

Bachelorarbeit
**Cloud Service Provider Evaluierung
auf Basis von
Infrastructure as Code Unterstützung**

im Studiengang Softwaretechnik und Medieninformatik
der Fakultät Informationstechnik
Wintersemester 2021/22

Julian Schallenmüller

Zeitraum: 15.10.2021-31.01.2022

Prüfer: Prof. Dr.-Ing. Kai Warendorf

Zweitprüfer: Prof. Dr. Rer. Nat. Mirko Sonntag

Firma: Novatec Consulting GmbH

Betreuer: Dipl.-Ing (BA) Matthias Häussler



Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 26. Januar 2022 _____
Unterschrift

Sperrvermerk

Die nachfolgende Bachelorarbeit enthält vertrauliche Daten der Novatec Consulting GmbH. Veröffentlichungen oder Vervielfältigungen dieser Arbeit – auch nur auszugsweise – sind ohne ausdrückliche Genehmigung der Novatec Consulting GmbH nicht gestattet. Diese Arbeit ist nur den Prüfern sowie den Mitgliedern des Prüfungsausschusses zugänglich zu machen.

Zitat

„Showing a strong success and visible benefits is key to getting others to agree to try your way of doing things.“

- Frederic Rivain

Vorwort

Ich möchte mich an dieser Stelle bei der Firma Novatec bedanken, in der ich seit meiner Zeit als Praktikant, danach als Werkstudent und nun auch während meiner Bachelorarbeit immer willkommen war und bei allen Herausforderungen und Problemen stets unterstützt wurde.

Besonderer Dank geht hierbei an die Mitarbeiter der PA TC und insbesondere an meinen Betreuer Matthias Häussler. Herr Häussler stand bereits vor und vor allem während meiner Zeit als Bachelorant immer für Fragen und Ratschläge, auch über die normalen Arbeitszeiten hinaus, bereit.

Genauso möchte ich mich bei Herrn Prof. Dr.-Ing Warendorf für die Betreuung meiner Bachelorarbeit von Seiten der Hochschule Esslingen bedanken. Durch seine schnelle und unkomplizierte Art der Kommunikation konnten alle organisatorischen Fragen und Aufgaben rund um die Bachelorarbeit immer schnell beantwortet und bewältigt werden.

Kurz-Zusammenfassung

Im Rahmen dieser Arbeit wird ein Vergleich der Cloud Service Provider Microsoft und Google in Bezug auf deren Unterstützung von Infrastructure as Code mit Terraform durchgeführt.

Dieser Vergleich soll die Qualität der Unterstützung von deren Plattformen Microsoft Azure und Google Cloud Platform in einem gewöhnlichen Szenario evaluieren und die Frage nach der Einsatzreife beider Plattformen beantworten.

Zusätzlich soll diese Arbeit ermitteln, wie einheitlich die konkrete Umsetzung von Infrastructure as Code mit Terraform in einem funktional gleichwertigen Szenario für verschiedene Plattformen ausfällt.

Für die Beantwortung dieser Fragen wurde ein beispielhaftes Infrastruktur-System auf beiden Plattformen mithilfe von Terraform implementiert und deployed. Die in der Evaluierung eingesetzten Bewertungskriterien wurden aus den Kriterien des Softwarequalitätsmodells der ISO/IEC 25010 ausgewählt.

Zusätzlich zu der Untersuchung des Testsystems wurde eine Reihe unterstützender Versuche durchgeführt, durch die ein Teil der aufgeworfenen Fragen beantwortet und ein vollständigeres Bild bezüglich der Performance, Qualität und Unterschiede der untersuchten Plattformen aufgebaut werden konnte.

Das Ergebnis der Untersuchung lässt darauf schließen, dass beide Plattformen eine gute Unterstützung von Terraform bieten, Azure genießt hierbei einen leichten Vorsprung hinsichtlich Funktionsumfang und Performance.

Die Einheitlichkeit der Umsetzung fällt jedoch eher gering aus. Sie begrenzt sich auf die Anwendung einer einheitlichen Sprache und Bedienung durch Terraform, es ist jedoch weiterhin ein umfangreiches Verständnis der individuellen Cloud Plattform notwendig, um diese erfolgreich einsetzen zu können.

Dennoch ist es sinnvoll, Terraform für das Deployment von Infrastruktur zu nutzen, durch die Umsetzung der Prinzipien von Infrastructure as Code werden die Vorteile moderner Cloud Plattformen in vollem Umfang ausgeschöpft und die Wertschöpfungskette von der Idee zur Auslieferung an Kunden kann deutlich beschleunigt werden.

Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen	3
2.1	Funktionsprinzip, Vorteile und Herausforderungen	3
2.1.1	Definition und Funktionsweise	3
2.1.2	Vor- und Nachteile des Einsatzes von Cloud Computing	6
2.1.3	Überblick über die wichtigsten Cloud Service Provider	8
2.2	Infrastructure as Code	10
2.2.1	Technologischer Wandel und das Cloud Age Mindset	11
2.2.2	Vorteile von Infrastructure as Code im Vergleich zu manuellem Infrastruktur-Provisioning	12
2.2.3	Herausforderungen und Argumente gegen den Einsatz von Infrastructure as Code	14
2.2.4	Die drei Kernverfahren von Infrastructure as Code	16
2.3	Funktionsprinzip und Rolle von Terraform im IaC-Anwendungsprozess	16
2.3.1	Überblick über Infrastructure as Code Tools	17
2.3.2	Funktionsweise von Terraform	18
2.3.3	Einführung in die Hashicorp Configuration Language	21
3	Evaluierungsanforderungen und Umsetzung	25
3.1	Evaluierungsanforderungen	25
3.1.1	Ziel der Evaluierung	25
3.1.2	Untersuchte Komponenten der Terraform Provider	25
3.1.3	Auswahl der Evaluierungskriterien	25
3.2	Umsetzung des Testsystems	25
3.2.1	Eingesetzte Software und Tools	25
3.2.2	High-Level Aufbau des Testsystems	25
3.2.3	Konkreter Aufbau auf Google Cloud Platform	25
3.2.4	Konkreter Aufbau auf Azure	25
3.2.5	Aufbau der ergänzenden Versuche	25
4	Ergebnisse und Bewertung	26
4.1	Evaluierung der Functional Completeness	27
4.2	Ergebnisse und Bewertung der Time Behaviour Tests	28
4.3	Ergebnisse und Bewertung der Recoverability Tests	29

4.4	Ergebnisse und Bewertung der Modifiability Tests	29
4.5	Evaluierung der Einheitlichkeit der Testsysteme für Azure und Google Cloud Platform	29
5	Schluss	30
5.1	Fazit	30
5.2	Ausblick	30
A	Kapitel im Anhang	31
	Literaturverzeichnis	32

Abbildungsverzeichnis

2.1	Die Cloud Service Modelle im Überblick[3]	5
2.2	CSP Market Share Q2 2021 nach Umsatz[14]	9
2.3	CSP Gartner Magic Quadrant Juli 2021[15]	10
2.4	Verhältnis von Geschwindigkeit und Qualität[24]	15
2.5	Überblick IaC Tools[27]	17
2.6	Terraform Funktionsprinzip[28]	19
2.7	Terraform Workflow[32]	20

Tabellenverzeichnis




2.1	„Iron vs Cloud Age“ [17]	11
4.1	Anaylse der Functional Completeness	27
4.2	Ergebnisse der Time Behaviour Tests (1/2)	28
4.3	Ergebnisse der Time Behaviour Tests (2/2)	28
4.4	Ergebnisse der Recoverability Tests	29
4.5	Ergebnisse der Modifiability Tests	29



Listings

2.1	Grundlegende Elemente von HCL	21
2.2	Beispiel Input-Variable	22
2.3	Beispiel Output-Variable	22
2.4	Beispiel Terraform Module	23
2.5	Beispiel Terraform Provider Konfiguration	23

1 Einleitung

Eines der wichtigsten Schlagworte im Zeitalter der fortschreitenden Digitalisierung ist der Begriff des Cloud Computings. Cloud Computing spielt heute längst nicht mehr nur in der IT-Industrie eine wichtige Rolle. Selbst in Bereichen wie der Finanzbranche, die besonders hohen Sicherheitsansprüchen gerecht werden muss, findet Cloud-basierte Software eine zunehmende Verbreitung [1].

Die Nutzung von Cloud Technologien verspricht die Möglichkeit  schneller auf Anforderungen von Kunden reagieren zu können, kostengünstige und flexible Skalierung der eigenen Rechenkapazitäten, Einsparungen durch den Wegfall eigener IT-Infrastruktur-Fachleute und mehr. Gemeinsam mit der Eröffnung neuer Möglichkeiten  bringt die Einführung neuer Technologie jedoch auch immer eine Reihe eigener Herausforderungen mit sich. Für den erfolgreichen und gewinnbringenden Einsatz dieser Technologie ist es daher essentiell  diese zu verstehen und die neuen Herausforderungen mit angepasster Denkweise und neuen Werkzeugen anzugehen.

Das Thema mit dem sich diese Arbeit befassen wird  ist das automatisierte Managen und Bereitstellen von IT-Infrastruktur-Ressourcen, ein Teil der größeren Fachthematik Infrastructure as Code (IaC). Die Grundlagenkapitel werden zu diesem Zweck auf den technischen Kontext und die Relevanz von Cloud Computing, IaC und das Software Tool Terraform eingehen. Es wird erläutert werden  in welcher Stelle die entsprechenden Plattformen und Software zum Einsatz kommen, welche Probleme durch diese gelöst werden, wofür deren Vorteile und Grenzen liegen sowie welche Alternativen existieren und wo ergänzende Werkzeuge zum Einsatz kommen können.

Den Kern der Arbeit bildet ein Vergleich verschiedener Cloud Service Provider unter dem zentralen Kriterium derer Unterstützung von IaC mit Terraform. Zu diesem Zweck wurde ein Infrastruktur-System auf Basis des Infrastruktur-Showcase der Firma Novatec¹ für ver-

¹ Link: <https://github.com/NovatecConsulting/technologyconsulting-showcase-infrastructure>

schiedene Cloud Plattformen in Terraform Code implementiert und auf diesen deployed. Als Grundlage für den Vergleich dienen eine Auswahl von Aspekten aus dem Softwarequalitätsmodell der ISO/IEC 25010. Da jedoch nicht alle enthaltenen Qualitätsaspekte der Norm für diesen konkreten Vergleich geeignet sind, werden diese zunächst auf ihre Anwendbarkeit im vorliegenden Fall hin analysiert und bewertet, anschließend werden einige der relevantesten Kriterien ausgewählt und die Untersuchung anhand derer durchgeführt. Da sich durch Unterschiede zwischen den Cloud Plattformen, zum Beispiel bei Auswahl der Leistungsfähigkeit verwendeter Virtueller Maschinen, einige Auffälligkeiten ergeben, werden zusätzliche Tests definiert und durchgeführt, um ein insgesamt vollständigeres Bild liefern zu können.

Im Anschluss werden die Ergebnisse des Vergleichs bewertet und die daraus resultierenden Erkenntnisse zusammengefasst. Aus diesen Erkenntnissen kann dann ein Fazit gezogen werden, aus deren Bedeutung im Kontext des aktuellen Stands der Technik interpretiert und bewertet. Den Abschluss bildet ein Ausblick auf zusätzliche Themen, die als nächstes betrachtet werden sollten, wenn es darum geht Infrastructure as Code für das Infrastruktur-Management in einer realen Produktivumgebung einzusetzen. Diese Themen werden im Verlauf der Arbeit bereits angesprochen, können hier jedoch noch nicht in zufriedenstellendem Umfang betrachtet werden.

2 Grundlagen

2.1 Funktionsprinzip, Vorteile und Herausforderungen des modernen Cloud Computings

Um die Rolle von Infrastructure as Code und Terraform vollständig erläutern zu können, sollte zuerst das grundlegende Funktionsprinzip und die verschiedenen Service- und Bereitstellungs-Modelle moderner Cloud Plattformen erklärt werden. Die am meisten verwendete Definition von Cloud Computing wurde vom National Institute of Standards and Technology der Vereinigten Staaten von Amerika veröffentlicht und wird im folgenden Kapitel zusammengefasst.

2.1.1 Definition und Funktionsweise

Das National Institute of Standards and Technology (NIST) der Vereinigten Staaten von Amerika definiert Cloud Computing im Abstract der NIST SP-800-145[2] folgendermaßen:

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

Cloud Computing beschreibt ein Modell, das es ermöglicht ortsunabhängig, zweckdienlich und zeitunabhängig auf einen konfigurierbaren Pool an Computing Ressourcen (Netzwerke, Server, Datenspeicher, Anwendungen und Services) zuzugreifen, die schnell und mit

minimalem Aufwand und minimaler notwendiger Interaktion bereitgestellt und wieder abgebaut werden können. Dieses Cloud Modell beschreibt fünf essentielle Charakteristiken, drei Servicemodelle und vier Bereitstellungsmodelle.

Weiter definiert das Dokument die fünf Charakteristiken in den folgenden Punkten:

On-demand-self-service: Der Nutzer kann eigenmächtig die benötigten Computing Ressourcen automatisch bereitstellen, es wird keine menschliche Interaktion benötigt.

Broad network access: Auf Leistungen wird über das normale Internet mit standardmäßigen Mechanismen wie der Nutzung von Thin Clients und Fat Clients (Smartphones, Tablets, Laptops oder Workstations) zugegriffen.

Resource Pooling: Die Computing Ressourcen des Anbieters werden in einem gemeinsamen Pool für mehrere Kunden in einem Multi-Tenancy-fähigen Modell bereitgestellt, physische und virtuelle Ressourcen werden dynamisch zugewiesen und entsprechend der Nachfrage kontinuierlich neu verteilt. Es wird eine empfundene Ortsunabhängigkeit hergestellt indem der Nutzer kein genaues Wissen darüber besitzt wo sich dessen Ressourcen befinden auf höherem Level wie beispielsweise dem Staat, der Region oder auch Rechenzentrum kann der Ort vom Nutzer spezifiziert werden. Die bereitgestellten Ressourcen beinhalten zum Beispiel Datenspeicher, Rechenleistung, Arbeitsspeicher und Netzwerkbandbreite.

Rapid Elasticity: Rechenkapazitäten werden dehnbar bereitgestellt und abgebaut, teilweise automatisch, um entsprechend der Nachfrage schnell hoch- und wieder zurück skalieren zu können. Rechenkapazitäten erscheinen dadurch unbegrenzt und können zu jeder Zeit und in jedem Umfang bereitgestellt werden.

Measured Service: Cloud Systeme kontrollieren und optimieren Ressourcennutzung automatisch mithilfe eines Mess-Systems das auf einer abstrakten Ebene den entsprechenden Service (Datenspeicher, Rechenleistung, Benutzerkonten, etc.) überwacht, kontrolliert und Bericht erstattet um sowohl für Anbieter als auch Kunden Transparenz herzustellen.

Es wird zwischen drei grundlegende Cloud Service Modellen unterschieden: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS) (Abb. 2.1).

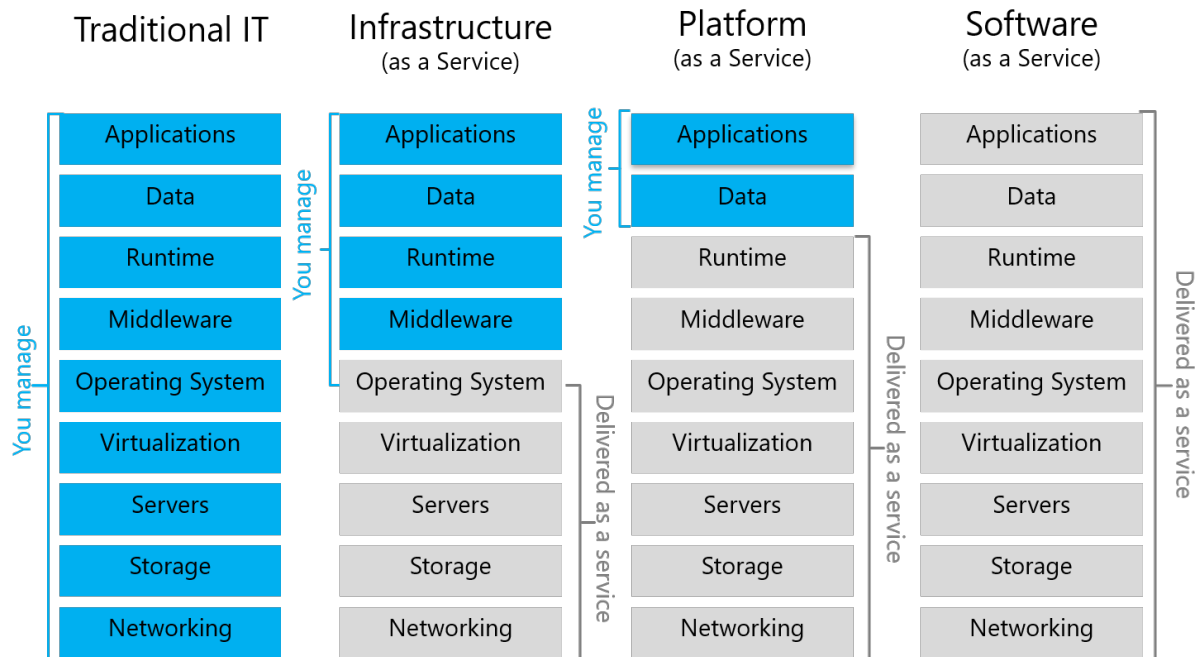


Abb. 2.1: Die Cloud Service Modelle im Überblick[3]

Infrastructure as a Service: Der Nutzer hat die Fähigkeit Rechenleistung, Datenspeicher, Netzwerke und weitere fundamentale Computing Ressourcen bereitzustellen und beliebige Software darauf zu betreiben, dazu können Betriebssysteme und Anwendungen gehören. Die darunterliegende Infrastruktur wird vom Anbieter betrieben, der Nutzer kann aber eingeschränkte Kontrolle über bestimmte Komponenten haben, dazu gehören beispielsweise Firewalls.

Platform as a Service: Der Nutzer verfügt über die Fähigkeit, eine eingekaufte oder selbst erstellten Anwendungen auf der Cloud Infrastruktur zu betreiben, die notwendige Umgebung die über Sprachen, Bibliotheken, Tools und Services verfügt wird vom Cloud Service Provider (CSP) bereitgestellt. Die darunter liegende Infrastruktur mit Netzwerken, Servern, Betriebssystemen und Speicher wird vom CSP betrieben, der Nutzer hat die Kontrolle über Anwendung und Konfiguration der Umgebung in der die Anwendung betrieben wird.

Software as a Service: Dem Nutzer wird der Zugriff auf die vom CSP in der Cloud Infrastruktur betriebenen Softwareanwendung gewährt. Auf diese wird mithilfe eines Thin oder Fat Client zugegriffen, dabei kümmert sich der Nutzer nicht um den Betrieb und die Konfiguration der darunterliegenden Cloud Infrastruktur (Netzwerke, Server, Betriebssystem,

Speicher) und die Anwendung selbst mit Ausnahme eingeschränkter Nutzereinstellungen.

In Art der Bereitstellung eines Cloud Services werden vier grundlegende Modelle unterschieden. Es existieren Public, Private, Hybrid und Community Cloud Modelle.

Private Cloud: Die Cloud Infrastruktur wird ausschließlich für die Nutzung durch eine einzige Organisation mit mehreren Nutzern bereitgestellt. Besitz und Betrieb liegen dabei entweder bei der selben Organisation, einer Drittpartei oder einer Kombination beider, die Infrastruktur kann dabei On- oder Off-Premises¹ betrieben werden.

Public Cloud: Die Public Cloud steht für die Nutzung durch die allgemeine Öffentlichkeit bereit. Die Cloud Infrastruktur befindet sich im Besitz eines Unternehmens, Bildungseinrichtung, Regierungsorganisation oder einer Kombination aus diesen und wird auch von der selben Organisation On-Premises betrieben.

Community Cloud: Eine Community Cloud wird von einer Gemeinschaft von Nutzern mit gemeinsamen Anliegen eingesetzt. Der Besitz und Betrieb liegen dabei bei einem oder mehreren Mitgliedern dieser Gemeinschaft, einer Drittpartei und kann Off- oder On-Premises betrieben werden.

werden.

Hybrid Cloud: Die Hybrid Cloud besteht aus einer Kombination der beschriebenen Modelle (Public, Private und Community). Diese bilden dabei eigene Instanzen, die aber durch standardisierte oder proprietäre Schnittstellen den Transfer von Daten und Anwendungen zwischen den Instanzen erlauben.

2.1.2 Vor- und Nachteile des Einsatzes von Cloud Computing

Vorteile und Treiber der Adoption von Cloud Computing

Wirtschaftliche Vorteile: Ein Vorteil in der Nutzung von Cloud Computing kann darin liegen, dass ein Großteil der für den Betrieb notwendigen Infrastruktur nicht mehr vom

¹ Hardware ist lokal vor Ort/nicht vor Ort

Unternehmen selbst eingekauft, eingerichtet und betrieben werden muss (Abb. 2.1). Potenziell hohe Kosten¹ die bereits vor der Inbetriebnahme eines Systems mit einem höheren Risiko aufgewendet werden müssen² stellen in Form von individuell geringeren laufenden Beträgen ein deutlich reduziertes Risiko dar[4].

Sofern der Einsatz von Cloud Computing in einer sinnvollen und korrekten Weise erfolgt³ können je nach Fall die Gesamtkosten um einen hohen Anteil reduziert werden[5].

Die Gesamtkostenersparnisse stehen auch im Zusammenhang mit Skaleneffekten⁴ die für große Cloud Service Provider gelten. Der Betrieb eines einzelnen Servers ist im Verhältnis mit bedeutend höheren Kosten verbunden als das hinzufügen eines äquivalenten Systems zu einem Rechenzentrum im Betrieb von Microsoft oder einem vergleichbaren Anbieter[6].

Skalierbarkeit: Besonders für schnell wachsende Unternehmen ist die Möglichkeit schneller Skalierbarkeit einer der prominentesten Vorteile der Cloud. Es kann nicht nur auf vorhersagbare Anstiege (zum Beispiel ausgelöst durch eine Verkaufsaktion⁵) sondern auch auf unvorhersehbare Ereignisse reagiert werden. Zusätzlich ist es möglich⁶ diese Skalierung nicht nur bis zu einem bestimmten Limit, sondern nahezu unendlich zu betreiben. Wichtig ist auch⁷ dass sowohl auf steigende als auch sinkende Nachfrage reagiert werden kann[7].

Resilienz: In einem Worst Case-Szenario kann ein ganzes Rechenzentrum durch unvorhergesehen Ereigniss⁸ die beispielsweise Brände, Naturkatastrophen oder anderes vollständig zerstört werden. Selbst wenn Backup-Rechenzentren verfügbar sind⁹ ist eine Übertragung der Operationen kein trivialer Ablauf und birgt oft nicht außer Acht zu lassende Risiken. Die Flexibilität der Cloud erlaubt es¹⁰ die gesamte Infrastruktur mit sehr geringem Aufwand in nicht betroffene Regionen zu verlagern und die Kontinuität der Geschäftstätigkeiten mit minimaler Unterbrechung aufrecht zu erhalten[8].

Security: Sicherheitsaspekte können sowohl einen Vor- als auch Nachteil von Cloud Computing darstellen. Hier sollen zuerst Vorteile dargelegt werden, potentielle Probleme sind im nächsten Abschnitt beschrieben.

Die technischen Möglichkeiten und besonders auch die Wahrnehmung des Themas Sicherheit in der Cloud unterlagen und unterliegen auch noch immer einem deutlichen Wandel. Cloud Anbieter investieren viele Ressourcen in Sicherheit und stellen dem Nutzer zum Beispiel bereits sicher implementierte Verschlüsselungen zur Verfügung oder bieten einen gewissen Schutz vor Denial-of-Service Angriffen durch ihre „natürliche“ Skalierbarkeit[9].

1 Zusammenhang zwischen produziertem Ertrag und eingesetzter Ressourcen

Herausforderungen und Risiken

Netzwerkabhängigkeit: Da der Zugriff auf Cloud Dienste über das Internet erfolgt entsteht dadurch entsprechend auch eine hohe Abhängigkeit. Stabile und schnelle Netzwerkanbindung ist eine kritische Voraussetzung für effektives Arbeiten[10], bei lokal gehosteten Systemen ist diese Abhängigkeit entsprechend geringer.

Vendor Lock-in: Bei der Nutzung eines Cloud Anbieters entsteht die Gefahr sich zu sehr in Abhängigkeit eines einzelnen Anbieters zu begeben. Im Fall einer Änderung der Nutzungsbedingungen oder einer Änderung im Kostenmodell die den eigenen Interessen stark entgegen steht, besteht die Gefahr bereits so abhängig von diesem Anbieter zu sein, dass die Kosten eines Wechsels derart hoch ausfallen dass man gezwungen ist die Bedingungen zu akzeptieren[12].

Security und Privacy: Sicherheitsrisiken sind einer der meistgenannten Gründe gegen Cloud Computing sprechen[11], besonders im Fall der Nutzung einer Public Cloud. Die Gefahr dass Daten in die Hände dritter gelangen kann zum beispielsweise nicht vollständig ausgeschlossen werden. Da die Verantwortung über die Sicherheit der Daten dem Cloud Anbieter unterliegt kann es auch zu Problemen hinsichtlich Privatheit der Daten kommen, sollte etwa eine Regierungsorganisation Zugriff auf bestimmte Daten eines Nutzer verlangen könnte dieser ohne dessen Einverständnis gewährt werden.

Kosten: Auch wenn die Nutzung von Cloud Computing mit dem Vorteil geringerer Kosten beworben wird, ist dies nicht zwingend garantiert. Werden die vorhandenen Systeme ungünstig verwendet, bleiben zum Beispiel viele gebuchte CPUs und IP-Adressen sowie weitere Ressourcen ungenutzt, können unnötig hohe Kosten entstehen. Auch während der Migrationsphase, in der möglicherweise beide Systeme parallel betrieben werden müssen, können höhere Kosten entstehen als in einem vergleichbaren Zeitraum davor. Nicht zuletzt stellt die Migration komplexer Systeme häufig eine große technische und dadurch auch finanzielle Herausforderung dar[13].

2.1.3 Überblick über die wichtigsten Cloud Service Provider

In diesem Abschnitt soll ein knapper Überblick über die wichtigsten Public Cloud Service Provider gegeben werden. Abbildung 2.2 stellt den Marktanteil der aktuell umsatz-

stärksten Cloud Service Provider sowie das globale Umsatzwachstum insgesamt dar. Zu beachten ist, dass in dieser Abbildung das Service Modell SaaS nicht beinhaltet ist.

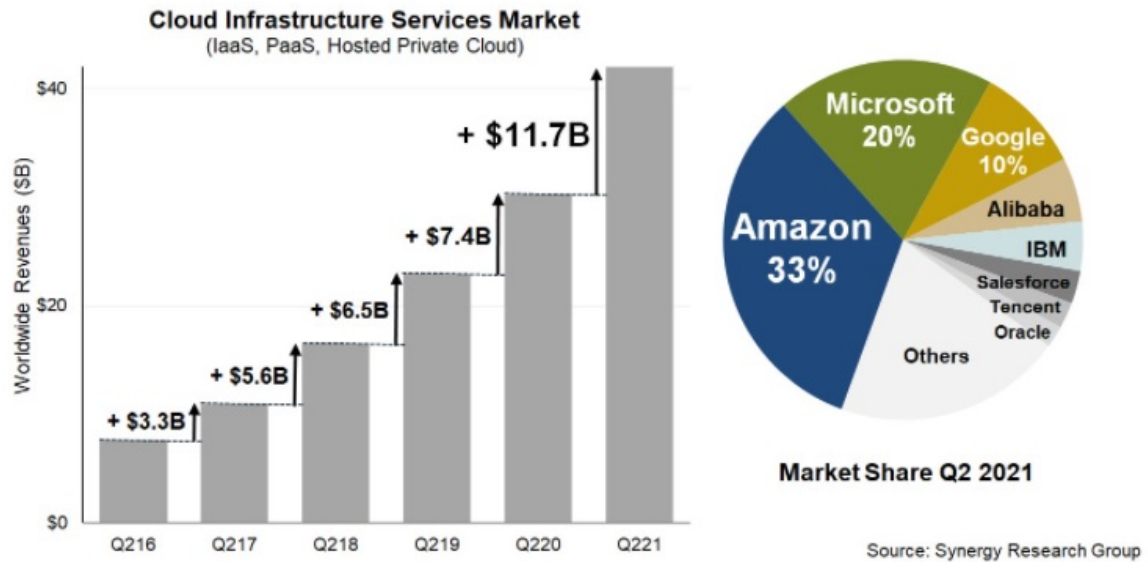


Abb. 2.2: CSP Market Share Q2 2021 nach Umsatz^[14]

Aus Abb. 2.2 wird ersichtlich, dass Amazon, Microsoft und Google gemeinsam mehr als die Hälfte des Marktes beherrschen, Amazon allein stellt etwa ein Drittel aller Infrastruktur Services bereit.



Abb. 2.3: CSP Gartner Magic Quadrant Juli 2021[15]

Der Gartner Magic Quadrant für Cloud Service Provider bietet einen groben Überblick über den Umfang der Angebote (Completeness of Vision) und die Ausgereiftheit einer Plattform (Ability to Execute). Deutlich erkennbar ist hier die Vormachtstellung von Amazon gegenüber Microsoft und Google, auffällig ist auch die Stellung von IBM die auf ein mögliches Problem in der Ausgereiftheit der IBM Cloud hinweisen könnte.

2.2 Infrastructure as Code

Automatisierung spielt in der modernen Softwareentwicklung eine wichtige Rolle. Automatisierte Unit- und Integration-Tests gehören inzwischen schon lange zum Standardrepertoire eines Softwareentwicklers. Die Verbreitung der DevOps-Kultur förderte in den

letzten Jahren auch die Automatisierung des Build- und Deployment-Prozesses und mit der steigenden Verbreitung Cloud-basierter Software erweitert sich diese Automatisierung nun auch auf das Provisioning der Infrastruktur.

Infrastructure as Code beschreibt im allgemeinen einen Ansatz zur Automatisierung des Infrastruktur-Provisionings basierend auf Methoden aus der Softwareentwicklung[16].

Statt eines manuellen Aufbaus und händischer Konfiguration der einzelnen Komponenten werden maschinenlesbare Dateien verfasst, welche dann von einem IaC Tool eingelesen und verarbeitet werden. Dabei kommen bevorzugt deklarative Sprachen zum Einsatz, deren höhere Abstraktion mehr Flexibilität als ein imperativer Ansatz erlaubt[17].

2.2.1 Technologischer Wandel und das Cloud Age Mindset

Durch die Technologien der Cloud ist es heute möglich eine gewünschte IT-Infrastruktur sehr viel schneller bereitzustellen als zuvor. Statt Einkauf, Anschließen und Einrichten eines physischen Servers das, je nach Szenario einen Zeitraum von mindestens mehreren Stunden oder Tagen bis hin zu Wochen dauern kann, können virtuelle Ressourcen in der Cloud in wenigen Minuten verfügbar gemacht werden. Der schnellere Ablauf wird durch die Automatisierung von Prozessen, die der Bereitstellung der Infrastruktur mithilfe von IaC Tools noch verstärkt. Mit diesen Veränderungen wird das Management und die Erweiterung der bestehenden Systeme jedoch nicht unbedingt einfacher[18], die Anwendung von Arbeitsprozessen die sich bisher bewährt haben sind oftmals ineffektiv und verhindern das Ausnutzen des vollen Potentials der Cloud. Kief Morris, Autor von *Infrastructure as Code - Dynamic Systems for the Cloud Age*, stellt die fundamentalen Unterschiede des Arbeitens mit Cloud-Technologien mithilfe der folgenden Tabelle (Tab. 2.1) dar.

Iron Age	Cloud Age
Cost of change is high	Cost of change is low
Changes represent failure (changes must be "managed," "controlled")	Changes represent learning and improvement
Reduce opportunities to fail	Maximize speed of improvement
Deliver in large batches, test at the end	Deliver small changes, test continuously
Long release cycles	Short release cycles
Monolithic architectures (fewer, larger moving parts)	Microservices architectures (more, smaller parts)
GUI-driven or physical configuration	Configuration as Code

Tab. 2.1: „Iron vs Cloud Age“[17]

Veränderungen im „Iron Age“ sind aufwändig und teuer und stellen ein Risiko dar, es wird versucht diese Risikopunkte zu reduzieren, daher werden viele Veränderung gebündelt getestet und eingeführt, wodurch lange Release-Zyklen entstehen. Die Architekturen die dadurch befördert werden sind monolytisch, die Konfiguration erfolgt eher mithilfe von GUI-gesteuerten Programmen oder direkter physischer Interaktion, zum Beispiel wenn ein neuer Server in ein Netzwerk eingebunden wird. Veränderungen in der Cloud stellen fast genau das Gegenteil dar, daher wird erkennbar, dass eine auf das „Iron Age“ zugeschnittene Arbeitsweise für die Cloud nicht effektiv ist. Ein neues „Cloud Age Mindset“ das die rechte Spalte der Tabelle verinnerlicht ist erforderlich um die Vorteile der Cloud wirklich effektiv und in vollem Umfang nutzen zu können.

2.2.2 Vorteile von Infrastructure as Code im Vergleich zu manuellem Infrastruktur-Provisioning

Kein Configuration Drift durch einheitliche Codebasis: Configuration Drift bezeichnet eine über die Zeit wachsende Abweichung zweier ursprünglich identischer Systeme. Wird ein gleiches System, zum Beispiel ein Applikationsserver, in verschiedenen Umgebungen eingesetzt stellen diese Umgebungen oftmals auch leicht verschiedene Anforderungen an diesen Server. Auf diese kann dann mit Optimierungen, etwa in Form spezifischer Konfigurationsdetails, reagiert werden. Wird nun das ursprüngliche Basis-System geupdated werden individuelle und oft undokumentierte Anpassungen nicht berücksichtigt wodurch ein Update unbequeme Konsequenzen nach sich ziehen kann. Werden alle Veränderungen in einer einheitlichen Codebasis verwaltet und Updates häufig vorgenommen verhindert man starken Configuration Drift der ansonsten oftmals stattfindet[19].

Wiederverwendbarkeit durch einheitlichen Code: Ein weiterer Vorteil der durch die Verwendung einer einheitlichen Codebasis entsteht ist die Wiederverwendbarkeit und Reproduzierbarkeit eines Systems. Wenn ein identisches System an einer anderen Stelle aufgebaut werden soll oder geht ein System aus unvorhergesehenen Gründen in seiner Gesamtheit verloren kann es schnell und mit verhältnismäßig geringem Aufwand reproduziert werden. Ein dazu passender Ausdruck in Bezug auf Server ist „Cattle not Pets“. Statt sich individuell und mit großem Aufwand um einzelne Server zu kümmern, wie man es etwa mit dem eigenen Haustier tut, sollten Server wie leicht ersetzbares Vieh behandelt werden[20].

Schnelleres Provisioning durch Cloud: Ein bereits häufig angesprochener Vorteil ist die schnelle Bereitstellung, durch diese Verkürzung bzw. Eliminierung des Hardware-Anschaffungs-Prozesses. Dadurch können sowohl frühe Developer-Builds als auch spätere Release-Versionen schneller getestet und deployed werden wodurch der gesamte Release-Cycle beschleunigt wird.

Schnellerer Profit: Die logische Schlussfolgerung aus dem vorangegangenen Punkt ist die schnellere Erzielung von Umsätzen. Den Markt als erstes mit einem neuen Produkt zu betreten bringt wirtschaftliche Vorteile mit sich, Voraussetzung dabei ist eine adäquate Qualität, welche durch IaC ebenfalls gefördert wird.

Einheitlicheres Tooling in Dev, Ops und weiteren Beteiligten Teams: Verwendung von IaC fördert ein einheitlicheres Tooling in allen Bereichen, die mit einem Softwareprodukt in Zusammenhang stehen. Ein einfaches Beispiel hierfür ist die Verwendung eines Source-Code Editors, der sowohl für das Schreiben von Programm- und Infrastruktur-Code gleichermaßen notwendig ist.

Stärkere Automatisierung im Arbeitsablauf: Automatisierung bedeutet immer einen gewissen initialen Zusatzaufwand, jedoch kann auf längere Sicht deutlich von automatisierten Abläufen profitiert werden. Testen, Builds und Deployment wurden bereits angesprochen und sind klassische Prozesse, deren Automatisierung längerfristig vorteilhaft ist.

Höhere Zuverlässigkeit und Sicherheit durch schnelle Updates: Veränderungen stellen die größte Gefahr für ein Produktionssystem dar, gleichzeitig sind Veränderungen aber unvermeidbar, um die Sicherheit des Systems zu gewährleisten und es zu verbessern. Durch die Fähigkeit, Veränderungen schnell und dennoch zuverlässig durchzuführen, wird die Reaktionszeit auf unvorhergesehene Ereignisse und Anforderungen wesentlich verkürzt[21].

Schnellere Fehlersuche und -behebung: Potentielle Fehler können durch Automatisierung und Modularität (siehe. Kapitel 2.2.4) schneller isoliert, gefunden und behoben werden. Infolgedessen profitieren Sicherheit und Stabilität von einer Struktur, die eine einfache und schnelle Integration von Veränderungen fördert.

2.2.3 Herausforderungen und Argumente gegen den Einsatz von Infrastructure as Code

Kief Morris benennt drei Argumente, die gegen die Einführung von IaC genannt werden, diese sollen hier im Kontext der zuvor genannten Vorteile betrachtet werden.

1. „Veränderungen werden nicht oft genug durchgeführt um Automatisierung zu rechtfertigen.“

Die Idee, dass ein System einmal erstellt und dann „fertig“ ist, wodurch eine Automatisierung der Veränderungen überflüssig wird, entspricht sehr selten der tatsächlichen Realität. IT-Systeme und damit auch IT-Infrastruktur wird während ihres gesamten Lebenszyklus mehr oder weniger kontinuierlich verändert und erweitert.

Sicherheitslücken in alten Versionen von Softwarepackages oder Betriebssystemen sind keine Seltenheit und müssen gepatcht werden, um einen sicheren und zuverlässigen Betrieb zu gewährleisten. Neue Features in bestehender Software kann neue Infrastruktur, zum Beispiel in Form einer zusätzlichen Datenbank, notwendig machen oder eine veränderte Konfiguration erfordern. Gerade bei Sicherheitslücken ist es wichtig, Anpassungen nicht erst nach längerer Zeit sondern so schnell wie möglich durchzuführen, um Sicherheit und Stabilität nicht zu gefährden. Ein weiteres Szenario, das die Stabilität eines Systems gefährden kann, ist ein schneller Zuwachs an Last, die ein System erfährt, können die Kapazitäten automatisiert erweitert werden, um einen Zusammenbruch des Services auf einfache Art verhindert zu werden [21].

„A fundamental truth of the Cloud Age is: Stability comes from making changes.“ [22]

Eine fundamentale Wahrheit des Cloud Zeitalters ist: Stabilität entsteht durch Veränderung.

2. „Infrastruktur soll zuerst aufgebaut, danach automatisiert werden.“

Die Umsetzung von Infrastructure as Code stellt eine durchaus eine große Herausforderung dar, umso mehr wenn die entsprechenden Kompetenzen noch aufgebaut werden müssen. Der Nutzen wird deshalb nicht unbedingt direkt ersichtlich, da es zu Situationen kommt in denen es attraktiv erscheint Infrastruktur zuerst aufzubauen und sich erst später um die Automatisierung zu kümmern. Mit diesem Ansatz werden viele der Vorteile von IaC jedoch verwirkt. Ein bestehendes, komplexes Infrastruktur-System zu automatisieren

ist eine deutlich größere Herausforderung als ein System von Grund auf mit dem Gedanken an Automatisierung aufzubauen[23].

3. „Es muss zwischen schneller Umsetzung und hoher Qualität gewählt werden.“

Die Idee dass der Fokus auf hohe Geschwindigkeit und hohe Qualität sich gegenseitig behindert oder ausschließt mag logisch erscheinen, in der Praxis ist dies jedoch nicht der Fall. Ein unausgeglichener Fokus auf eines der beiden Kriterien führt im Lauf der Zeit zu einem „Fragile Mess“ (Abb. 2.4).

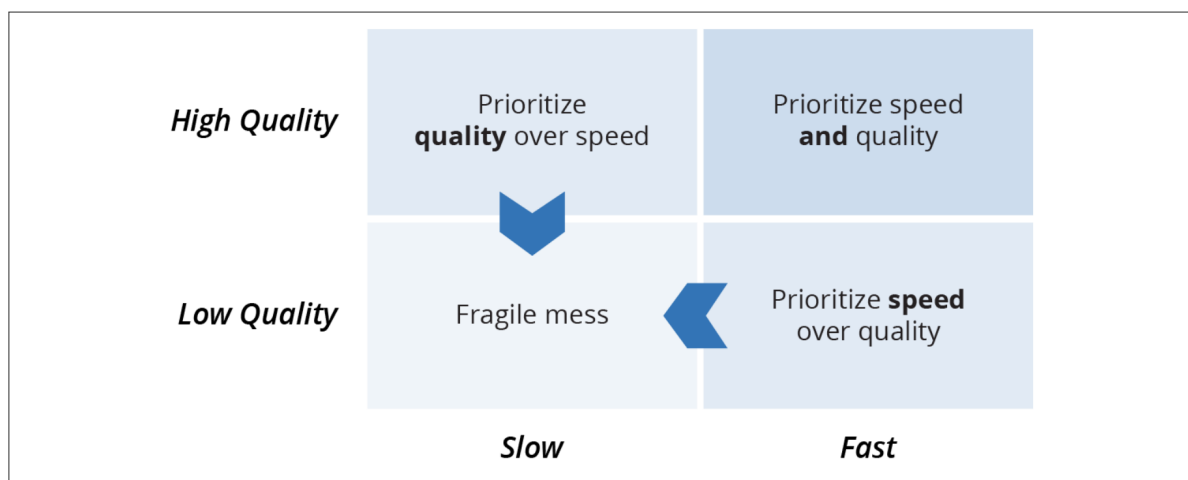


Abb. 2.4: Verhältnis von Geschwindigkeit und Qualität[24]

Wird Geschwindigkeit über Qualität gestellt (Quadrant links oben) entstehen mit der Zeit chaotische und instabile Systeme an denen Veränderungen mit der Zeit nur noch erschwert und deshalb langsam durchgeführt werden können.

Wird Geschwindigkeit zu niedrig priorisiert (Quadrant rechts unten) führt es allerdings auch dazu dass letzten Endes durch Druck von Deadlines und schnellen Workarounds technische Schulden aufgebaut werden welche ebenfalls zu einem qualitativ schlechten System führen.

Aufgrund dieser Probleme ist es wichtig Geschwindigkeit und Qualität gleichermaßen zu priorisieren.

2.2.4 Die drei Kernverfahren von Infrastructure as Code

- **Define everything as Code:** Alle Teile eines Systems in Form von Code zu definieren bringt mehrere Vorteile mit sich. Der Konfigurations-Code kann mehrfach ausgeführt werden, daher ist ein als Code definiertes System wiederverwendbar. Es können unkompliziert mehrere identische Instanzen erstellt werden, das gilt auch für den Fall, wenn Fehler auftreten und ein Neuaufbau erforderlich ist. Das Verhalten des Systems ist vorhersehbarer, fortlaufendes automatisches Testen ist möglich und damit auch zuverlässiger. Definition in Code macht auch den Aufbau eines Systems transparenter, da dieser immer dem tatsächlichen Zustand entspricht und diesen damit auch dokumentiert[25].
- **Continuously Test and Deliver All Work in Progress:** Fortlaufendes, automatisiertes Testen und Integrieren aller Komponenten, die sich in Entwicklung befinden, dient dem Ziel, die Qualität eines Systems nicht nur „einzutesten“, sondern von Beginn an und kontinuierlich einzubauen[25].
- **Build Small, Simple Pieces That You Can Change Independently:** Systeme, die aus mehreren kleineren, voneinander unabhängigen Komponenten bestehen, sind generell stabiler als große Monolithen. Eine Änderung, die einen Fehler verursacht, betrifft nur die Komponente, in der die Änderung stattfindet; diese kann dann leichter isoliert und das Problem behoben werden. Kleine Komponenten sind in der Regel auch weniger komplex und dadurch leichter zu verstehen. Ein einzelner Fehler nach einem Update hat auch den Vorteil, dass nur diese Komponente und nicht das gesamte System auf eine ältere Version zurückgesetzt werden muss, um den Betrieb wieder herzustellen[26].

2.3 Funktionsprinzip und Rolle von Terraform im IaC-Anwendungsprozess

Während der Anwendung von Infrastructure as Code kann primär zwischen zwei wichtigen Phasen unterschieden werden, einer initialen Einrichtungsphase und der darauf folgenden Wartungs- und Betriebsphase. Während der Einrichtung wird die Infrastruktur bereitgestellt und konfiguriert, genauso wird auch Software installiert und eingerichtet. Nachdem

das System dann in Betrieb genommen wurde können Anpassung notwendig sein, Server werden hinzugefügt und abgebaut, Software wird aktualisiert und neu konfiguriert.

2.3.1 Überblick über Infrastructure as Code Tools

Infrastructure as Code beinhaltet verschiedene konkrete Anwendungsfälle und entsprechend existieren auch Tools die zum Teil ein breiteres Spektrum von IaC abdecken, zum Teil aber auch eher spezialisiert sind; Terraform ist dabei eher ein Beispiel eines spezialisierten Tools. Die unten stehende Abbildung 2.5 soll einen Überblick über verschiedene IaC Tools und deren Aufgabengebiete verschaffen, die Einordnungen sind dabei aber nicht unbedingt als absolut anzusehen. Es ist zum Beispiel möglich innerhalb eines Terraform-Deployments auch Software zu installieren und zu konfigurieren, allerdings wird die tatsächliche Installation dann eher per von Terraform aufgerufenen Skripten vorgenommen statt von Terraform selbst verwaltet zu werden, daher ist Terraform hier ausschließlich als Infrastruktur-Management Tool eingeordnet.

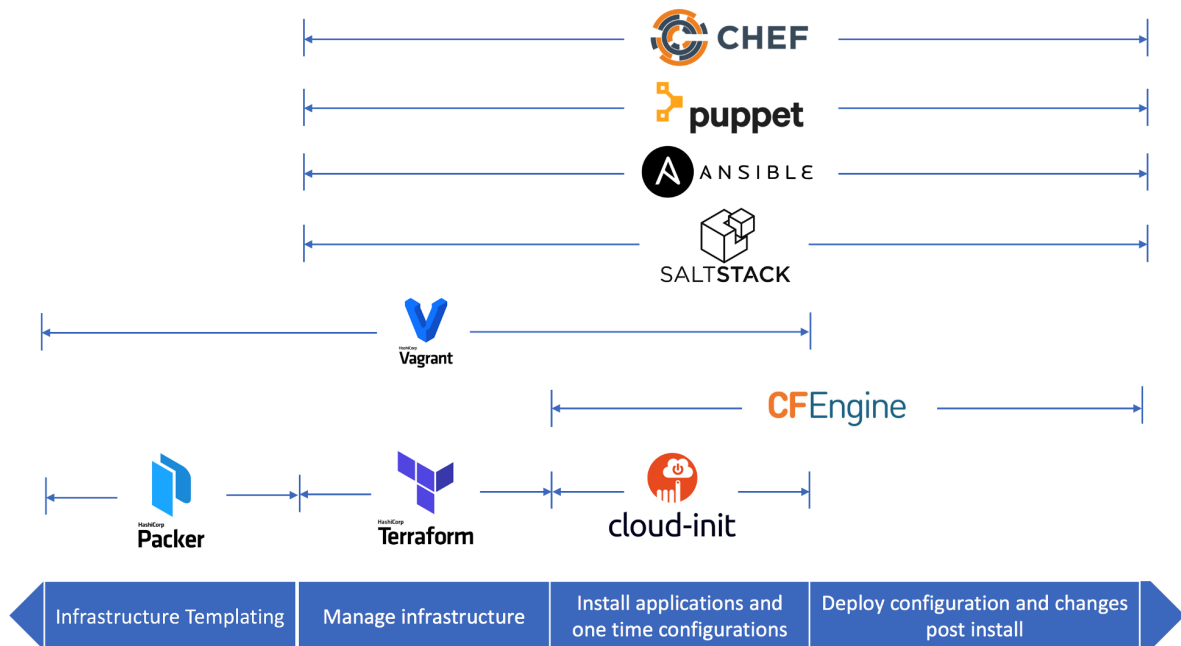


Abb. 2.5: Überblick IaC Tools[27]

Vorteile und Limiterungen von Terraform

Da sich diese Arbeit primär mit dem Provisioning von grundlegender Cloud Computing Infrastruktur in Form von VM's, Netzwerken, Datenspeicher und anderen grundlegenden Komponenten beschäftigen soll, setzt sich der Einsatz eines darauf spezialisierten Tools an. Neben Terraform existiert in diesem Bereich auch das Tool Pulumi, in dieser Arbeit wird Terraform jedoch aus den folgenden Gründen verwendet:

- **Modularität:** Terraform Module erlauben es ein System in mehrere klar definierte Komponenten zu strukturieren. Dadurch wird die Wiederverwendbarkeit dieser ermöglicht und gefördert, die daraus resultierenden Vorteile wurden bereits in den vorangegangenen Kapiteln angesprochen.
Pulumi strukturiert Infrastruktur Code entweder in einem großen monolithischen Projekt oder vielen kleinen Mikroprojekten, beide Optionen sind weniger flexibel und umständlicher als die von Terraform implementierte Lösung.
- **Weitere Verbreitung und größere Popularität:** Terraform wurde 2014, Pulumi 2017 veröffentlicht, entsprechend ist Terraform deutlich weiter verbreitet und verfügt über all die Vorteile die eine größere Community mit sich bringt. Dazu gehören mehr Lernressourcen, mehr Codebeispiele, größere Bekanntheit und mehr Jobs für die Arbeit mit Terraform.
- **Dokumentation:** Einen weiteren Vorteil von Terraform stellt dessen umfangreiche und ausgereifte Dokumentation, wie auch die Dokumentation der einzelnen Provider dar. Der genaue Aufbau der einzelnen Ressourcen wie etwa einer VM auf Google Cloud Platform (`google_compute_instance`) ist mit einem Beispiel und der zugehörigen Argument Reference versehen aus der direkt ersichtlich wird, welche Argumente notwendig (required), was der Zweck jedes einzelnen Arguments ist und wo anstelle eines einfachen Wertes ein Block oder eine Liste erwartet wird.

2.3.2 Funktionsweise von Terraform

Terraform ist ein Infrastructure as Code Tool das es ermöglicht Infrastruktur sicher und effizient aufzubauen, zu verändern und zu versionieren. Die Funktionsweise von Terraform wird in Abbildung 2.6 dargestellt.

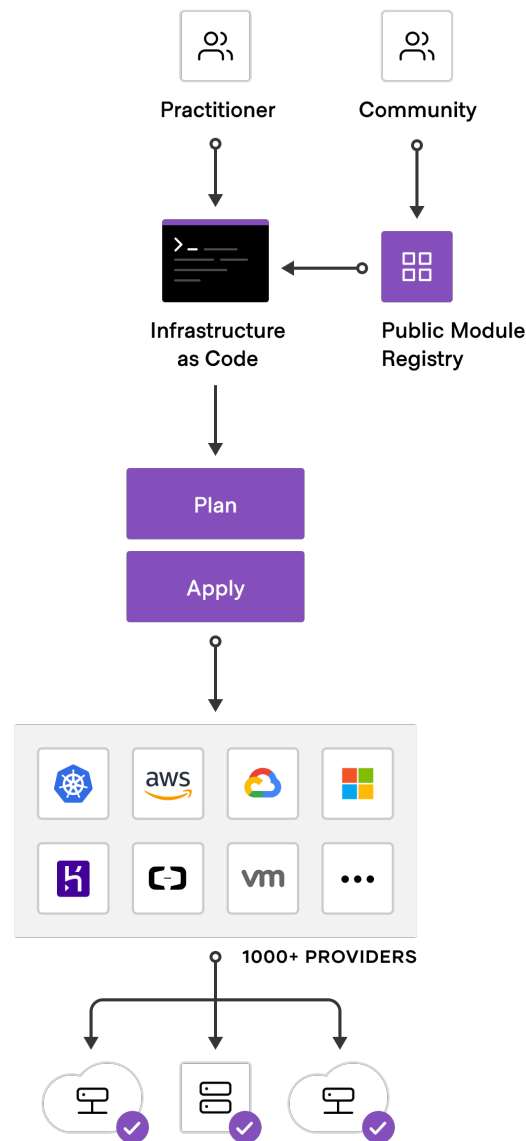


Abb. 2.6: Terraform Funktionsprinzip[28]

Terraform verwendet eine deklarative Domänenspezifische Sprache (DSL) ¹ die es erlaubt Infrastruktur in Konfigurationsdateien zu beschreiben. Diese werden versioniert, geteilt und wiederverwendet werden können. Terraform Module können auf einer Public Module Registry ¹ zur allgemeinen Verfügung bereitgestellt und von der breiteren Terraform-Community wiederverwendet werden. Terraform unterstützt über Plug-Ins, die Terraform Provider, eine große Anzahl an Plattformen.

¹ Vergleichbar mit der npm-Registry von Node.js

Terraform State

Das Terraform State-File ist die wichtigste Datei in einem Terraform-Projekt. Sie enthält die Informationen der aktuell existierenden und von Terraform verwalteten Ressourcen und stellt ein Mapping zwischen den Konfigurationsdateien und der realen Welt dar[31]. Zusätzlich befinden sich essentielle Metadaten und Informationen über die Abhängigkeiten der verschiedenen Ressourcen im Terraform State.

Der Terraform State kann entweder lokal oder in einem Remote Backend, zum Beispiel der Terraform Cloud, gespeichert werden. Lokaler State ist ausreichend für ein kleineres Forschungsprojekt eines einzelnen Entwicklers, für wichtige Projekte und die Kollaboration in einem Team sollte jedoch ein Remote Backend gewählt werden. Sollten zwei Entwickler gleichzeitig Änderungen an der Infrastruktur vornehmen wollen muss ausgeschlossen werden, dass der State korrumpiert und inkonsistent wird. Sobald der erste Anpassungsvorgang begonnen wird, wird der Zugriff auf den State gesperrt und erst am Ende wieder freigegeben, auf diese Weise bleibt der State konsistent.

Terraform Workflow

Der grundlegende Terraform Workflow wird in Abbildung 2.7 dargestellt. Er besteht primär aus vier Befehlen: einem Init-Befehl zur Initialisierung des Projekts, der Erstellung des Execution Plan, Durchführung des Execution Plan und einem Destroy-Befehl zur Freigabe/Abbau aller von Terraform verwalteten Ressourcen.

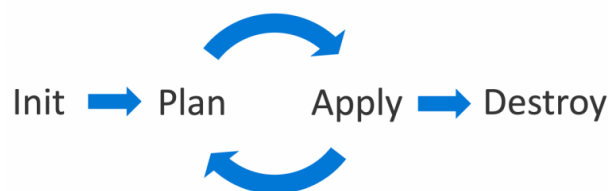


Abb. 2.7: Terraform Workflow[32]

Mit **terraform init** wird das Arbeitsverzeichnis initialisiert, die Provider und Module installiert sowie das Backend für den Terraform State konfiguriert.

Durch **terraform plan** wird der Execution Plan erstellt. Der Execution Plan definiert alle Schritte, die von Terraform durchgeführt werden müssen, um vom aktuellen Zustand, gespeichert

im Terraform State, zum in den Konfigurationsdateien mit Terraform Code beschriebenen Zustand zu gelangen. Dieser Execution Plan kann ausgegeben und manuell reviewt werden bevor die tatsächliche Ausführung dessen vorgenommen wird. Durch die Erstellung dieses Plans werden Abhängigkeiten betrachtet die Reihenfolge der Ressourcen-Erstellung bestimmen, voneinander unabhängige Ressourcen können zum Beispiel parallel erstellt werden.

Die Ausführung des Execution Plan wird durch **terraform apply** ausgeführt. Es gibt zwei Modi: Der **Automatic Plan Mode** erstellt automatisch einen neuen Execution Plan, der **Saved Plan Mode** verwendet einen in einer Datei gespeicherten Execution Plan.

Und zuletzt dient **terraform destroy** dazu alle in Terraform verwalteten Ressourcen zu „zerstören“ und damit wieder freizugeben. Dies wird in einem Produktionssystem selten Anwendung finden, während der Entwicklung und für Testprojekte ist dies jedoch extrem nützlich.

2.3.3 Einführung in die Hashicorp Configuration Language

Die von Terraform verwendete Konfigurationssprache wurde mit dem Ziel entwickelt einen Kompromiss zwischen Maschinenfreundlichkeit und Menschenlesbarkeit zu erzielen[29]. Existierende Serialisierungsformate, Konfigurationssprachen und Programmiersprachen konnten die Ziele der Terraform-Entwickler nicht erfüllen daher kommt nun bei Terraform eine DSL in Form der Hashicorp Configuration Language (HCL) zum Einsatz.

Grundlegende Elemente

HCL besteht aus drei grundlegenden Elementen: Blöcken (Blocks), Argumenten (Arguments) und Ausdrücken (Expressions).

```
1 blockType "label1" "label2" {  
2     ...  
3     /*This is a block*/  
4     ...  
5  
6     argument = 5  
7  
8     arguments + can + build = expressions
```


9 }

List. 2.1: Grundlegende Elemente von HCL

Blöcke stellen für gewöhnlich ein Objekt, im Fall von Infrastruktur-Code meistens eine Computing Ressource, dar. Blöcke besitzen einen Typ und Null bis mehrere Label. Blöcke beinhalten Argumente und weitere verschachtelte Blöcke.

Argumente sind das, was in den meisten Programmiersprachen die Variablen darstellen: Ein Wert für einen Namen zugewiesen wird.

Ausdrücke sind ähnlich wie in anderen Sprachen ein aus anderen Ausdrücken und Argumenten berechneter Wert, ein Argument ist in diesem Sinne die simpelste Form eines Ausdrucks.

Input und Output-Variablen

Input-Variablen sind nützlich, um Parameter außerhalb des eigentlichen Terraform Codes anzupassen. Die ID des Projektes stellt einen solchen Parameter dar, befindet sich diese außerhalb des Codes, muss nur die Datei, welche die Input-Variablen enthält, angepasst werden, alles andere kann ohne Veränderungen wiederverwendet werden.

```
1 variable "project" {  
2     default = "some_value"  
3     description = "a description"  
4 }
```

List. 2.2: Beispiel Input-Variable

Input-Variablen können entweder während der Ausführung über die Kommandozeile oder über eine Datei mit einfachen Key-Value Paaren befüllt werden.

Output-Variablen werden in der Regel verwendet, um nach dem Aufbau des Systems auf spezifische Parameter einfachen und schnellen Zugriff zu ermöglichen.

```
1 output "something_important" {  
2     value = resource-group.vm-name.ip-address  
3 }
```

List. 2.3: Beispiel Output-Variable

Werte wie die IP einer Virtuellen Maschine auf einer Public Cloud sind oft vor der Bereitstellung dieser nicht bekannt und können erst danach ausgegeben werden. Terraform kann Output-Variablen mithilfe eines Kommandos schnell und übersichtlich ausgeben.

Terraform Module

Die Verwendung von Modulen gestaltet sich ebenfalls relativ einfach. Es wird immer eine Quelle benötigt, aus der das Modul bezogen wird, diese kann ein lokales Verzeichnis oder eine Module Registry sein. Zusätzlich benötigen die meisten Module eine Reihe von Input-Variablen wie im folgenden Listing 2.4 dargestellt.

```
1 module "showcase_kubernetes" {
2   source = "./showcase_kubernetes"
3
4   project    = var.project
5   network    = google_compute_network.jsa-vpc_network.name
6   zone       = var.zone
7   num_nodes  = 2
8 }
```

List. 2.4: Beispiel Terraform Module

Terraform Provider Konfiguration

Terraform Provider benötigen ähnlich wie Module eine Quelle und eine optionale zusätzliche Konfiguration. In Listing 2.6 wird beispielhaft der Google Cloud Platform Provider eingebunden und für ein spezifisches Projekt konfiguriert.

```
1 terraform {
2   required_providers {
3     google = {
4       source = "hashicorp/google"
5       version = "4.1.0"
6     }
7   }
8
9   provider "google" {~
10     project = "active-woodland-324808"
```

```
11     region = "europe-west3"
12     zone   = "europe-west3-b"
13 }
14 }
```

List. 2.5: Beispiel Terraform Provider Konfiguration

3 Evaluierungsanforderungen und Umsetzung

3.1 Evaluierungsanforderungen

3.1.1 Ziel der Evaluierung

3.1.2 Untersuchte Komponenten der Terraform Provider

3.1.3 Auswahl der Evaluierungskriterien

3.2 Umsetzung des Testsystems

3.2.1 Eingesetzte Software und Tools

3.2.2 High-Level Aufbau des Testsystems

3.2.3 Konkreter Aufbau auf Google Cloud Platform

3.2.4 Konkreter Aufbau auf Azure

3.2.5 Aufbau der ergänzenden Versuche

4 Ergebnisse und Bewertung

4.1 Evaluierung der Functional Completeness

	Google Cloud Platform	Microsoft Azure
VM Ubuntu 20.04	Funktionalität wird durch Ressource <code>google_compute_instance</code> erfüllt.	Funktionalität wird durch Ressource <code>azurerm_virtual_machine</code> erfüllt.
PostgreSQL Datenbankserver	Funktionalität wird durch Ressource <code>google_sql_database_instance</code> erfüllt.	Funktionalität wird durch Ressource <code>azurem_postgresql_database</code> erfüllt.
Anlegen von Datenbanken mit Charset UTF-8 und Collation English_Unites States.1252	Funktionalität wird durch die Ressource <code>google_sql_database</code> teilweise erfüllt. Collation kann bei Erstellung nicht wie erwünscht definiert werden. Default ist <code>en_US.UTF8</code>	Funktionalität wird durch die Ressource <code>azurem_postgresql_database</code> erfüllt. Charset und Collation werden bei Erstellung definiert.
Secret Storage	Funktionalität wird durch Ressource <code>google_secret_manager_secret</code> erfüllt. Es existiert ein Secret Storage je Google Projekt.	Funktionalität wird durch Ressource <code>azurem_key_vault_secret</code> erfüllt. Mehrere secret storages in Ressource Group möglich.
Kubernetes Cluster	Funktionalität wird durch Ressource <code>google_container_cluster</code> erfüllt-	Funktionalität wird durch Ressource <code>azurem_kubernetes_cluster</code> erfüllt.
Einbindung vom Container Registry als Data Source	Funktionalität wird durch Data Source <code>google_storage_bucket</code> erfüllt	Funktionalität wird durch Data Source <code>azurem_container_registry</code> erfüllt

Tab. 4.1: Analyse der Functional Completeness

4.2 Ergebnisse und Bewertung der Time Behaviour Tests

	Testlauf	Google Cloud Platform		Microsoft Azure	
		Aufbau	Abbau	Aufbau	Abbau
Aufbau/Abbau Testsystem	1	15m48s	7m54s	7m7s	6m5s
	2	17m37s	8m11s	7m59	7m44s
	3	15m12s	6m52s	6m54s	6m6s
Aufbau/Abbau VM	1	0m30s	1m3s	1m12s	6m5s
	2	0m23s	1m4s	1m11s	1m37s
	3	0m25s	1m4s	1m11s	1m40s
Aufbau/Abbau PostgreSQL- Datenbankserver	1	3m59s	1m3s	2m28s	0m42s
	2	3m58s	0m52s	2m32s	0m43s
	3	3m57s	0m52s	2m28s	0m40s
Aufbau/Abbau Kubernetes- Cluster mit default Node Pool	1	3m59s	2m52s	4m36s	5m33s
	2	3m39s	3m3s	6m5s	5m32s
	3	3m31s	2m53s	5m13s	5m29s

Tab. 4.2: Ergebnisse der Time Behaviour Tests (1/2)

	Testlauf	Google Cloud Platform	Microsoft Azure
Execution Plan für Testsystem	1	1.2s	14.6s
	2	1.2s	15.2s
	3	1.6s	15.9s
Execution Plan für Datenbank- server	1	0.9s	22.2s
	2	1.0s	23.2s
	3	1.0s	23.3s
Execution Plan für Kubernetes Cluster mit default Node Pool	1	0.8s	13.5s
	2	0.9s	13.3s
	3	0.9s	12.3s

Tab. 4.3: Ergebnisse der Time Behaviour Tests (2/2)

4.3 Ergebnisse und Bewertung der Recoverability Tests

	Google Cloud Platform	Microsoft Azure
Längere Unterbrechung des Internets, State in Remote Backend	State korumpiert, erneutes terraform apply führt zu Fehler, manuelles entfernen der Ressource notwendig.	State korumpiert, erneutes terraform apply führt zu Fehler. Zusätzlich zur VM Ressource muss die Festplatte ebenfalls separat manuell gelöscht werden.
Temporäre Unterbrechung des Internets, lokaler State	Terraform state wird auch lokal korumpiert, manuelles löschen der VM-Ressource notwendig.	
Soft Cancel von Terraform apply	State korumpiert	Shutdown sofort, state korumpiert
Hard Cancel von Terraform apply	State korumpiert	Nicht durchgeführt

Tab. 4.4: Ergebnisse der Recoverability Tests

4.4 Ergebnisse und Bewertung der Modifiability Tests

	Google Cloud Plattform	Microsoft Azure
Änderung des Betriebssystems einer virtuellen Maschine	Ressource wird zerstört und neu erstellt.	
vergrößerung der Festplatte einer virtuellen Maschine	Ressource wird zerstört und neu erstellt.	Führt zu Error.
Veränderung des Machine-Type einer virtuellen Maschine	Ressource wird modifiziert.	
Vergrößerung des default Node Pools von 1 auf 2 Nodes	Ressource wird zerstört und neu erstellt.	Ressource wird modifiziert.

Tab. 4.5: Ergebnisse der Modifiability Tests

Blabla

4.5 Evaluierung der Einheitlichkeit der Testsysteme für Azure und Google Cloud Platform

5 Schluss

5.1 Fazit

5.2 Ausblick

öö

A Kapitel im Anhang

Literaturverzeichnis

- [1] Asadi, S., Nilashi, M., Husin, A.R.C. et al. - Customers perspectives on adoption of cloud computing in banking sector. Inf Technol Manag 18, 305?330 (2017). [Online] Verfügbar unter: <https://doi.org/10.1007/s10799-016-0270-8> (Zugriff am: 23.01.2022)
- [2] Mell, Peter, and Tim Grance. - "The NIST definition of cloud computing."(2011). [Online] Verfügbar unter: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf> (Zugriff am: 28.10.2021)
- [3] Chou, David. Cloud Service Models (IaaS, PaaS, SaaS) Diagram (2018). [Online] Verfügbar unter: <https://dachou.github.io/2018/09/28/cloud-service-models.html> (Zugriff am: 28.10.2021)
- [4] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.23.
- [5] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.24.
- [6] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.24-25.
- [7] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.27.
- [8] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.26.
- [9] Lisdorf, Anders. - Cloud computing basics a non-technical introduction. Berkeley, CA: Apress, 2021. S.25.

- [10] NANDGAONKAR, Suruchee V.; RAUT, A. B. A comprehensive study on cloud computing. International Journal of Computer Science and Mobile Computing, 2014, 3. Jg., Nr. 4, S. 737.
- [11] NANDGAONKAR, Suruchee V.; RAUT, A. B. A comprehensive study on cloud computing. International Journal of Computer Science and Mobile Computing, 2014, 3. Jg., Nr. 4, S. 737.
- [12] OPARA-MARTINS, Justice; SAHANDI, Reza; TIAN, Feng. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. Journal of Cloud Computing, 2016, 5. Jg., Nr. 1, S. 1-18. [Online] Verfügbar unter <https://ieeexplore.ieee.org/abstract/document/7009018> (Zugriff am: 24.01.2022)
- [13] BAI, Kun, et al. What to discover before migrating to the cloud. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013). IEEE, 2013. S. 320. [Online] Verfügbar unter: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6573001> (Zugriff am: 24.01.2022)
- [14] Kein Autor. Enterprise spending on cloud infrastructure services grows to USD 42 billion in Q2, 2021: SRG (2021) [Online] Verfügbar unter: <https://www.cioandleader.com/article/2021/07/30/enterprise-spending-cloud-infrastructure-services-grows-usd-42-billion-q2-2021> (Zugriff am: 24.01.2022)
- [15] Bala, Raj; Gill, Bob; Smith, Dennis; Ji, Kevin; Wright, David. Magic Quadrant for Cloud Infrastructure and Platform Services (2021) [Online] Verfügbar unter: <https://www.gartner.com/doc/reprints?id=1-2710E4VR&ct=210802&st=sb> (Zugriff am: 24.01.2022)
- [16] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.4
- [17] Jacobs, Mike; Kaim, Ed. What is Infrastructure as Code? (2021) [Online] Verfügbar unter: <https://docs.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code> (Zugriff am: 24.01.2022)
- [18] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.3
- [19] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.17-18
- [20] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.16

- [21] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.4-5
- [22] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.5
- [23] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.6
- [24] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.8
- [25] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.10
- [26] Kief Morris. Infrastructure as Code. O'Reilly Media, 2. edition, 2020. S.11
- [27] Nayak, Ramnath. When to use which Infrastructure-as-code tool (2019)
[Online] Verfügbar unter: <https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde> (Zugriff am: 25.01.2022)
- [28] Terraform Website. [Online] Verfügbar unter: <https://www.terraform.io/> (Zugriff am: 25.01.2022)
- [29] Introduction to HashiCorp Configuration Language (HCL). [Online] Verfügbar unter: <https://www.linode.com/docs/guides/introduction-to-hcl/> (Zugriff am: 25.01.2022)
- [30] Terraform Provider Documentation. [Online] Verfügbar unter: <https://www.terraform.io/language/providers> (Zugriff am: 26.01.2022)
- [31] Terraform State Documentation. [Online] Verfügbar unter: <https://www.terraform.io/language/state/purpose> (Zugriff am 26.01.2022)
- [32] Kumar, Rajesh. Terraform Basics Workflow loop explained!!! (2019).
[Online] Verfügbar unter: <https://www.devopsschool.com/blog/terraform-basics-workflow-loop-explained/> (Zugriff am: 26.01.2022)