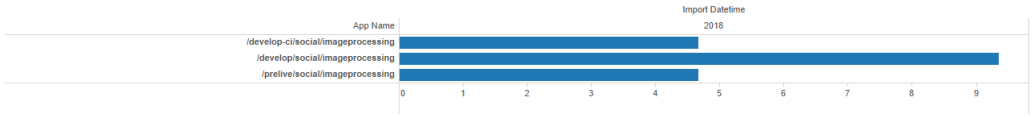

Abbildung k.: Kosten für setFree-Teil-Servie in der alte Umgebung (live kostet soviel wie develop)

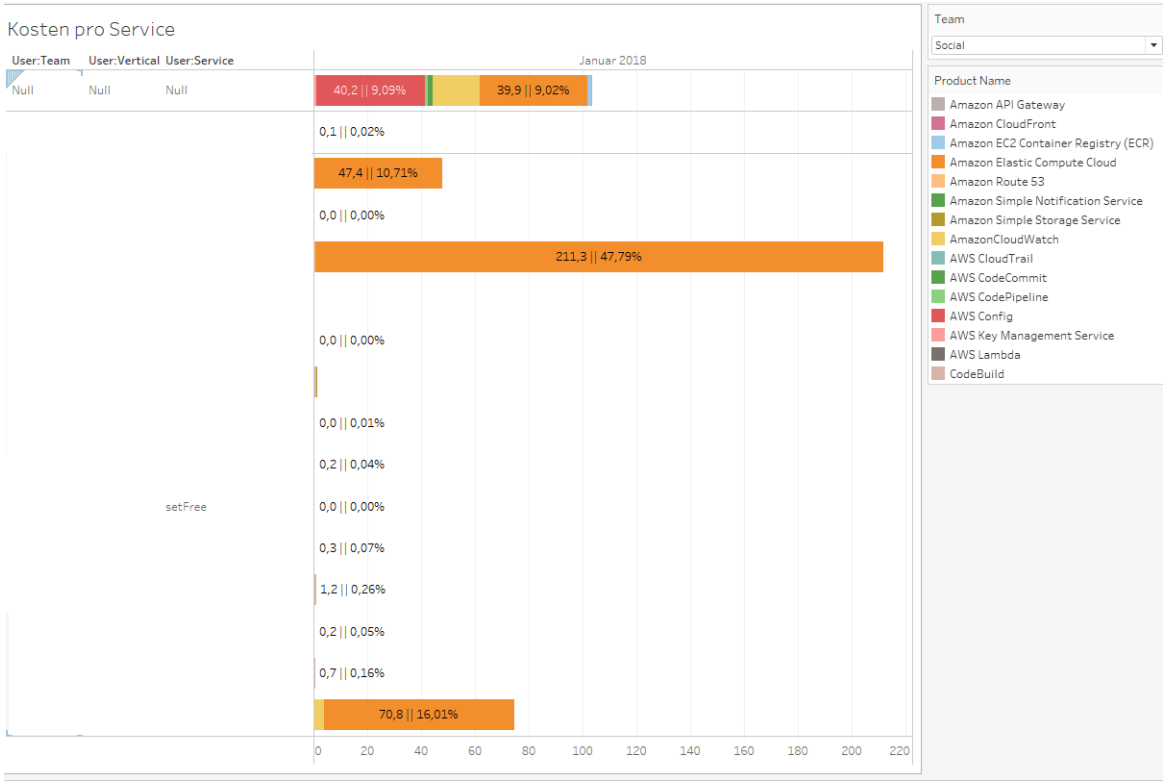

Abbildung I.: Kosten für setFree-Teil-Servie in der neue Umgebung

B. Tabellen

Name	Funktion	Aufgaben innerhalb des Projektes
Antje Radkiewicz	Auszubildende OTTO (GmbH & Co. KG)	Eigenständige Migration des Teil-Microservice
Christian F.	Technischer Entwickler Team Social	Abnahme der Projektarbeit, Anforderungsmanagement
Marco H.	K2-Pate für Team Social	Eskalation Supportanfrage
Christopher R.	Business Analyst	Kostenübersicht Infrastruktur

Tabelle d.: Projektbeteiligte Personen

	Teilaufgabe	Geplante Durchführungszeit in Stunden
1.	Analyse des Services	3
2.	Umsetzungsstrategien erstellen	3
3.	Entscheidung für eine Umsetzungsstrategie	1
4.	Aufsetzen der Infrastruktur	3
5.	Anpassung der Software	5
6.	Aufbau einer lokalen Testumgebung	3
7.	Automatisierte Tests lauffähig machen	1
8.	Logging transferieren	2
9.	Alarmierungsfunktion überführen	1
10.	Monitoring implementieren	2
11.	Performancetest	1
12.	Kostenanalyse	1
13.	Übergabe an das Team	1
14.	Dokumentation aller Prozesse und Ergebnisse	8
	Summe	**35**

Tabelle e.: Zeitplanung pro Teilaufgabe vor der Durchführung

	Teilaufgabe	Geplante Durchführungszeit in Stunden
1.	Analyse des Services	1 (-2)
2.	Umsetzungsstrategien erstellen	3
3.	Entscheidung für eine Umsetzungsstrategie	1
4.	Aufsetzen der Infrastruktur	3
5.	Anpassung der Software	8 (+3)
6.	Aufbau einer lokalen Testumgebung	3
7.	Automatisierte Tests lauffähig machen	1
8.	Logging transferieren	1 (-1)
9.	Alarmierungsfunktion überführen	1
10.	Monitoring implementieren	2
11.	Performancetest	1
12.	Kostenanalyse	1
13.	Übergabe an das Team	0,5 (-0,5)
14.	Dokumentation aller Prozesse und Ergebnisse	8,5 (+0,5)
	Summe	**35**

Tabelle f.: Tatsächlicher Zeitaufwand pro Teilaufgabe nach der Durchführung

C. Glossar

Amazon EC2 Auto Scaling Group	Mittels der Amazon EC2 Auto Scaling Group können Parameter gesetzt werden, wodurch bei Bedarf EC2-Container automatisch hoch- und runtergefahren werden.
Amazon Web Services	Ein Tocherterunternehmen von Amazon. Bietet Cloud-Computing mit weltweit verteilten Rechenzentren an.
Autostacker24	Dies ist ein kleines Tool, welches es erlaubt CloudFormation Ressourcen schnell zu erstellen oder zu updaten.
AWS Identity and Access Management	Ein in die AWS integriertes Benutzerverwaltungstool, welches ein granulares Rechte- und Rollenkonzept erlaubt.
AWS Serverless Application Model CLI	Die AWS SAM CLI erlaubt, es über die Kommandozeile AWS Ressourcen zu erzeugen und zu testen.
Bootstrap	Bootstrapping bezeichnet den Vorgang, bei dem ein Mechanismus zum Starten von Software benötigt wird, die dann weitere Software startet.
CloudFormation	Sprache, mit der Ressourcen in der AWS erzeugt werden können.
CloudFormation-YAML	CloudFormation-Templates können seit 2016 auch im YAML-Format definiert werden.
CloudWatch	Amazon CloudWatch ist ein Überwachungsservice der AWS. Hier können Logs, Alarmierungsfunktionen und Monitoring Dashboards genutzt werden.
CodeCommit	System zur verteilten Versionsverwaltung von Quellcode, welches nativ in der AWS genutzt werden kann.
CodePipeline	Eine integriertes Pipelinetool in der AWS. Die einzelnen Steps können aus CloudFormation-Template erzeugt werden.

Elastic Compute Cloud	Eine Amazon Elastic Compute Cloud ist eine definierte Rechenkapazität in der AWS. Es gibt diverse Größen und Bereitstellungstypen.
GraphicsMagick	Ein Ableger von ImageMagick, welches Bilder mit sehr hoher Farbtiefe unterstützt.
ImageMagick	Ist eine freie Software zur Bearbeitung von Grafiken. Es kann auf der Kommandazeile aufgerufen werden.
Lambda	Eine AWS Lambda ist ein Datenverarbeitungsservice. Hiermit können Prozesse hochgradig automatisiert werden.
Lambda-Trigger	Lambda-Funktionen können über zahlreiche Trigger ausgelöst werden, wie zum Beispiel, ein Zeitpunkt, das eintreten einer Veränderung in einem AWS Service oder API-Gateway Requests.
Microservice	Ein Microservice dient dazu große monolithische Architekturen zu vermeiden. Dies geklingt indem kleine entkoppelte Servies bereitgestellt werden.
Root-Account	Auch Stammbenutzer des AWS-Konto gennant. Dieses Anmeldekonto hat sämtliche rechte auf AWS-Ressourcen und -Services.
Virtual Private Cloud	Die AWS Virtual Private Cloud ermöglicht es, einen Netzwerkbereich in der AWS zu definieren.

D. Abkürzungsverzeichnis

Amazon Web Services - **AWS**, Seite 1
AWS Identity and Access Management - **IAM**, Seite 4
Elastic Compute Cloud - **EC2**, Seite 5
Virtual Private Cloud - **VPC**, Seite 5
AWS Serverless Application Model CLI - **SAM CLI**, Seite 11

E. Abbildungsverzeichnis

F. Tabellenverzeichnis

G. Literaturverzeichnis

Internetquellen

Name des Autors unbekannt: AWS Lambda. Developer Guide. In: AWS documentation. URL: https://aws.amazon.com/documentation/lambda (letzter Aufruf am: 10.05.2018)

Name des Autors unbekannt: AWS Lambda. Preise. In: AWS Lambda. URL: https://aws.amazon.com/de/lambda/pricing (letzter Aufruf am: 11.05.2018)

Name des Autors unbekannt: Enable Binary Support Using the API Gateway Console. In: AWS documentation. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-payload-encodings-configure-with-console.html (letzter Aufruf am: 09.05.2018)

Name des Autors unbekannt: Content Type Conversions in API Gateway. In: AWS documentation. URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-payload-encodings-workflow.html (letzter Aufruf am: 09.05.2018)

Name des Autors unbekannt: Build Specification Reference for AWS CodeBuild. In: AWS documentation. URL: https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html (letzter Aufruf am: 08.05.2018)

Name des Autors unbekannt: Lambda-Funktions-Handler (Node.js). In: AWS Lambda Entwicklerhandbuch. URL: https://docs.aws.amazon.com/de_de/lambda/latest/dg/programming-model.html (letzter Aufruf am: 08.05.2018)

Jean-Christophe Lavocat: Image conversion using Amazon Lambda and S3 in Node.js. In: jice lavocat Blog. URL: http://jice.lavocat.name/blog/2015/image-conversion-using-amazon-lambda-and-s3-in-node.js (letzter Aufruf am: 07.05.2018)

Daniel Lemire: Ridiculously fast base64 encoding and decoding. In: Daniel Lemire's blog. URL: https://lemire.me/blog/2018/01/17/ridiculously-fast-base64-encoding-and-decoding (letzter Aufruf am: 11.05.2018)