Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Einführung von Cloud Technologien am Beispiel Amazon Web Services im Bereich DevOps (Continuous Integration / Continuous Delivery) anhand eines Praxisfalls

Abschlussarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich 4 – Informatik, Kommunikation und Wirtschaft
Studiengang Wirtschaftsinformatik

1. Prüfer:	Prof. Dr. Jörg Courant
2. Prüfer:	M. Sc. Nico Schönnagel

Eingereicht von:	Janek Hornung
Matrikelnummer:	s0549662
Datum der Abgabe:	05.08.2019

Abstract

Ziel dieser Arbeit ist die Einführung aktuellster Cloud-Technologien zur Optimierung des Betriebs eines Webportals, sowie der Automatisierung dessen Software-Release-Prozesses. Bei der Dokumentation und Analyse des alten Prozesses wurden Probleme festgestelllt, wie z.B. zu viele manuelle Tätigkeiten und voneinander abweichende Infrastrukturen.
Die Anforderungen an den neuen Release-Prozess wurden so definiert, dass sie eine cloud-unabhängige Lösung bieten. Durch die Einführung der von Amazon Web Services speziell für Software-Releases ausgelegten Dienste und Werkzeuge, konnte eine sichere, den DevOps-Prinzipien entsprechende Umgebung aufgebaut werden. Diese bietet die Möglichkeit ‚Änderungen verschiedener Entwickler häufiger zu integrieren, erhöht so die Software-Qualität und verkürzt die Zeit, die ein neues Feature braucht, um beim Kunden anzukommen. Durch das Wegfallen manueller Tätigkeiten während des Release bleibt den Entwicklern mehr Zeit für die Bearbeitung von anderen bzw. neuen Kundenwünschen.
Infolge der Einführung von Infrastruktur als Code sind alle Ressourcen der Anwendung dokumentiert und es können Änderungen an ihr Nachvollzogen werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Seit einigen Jahren entsteht durch eine Art Prozessverbesserungsansatz in Softwareentwicklungsfirmen eine neue Unternehmenskultur - das so genannte DevOps. Darunter versteht man die voranschreitende Verbindung der Entwicklung einer Anwendung (Development) mit dem Erstellen, Warten und Monitoren der zum Betrieb benötigten Infrastruktur (Operation). Zusätzlich lassen Trends im Bereich der Cloud-Technologien hier die Grenzen weiter verschwimmen. Zur Einführung von DevOps eignen sich besonders gut die Techniken Continuous Integration (CI) und Continuous Delivery (CD)[1].

1.1 Ausgangssituation und Problemstellung

Der Prozessverbesserungsansatz DevOps mit den Techniken CI und CD soll an einer in der Cloud manuell erstellten Infrastruktur eingeführt werden. Diese hostet ein Webportal.
Hier lassen sich nun folgende Probleme nennen: Bei der Konfiguration dieser Infrastruktur lässt sich nicht nachvollziehen, wer hier Änderungen vorgenommen hat. Müssen der Anwendung neue Funktionen hinzugefügt werden, sind darüber hinaus viele manuelle Schritte notwendig, bis diese in der Produktion ankommen. Durch unterschiedlich konfigurierte Entwicklungs- und Produktivumgebungen, sowie einer fehlenden Staging-Umgebung, kann des Weiteren fehlerhafter Code unbemerkt bis zum Kunden ins produktive System gelangen.

[1] Wolff, E. (2016). *Continuous delivery: der pragmatische Einstieg.* dpunkt. verlag, S. 239.

1.2 Ziele der Arbeit

Hauptziel der Arbeit ist die Optimierung des Software-Release-Prozesses für ein in der Cloud betriebenes Marketingportal. Änderungen am Code, z.B. das Hinzufügen von neuen Features oder Bugfixes, sollen dem Kunden schnell und sicher zur Verfügung gestellt werden. Im Fokus stehen dabei die Automatisierung der anfallenden Tätigkeiten sowie die Transparenz und Nachvollziehbarkeit der geleisteten Arbeit. Hierdurch soll das Sicherheitsgefühl der Entwickler bei ihrer Arbeit steigen.
Durch Reduzierung manueller Eingriffe sowie Tests auf getrennten Umgebungen, welche die Produktion exakt simulieren, sollen Möglichkeiten über Ausfälle nach dem Release minimiert werden. Gerät dennoch ein Fehler in die produktive Umgebung, soll es möglich sein, schnell und einfach auf den letzten funktionierenden Stand zurückzukehren.

1.3 Methodik und Vorgehensweise

Zur Erarbeitung des technischen Grundlagenwissens wurde Literaturrecherche betrieben. Die quantitative Datenerhebung erfolgte durch die Durchführung von Experimenten. Um die ausgearbeiteten Prozesse gegenüberstellen zu können, wurden diese häufig wiederholt und aus den hierbei erhaltenen Daten Key Performance Indicators (KPI) abgeleitet. Vor der Implementierung der umgesetzten Abläufe wurden Ressourcen-Diagramme erstellt.

1.4 Thematische Abgrenzung

Der Wandel einer Unternehmenskultur basierend auf dem DevOps-Ansatz ist ein sich ständig entwickelnder Prozess, in den alle hierbei beteiligten Mitarbeiter integriert werden müssen. Er ist daher sehr umfassend und komplex und geht somit über den Rahmen einer Bachelorarbeit hinaus. Deshalb wird in der vorliegenden Arbeit nur die Umsetzung der technischen Grundlagen von CI/CD als DevOps Tools in der Cloud betrachtet. Behandelt wird ausschließlich der Prozess des Einspielens von Änderungen am Content-Management-System und

an der Infrastruktur eines Marketingportals und die damit verbundenen Aufgaben. Code-Änderungen an weiteren, dem Portal angeschlossene Microservices o. Ä. werden nicht berücksichtigt.

1.5 Aufbau der Arbeit

Das erste Kapitel enthält die Einleitung und gibt einen Einblick in die Arbeit. Im nächsten Kapitel werden die für diese Arbeit notwendigen Systeme, Anwendungen und Definitionen erläutert. Darauf folgt die Vorstellung des Praxisfalls und die Analyse des Deployment Prozesses. Anhand dieser Analyse werden im 4. Kapitel cloud-agnostische Anforderungen definiert, welche die beschriebenen Ziele erfüllen sollen. Darauf folgt das Entwerfen der auf die Amazon Web Services (AWS)-Cloud ausgerichteten Prozesse und Abläufe, sowie ihre Implementierung.
Abschließend enthält das letzte Kapitel eine Zusammenfassung der in dieser Arbeit vorliegenden Ergebnisse, sowie einen Ausblick auf Möglichkeiten, diese Prozesse zu optimieren und zu erweitern.

Kapitel 2

Theoretischer Hintergrund

In diesem Kapitel werden u. A. die Grundlagen des Cloud Computing (2.1), sowie der darin betriebenen Infrastruktur (2.3) erläutert. Der letzte Abschnitt befasst sich mit der Abgrenzung von Continuos Integration (2.4), Devlivery (2.5) und Deployment (2.6), sowie einer Definition von DevOps (2.7). Voraussetzung für das Verständnis sind außerdem die Grundlagen der Softwareentwicklungs- und Deployment-Prozesse.

2.1 Cloud Computing

Das National Institute of Standards and Technology (NIST) definiert Cloud Computing als ein Modell, das es erlaubt, bei Bedarf jederzeit und von überall bequem über das Internet auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z.B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.[1]

Davis und Daniels vereinfachen die Definition mit ihrer Beschreibung von Cloud Computing als eine Art von gemeinsam genutzten internetbasierten Rechnern, bei denen Kunden be-

[1] Mell, P., Grance, T. et al. (2011). The NIST definition of cloud computing. Zugriff unter https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, S. 2.

nötigte Ressourcen bei verschiedenen Anbietern nach Bedarf erwerben und teilen können. Dies kann die Fixkosten für Einkauf, Installation und Wartung eigener Hardware senken.[2]

NIST unterteilt die Cloud dabei in vier Kategorien, die wie folgt unterschieden werden:[3]

1. **Private Cloud**
   Die Infrastruktur wird exklusiv für ein Unternehmen bereitgestellt. Das Unternehmen kann die Infrastruktur selbst betreiben oder von einem Drittanbieter betreiben lassen.

2. **Öffentliche Cloud**
   Die Infrastruktur wird der Öffentlichkeit zur Verfügung gestellt. Sie kann von Unternehmen, akademischen oder staatlichen Einrichtungen oder einer Kombination daraus betrieben werden.

3. **Gemeinschafts-Cloud**
   Die Infrastruktur wird von einer Interessensgemeinschaft von Unternehmen geteilt. Die Infrastruktur kann von der Gemeinschaft oder einem Dienstleister betrieben werden.

4. **Hybride Cloud**
   Die Infrastruktur besteht aus zwei oder mehr eigenständigen Cloud Infrastrukturen, (privat, öffentlich oder gemeinschaftlich), die über Schnittstellen miteinander verbunden sind.

2.2 Versionskontrollsystem

Arbeiten mehrere Entwickler an einer Codebasis und ändern dieselbe Datei, kommt es beim Speichern auf der zentralen Ablage zu Änderungskonflikten. Davis und Daniels schreiben, dass ein Versionskontrollsystem (VKS) alle Änderungen der in ihm abliegenden Dateien oder Sammlungen von Dateien aufzeichnet. Dabei kann es sich um Quellcode, Anhänge oder andere Dokumente des jeweiligen Projekts handeln. Zusammengefasste Änderungen der Entwickler werden Commits genannten. Jeder Commit ist mit Metadaten, wie u. a. Datum,

[2] Davis, J. & Daniels, R. (2016). *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale.* O'Reilly Media, Inc., S. 41.
[3] Mell, Grance et al., 2011, S.3.

Entwickler und Kommentar im System gespeichert.[4]
Git[5] ist ein VKS, das für den Einsatz von mehreren Entwicklern und deren Kollaboration ausgelegt ist. Alle Dateien einer Anwendung werden in Repositories (Projektarchive) auf sogenannten Branches abgelegt. Die Branch auf welcher der Code des produktiven Systems abliegt heißt Master. Arbeitet ein Entwickler z.B. an einem neuen Feature, erstellt er eine neue Branch auf Basis des Masters mit dem Namen dieses Features. Ist er der Meinung mit der Arbeit am Feature fertig zu sein, fügt er alle Änderungen, die er auf seiner Feature-Branch gemacht hat, mit dem Master zusammen. Diesen Vorgang nennt man Mergen.

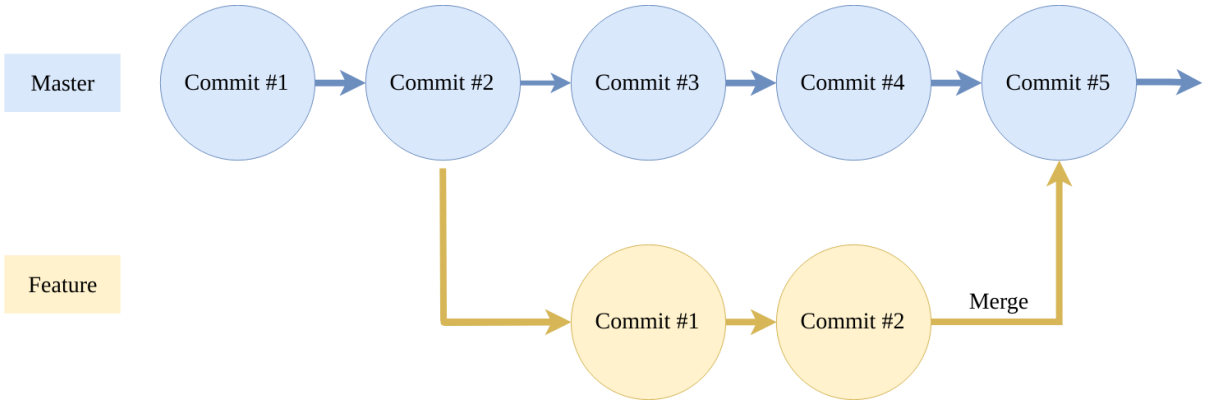

Abbildung 2.1: Git-Flow mit Master- und Feature-Branch

Arbeiten mehrere Entwickler gleichzeitig auf jeweils ihrer eigenen Feature-Branch an derselben Datei und mergen diese auf die Masterbranch, ergibt sich ein Merge-Konflikt. VKS zeigen den Entwicklern die vom Konflikt betroffenen Codezeilen in der Datei an, damit andere Entwickler diese Änderungen übernehmen oder gegebenenfalls anpassen können, um den Merge abzuschließen.

Änderungen unterschiedlicher Entwickler zu vergleichen und zusammenzuführen, erlaubt eine bessere Zusammenarbeit in und zwischen Entwicklerteams. Die Möglichkeit, auf vergangene Änderungen zurückgehen zu können, minimiert die Risiken eines fehlerhaften Deployments.[6]

Um sicherzustellen, dass das Produktivsystem kontinuierlich, schnell und vorhersehbar wiederhergestellt werden kann, müssen folgende Objekte (Assets) in das zentral genutzte VKS

[4] Davis und Daniels, 2016, S. 37.
[5] GitHub, Inc., 2019.
[6] Davis und Daniels, 2016, S. 178.

abgelegt werden:[7]

1. Der gesamte Anwendungscode sowie die dazugehörigen Abhängigkeiten, wie z.B. Bibliotheken,
2. Skripte zum Erstellen von Datenbanken und Anwendungsreferenzdaten,
3. Tools und Artefakte zum Erstellen von Umgebungen (z.B. VMware- oder AMI-Images, Automatisierungsskripte),
4. Container Definitionsdateien (z.B. Dockerfile),
5. Alle Skripte zur Automatisierung der Deployments (Rollout)
6. Anforderungsdokumente, Release-Hinweise und Infrastruktur-Skripte

Liegen alle oben genannten Assets in einem zentralen VKS ab, erzielt man vollständige Transparenz über dessen Anwendung und die gesamte Infrastruktur. Man ist somit in der Lage, Review-Prozesse für den Anwendungs-, sowie den Infrastruktur Code zu definieren und verbessert dadurch die generelle Arbeitsbedingung für die Entwickler.

2.3 IT-Infrastrukturen

Jede Computer-Software wird auf einer Form von IT-Infrastruktur betrieben. Dabei kann es sich um Hardware handeln, die eine Firma besitzt und selbst verwaltet, um geleaste Ausrüstung, bei der die Wartungsverantwortung abgegeben wurde, oder um bei Bedarf verfügbare Computerressourcen, die einfach hoch- und runter-skaliert werden können.[8]

IT-Infrastrukturen können klassisch als Hardware-Server in einem Rechenzentrum, virtuell auf entsprechender Hardware oder in der Cloud betrieben werden.

[7] Kim, G., Humble, J., Debois, P. & Willis, J. (2017). *Das DevOps-Handbuch: Teams, Tools und Infrastrukturen erfolgreich umgestalten.* O'Reilly Media, Inc., S. 164 f.
[8] Davis und Daniels, 2016, S. 40.

Infrastruktur als Code

Nach Hewlett Packard Enterprise wird bei einer Infrastruktur als Code (IaC) die Konfiguration der Infrastruktur genau so wie ein Programmcode von Software behandelt. Dadurch verschwimmen die Grenzen vom Schreiben der Anwendung und der Umgebung auf der sie ausgeführt wird. So können beispielsweise Anwendungen mit Skripten ausgestattet werden, um ihre eigenen Server zu starten. Dies bildet eine Grundlage von Cloud Computing und ist wesentlich für DevOps.[9]

„Das IaC-Konzept ist das Fundament, aus dem DevOps hervorgegangen ist. Durch die zunehmende Gratwanderung zwischen dem Code zur Ausführung von Anwendungen und dem Code zur Konfiguration der Infrastruktur nähern sich die Aufgabenbereiche von Entwicklern und IT-Administratoren immer mehr an.“[10]

Nach Wolf ergeben sich durch IaC folgende Vorteile:[11]

- Die Infrastruktur wird durch einen automatisierten Prozess erstellt und muss nicht mehr aufwändig von Hand aufgebaut und konfiguriert werden.
- Zusätzliche Sicherheit durch Vermeidung von Fehlern beim Umgebungsaufbau ist gegeben, da jede Umgebung derselben Konfiguration zugrunde liegt. Dadurch kann garantiert werden, dass die Staging-Umgebung der Produktivumgebung entspricht.
- Da der Infrastrukturcode im VKS abliegt, können nachvollziehbare Prozesse mit Codereviews erstellt werden.
- Durch die gemeinsame Versionierung wird sichergestellt, dass die Infrastruktur mit der Software kompatibel ist.
- Alle Ressourcen und deren Änderungen sind über die Konfigurationsdatei dokumentiert.
- Theoretisch kann ohne Aufwand eine beliebige Anzahl an Umgebungen erstellt werden, was gerade in Testphasen eine große Rolle spielt.

[9] HP, Enterprise Development. (2019). Was ist Infrastructure as Code? Zugriff 6. Juni 2019 unter https://www.hpe.com/de/de/what-is/infrastructure-as-code.html.
[10] HP, Enterprise Development, 2019.
[11] Wolff, 2016, S. 75.

Hierdurch zeigt sich, dass IaC ein wichtiger Bestandteil dieser Arbeit sein wird um zum einen eine transparente, nachvollziehbare Infrastruktur betreiben zu können und zum anderem, um durch den automatisierten Aufbau einer Staging-Umgebung den Entwicklern Sicherheit beim Release in die Produktion zu geben.

2.4 Continuous Integration

Nach Davis und Daniels ist Continuous Integration (CI) ein Prozess, bei dem neuer Code kontinuierlich in eine Masterbranch aufgenommen wird. Dies steht im Gegensatz zu Entwicklern, die in ihren unabhängigen Feature-Branches arbeiten und ihre Änderungen erst dann veröffentlichen, wenn sie diese abgeschlossen haben. Hierbei vergeht viel Zeit bei dem Zusammenfügen ihrer Änderungen mit dem Master. Wurden auf diesem schon mehrere Änderungen eingespielt, erhöht dies die Wahrscheinlichkeit für Fehler.[12]
Damit keine größeren Nacharbeiten aufgrund von Konflikten nach dem Mergen von Änderungen an einer Datei durch unterschiedliche Entwickler vorgenommen werden müssen, ist es wichtig, den Vorgang des Mergens kontinuierlich zu wiederholen. Das regelmäßige Integrieren der Software in den Masterbranch stellt somit sicher, dass die Änderungen der Entwickler erfolgreich gebaut werden können.

Ein Build-Server checkt die aktuellste Version aus dem VKS aus und führt automatisiert das Bauen des Quellcode aus.[13]

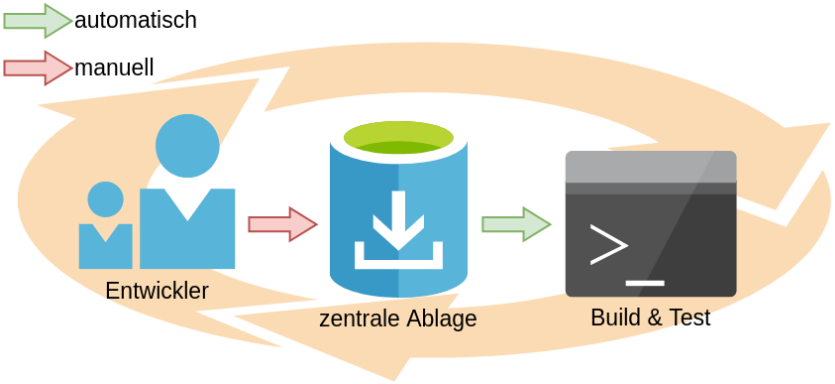

Abbildung 2.2: Continuous Integration

[12] Davis und Daniels, 2016, S. 38.
[13] Wolff, 2016, S. 107.

Abbildung 2.2 zeigt den Kreislauf vom Mergen des Codes durch die Entwickler in die zentrale Ablage mit VKS bis zum Auschecken des Codes durch den CI-Server für das automatisierte Bauen. Schlägt ein Build fehl, wird der entsprechende Entwickler informiert und kann den Kreislauf von Neuem beginnen. Erst wenn dessen Änderungen gebaut werden können, sind sie erfolgreich in die Code-Änderungen der anderen Entwickler integriert.

2.5 Continuous Delivery

„Continuous Delivery ermöglicht es, Software viel schneller und mit wesentlich höherer Zuverlässigkeit in Produktion zu bringen, als es bisher möglich war. Grundlage dafür ist eine Continuous Delivery Pipeline, die das Ausrollen der Software weitgehend automatisiert und so einen reproduzierbaren, risikoärmeren Prozess für die Bereitstellung neuer Releases bietet.“[14]

Somit stellt Continuous Delivery (CD) eine Erweiterung von CI dar und umfasst den kompletten Release Prozess einer Software. In dem Buch *Effective Devops* wird CD als eine Sammlung von allgemeinen Softwareentwicklungswerkzeugen beschrieben, die regelmäßige Releases durch den Einsatz von automatisierten Tests und CI ermöglichen. Über das erfolgreiche Integrieren hinaus stellt CD sicher, dass die Änderungen auch ausgerollt (deployt) werden können.[15]

Es wird also mithilfe geeigneter Tools sichergestellt, dass die zuvor auf Integration getesteten Änderungen auch in das produktive System eingespielt werden können. Dafür werden mittels IaC automatisiert Staging-Umgebungen erstellt, die somit exakt der Produktivumgebung entsprechen. Das dort eingespielte Feature wird sich daher auch genauso in der produktiven Umgebung verhalten, wie es sich in der Staging-Umgebung verhält.

Das folgende Schaubild 2.3 erweitert den in Abbildung 2.2 gezeigten CI-Prozess zu einer CD-Pipeline.

[14] Wolff, 2016, S. 1.
[15] Davis und Daniels, 2016, S. 39.

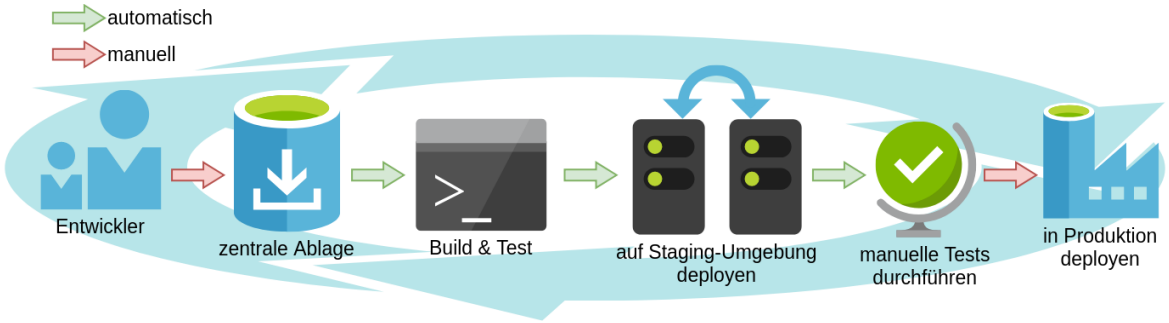

Abbildung 2.3: Continuous-Delivery-Pipeline

Die CD-Pipeline veranschaulicht wie die Code-Änderungen weiter auf die Staging-Umgebung deployt und anschließend getestet werden. Nach einer manuellen Freigabe werden die Änderungen schließlich in das Produktiv-System eingespielt.

2.6 Continuous Deployment

Während Continuous Delivery sicherstellt, dass neue Änderungen deployt werden können, bedeutet Continuous Deployment, dass diese auch in die Produktion deployt werden.[16] Der wesentliche Unterschied zu Continuous Delivery ist somit die automatische statt manuelle Freigabe vor dem Produktiv-Deployment.

2.7 DevOps

Die Entwicklung (Dev) und der Betrieb (Ops) einer Software-Anwendung ist klassisch in zwei unterschiedliche Abteilungen aufgeteilt, die erst weit oben im IT-Management zusammengefasst werden, so Wollf.[17] Weiter fügt er hinzu, dass der zusammenfassende Begriff DevOps schon zeige, dass der Kern des Gedankens die Zusammenführung dieser beiden Abteilungen ist und die Aufteilung der Teams eher anhand ihrer Fähigkeiten und Komponenten erfolgt.[18]

[16] Davis und Daniels, 2016, S. 39.
[17] Wolff, 2016, S. 235.
[18] Wolff, 2016, S.237.

Davis und Daniels beschreiben DevOps als eine Bewegung, die sich durch Geschichten und Ideen einzelner Personen, Teams und Organisationen definiert. Es seien die ständige Konversation und Evolution von Prozessen und Ideen, die zu Wachstum und Änderung führen.[19]

Nach dem *DevOps Handbuch* ermögliche dieser Ansatz das Enstehen einer Kultur, die eine Atmosphäre zum Experimentieren und Eingehen von Risiken unterstützt und das firmenweite Lernen ermöglicht. Außerdem werde eine ständige Erweiterung einer Feedback-Schleife in den Fokus gestellt, um eher Risiken eingehen und Experimente durchführen zu können. Dies solle letztlich dazu führen, dass eine Organisation schneller als die Konkurrenz lernen könne und dadurch im Markt erfolgreicher sei.[20]

Da DevOps Teil einer Unternehmenskultur ist, muss es auch für jedes Unternehmen spezifisch definiert werden. Laut Wolff lässt sich CD im Rahmen von DevOps besonders gut einsetzen, da für CD das Wissen der beiden betroffenen Abteilungen benötigt wird. Die Entwicklung kennt die Anwendung und ihre Konfiguration und der Betrieb kann Tools zum Monitoring bereitstellen.[21]

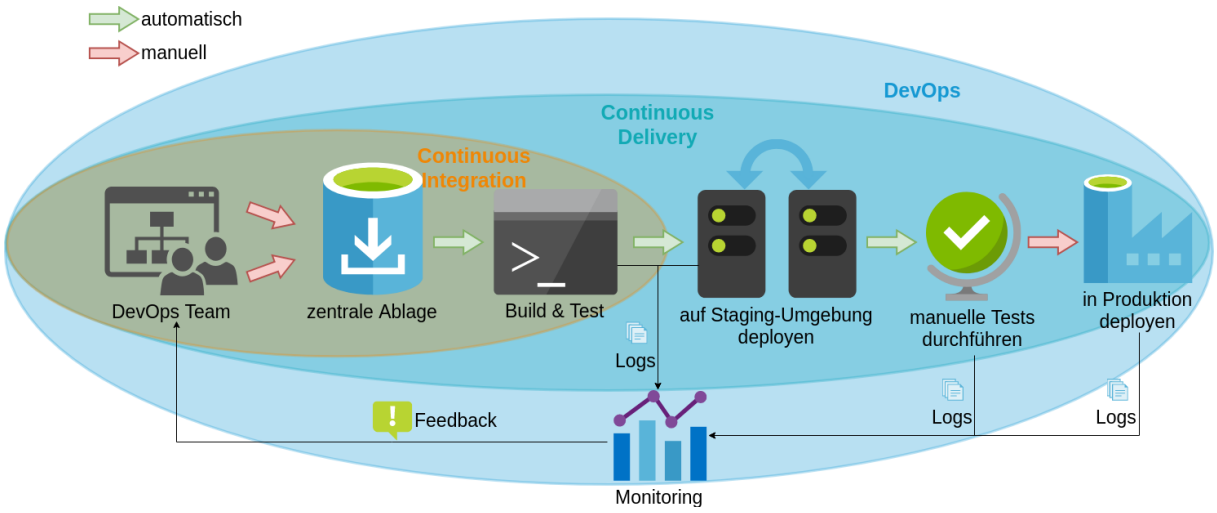

Abbildung 2.4: Übersicht DevOps

In der vorliegenden Arbeit wird DevOps, wie in Abbildung 2.4 gezeigt, als Rahmen um die CD Pipeline gelegt. Die Verbindung von Anwendungsentwicklung und Betrieb in einer Abtei-

[19] Davis und Daniels, 2016, S. 52.
[20] Kim et al., 2017, S. 57.
[21] Wolff, 2016, S. 239.

lung, sowie das ständige Überprüfen der Anwendung durch Protokolldateien, steht somit im Vordergrund. Dadurch bildet anstelle eines Entwicklers ein DevOps-Team den Schwerpunkt, mit den für beide Aufgaben notwendigen Fähigkeiten.

Kapitel 3

Analyse des Praxisfalls

In diesem Kapitel wird zunächst das in der AWS-Cloud betriebene Marketingportal vorgestellt, für welches DevOps eingeführt werden soll. Danach erfolgt eine Analyse des bestehenden manuellen Deployment-Prozesses mittels Erhebung von Key-Performance-Indikatoren, die zum späteren Vergleich dienen sollen.

3.1 Vorstellung Marketingportal

Das Marketingportal ukv-verona.de ist eine nicht-öffentliche Webplattform für Vertriebsmitarbeiter der UKV (Union Krankenversicherung). Diese können sich dort Vertriebsmaterialien bestellen und von Redakteuren eingestellte Artikel lesen. Es ist die erste Anwendung der UKV, in einer öffentlichen Cloud. Entwicklung und Betrieb werden von einer einzigen Abteilung übernommen. Als Grundlage für das Portal dient das Content Management System (CMS) WordPress, das durch eigene Entwicklungen erweitert wurde.

Abbildung 3.1 zeigt eine vereinfachte Darstellung der Infrastruktur (die genaue Infrastruktur ist im Anhang A.1 zu sehen).

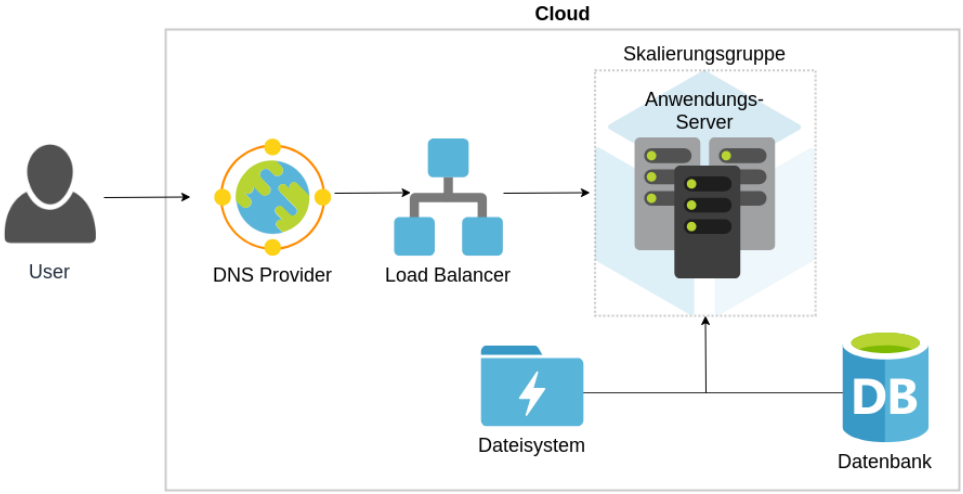

Abbildung 3.1: Infrastruktur Marketingportals ukv-verona.de

- Der **DNS Provider** stellt eine eindeutige für den Benutzer lesbare Adresse für die Anwendung zur Verfügung. Hier ukv-verona.de.
- Auf dem **Anwendungsserver** läuft ein HTTP-Dienst, der die Website hostet.
- Kommen zu viele Anfragen gleichzeitig an einem Server an, kann dies zu einer Überlastung und damit im schlimmsten Fall zu seinem Ausfall führen. Die **Skalierungsgruppe** überprüft die Last der Server und entscheidet anhand vordefinierter Grenzwerte, ob weitere Instanzen hinzugefügt werden müssen.
- Der **Load Balancer** verteilt die anfallende Last auf die Server in der Skalierungsgruppe.
- Das **Dateisystem** und die **Datenbank** werden aus Sicherheits- und Wartungsgründen vom Server getrennt betrieben. So sind beim Ausfall des Anwendungsservers die Einträge in der Datenbank und hochgeladene Dateien noch vorhanden.

3.2 Deployment-Prozess

Der Deployment-Prozess beginnt, wenn der Entwickler mit seinen Änderungen fertig ist und diese in das produktive System einspielen will. Abbildung 3.2 zeigt die Schritte, die bis zu

diesem Zeitpunkt notwendig sind.[1]

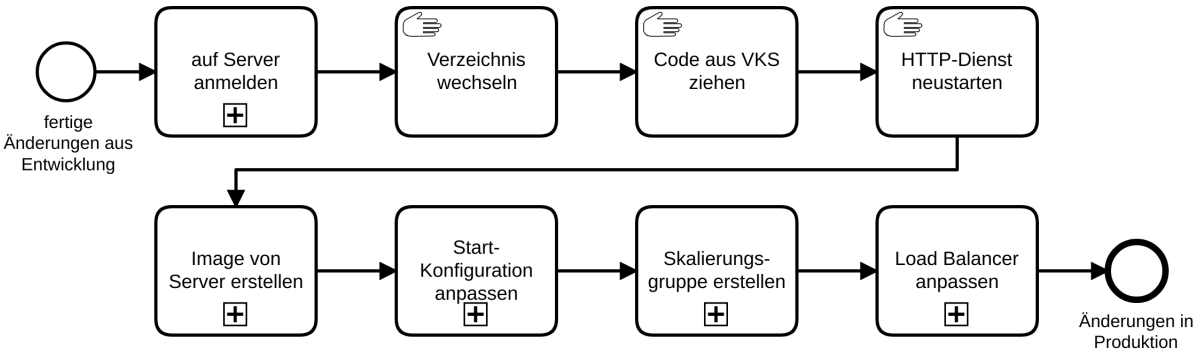

Abbildung 3.2: Deployment-Prozess WordPress

Zuerst muss sich der Entwickler per SSH auf einen Server der Skalierungsgruppe anmelden. Nachdem er in das Anwendungsverzeichnis gewechselt hat, zieht er sich mit dem Befehl "git pull origin"die geänderten Dateien aus der zentralen Ablage und startet den HTTP-Dienst neu - die Änderungen sind damit auf dem Server eingespielt. Weil das Portal zur Ausfallsicherheit gleichzeitig auf mindestens zwei Servern betrieben wird und je nach Last neue Server dazugeschaltet werden, muss die Konfiguration der Skalierungsgruppe so angepasst werden, dass neu dazugeschaltete Server einer Kopie desjenigen Servers entsprechen, auf dem die Änderungen schon eingespielt wurden. Mit Änderungen an der Konfiguration muss eine neue Skalierungsgruppe erstellt werden, auf welche dann der Load Balancer umgeleitet wird.

Zur Analyse und Auswertung wurde der Deployment Prozess mehrmals durchgeführt und ein Ablaufdiagramm (s. Anhang A.2 und A.3) erstellt. Hierbei werden folgende KPIs betrachtet:

Anzahl manueller Tätigkeiten

Die Zahl beschreibt jene Tätigkeiten bei, der ein Entwickler von Hand in den Deployment-Prozess eingreifen muss. Bei jeder manuellen Handlung kann es zu Fehlern kommen. Es wurden insgesamt 28 manuelle Tätigkeiten identifiziert.

[1] ein detalierte Ablauf ist in Anhang A.2 und A.3 zu sehen

Anzahl an Risiken

Diese Zahl nennt jene manuellen Tätigkeiten, bei denen ein Fehler zu Betriebsstörungen führen oder ähnliche Folgen haben kann.

1. Beim Erstellen der Konfiguration der Skalierungsgruppe, muss eines jener Images, auf denen die neuen Server basieren, ausgewählt werden. Beim Auswählen aus einer Liste vieler Image-IDs könnte ein falsches Image ausgewählt werden. Anstatt die Änderungen zu deployen, würde jetzt ein alter Stand bzw. im schlimmsten Fall ein komplett anderer Server geladen werden.

2. Das Erstellen der eigentlichen Skalierungsgruppe birgt genau dasselbe Risiko wie das Anlegen der Konfiguration, da diese manuell aus einer Liste herausgesucht werden muss.

3. Wird der Traffic vom Load Balancer auf die neu erstellte Skalierungsgruppe umgeleitet, könnte es passieren, dass aus Versehen die falsche Gruppe angeklickt wird und somit alle Anfragen auf die falschen Server laufen.

Gesamtdauer des Deployments

Wie lange dauert es, bis die Code-Änderungen nach dem Ablegen im zentralen VKS im Produktivsystem angekommen sind? Nach 10 Test-Deployments wurde eine durchschnittliche Dauer von 15 Minuten ermittelt.

Benötigtes Wissen

Über welche Kenntnisse muss der Entwickler verfügen, um ein erfolgreiches Deployment durchzuführen? Was für unterschiedliche Dienste und Tools werden hierbei benötigt?

1. **Git-Versionskontrolle**
   Jeder Entwickler muss wissen, wie er den aktuellen Code der Anwendung bekommt, seine Änderungen hinzu- und die seiner Kollegen zusammenfügt.

2. **SSH und Linux Terminal**
   Zum Einloggen auf den Server muss ein SSH Client vorhanden sein. Der Umgang mit dem Linux Terminal ist zum Wechseln in das Anwendungsverzeichnis erforderlich.

3. **AWS EC2**
   Für die Erstellung eines Images nach dem Deployment auf dem ersten Server ist der Umgang mit der EC2-Konsole notwendig.

4. **AWS-Skalierungsgruppen**
   Es muss eine Startkonfiguration angelegt und mit ihr eine neue Auto Scaling-Gruppe (ASG) erstellt werden.

5. **AWS Load Balancer**
   Alle Anfragen müssen auf die neue Skalierungsgruppe umgeleitet werden. Dazu müssen die Einstellungen des Load Balancers geändert werden.

Zur späteren Gegenüberstellung werden die Ergebnisse in folgender Tabelle zusammengefasst:

Indikator	**Wert**
Anzahl manuelle Tätigkeiten	28
Anzahl an Risiken	3
Gesamtdauer des Deployment	15 Minuten
Anzahl eingesetzter Software	6

Tabelle 3.1: Bewertung Deployment-Prozess anhand KPIs

Kapitel 4

Anforderungen an den verbesserten Deployment-Prozess

Die in diesem Kapitel beschriebenen, cloud-agnostischen Anforderungen sollen die Abläufe des Deployment-Prozesses nachvollziehbar (4.1) und in seiner Zielform die manuellen Tätigkeiten reduzieren (4.4). Durch die Übertragung aller Ressourcen in Infrastruktur-Skripte wird eine für DevOps grundlegende Technologie eingeführt (4.2). Diese ermöglicht die Nutzung von temporären Staging-Umgebungen (4.3) und bietet die Möglichkeit, einfach auf alte Versionen zurückzukehren (4.5).

4.1 Nachvollziehbarkeit aller Änderungen

Um den Grundgedanken von CI das kontinuierliche Ablegen und Integrieren der Änderungen des einen Entwicklers mit den Änderungen anderer Entwickler zu verwirklichen, muss bei Konflikten einfach nachvollzogen werden können, wer was und wann gemacht hat. Dazu muss die Arbeit eines Entwicklers so transparent wie möglich sein. Alle Änderungen, die an der Infrastruktur oder dem Anwendungscode gemacht werden, müssen darüber hinaus nachvollziehbar sein. Um dies zu erreichen, müssen alle für den Anwendungsbetrieb notwendigen Dateien, Konfigurationen, Skripte und Dokumentationen zentral im VKS abgelegt werden.

4.2 Vollständige Umsetzung der Ressourcen als Code

Alle Ressourcen der Infrastruktur müssen als Code in Form von Konfigurationsdateien angelegt werden. Das hat zum einen den Vorteil, dass durch die Ablage im VKS alle Änderungen an den Ressourcen nachvollziehbar sind, zum anderen erhält man so gleichzeitig auch eine exakte Dokumentation der eingesetzten Infrastruktur. Im Rahmner dieser Arbeit werden die für den Betrieb kritischen Ressourcen[1] der bestehenden Infrastruktur und die komplette Staging-Umgebung als IaC-Skript angelegt. Eine Pipeline exklusiv für die produktive Infrastruktur sorgt dafür, dass Änderungen an den Ressourcen sicher und nachvollziehbar ausgerollt werden.

4.3 Sicherheit beim Release durch getrennte Umgebungen

Die Trennung von Entwicklungs-, Staging-, und Produktiv Umgebungen vereinfacht das gemeinsame Arbeiten und ermöglicht sichere Softwareentwicklung.
Die Entwickler arbeiten auf ihren jeweils eigenen Entwicklungsservern, die auf einer vereinfachten Infrastruktur in der Cloud betrieben werden. Da die Entwicklungs- und Produktionsumgebung nicht die selben Ressourcen nutzen, kann es zu umgebungsabhängigen Fehlern kommen. Damit in der Entwicklungsumgebung nicht vorhandene Fehler nicht erst nach dem Deployment in das Produktivsystem entdeckt werden, soll davor eine temporäre Staging-Umgebung erstellt werden. Diese entspricht der Infrastruktur der Produktivumgebung. Spätestens nach dem Deployment auf diese Zwischenumgebung und anschließenden Tests, kann man sicher sein, dass sich die Anwendung auch identisch in der Produktivumgebung verhalten wird. Diese Umgebung ist nur in Betrieb, wenn Deployments gemacht werden und wird nach ihrer Nutzung gelöscht.

[1] Load Balancer, Skalierungsgruppe und ihre abhängigen Ressourcen

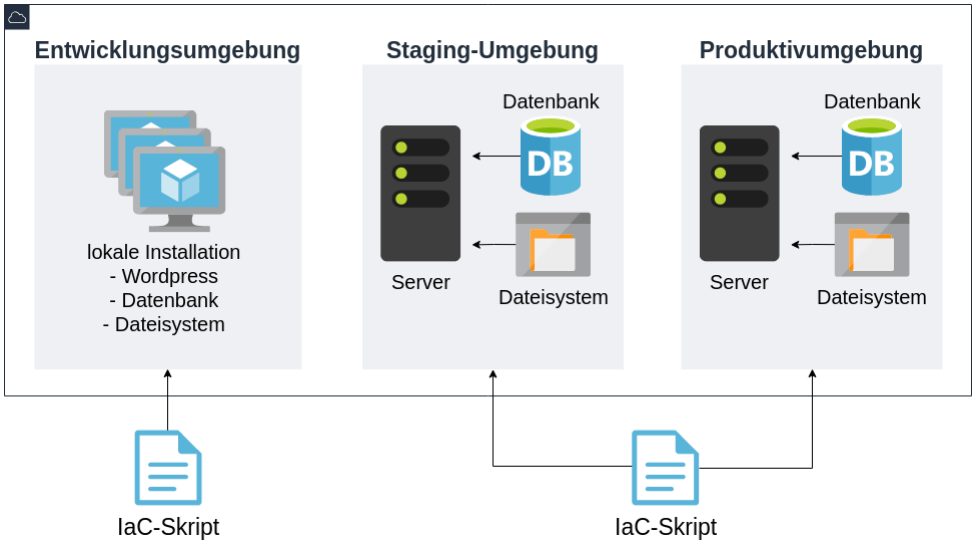

Abbildung 4.1: Getrennte Entwicklungs-, Staging- und Produktivumgebung

4.4 Automatisierung des Deployments

Um die Anzahl manueller Tätigkeiten zu minimieren und die Deployments weitestgehend zu automatisieren, soll eine CI/CD-Pipeline für den WordPress-Code eingeführt werden. Die Hauptstationen sind die zentrale Ablage (Source), das Vorbereiten der Staging-Umgebung (Build) und das Einspielen des Codes in das Produktivsystem (Deploy).
Die Staging-Umgebung wird im Build-Teil der Pipeline automatisch auf Grundlage eines IaC-Skripts aufgebaut. Ist sie erstellt, werden die Änderungen eingespielt und können manuell getestet und abgenommen werden. Nach der Freigabe durch den Product Owner (PO) oder einem anderen Verantwortlichen werden die Änderungen in Produktion gebracht.

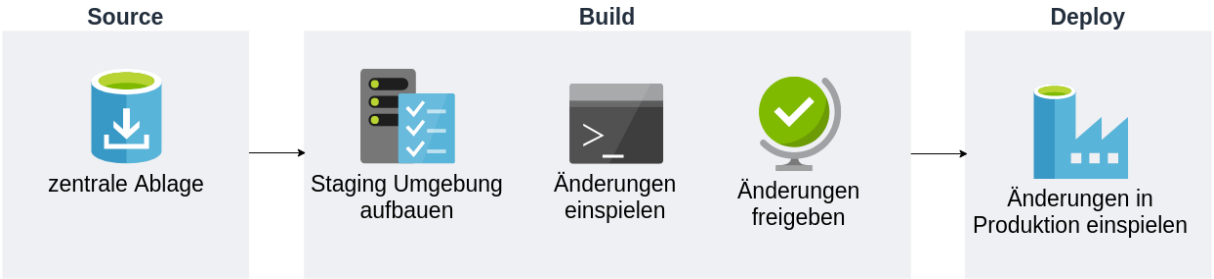

Abbildung 4.2: Entwurf Deployment Pipeline

4.5 Wiederherstellung der Umgebung nach Fehlern

Die Möglichkeit, nach einem fehlerhaften Deployment auf den zuvor in Betrieb gewesenen, funktionierenden Versionsstand der Anwendung zurückzukehren (Rollback), führt zu erhöhtem Sicherheitsgefühl während der Ausführung des Deployment-Prozesses. Das hat zur Folge, dass Änderungen von Entwicklern unbedenklich und öfter eingespielt werden können. Um sicherer arbeiten zu können, sollte Entwicklern diese Funktion zur Verfügung stehen. Dabei soll der Rollback ebenfalls so automatisiert wie möglich ablaufen.

Kapitel 5

Design & Implementierung in der AWS Cloud

Dieses Kapitel beschreibt zuerst die IaC-Lösung von AWS (5.1) und ihren Einsatz in der vorliegenden Arbeit. Danach werden jene Tools vorgestellt, die für die Einführung von DevOps von AWS zur Verfügung gestellt werden (5.2). Über eine Pipeline werden alle für das Deployment angelegten Ressourcen verbunden (5.3). Zum Abschluss folgt eine Gegenüberstellung (5.4) des alten und des neuen Prozesses.

5.1 Nutzung von CloudFormation zur Erstellung der Infrastruktur

CloudFormation[1] ist der IaC Dienst von AWS. Er erlaubt es, alle für eine Anwendung notwendigen Ressourcen in der AWS Cloud in einer Textdatei zu definieren und diese dann hochzuladen und automatisiert erstellen zu lassen. Ein in CloudFormation ausgerolltes Skript, wird als *Stack* bezeichnet. Anhang A.12 zeigt das CloudFormation Template der produktiven Infrastruktur. Es wurden nur die für den Betrieb notwendigen und für die vorliegende Arbeit

[1] Amazon Web Services. (2019a). AWS CloudFormation. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/cloudformation/.

wichtigen Ressourcen[2] umgesetzt. Dadurch können Änderungen an der produktiven Infrastruktur transparent und nachvollziehbar implementiert werden.
Werden mehrere Skripte über ein Hauptskript verbunden, spricht man von einem *Nested-Stack*. Im *Master-Stack* (master.yml) befinden sich so genannte *Child-Stacks*. Das Skript für die Staging-Umgebung soll aus fachlich getrennten Templates bestehen: einen *Child-Stack* für die Netzwerk-Ressourcen (network.yml) und einen *Child-Stack* für die Anwendung (app.yml):

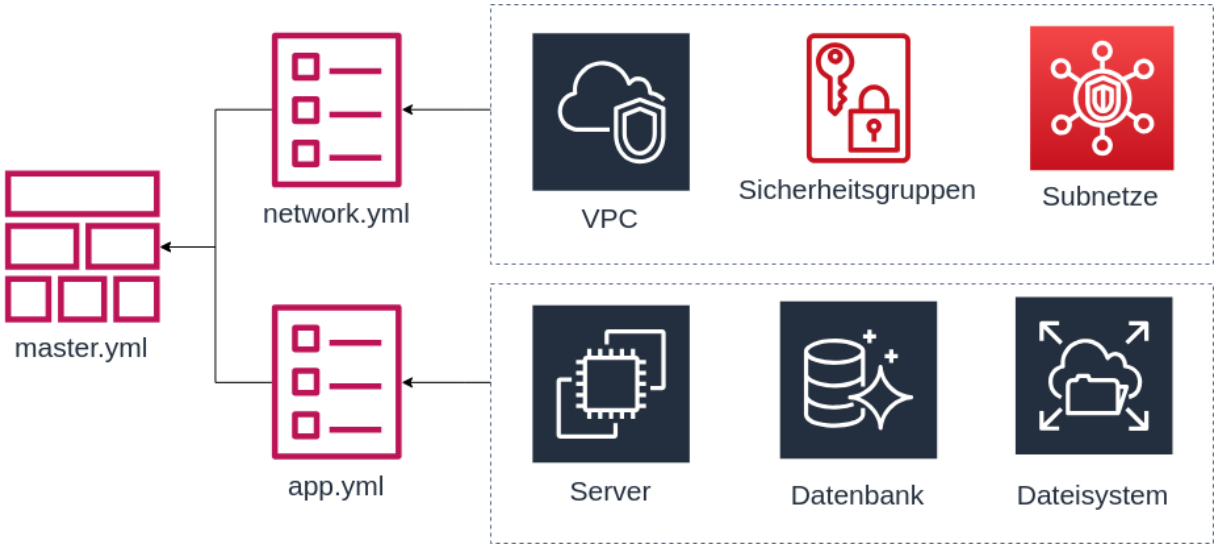

Abbildung 5.1: Aufbau Staging CloudFormation Skript als Nested-Stack

Der Netzwerk-Stack (s. Anhang A.3) enthält ein virtuelles Netz (VPC) mit Sicherheitsgruppen und Subnetze, indem später die Ressourcen aus dem Anwendungs-Stack (s. Anhang A.4) (Server, Datenbank und Dateisystem) erstellt werden. Diese sind im Haupt-Stack (s. Anhang A.1) über einen relativen Pfad verlinkt (Zeile 7 + 13). Der Anwendungs-Stack ist so konfiguriert, dass beim Ausrollen ein Server auf Grundlage eines aktuell produktiven Images gestartet wird. Die Datenbank wird anhand eines Snapshots angelegt, der jede Nacht durch ein automatisches Backup erstellt wird.

[2] Application Load Balancer (ALB) mit Abhängigkeiten, ASG, Elastic Compute Cloud (EC2), Launch Configuration (LC)

5.2 AWS Developer Tools

AWS fasst die für CI/CD benötigten Dienste als Developer Tools zusammen. CodeCommit, CodeBuild und CodeDeploy bieten alle dafür notwendigen Funktionen und können über den AWS-Dienst CodePipeline[3] verbunden werden.

5.2.1 Versionskontrolle über CodeCommit

CodeCommit[4] ist ein selbstverwaltetes, serverloses VKS das auf Git basiert. Es bietet eine zentrale Ablage für Code und Konfigurationen in der AWS Cloud. Für die Anwendung werden in CodeCommit drei Repositories angelegt:

- verona-wordpress-repo
  enthält die WordPress Dateien, die buildspec.yml (s. Abschnitt 5.2.2) und die Deployment-Skripte (s. Abschnitt 5.2.3)
- verona-staging-infra-repo
  enthält die Dateien des CloudFormation Nested-Stack und auch eine buildspec.yml
- verona-prod-infrastructure-repo
  enthält das CloudFormation Template für die produktive Infrastruktur (s. Anhang A.12)

5.2.2 CodeBuild als Integrations-Umgebung

Mit CodeBuild[5] stellt AWS einen vollständig verwalteten Dienst für CI zur Verfügung. Er startet einen Docker Container, indem man seinen Quellcode zur Implementierung vorbereiten und testen kann. Der Quellcode wird als Artefakt von CodeCommit übernommen.
Für die CI/CD-Pipeline werden zwei Build Projekte angelegt: Eines für den CloudFormation *Nested-Stack* (verona-cf-build) und ein weiteres um die Deployment-Skripte (s. Abschnitt

[3] Amazon Web Services. (2019b). AWS CodePipeline. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codepipeline/.
[4] Amazon Web Services. (2019c). AWS CodeCommit. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codecommit/.
[5] Amazon Web Services. (2019d). AWS CodeBuild. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codebuild/.

5.2.3) der Stating-Umgebung mit jener der Produktivumgebung auszutauschen (verona-prod-build). Die Projekte werden mit folgenden Einstellungen angelegt:

- CodeBuild standard Linux
- 3 GB Arbeitsspeicher, 2 vCPUs
- Logs aktiviert
- Service Rolle automatisch anlegen

Jedes Projekt hat eine unterschiedliche Konfigurationsdatei *buildspec.yml*. In ihr wird definiert, was mit den Dateien aus dem Artefakt auf dem Container gemacht werden soll.
Die CloudFormation *Child-Stacks*, die in den *Master-Stack* eingebundenen sind, müssen auf einem S3-Bucket abliegen, um von CloudFormation genutzt werden zu können.[6] Dazu bietet das Kommandozeilentool (CLI) den Befehlt *aws cloudformation package*, der in der buildspec.yml des Build-Projekts angegeben ist:

```
3   install:
4     commands:
5       - aws cloudformation package --template master.yml --s3-bucket ↩
            erona-infrastruktur-bucket --output-template-file master.yml
```

Abbildung 5.2: Auszug der buildspec.yml – Ausführung des CLI-Befehls

Die CLI ist im standard-Image von CodeBuild schon vorinstalliert. Abbildung 5.4 zeigt die Installationsphase des Skripts (das komplette Skript befindet sich in Anhang A.5). Wird der Befehl *aws cloudformation package* ausgeführt, werden die in dem Master-Stack *master.yml* verlinkten CloudFormation Templates auf den S3-Bucket *verona-infrastruktur-bucket* hochgeladen. Die vorher relativen Pfadangaben zu den Child-Stack Templates sind jetzt in absolute Pfade zu den jeweiligen Dateien im S3-Bucket umgewandelt (s. Anhang A.2). Das Template kann jetzt von CloudFormation ausgerollt werden.

Der in Abbildung 5.3 dargestellte Teil der buildspec.yml erstellt ein Artefakt im ZIP-Format mit der neu erstellten Datei *master.yml*.

[6] Amazon Web Services. (2019e). AWS::CloudFormation::Stack. Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-stack.html.

```
6 artifacts:
7   type: zip
8   files:
9     - master.yml
```

Abbildung 5.3: Auszug der buildspec.yml – Erstellung des Artefakts

Die Konfigurationsdatei für den CodeDeploy Agent *appspec.yml* (s. Abschnit 5.2.3) muss im Root-Verzeichnis des Artefakts liegen.[7] Weil die Deployment-Skripte für die Staging- und Produktivumgebung unterschiedlich sind, liegt die für das Produktiv-Deployment benötigte Datei in einem Unterordner ab. Der folgende Teil (das komplette Skript befindet sich in Anhang A.6) der *buildspec.yml* des *verona-prod-build*-Projekts, überschreibt die *appspec.yml* im Root-Verzeichnis mit jener aus einem Unterordner (Zeile 5). Zum Schluss erstellt CodeBuild ein Artefakt, das alle Dateien (WordPress Code und Deployment Skripte) enthält (Zeile 6-9):

```
4     commands:
5       - mv -f scripts/prod/appspec.yml /appspec.yml
6 artifacts:
7   type: zip
8   files:
9     - '**/*'
```

Abbildung 5.4: Auszug der buildspec.yml – Verschiebung Skripte & Artefakterstellung

Durch die selbstverwalteten Container und die einfache Konfigurationsdatei ist CodeBuild ideal für CI in der Cloud geeignet. Nach erfolgreichem Abschließen der Build-Projekte stehen im Deployment Prozess jetzt jeweils ein Artefakt mit dem vorbereiteten Code für die Infrastruktur und dem Anwendungscode für das Webportal zur Verfügung.

[7] Amazon Web Services. (2019f). CodeDeploy AppSpec File Reference. Zugriff 18. Juli 2019 unter https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html.

5.2.3 CodeDeploy für die Releases

CodeDeploy[8] ist ein agentenbasierter Deployment-Dienst. Er kopiert den Quellcode auf die Server und kann davor oder danach Skripte mit Root-Rechten ausführen. Damit können z.B. abhängige Dienste vor dem Deployment gestoppt werden. In der AWS-Konsole wird im CodeDeploy eine Anwendung *verona* angelegt. Es wird eine Bereitstellungsgruppe für die Staging-Umgebung eingerichtet (*deploy-to-staging*) und eine für die Produktivumgebung (*blue-green-deployment*). Über sie wird angegeben, ob das Deployment auf über Tags definierte Server, oder auf einer ASG ausgeführt werden soll. Der Agent wird über die Datei appspec.yml konfiguriert. Der in Abbildung 5.5 dargestellte Ausschnitt (das komplette Skript befindet sich in Anhang A.7) gibt an, dass der Agent alle Dateien aus dem Root-Verzeichnis des Artefakts (Zeile 4) in das Root-Verzeichnis des HTTP-Servers (Zeile 5) kopieren soll:

```
3  files:
4    - source: /
5    destination: /var/www/html
```

Abbildung 5.5: Auszug der appspec.yml – Angabe von Quell- und Zielverzeichnis

Das Deployment ist in Phasen (Hooks) aufgeteilt. Jeder Hook führt eine Skriptdatei auf dem Server aus. Für das WordPress Deployment werden folgende Hooks angegeben:

- **ApplicationStop:** Stoppt den HTTP Dienst (s. Anhang A.8)
- **BeforeInstall:** Trennt die Verbindung zum Dateisystem (s. Anhang A.11)
- **AfterInstall:** Eine Variable wird geändert um für das jeweilige Deployment (Staging oder Produktion) die richtigen Pfade zu haben (s. Anhang A.10).
- **ApplicationStart:** Bindet das Dateisystem ein und startet den HTTP Dienst (s. Anhang A.9)

Für jede Phase wird der Pfad zu dem auszuführenden Skript und die Berechtigung, mit der es ausgeführt werden soll, angegeben. Zusätzlich kann ein Timeout übergeben werden. Ist

[8] Amazon Web Services. (2019g). AWS CodeDeploy. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codedeploy/.

das Skript in dieser Zeit nicht abgeschlossen, bricht das Deployment ab. Abbildung 5.6 zeigt anhand des *ApplicationStop* Hooks, wie sie in der *appspec.yml* definiert werden.

```
7   ApplicationStop:
8   - location: scripts/stop_server.sh
9     timeout: 300
10    runas: root
```

Abbildung 5.6: Auszug der appspec.yml – Definition eines Hooks

Die Bereitstellungsgruppe der Staging-Umgebung ist so eingestellt, dass das Deployment auf allen Server mit dem Namen *verona-staging* ausgeführt wird.

Blue-Green Deployment

Für das produktive Deployment bietet der CodeDeploy Agent die Möglichkeit des Blue-Green Deployments. Dabei wird zuerst die ASG der Anwendung kopiert und darauf die Änderungen eingespielt.

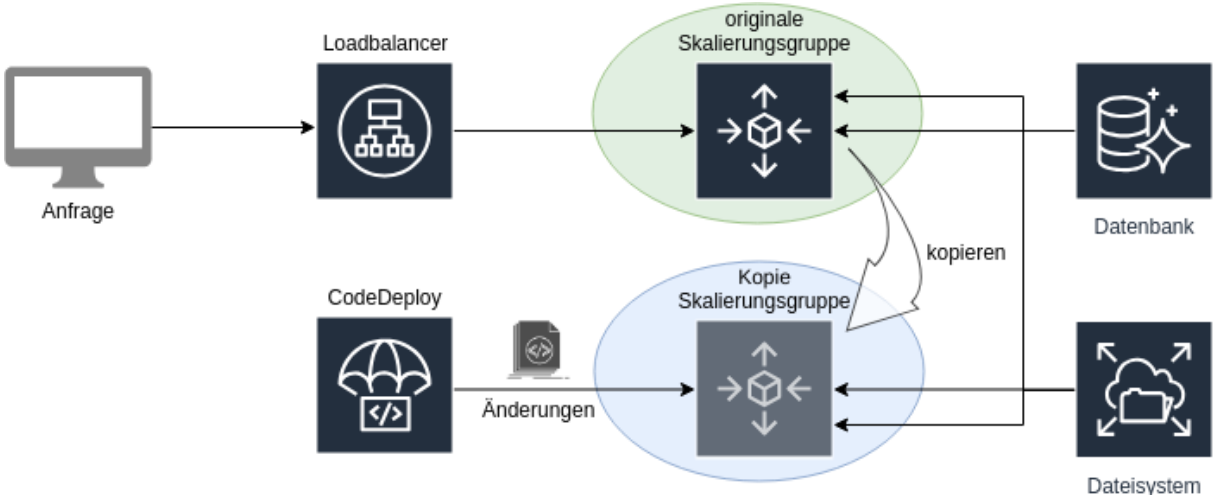

Abbildung 5.7: Phase 1 – Kopieren der ASG und Einspielen der Änderungen

Nach dem Deployment werden die Server von AWS automatisch über eine HTTP-Anfrage auf ihre Verfügbarkeit geprüft. Antwortet der Server, werden alle Anfragen, die auf den ALB

gehen auf die kopierte ASG umgeleitet. Antwortet der Server nicht, stoppt das Deployment und startet automatisch ein Rollback. Dabei wird die kopierte ASG mit den fehlerhaft eingespielten Änderungen gelöscht und alle Anfragen gehen weiterhin auf die originale ASG ohne die neuen Änderungen.

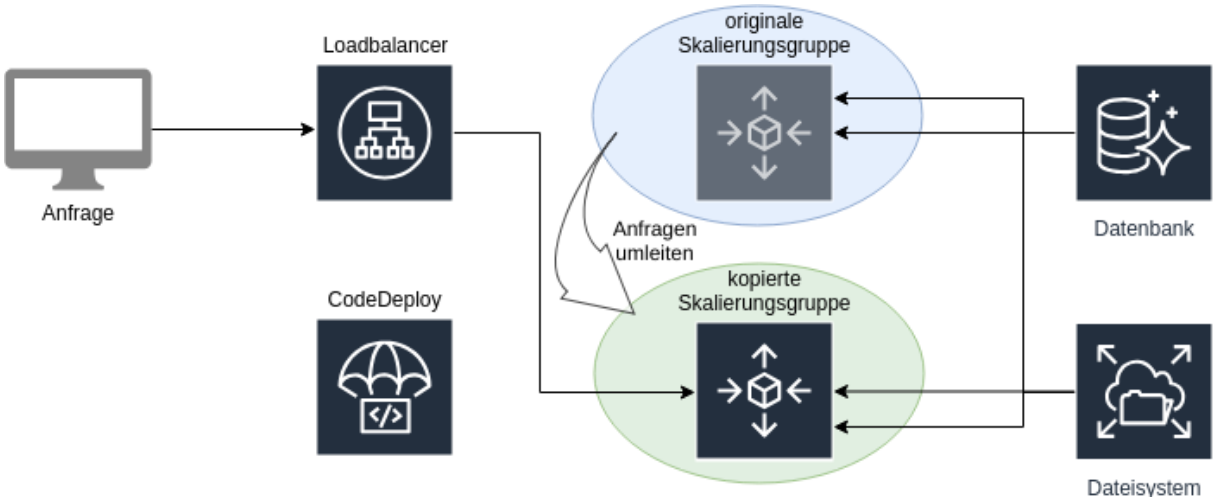

Abbildung 5.8: Phase 2 – Umleiten der Anfragen nach erfolgreichem Deployment

Mit CodeDeploy kann man also die zuvor vorbereiteten Änderungen sicher in das produktive System einspielen. Durch das Blue-Green Deployment können sich die Entwickler sicher sein, dass ihre Anwendung selbst bei einem fehlerhaften Deployment dem Kunden weiterhin zur Verfügung steht.

Dadurch, dass alle Dienste der Developer Tools von AWS verwaltet werden, erspart man sich die Arbeit für das Erstellen und Warten von Servern für eine Integrationsumgebung oder eine zentrale Ablage.

5.3 Continuous Delivery Pipeline

Die in den Developer Tools angelegten Ressourcen werden hier zu der in 5.3 abgebildeten, automatisierten CI/CD-Pipeline verbunden. Die Pipeline beginnt mit den CodeCommit Repositories (5.3.1) und geht weiter über die CodeBuild-Projekte (5.3.2). Anschließend werden die Änderungen über CodeDeploy zuerst auf der Staging-Umgebung (5.3.3) und dann auf

der Produktivumgebung eingespielt (5.3.4). Zum Schluss besteht die Möglichkeit die Staging-Umgebung zu löschen (5.3.4).

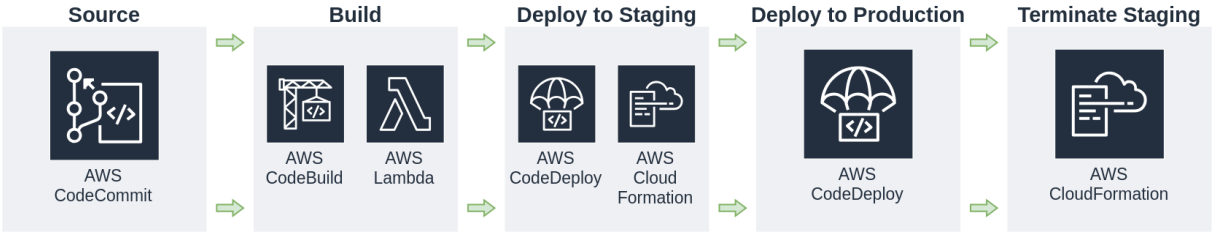

Abbildung 5.9: Vollständige Pipeline mit den jeweils genutzten Tools

5.3.1 Source-Phase

Die erste Phase heißt *Source* (s. Abbildung 5.10). Ausgelöst wird sie, wenn Änderungen vom Entwickler in das VKS der Wordpress-Anwendung eingespielt werden. Sie enthält dazu noch das Repository der Staging-Infrastruktur. Änderungen dort lösen aber nicht die Pipeline aus. In dieser Phase werden alle Anwendungsdateien aus den angegebenen Repositories für die Build-Phase zu jeweils einem Artefakt gepackt. Diese werden auf einen S3-Bucket hochgeladen, um der nächsten Phase zur Verfügung zu stehen.

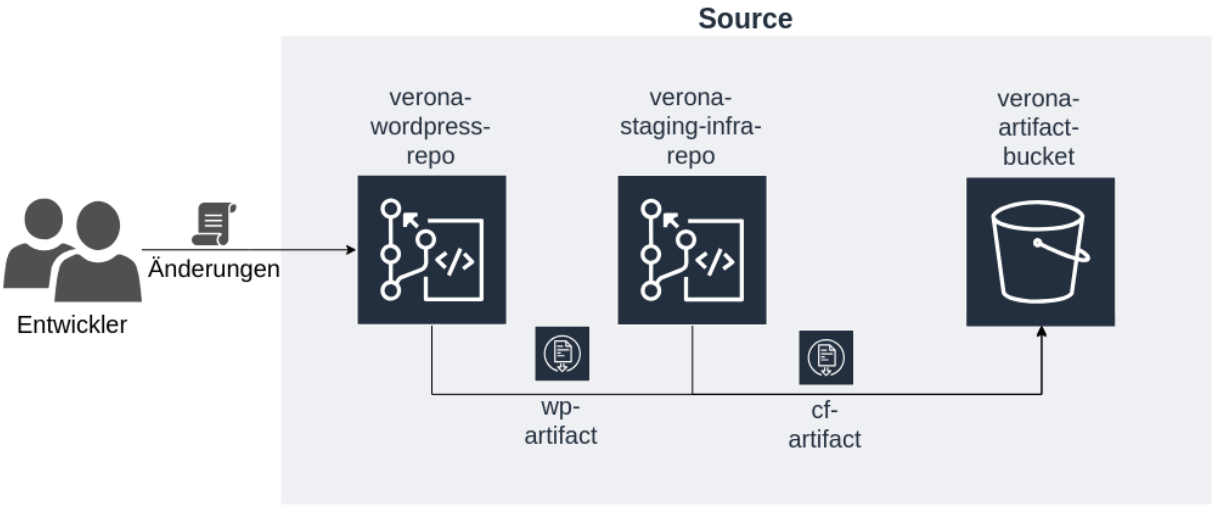

Abbildung 5.10: Source-Phase der CD-Pipeline mit beiden Repositories

5.3.2 Build-Phase

Die *Build*-Phase (s. Abbildung 5.11) übernimmt die Artefakte von CodeCommit und führt parallel die beiden Build-Projekte (s. 5.2.2) *verona-cf-build* und *verona-prod-build* aus. Die dabei erstellten Artefakte werden wieder in den S3-Bucket hochgeladen. Außerdem wird in dieser Phase die Lambda-Funktion *update-db-parameter* (s. Anhang A.13) ausgeführt, die einen Parameter mit dem Snapshot-Namen der letzten Nacht aktualisiert.[9]

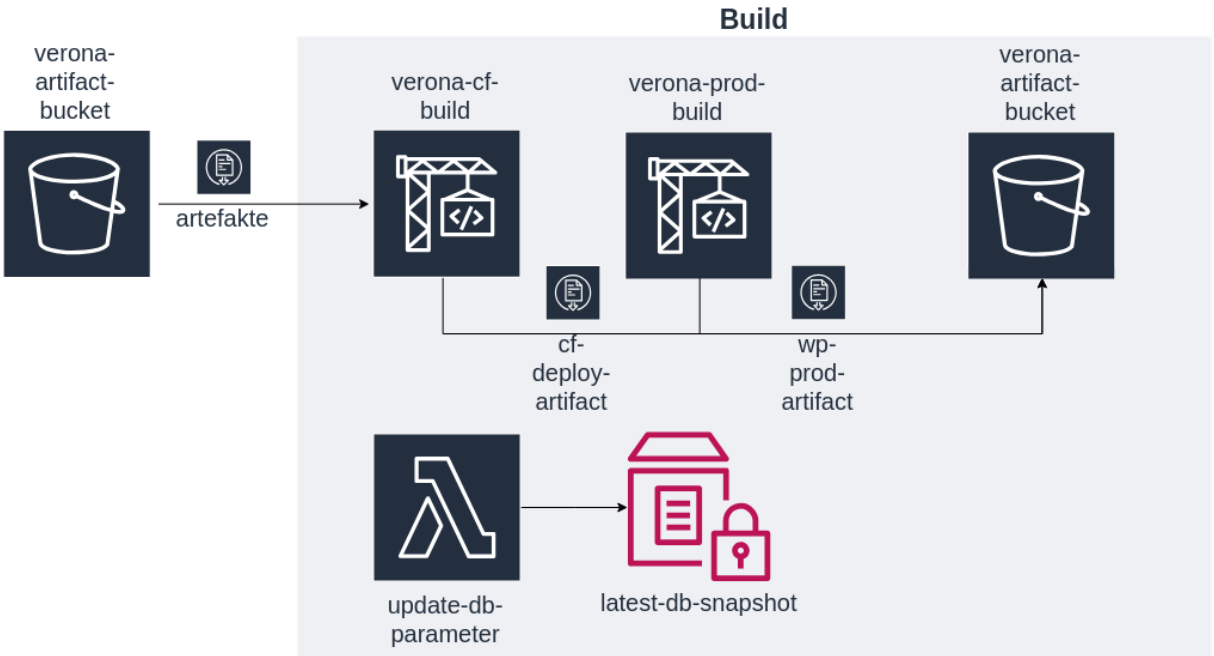

Abbildung 5.11: Build-Phase der CD-Pipeline

5.3.3 Phase Deploy to Staging

Die Phase *Deploy to Staging* (s. Abbildung 5.12) bekommt das von CodeBuild erstellte Infrastruktur-Artefakt übergeben und rollt das darin enthaltene CloudFormation-Skript der Staging-Umgebung aus. Nachdem die Umgebung erstellt wurde, wird der Wordpress-Code aus der Source-Phase über den CodeDeploy Agent eingespielt (das vorbereitete Artefakt aus der Build-Phase wird später für das Produktiv-Deployment genutzt). Die Staging-Umgebung mit den eingespielten

[9] Dieser Parameter wird im ParameterStore, einem Dienst zum Speichern von Key-Value-Paaren, abgelegt und später beim Aufbau der Umgebung von dem CloudFormation Skript ausgelesen

Änderungen kann über eine URL aufgerufen und von dem PO oder anderen Stakeholdern getestet zu werden.

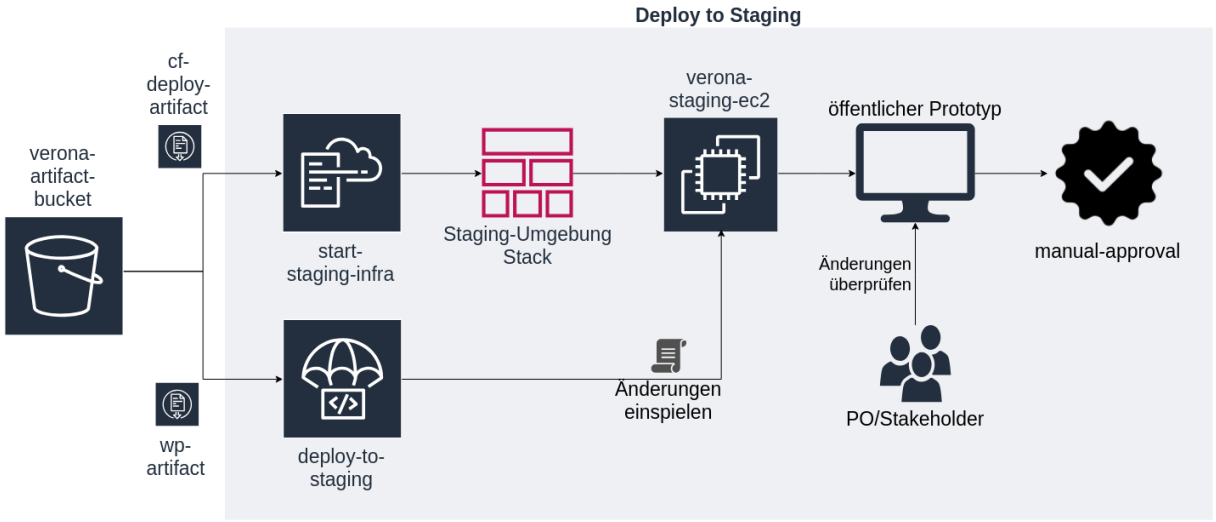

Abbildung 5.12: Phase Deploy to Staging der CD-Pipeline

5.3.4 Phase Deploy to Production & Terminate Staging

Werden keine Fehler gefunden und die Änderungen akzeptiert, müssen diese manuell in der AWS-Konsole freigegeben werden. Danach wird die letzte Phase der Pipeline *Deploy to Production* angestoßen. Diese führt das in Abschnitt 5.2.3 beschriebene Blue-Green Deployment aus. In der letzten Phase *Terminate Staging* besteht für den PO die Möglichkeit, die Staging-Umgebung zu löschen. Damit wird der CloudFormation Stack terminiert und die Ausführung der Pipeline beendet. Sind z.B. wegen eines Sprints weitere Deployments geplant, kann diese Phase übersprungen werden und die Staging-Umgebung bleibt bestehen.

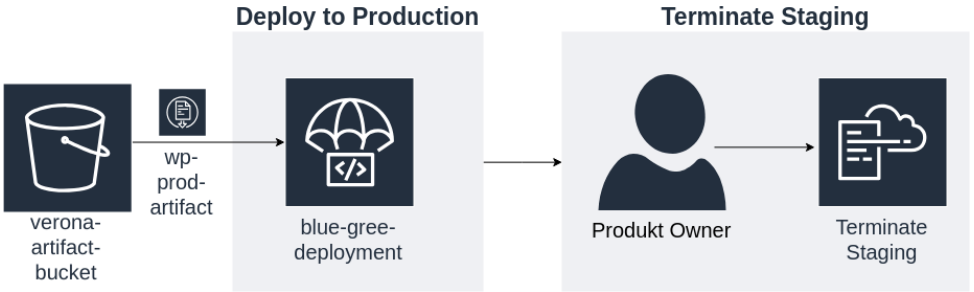

Abbildung 5.13: Produktiv-Deployment und Abbau Staging-Umgebung

5.3.5 Pipeline der produktiven Infrastruktur

Für die Infrastruktur der produktiven Anwendung wird extra eine Pipeline angelegt. Diese besteht aus einer Source-Phase, welche die Template-Dateien enthält, sowie der Deploy-Phase, die den CloudFormation Stack erstellt. Dadurch, und weil die Template Dateien in einem VKS liegen, können Änderungen an Ressourcen der Infrastruktur, genau wie jene an der Anwendung, automatisch und risikofrei eingespielt werden.

5.4 Gegenüberstellung alter & neuer Prozess

Der neue Deployment-Prozess wird nach den in Kapitel 3.2 definierten KPIs bewertet und verglichen.

Anzahl manueller Tätigkeiten

Der Entwickler legt seine fertigen Änderungen manuell im VKS ab. Das führt im Sinne von CD automatisch zum Deployment in die Staging-Umgebung. Die Überprüfung und die Freigabe der Änderungen sind weitere manuelle Tätigkeiten. Darüber hinaus sind keinerlei Eingriffe in den Prozess notwendig.

Anzahl an Risiken

Durch das Deployment in eine getrennte, der produktiven identischen Umgebung, kann vor jedem Produktiv-Deployment davon ausgegangen werden, dass sich die Änderungen ohne Fehler einspielen lassen. Durch die Minimierung der manuellen Tätigkeiten auf einfache Ablage- b.z.w. Freigabe-Aufgaben, ist das Risiko eines menschlichen Fehlers nahezu auszuschließen.

Gesamtdauer des Deployment

Das Aufbauen der Staging-Umgebung dauert 15 Minuten. Das Blue-Green Deployment benötigt 8 Minuten. Mit den Zeiten der anderen Phasen kommt man beim ersten Deployment auf eine Durchlaufzeit von 25 Minuten.
Ist die Staging-Umgebung jedoch einmal aufgebaut, wird diese Phase in der Pipeline übersprungen und die Zeit zum Produktiv-Deployment nimmt drastisch ab. Alle weiteren Deployments sind dann nach ca. 10 Minuten in Produktion.[10]

Benötigtes Wissen

Die Entwickler müssen sich nicht mehr mit den AWS-Diensten und seinen Ressourcen auskennen, sondern nur noch ihre Änderungen im VKS ablegen. Der PO muss sich einmal in der Konsole anmelden, um nach der Abnahme der Änderungen die Pipeline auszuwählen und auf *Freigabe* zu klicken. Das Wissen, welches zum Deployment benötigt wird, begrenzt sich also auf Fähigkeiten, welche die jeweiligen Personen schon besitzen bzw. die schnell erlernt werden können.
Die Person in der Rolle des DevOps-Verantwortlichen muss sich natürlich mit den in der vorliegenden Arbeit vorgestellten Diensten und Techniken auskennen.

Indikator	**mit DevOps**	**bisher** (s. Tabelle 3.1)
Anzahl manuelle Tätigkeiten	3	28
Anzahl an Risiken	0	3
Gesamtdauer des Deployment	10 Minuten	15 Minuten
Anzahl eingesetzter Software	2 (VKS & AWS-Konsole)	6

Tabelle 5.1: Gegenüberstellung KPIs alter & neuer Deployment-Prozess

Die automatisierte Pipeline beschränkt die notwendigen Tätigkeiten der beim Deployment Beteiligten auf ein Minimum: das Ablegen des Codes und das Überprüfen und Freigeben der Änderungen. Dies macht jedes Deployment zu einer sicheren, alltäglichen Aufgabe, die ständig und unter denselben Bedingungen ausgeführt werden kann. Die Gesamtzeit wurde nicht nur um 5 Minuten verkürzt, die Entwickler können während des Deployments auch an

[10] Es wird davon ausgegangen, dass die eingespielten Änderungen fehlerfrei sind und sofort freigegeben werden.

anderen Aufgaben arbeiten. Durch die Verringerung des benötigten Wissens b.z.w. der eingesetzten Software, müssen sich Entwickler nicht erst lange in neue Umgebungen einarbeiten, sondern können früher mit der Arbeit an Kundenwünschen beginnen.

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Rahmen diese Bachelorarbeit wurde eine Continuous Integration / Continuous Delivery Umgebung für die Einführung von DevOps am Marketingportal ukv-verona.de geschaffen. Ziel dabei war es, Pipelines zu entwickeln und Techniken einzuführen, mit denen es möglich ist, neue Funktionen oder Änderungen automatisiert und somit schnell und sicher dem Kunden zur Verfügung zu stellen. Dazu wurde zunächst das bestehende Produkt mit seiner Infrastruktur und seinem aktuellen Deployment Prozess analysiert und vorgestellt.
Es wurden cloud-agnostische Anforderungen definiert, die eine transparente und nachvollziehbare sowie automatisierte Anwendungsentwicklung in der Cloud ermöglichen. Erreicht wurde dies durch die Nutzung eines Versionskontrollsystems, das eine für die Anwendung erstellte Code-Pipeline ausführt. Durch die Einführung von Infrastruktur als Code ist außerdem die für DevOps wichtige Transparenz der Infrastruktur gegeben.

Zum Schluss wurden die Anforderungen für die Amazon Web Services Cloud entworfen und umgesetzt. Mit der implementierten Lösung ist es den Entwicklern künftig möglich, ihre Code-Änderungene sicher in das produktive System und damit zum Kunden zu bringen. Statt jedes Deployment von Anfang bis Ende zu begleiten, um bei Fehlern schnell reagieren zu können, steht durch eine fast vollständige Automatisierung mehr Zeit für andere Aufgaben, z.B. zum Entwickeln neuer Features, zur Verfügung.

6.2 Ausblick

Die Einführung von DevOps bzw. Continuous Integration/Continuous Delivery ist ein Prozess, der für sich selbst kontinuierliche Weiterentwicklung benötigt. Die hier umgesetzten Pipelines können mit automatisierten Testphasen erweitert werden. Durch die Einführung von Unit Test oder CloudFormation Validierungs-Tools, die in der Build-Phase ausgeführt werden, lässt sich die Qualität des Deployment-Prozesses weiter verbessern. Mit zunehmender Automatisierung und dem Ersetzen von manuellen Tests durch automatisierte Skripte, würde sich Continuous Delivery zu Continuous Deployment erweitern lassen.
Durch die ständige Weiterentwicklung der Cloud Technologien, werden neue Dienste bereitgestellt, die bei Bedarf und Notwendigkeit zur Optimierung in die Pipeline implementiert werden können.
Um im Sinne von DevOps mehr Monitoring und damit Transparenz über seine Cloud Umgebung zu bekommen, könnten die in den einzelnen Phasen der Pipeline geloggten Metriken zu einem Dashboard zusammengefasst werden, um eventuelle Änderungen am Deployment-Prozess frühzeitig festzustellen.

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Glossar

ALB Ein Load Balancer dient als zentraler Kontaktpunkt für Clients. Er verteilt eingehenden Anwendungsdatenverkehr automatisch auf mehrere Ziele wie Amazon EC2-Instances, Container, IP-Adressen und Lambda-Funktionen.[1]. 24, 29

Auto Scaling-Gruppe Eine Auto Scaling-Gruppe enthält eine Sammlung von Amazon EC2-Instances, die als logische Gruppierung zum Zweck der automatischen Skalierung und Verwaltung behandelt werden.[2]. 18

EC2 Der Web-Service Amazon Elastic Compute Cloud (Amazon EC2) stellt sichere, skalierbare Rechenkapazitäten in der Cloud bereit. Der Service ist darauf ausgelegt, Cloud Computing für Entwickler zu erleichtern.[3]. 24

Kollaboration Gemeinsam vernetzte Arbeit. 6

Lambda AWS Lambda ist ein Datenverarbeitungsservice, mit dem man Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. AWS Lambda führt Code nur bei Bedarf aus und skaliert automatisch.[4]. 32

Route 53 Amazon Route 53 ist ein hochverfügbarer und skalierbarer Domain Name System (DNS)-Web-Service für die Cloud.[5]. 24

[1] Amazon Web Services. (2019h). Application Load. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/elasticloadbalancing/.

[2] Amazon Web Services. (2019i). Auto Scaling-Gruppen. Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/de_de/autoscaling/ec2/userguide/AutoScalingGroup.html.

[3] Amazon Web Services. (2019j). Amazon EC2. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/ec2/.

[4] Amazon Web Services. (2019k). Was ist AWS Lambda? Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/de_de/lambda/latest/dg/welcome.html.

[5] Amazon Web Services. (2019l). Amazon Route 53. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/route53/.

S3-Bucket Amazon Simple Storage Service (Amazon S3) ist ein Objektspeicherservice, der Skalierbarkeit, Datenverfügbarkeit, Sicherheit und Leistung bietet.[6]. 26

[6]Amazon Web Services. (2019m). Amazon S3. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/s3/.

Abkürzungsverzeichnis

ALB Application Load Balancer. 24, *Glossar:* ALB

ASG Auto Scaling-Gruppe. I, 18, 24, 28–30, *Glossar:* Auto Scaling-Gruppe

Aurora Amazon Aurora (Aurora). *Glossar:* Amazon Aurora RDS

AWS Amazon Web Services. 3, 14, 23, 25, 28–30, 33, 35

CD Continuous Delivery. 1, 2, 10–12, 25, 30, 34

CI Continuous Integration. 1, 2, 10, 19, 25, 27, 30

CMS Content Management System. 14

EC2 Elastic Compute Cloud. 24, *Glossar:* EC2

EFS Elastic File System. *Glossar:* Elastic File System

IaC Infrastruktur als Code. 8–10, 20, 21, 23

KPI Key Performance Indicators. 2, 16, 34

Launch Configuration Launch Configuration. 24, *Glossar:* Route 53

NIST National Institute of Standards and Technology. 4, 5

PO Product Owner. 21, 33, 35

Route 53 Route 53. *Glossar:* Route 53

S3-Bucket Simple Storage Service Bucket. 26, 31, 32, *Glossar:* S3-Bucket

VKS Versionskontrollsystem. 5–10, 17, 19, 20, 25, 31, 34, 35

Literaturverzeichnis

Wolff, E. (2016). *Continuous delivery: der pragmatische Einstieg*. dpunkt. verlag.

Mell, P., Grance, T. et al. (2011). The NIST definition of cloud computing. Zugriff unter https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Davis, J. & Daniels, R. (2016). *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. O'Reilly Media, Inc.

GitHub, Inc. (2019). Git. Zugriff 30. Juli 2019 unter https://github.com/git/git

Kim, G., Humble, J., Debois, P. & Willis, J. (2017). *Das DevOps-Handbuch: Teams, Tools und Infrastrukturen erfolgreich umgestalten*. O'Reilly Media, Inc.

HP, Enterprise Development. (2019). Was ist Infrastructure as Code? Zugriff 6. Juni 2019 unter https://www.hpe.com/de/de/what-is/infrastructure-as-code.html

Amazon Web Services. (2019a). AWS CloudFormation. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/cloudformation/

Amazon Web Services. (2019b). AWS CodePipeline. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codepipeline/

Amazon Web Services. (2019c). AWS CodeCommit. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codecommit/

Amazon Web Services. (2019d). AWS CodeBuild. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codebuild/

Amazon Web Services. (2019e). AWS::CloudFormation::Stack. Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-stack.html

Amazon Web Services. (2019f). CodeDeploy AppSpec File Reference. Zugriff 18. Juli 2019 unter https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file.html

Amazon Web Services. (2019g). AWS CodeDeploy. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/codedeploy/

Amazon Web Services. (2019h). Application Load. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/elasticloadbalancing/

Amazon Web Services. (2019i). Auto Scaling-Gruppen. Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/de_de/autoscaling/ec2/userguide/AutoScalingGroup.html

Amazon Web Services. (2019j). Amazon EC2. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/ec2/

Amazon Web Services. (2019k). Was ist AWS Lambda? Zugriff 30. Juli 2019 unter https://docs.aws.amazon.com/de_de/lambda/latest/dg/welcome.html

Amazon Web Services. (2019l). Amazon Route 53. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/route53/

Amazon Web Services. (2019m). Amazon S3. Zugriff 30. Juli 2019 unter https://aws.amazon.com/de/s3/

Anhang A

A.1 Diagramme

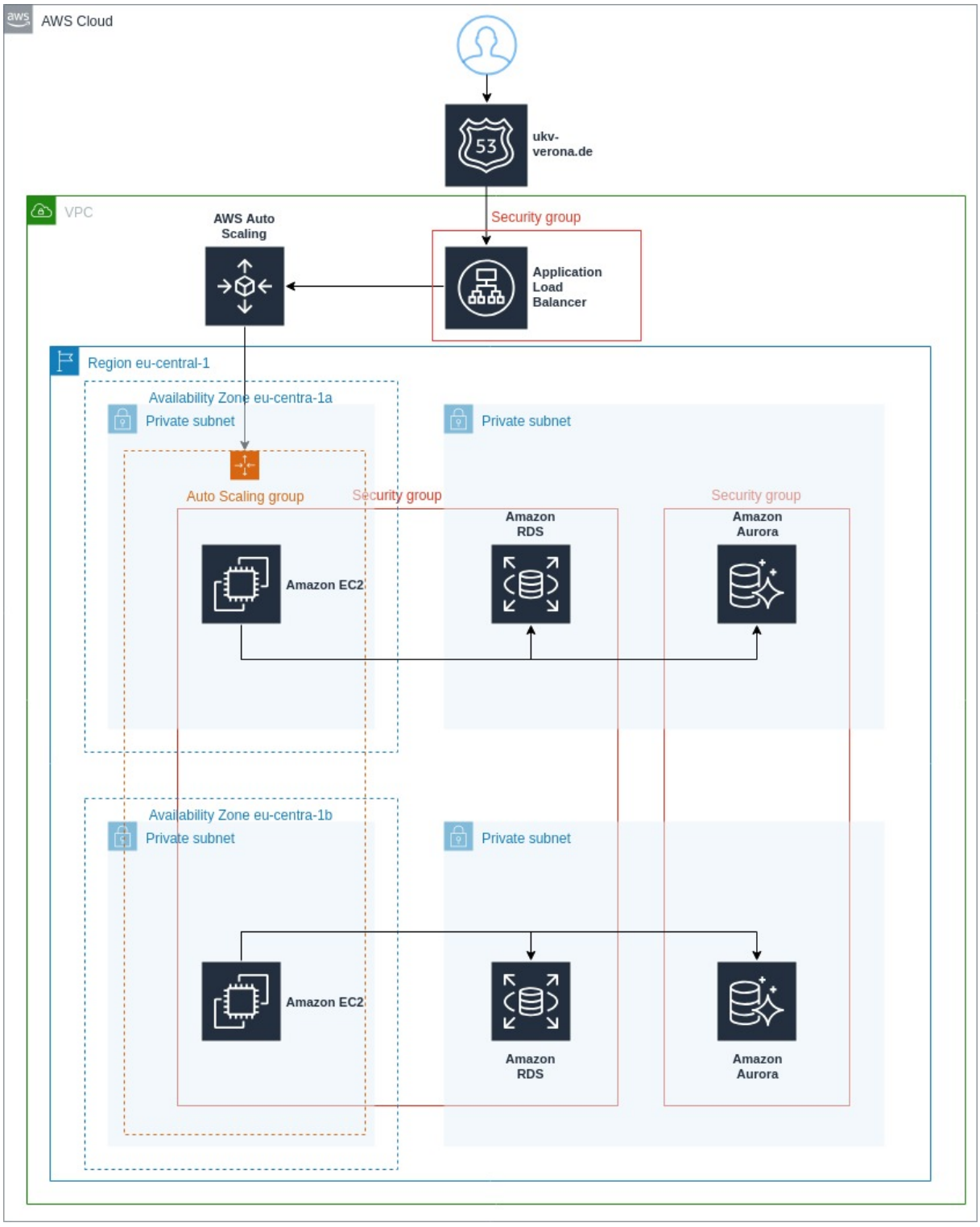

Abbildung A.1: Infrastruktur Diagramm ukv-verona.de

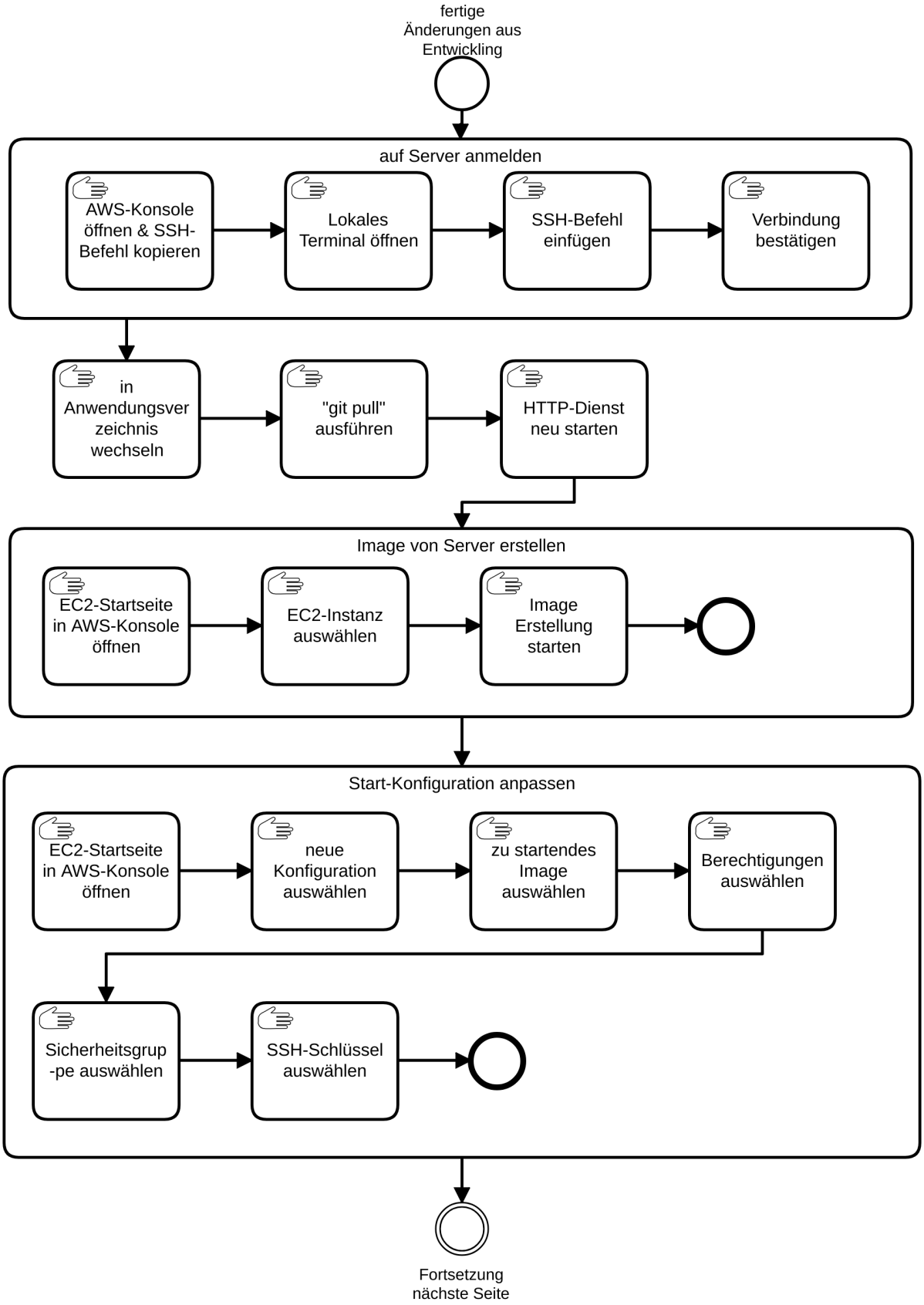

Abbildung A.2: Ablaufplan Deployment-Prozess – 1

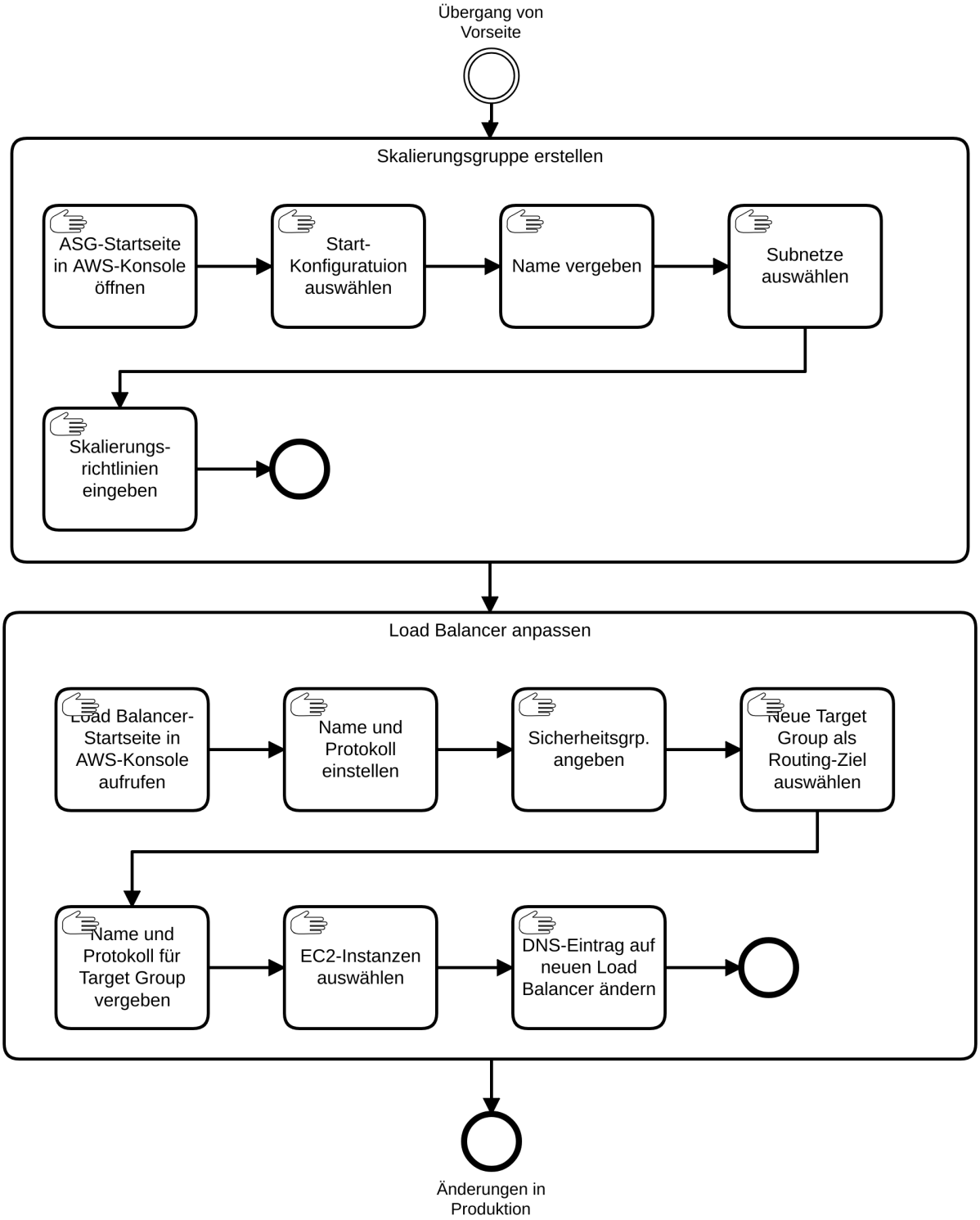

Abbildung A.3: Ablaufplan Deployment-Prozess – 2

A.2 Code-Auszüge

```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'verona staging environment'
Resources:
    Network:
        Type: AWS::CloudFormation::Stack
        Properties:
            TemplateURL: ./network.yml
            TimeoutInMinutes: 30
    Application:
        Type: AWS::CloudFormation::Stack
        DependsOn: Network
        Properties:
            TemplateURL: ./app.yml
            TimeoutInMinutes: 60
Outputs:
    StackName:
      Description: The name of the Stack
      Value: !Sub '${AWS::StackName}'
      Export:
        Name: verona-staging-master
```

Code-Auszug A.1: master.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Description: verona staging environment
Metadata: {}
Mappings: {}
Conditions: {}
Resources:
  Network:
    Type: AWS::CloudFormation::Stack
    Properties:
```

```
10        TemplateURL: https://s3.amazonaws.com/verona-infrastruktur-bucket/
11        e3dfbd38b2c9ba51ca7599e202b689cf.template
12        TimeoutInMinutes: 30
13      Application:
14        Type: AWS::CloudFormation::Stack
15        DependsOn: Network
16        Properties:
17          TemplateURL: https://s3.amazonaws.com/verona-infrastruktur-bucket/
18          4d0c6a23dfbda56dbe8ec6df6394b6e6.template
19          TimeoutInMinutes: 60
20    Outputs:
21      StackName:
22        Description: The name of the Stack
23        Value:
24          Fn::Sub: ${AWS::StackName}
25        Export:
26          Name: VeronaIntegrationMaster
```

Code-Auszug A.2: master.yml, nach dem der Befehl *cloudformation package* ausgeführt wurde

```
1     AWSTemplateFormatVersion: 2010-09-09
2     Description: deploys a basic VPC / Network.
3     Resources:
4       VPC:
5         Type: 'AWS::EC2::VPC'
6         Properties:
7           CidrBlock: 192.168.0.0/16
8           EnableDnsHostnames: true
9           EnableDnsSupport: true
10          Tags:
11            - Key: Name
12              Value: !Sub '${AWS::StackName}-VPC'
13            - Key: project
14              Value: verona
```

```
15      InternetGateway:
16        Type: 'AWS::EC2::InternetGateway'
17        DependsOn: VPC
18        Properties:
19          Tags:
20            - Key: Name
21              Value: !Sub '${AWS::StackName}-IGW'
22            - Key: project
23              Value: verona
24      AttachGateway:
25        Type: 'AWS::EC2::VPCGatewayAttachment'
26        Properties:
27          VpcId: !Ref VPC
28          InternetGatewayId: !Ref InternetGateway
29      PublicSubnetA:
30        Type: 'AWS::EC2::Subnet'
31        Properties:
32          AvailabilityZone: !Select
33            - 0
34            - !GetAZs ''
35          VpcId: !Ref VPC
36          CidrBlock: 192.168.1.0/24
37          Tags:
38            - Key: Name
39              Value: !Sub '${AWS::StackName}-PublicSubnetA'
40            - Key: project
41              Value: verona
42      PublicSubnetB:
43        Type: AWS::EC2::Subnet
44        Properties:
45          AvailabilityZone: !Select
46            - 1
47            - !GetAZs ''
48          VpcId: !Ref VPC
49          CidrBlock: 192.168.2.0/24
50          Tags:
```

```
51            - Key: Name
52              Value: !Sub '${AWS::StackName}-PublicSubnetB'
53            - Key: project
54              Value: verona
55        PublicRouteTable:
56          Type: AWS::EC2::RouteTable
57          Properties:
58            VpcId: !Ref VPC
59            Tags:
60              - Key: Name
61                Value: !Sub '${AWS::StackName}-RouteTable'
62              - Key: project
63                Value: verona
64        PublicRoute:
65          Type: AWS::EC2::Route
66          DependsOn: AttachGateway
67          Properties:
68            RouteTableId: !Ref PublicRouteTable
69            DestinationCidrBlock: 0.0.0.0/0
70            GatewayId: !Ref InternetGateway
71        NATGateway:
72          Type: AWS::EC2::NatGateway
73          Properties:
74            AllocationId: !GetAtt ElasticIPAddress.AllocationId
75            SubnetId: !Ref PublicSubnetA
76            Tags:
77              - Key: Name
78                Value: !Sub '${AWS::StackName}-NATGateway'
79              - Key: project
80                Value: verona
81        ElasticIPAddress:
82          Type: AWS::EC2::EIP
83          Properties:
84            Domain: VPC
85        PublicSubnetARouteTableAssociation:
86          Type: AWS::EC2::SubnetRouteTableAssociation
```

```
87        Properties:
88          SubnetId: !Ref PublicSubnetA
89          RouteTableId: !Ref PublicRouteTable
90      SecurityGroup:
91        Type: AWS::EC2::SecurityGroup
92        Properties:
93          GroupName: !Sub '${AWS::StackName}-SecurityGroup'
94          GroupDescription: !Sub '${AWS::StackName}-SecurityGroup'
95          VpcId: !Ref VPC
96          SecurityGroupIngress:
97            - IpProtocol: tcp
98              FromPort: 80
99              ToPort: 80
100             CidrIp: 0.0.0.0/0
101           - IpProtocol: tcp
102             FromPort: 443
103             ToPort: 443
104             CidrIp: 0.0.0.0/0
105           - IpProtocol: tcp
106             FromPort: 22
107             ToPort: 22
108             CidrIp: 0.0.0.0/0
109           - IpProtocol: "-1"
110             CidrIp: 0.0.0.0/0
111         Tags:
112           - Key: Name
113             Value: !Sub '${AWS::StackName}-SecurityGroup'
114           - Key: project
115             Value: verona
116   Outputs:
117     PublicSubnetA:
118       Description: Public Subnet 192.168.0.0/24
119       Value: !Ref PublicSubnetA
120       Export:
121         Name: PublicSubnetA
122     PublicSubnetB:
```

```
123         Description: Public Subnet 172.16.0.0/12
124         Value: !Ref PublicSubnetB
125         Export:
126           Name: PublicSubnetB
127       VPC:
128         Description: VPC
129         Value: !Ref VPC
130         Export:
131           Name: IntegrationVPC
132       SecurityGroup:
133         Description: Information about the value
134         Value: !Ref SecurityGroup
135         Export:
136           Name: SecurityGroup
137       StackName:
138         Description: Name of the Stack
139         Value: !Sub ${AWS::StackName}
140         Export:
141           Name: NetworkStack
```

Code-Auszug A.3: network.yml

```
1     AWSTemplateFormatVersion: 2010-09-09
2     Description: deploys EC2, ASG, LB, etc.
3     Parameters:
4       latestAmi:
5         Description: ID of latest ami
6         Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
7         Default: latestAmi
8       KeyName:
9         Description: SSH Key
10        Type: String
11        Default: verona
12      InstanceType:
13        Description: Instance Type
```

```
14        Type: String
15        Default: t2.micro
16      EfsId:
17        Description: EfsId
18        Type: String
19        Default: efsid
20    Resources:
21      myEC2Instance:
22        Type: AWS::EC2::Instance
23        #DependsOn: ElasticFileSystem
24        Properties:
25          KeyName: !Ref KeyName
26          ImageId: !Ref latestAmi
27          InstanceType: !Ref InstanceType
28          Monitoring: true
29          IamInstanceProfile: CodeDeployInstanceRole
30          NetworkInterfaces:
31            - AssociatePublicIpAddress: true
32              DeleteOnTermination: true
33              Description: Verona Integration NI
34              DeviceIndex: '0'
35              SubnetId: !ImportValue PublicSubnetA
36              GroupSet:
37                - !ImportValue SecurityGroup
38          Tags:
39            - Key: Name
40              Value: verona-integration
41            - Key: project
42              Value: verona
43      ElasticFileSystem:
44        Type: AWS::EFS::FileSystem
45        Properties:
46          Encrypted: false
47          FileSystemTags:
48            - Key: Name
49              Value: verona-integration-efs
```

```
50          PerformanceMode: generalPurpose
51          ThroughputMode: bursting
52        MountTarget:
53          Type: AWS::EFS::MountTarget
54          Properties :
55            FileSystemId : !Ref ElasticFileSystem
56            SecurityGroups :
57              - !ImportValue SecurityGroup
58            SubnetId : !ImportValue PublicSubnetA
59        DatabaseSubnetGroup:
60          Type: AWS::RDS::DBSubnetGroup
61          Properties:
62            DBSubnetGroupDescription: CloudFormation managed DB subnet group.
63            SubnetIds:
64              - !ImportValue PublicSubnetA
65              - !ImportValue PublicSubnetB
66        AuroraDBCluster:
67          Type: AWS::RDS::DBCluster
68          Properties:
69            #AvailabilityZones:
70            # - !Select
71            # - 1
72            # - !GetAZs ''
73            Engine: aurora
74            DBClusterIdentifier: verona-integration
75            EngineMode: serverless
76            DBSubnetGroupName: !Ref DatabaseSubnetGroup
77            DBClusterParameterGroupName: default.aurora5.6
78            SnapshotIdentifier: rds:veronadb-2019-06-02-03-14
79            Tags:
80              - Key: Name
81                Value: verona-integration-db
82              - Key: project
83                Value: verona
84            VpcSecurityGroupIds:
```

```
85            - !ImportValue SecurityGroup
```

Code-Auszug A.4: app.yml

```
1  version: 0.2
2  phases:
3    install:
4      commands:
5        - aws cloudformation package --template master.yml --s3-bucket ↩
           verona-infrastruktur-bucket --output-template-file master.yml
6  artifacts:
7    type: zip
8    files:
9      - master.yml
```

Code-Auszug A.5: buildspec.yml für Build-Projekt verona-cf-build

```
1  version: 0.2
2  phases:
3    install:
4      commands:
5        - mv -f scripts/prod/appspec.yml /appspec.yml
6  artifacts:
7    type: zip
8    files:
9      - '**/*'
```

Code-Auszug A.6: buildspec.yml für Build-Projekt verona-prod-build

```
1    version: 0.0
2    os: linux
3    files:
4      - source: /
```

```
5     destination: /var/www/html
6   hooks:
7     ApplicationStop:
8     - location: scripts/stop_server.sh
9       timeout: 300
10      runas: root
11    BeforeInstall:
12    - location: scripts/cleanup_repo.sh
13      timeout: 300
14      runas: root
15    AfterInstall:
16    - location: scripts/set_staging_env.sh
17      timeout: 300
18      runas: root
19    ApplicationStart:
20    - location: scripts/start_server.sh
21      timeout: 300
22      runas: root
```

Code-Auszug A.7: appspec.yml

```
1   #!/bin/bash
2   service httpd stop
```

Code-Auszug A.8: stop-server.sh

```
1   #!/bin/bash
2   mkdir /var/www/html/wp-content/uploads
3   mount -t efs fs-1cd3f545.efs.eu-central-1.amazonaws.com ↩
        /var/www/html/wp-content/uploads
4   service httpd start
```

Code-Auszug A.9: start-server.sh

```
echo "<?php
define('ENVIRONMENT', 'stage');
//define('ENVIRONMENT', 'dev');
//define('ENVIRONMENT', 'dev2');" > /var/www/html/environment.php
```

Code-Auszug A.10: set-staging-env.yml

```
#!/bin/bash
umount /var/www/html/wp-content/uploads
rm -rf /var/www/html
```

Code-Auszug A.11: cleanup-repo.sh

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
    LoadBalancer:
        Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
        Properties:
            Scheme: internet-facing
            IpAddressType: ipv4
            Subnets:
                - subnet-8ca4e4e7
                - subnet-9bff47e6
            SecurityGroups:
                - sg-c6fe80ab
        Metadata:
            'AWS::CloudFormation::Designer':
                id: e02e067d-9c5b-41d4-a25f-1e1fda3069ec
    TargetGroup:
        Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
        Properties:
            Name: VeronaTG
            HealthCheckIntervalSeconds: 30
            HealthCheckProtocol: HTTP
```

```
22              HealthCheckTimeoutSeconds: 5
23              HealthyThresholdCount: 3
24              UnhealthyThresholdCount: 2
25              Matcher:
26                  HttpCode: '200,403,301,302'
27              Port: 80
28              Protocol: HTTP
29              TargetGroupAttributes:
30                  -
31                      Key: deregistration_delay.timeout_seconds
32                      Value: '100'
33              VpcId: vpc-90ff7efb
34              TargetType: instance
35          Metadata:
36              'AWS::CloudFormation::Designer':
37                  id: 16172071-362a-4ace-8206-b73a33bfaf89
38      ListenerHTTPS:
39          Type: 'AWS::ElasticLoadBalancingV2::Listener'
40          Properties:
41              DefaultActions:
42                  -
43                      Type: forward
44                      TargetGroupArn:
45                          Ref: TargetGroup
46              LoadBalancerArn:
47                  Ref: LoadBalancer
48              Port: 443
49              Protocol: HTTPS
50              SslPolicy: ELBSecurityPolicy-2016-08
51              Certificates:
52                  -
53                      CertificateArn:
54                      'arn:aws:acm:eu-central-1:240450801016:certificate/
55                      5fa54f5b-c067-4c90-a33c-5bb34584142d'
56          Metadata:
57              'AWS::CloudFormation::Designer':
```

```
58                  id: 53a12a5a-716e-46c8-bb83-4f068e5c7e1c
59      ListenerHTTP:
60          Type: 'AWS::ElasticLoadBalancingV2::Listener'
61          Properties:
62              DefaultActions:
63                  -
64                      Type: redirect
65                      RedirectConfig:
66                          Port: '443'
67                          Host: www.ukv-verona.de
68                          Path: '/#{path}'
69                          Query: '#{query}'
70                          Protocol: HTTPS
71                          StatusCode: HTTP_301
72              LoadBalancerArn:
73                  Ref: LoadBalancer
74              Port: 80
75              Protocol: HTTP
76          Metadata:
77              'AWS::CloudFormation::Designer':
78                  id: 34665d43-7bca-44c0-a63e-ac7762edd0d2
79      AutoScalingGroup:
80          Type: 'AWS::AutoScaling::AutoScalingGroup'
81          Properties:
82              LaunchConfigurationName:
83                  Ref: AutoScalingLC
84              AvailabilityZones:
85                  - eu-central-1a
86                  - eu-central-1b
87              VPCZoneIdentifier:
88                  - subnet-9bff47e6
89                  - subnet-8ca4e4e7
90              Tags:
91                  -
92                      Key: Name
93                      Value: Verona
```

```
94                  PropagateAtLaunch: true
95              HealthCheckGracePeriod: 10
96              Cooldown: '60'
97              MetricsCollection:
98                  -
99                      Granularity: 1Minute
100                     Metrics:
101                         - GroupPendingInstances
102                         - GroupTotalInstances
103                         - GroupDesiredCapacity
104                         - GroupMinSize
105                         - GroupTerminatingInstances
106                         - GroupMaxSize
107                         - GroupStandbyInstances
108                         - GroupInServiceInstances
109             DesiredCapacity: '2'
110             MaxSize: '4'
111             MinSize: '2'
112             TargetGroupARNs:
113                 -
114                     Ref: TargetGroup
115         UpdatePolicy:
116             AutoScalingReplacingUpdate:
117                 WillReplace: true
118         Metadata:
119             'AWS::CloudFormation::Designer':
120                 id: 02ac9fcd-3447-4e98-a978-f6ae5496e4e6
121     AutoScalingLC:
122         Type: 'AWS::AutoScaling::LaunchConfiguration'
123         Properties:
124             IamInstanceProfile: CodeDeployInstanceRole
125             InstanceType: t3.medium
126             ImageId:
127                 Ref: LatestAmi
128             KeyName: verona
129             InstanceMonitoring: true
```

```
130             SecurityGroups:
131                 - sg-2800b045
132         Metadata:
133             'AWS::CloudFormation::Designer':
134                 id: d6948c2d-78d6-4ece-b9b5-b319020ec58b
135     ScalingPolicy:
136         Type: 'AWS::AutoScaling::ScalingPolicy'
137         Properties:
138             AutoScalingGroupName:
139                 Ref: AutoScalingGroup
140             PolicyType: TargetTrackingScaling
141             TargetTrackingConfiguration:
142                 PredefinedMetricSpecification:
143                     PredefinedMetricType: ASGAverageCPUUtilization
144                 TargetValue: 30
145             EstimatedInstanceWarmup: 90
146         Metadata:
147             'AWS::CloudFormation::Designer':
148                 id: 9fecebda-26d2-48b1-9948-a7138d23e0b2
149     route53HealthCheck:
150       Type: AWS::Route53::HealthCheck
151       Properties:
152         HealthCheckConfig:
153           Port: 443
154           Type: HTTPS
155           ResourcePath: '/'
156           FullyQualifiedDomainName: 'ukv-verona.de'
157           RequestInterval: 30
158           FailureThreshold: 3
159           Regions:
160             - eu-west-1
161             - us-west-1
162             - us-west-2
163         HealthCheckTags:
164           - Key: "project"
165             Value: "verona"
```

```
166             - Key: "Name"
167               Value: "ukv-verona.de"
168   Parameters:
169       LatestAmi:
170           Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
171           Default: latestAmi
```

Code-Auszug A.12: verona-prod-infra.yml

```
1  def lambda_handler(event, context):
2      client = boto3.client('rds')
3      response = client.describe_db_cluster_snapshots(
4      DBClusterIdentifier='veronadb',
5      SnapshotType='automated',
6      )
7      for snapshots in response['DBClusterSnapshots']:
8          dbsnapshot = snapshots['DBClusterSnapshotIdentifier']
9      print (dbsnapshot)
10     client = boto3.client('ssm')
11     parameter = client.put_parameter(
12     Name='verona-staging-db-snapshot',
13     Description='The value of the latest veronadb snapshot to use in the ↩
           staging cloudformation template',
14     Value=dbsnapshot,
15     Type='String',
16     Overwrite=True,
17     )
18     print (event)
19     pipeline = boto3.client('codepipeline')
20     response = pipeline.put_job_success_result(
21         jobId=event['CodePipeline.job']['id']
22     )
23     print (response)
24     return response
```

Code-Auszug A.13: lambda.py

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 05.08.2019

Janek Hornung