Hochschule Esslingen
University of Applied Sciences

Bachelorarbeit

Cloud Service Provider Evaluierung auf Basis von Infrastructure as Code Unterstützung

im Studiengang Softwaretechnik und Medieninformatik
der Fakultät Informationstechnik
Wintersemester 2021/2022

Julian Schallenmüller

Zeitraum: 15.10.2021 - 15.01.2022
Prüfer: Prof. Dr.-Ing. Kai Warendorf
Zweitprüfer: Prof. Dr. rer. nat. Mirko Sonntag

Firma: Noavtec 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 7. Januar 2022 _____

Unterschrift

Sperrvermerk

Die nachfolgende Bachelorarbeit enthält vertrauliche Daten der Noavtec Consulting GmbH. Veröffentlichungen oder Vervielfältigungen dieser Arbeit – auch nur auszugsweise – sind ohne ausdrückliche Genehmigung der Noavtec 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

Dank an die Firma und die Firmenmitarbeiter, max. 1/2 Seite

Kurz-Zusammenfassung

„Aushängeschild"“ der Arbeit, max 1 Seite

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Einleitung

Eines der wichtigsten Schlagworte im Zeitalter der fortschreitenden Digitalisierung ist der Begriff des Cloud Computings. Auch in Bereichen der Industrie wie der Finanz- und Versicherungsbranche die sich zu großen Teilen aufgrund von Sicherheits- und anderen Bedenken lange Zeit gegen die Nutzung Cloud-basierter Systeme entschieden hatte gewinnt das Thema mehr und mehr Relevanz. 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 auch immer eine Reihe eigener Herausforderungen mit sich. Für eine erfolgreiche und gewinnbringende Einführung dieser ist es essentiell diese zu verstehen und die passende Denkweisen und Werkzeuge mit denen die aufkommenden Probleme gelöst werden können entsprechend einzusetzen.

1.2 Motivation und Ziele der Arbeit

Das Thema mit dem sich diese Arbeit befassen wird ist das automatisierte Managen und Bereitstellen von Cloud Computing Resourcen, zusammengefasst unter dem Begriff Infrastructure as Code (IaC). Die Grundlagenkapitel werden zu diesem Zweck auf den technischen Kontext und die Relevanz von Cloud Computing, IaC und dem Software Tool Terraform eingehen. Es wird erläutert werden an welcher Stelle die ausgewählten Plattformen und Software zum Einsatz kommen, welche Probleme dadurch gelöst werden, wo deren Vorteile und Grenzen liegen sowie welche Alternativen existieren und wo ergänzende Werkzeuge zum Einsatz kommen können.
Den Kern der Arbeit bildet ein Vergleich der ausgewählten Cloud Service Provider auf Basis der Unterstzützung Von IaC mit Terraform. Hierfür wird die Infrastruktur einer Schulung der Firma Novatec Consulting GmbH in der die wichtigsten grundlegenden Cloud Computing Resourcen Verwendung finden auf verschiedenen Plattformen mit Terraform bereitgestellt und einem Vergleich unterzogen. Als Vergleichsmetriken werden Kriterien der Norm ISO/IEC 25000 herangezogen um eine fachgerechte und neutrale Bewertung zu gewährleisten. Dabei soll auch in Betracht gezogen werden wie hoch der Aufwand für die

Migration eines bestehenden Systems zu IaC ausfällt. Nach der Auswertung der Vergleichsergebnisse werden die gewonnenen Erkentnisse zusammengefasst, weitere untersuchenswerte Aspekte aufgeführt und ein Ausblick auf aktuelle ud Zukünftige Entwicklungen gegeben. Durch die sehr aktuelle Relevanz des Themas werden sich mit hoher Wahrscheinlichkeit auch in Zukunft neue Software und Technologien durchsetzen, alte verdrängt und neue Lösungsansätze für bestehende Herausforderungen und Probleme etabliert werden. Diese Arbeit wird jedoch nur die aktuell relevantentesten PLattformen und Tools betrachten um eine Eintscheidungsbasis für deren Einsatz zum aktuellen Zeitpunkt bieten zu können.

2 Grundlagen

2.1 Funktionsprinzip, Vorteile und Herausforderungen des modernen Cloud Computings

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 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 Computerresourcen (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 Resourcen automatisch bereitstellen, es wird keine menschliche Interaktion benötigt.

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

Resource Pooling:: Die Computerresourcen des Anbieters werden in einem gemeinsamen Pool für mehrere Kunden in einem mandantentauglichen Modell bereitgestellt, physische

und virtuelle Resourcen werden nach dynamisch zugewiesen und entsprechend der Nachfrage neu verteilt. Es wird eine empfundene Ortsunabhängigkeit hergestellt indem der Nutzer kein genaues Wissen darüber besitzt wo sich dessen Resources befinden, auf höherem Level wie beispielsweise dem Staat, der Region oder auch Rechenzentrum kann der Ort vom Nutzer spezifiziert werden. Die bereitgestellten Resourcen beinhalten zum Beispiel Datenspeicher, Rechenleistung, Rechenspeicher 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 zu jeder Zeit in jedem Umfang bereitgestellt werden.

Measured Service: Cloud systeme kontrollieren und optimieren Resourcennutzung automatisch mithilfe eines Messungs-systems dass auf einer abstrakten Ebene den entsprechenden Service (Datenspeicher, Rechenleistung, Benutzerkonten) 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).

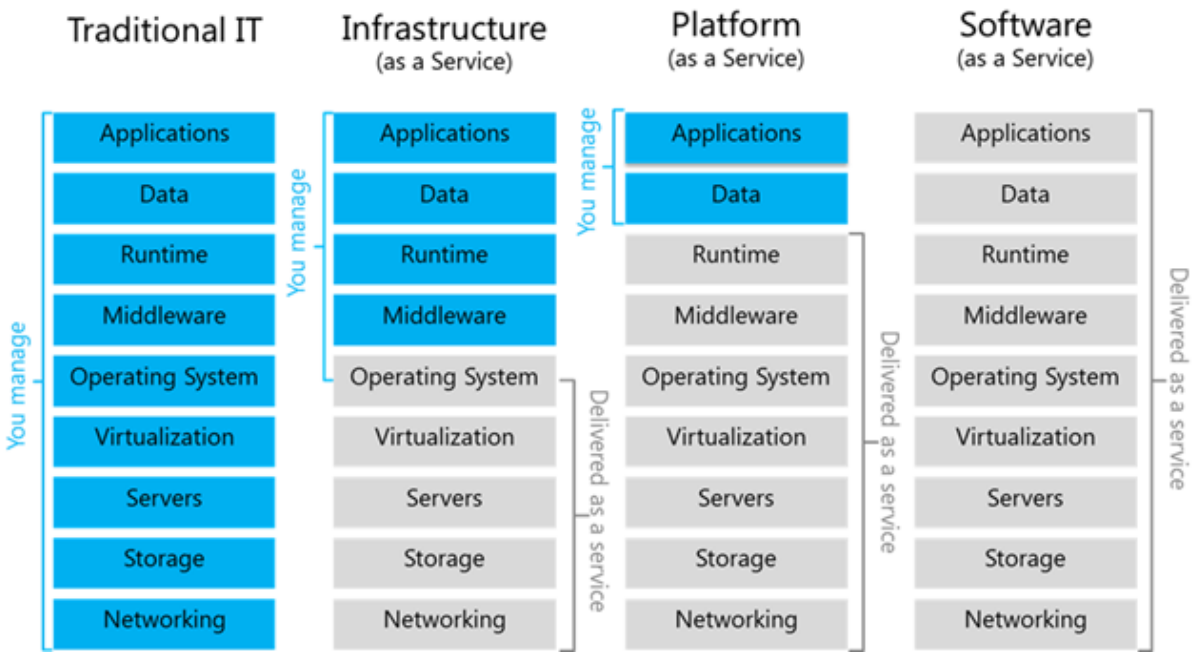

Abb. 2.1: Die grundlegenden Cloud Service Modelle im Überblick

Infrastructure as a Service: Der Nutzer hat die Fähigkeit Rechenleistung, Datenspeicher Netzwerke und weitere fundamentale Computerresourcen 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 seine eingkaufte oder selbst erstellten Anwendungen auf der Cloud Infrastruktur zu betreiben, die notwendige Umgebung die über Sprachen, Bibliotheken, Tools und Services verfügt wird vom Anbieter bereitgestellt. Die darunter liegende Infrastruktur mit Netzwerken, Servern, Betriebssystemen und Speicher wird vom Anbieter betrieben, der Nutzer hat die Kontrolle über die Anwendung und Konfiguration der Umgebung in der die Anwendung betrieben wird.

Software as a Service: Dem Nutzer wird der Zugriff auf die vom Anbieter in der Cloud Infrastruktur betriebenen Softwareanwendung gewährt. Auf diese wird mithilfe eines Thin oder Fat Clients zugegriffen, dabei kümmert sich der Nutzer nicht um den Betrieb und die Konfiguration der darunterliegenden Cloud Infrastruktur wie Netzwerke, Server, Betriebssystem, Speicher und die Anwendung selbst mit Ausnahme eingeschränkter Nutzereinstellungen.

Die von NIST unterschiedenen grundlegenden Service Modellen können noch weiter differenziert werden. Ein Beispiel hierfür ist das Modell Function as a Service (FaaS) als Subset von PaaS. FaaS erlaubt es Programmcode auszuführen ohne sich um die Bereitstellung weiterer Infrastruktur kümmern zu müssen wie es der Betrieb eines gewöhnlichen Microservices verlangt.

In Art der Bereitstellung eines Cloud Services werden vier grundlegende Modelle unterschieden; 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 Premise 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 Premise 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 Premise betrieben 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 Zu erwägende 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 Unternehmen selbst eingakuft, eingerichtet und betrieben werden muss (vgl. linke Seite Abb 2.1). Potentiell hohe Kosten die bereits vor der Inbetriebnahme eines Systems mit einem höheren Risiko aufgewendet werden mussten stellen in Form von individuell geringeren laufenden Beträgen ein deutlich reduziertes Risiko da.
Sofern der Einsatz von Cloud Computing in einer sinnvollen und korrekten Weise erfolgt können je nach Fall die Gesamtkosten um einen hohen Anteil reduziert werden.
Die Gesamtkostenersparnisse stehen auch im Zusammenhang mit Skaleneffekten die für große Cloudanbieter gelten. Der Betrieb eines einzelnen Servers ist im Verhältnis mit bedeutend höheren Kosten verbunden als das hinzufügen eines äquivalenten System zu einem Rechenzentrum im Betrieb von AWS oder einem vergleichbaren Anbieter.

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

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

Security: Security Aspekte können sowohl einen Vor- als auch Nachteil von Cloud Computings darstellen. Hier sollen zuerst Vorteile dargelegt werden, potentielle Probleme werden im nächsten Unterkapitel beschrieben. Die technischen Möglichkeiten und besonders auch die Wahrnung des Themas Sicherheit in der Cloud unterlagen und unterliegen auch immernoch einem deutlichen Wandel. Cloud Anbieter investieren viele Resourcesn in Sicherheit und stellen dem Nutzer zum Beispiel bereits sicher implementierte Verschlüsselungen zur Verfügung oder bieten einen gewissen Schutzen vor Denial-of-Service Angriffen durch die natürliche Skalierbarkeit.

Nachteile und Risiken

Netzwerkabhängigeit: Da der Zugriff auf Cloud Dienste über das Internet erfolgt entsteht dadurch entsprechend auch eine hohe Abhängigkeit. Eine stabiele und schnelle Netzwerkanbindung ist vorraussetzung dass effektiv gearbeitet werden kann, 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 einzelenen Anbieters zu begeben. Im Fall einer Änderung der Nutzungsbedingungen oder einer Änderung im Bezahlmodell die den eignen 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 Bedingugen zu akzeptieren.

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

Kosten: Auch wenn die Nutzung von Cloud Computing mit dem Vorteil geringerer Kosten beworben ist dies nicht zwingend garantiert. Werden die vorhandenen Systeme ungünstig verwendet, zum Beispiel bleiben viele gebuchte Ressourcen ungenutzt und IP-Adressen unverwendet, können hohe Kosten entstehen, auch in der Phase des Übergangs zu Cloud Computing können höhere Kosten entstehen als in einem vergleichbaren Zeitraum davor.

2.1.3 Grundlegende Erklärung von DevOps

Die exakte Definition und Abgrenzung des Begriffes DevOps ist ein Thema über das es keine uniform akzeptierte allein gültige Definition und Abgrenzung gibt. Im allgemeinen aber bezeichnet DevOps eine stärkere Vereinigung der Entwicklungs- (Dev) und Betriebs- (Ops) Teams eines Softwareprojekts durch die Anwendung einer effektiveren Arbeitskultur und -philosophie mit neuen Methoden, Werkzeugen und Prozessen. Verschiedene Organisationen legen hierbei in ihrer Definition die Schwerpunkte auf unterschiedliche Aspekte. Als Beispiel betont Amazon hierbei besonders die schnelle Auslieferung neuer Produkte (äbility to deliver applications and service at high velocity"), Microsoft die Kollaboration zwischen den beteiligten Teams ("DevOps enables formerly siloed roles - development, IT operations, quality engineering, and security - to coordinate and collaborate to produce better, more reliable products.").
Das DevOps Reasearch and Assessment (DORA) Team hat über sieben Jahre mit 32.000 Beteiligten die Praktiken und Fähigkeiten untersucht die hohe Leistungsfähigkeit bei Entwicklung und Auslieferung von Sofware fördern. Eine Übersicht über die Erkenntnisse

dieser Studie ist als Diagramm im Anhang ... zu finden.
Infrastructure as Code spielt als Werkzeug für die automatisierte Bereitstellung der benötigten Computerresourcen eine wichtige Rolle im DevOps Prozess.

GitOps als Werkzeug für Continuous Delivery im Rahmen von DevOps

GitOps wird von Gitlab als ein betriebliches Rahmenkonzept (operational Framework) defniert das DevOps Best Practices in der automatiserten Bereitstellung von Infrastruktur anwendet. GitOps benötigt dabei drei Hauptkomponenten:
Infrastructure as Code, Merge Requests und CI/CD. Organisationen die mit einer ausgereiften Anwendung der DevOps Kultur arbeiten können über GitOps hunderte Male pro Tag neuen Code auf den Produktionsservern installieren. Im praktischen Teil dieser Arbeit wird GitOps verwendet um da es eine einfache Integration mit Github ermöglicht, Github selbst wird als Versionskontrollsystem verwendet da es weithin etabliert und breit unterstützt ist und von der Novatec Consulting GmbH bevorzugt eingesetzt wird.

2.1.4 Überblick über die wichtigsten Public Cloud Service Provider

Da der Kern der Arbeit die Infrastructure as Code Unterstützung der wichtigsten Cloud Service Provider in deren Public Cloud untersucht soll hier ein kurzer Überblick über den aktuellen Markt in diesem Bereich gegeben werden.

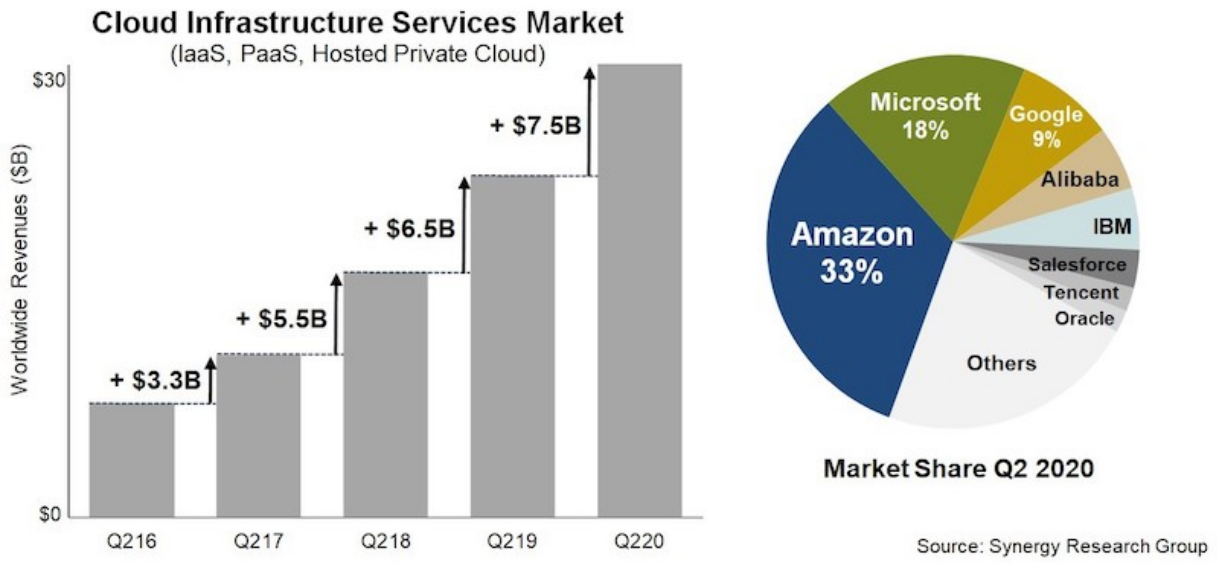

Abb. 2.2: CSP Market Share Q2 2020 nach Umsatz

1. Amazon: Amazon besitzt mit 33% den größten Anteil am Markt. Die Amazon Elastic Compute Cloud (EC2) ist die älteste der aktuell großen Cloud Computing Plattformen, der erste Release fand im August 2006 statt.

2. Microsoft: Microsoft stellt den zweitgrößten Anteil am Markt mit 18%, der erste Release von Microsoft Cloud Computing Service Microsoft Azure erfolgte im Oktober 2008. Gemeinsam besaßen Amazon und Microsoft 2020 über 50% des gesamten Marktes.

3. Google: Google's Google Cloud Platform steht mit 9% an dritter Stelle. Google Compute Engine, die IaaS Komponente der Cloud Services von Google wurde im Juni 2012 veröffentlicht.

4. Alibaba Cloud: Die viertgrößte Cloud Plattform ist die Alibaba Cloud der Alibaba Group. Alibaba Cloud bietet zusätzlichzu seinem Elastic Compute Service (ECS) einen dedizierte GPU basierten Service an.

5. IBM Cloud: Die aktuell fünftgrößte Cloud Computing Plattform ist die IBM Cloud von IBM. Bis Juni 2013 eine eigene Firma unter dem Namen Softlayer wurde diese von IBM übernommen und 2017 gemeinsam mit den anderen Cloud Services der Firma in ein einheitliches Portfolio unter dem Namen IBM Coud aufgenommen.

Abb. 2.3: CSP Gartner Magic Quadrant August 2020

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 Vormachtsstellung von Amazon gegenüber Microsoft und Google sowie der Abstand von Google zum nächst größten Anbieter Alibaba Cloud. TODO: Irgendwas mit IBM Ability to Execute vs Marktanteil verglichen mit Oracle

2.2 Infrastructure as Code

Infrastructure as Code beschreibt einen Ansatz zur Automatiesierung der Bereitstellung von Infrastruktur basierend auf Methoden aus der Softwareentwicklung (vgl Kief Morris Infrastructure as Code S.4).
Statt eines manuellen Aufbaus und direkter 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. (vgl. https://docs.microsoft.com/us/devops/deliver/what-is-infrastructure-as-code)

2.2.1 Technologischer Wandel und das Cloud Age Mindset

Durch die Technologien der Cloud ist es möglich eine gewünschte IT-Infrastruktur sehr viel schneller bereitzustellen als zuvor möglich. Statt des Einkaufs, Anschließens und Einrichtens eines physischen Servers das, je nach Szenario einen Zeitraum von mindestens mehreren Stunden oder Tagen bis zu Wochen gedauert hätte können virtuelle Resourcen in der Cloud in wenigen Minuten bereitgestellt werden. Der schnellere Ablauf wird durch die Automatisierung von Prozessen wie etwa der Bereitstellung der Infrastruktur mithilfe von IaC Tools noch verstärkt, nicht nur bei der initialen Mit diesen Veränderungen wird das Management und die Erweiterung der bestehenden Systeme jedoch nicht unbedingt einfacher (vgl IaC Kief Morris S.3), die Verwendung der alten IT-Governance Modelle (TODO Fußnotenerklärung) die sich bisher bewährt haben sind jedoch aufgrund der Veränderungen ungeeignet. Kief Morris stellt die fundamentalen Unterschiede des Arbeitens mit Cloud-Technologien mithilfe der folgenden Tabelle 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

Abb. 2.4: Iron Age vs Cloud Age

Veränderungen in der ’Iron Age’ sind aufwändig und teuer und stellen ein Risiko dar, es wird versucht diese Risikopunkte zu reduzieren daher werden viele Veränderung gebüdelt

getestet und eingeführt wodurch lange Release-Zyklen entstehen. Die Architekturen die dadurch befördert werden sind eher monolytisch die Konfiguration erfolgt eher mithilfe von GUI gesteuerten Programmen oder physischen, 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 die 'Iron Age' zugeschnittene Arbeitsweise für die Cloud nicht effektiv ist, es ist ein neues Cloud Age Mindset das die rechte Spalte der Tabelle verinnerlicht erforderlich um die Vorteile der Cloud wirklich effektiv nutzen zu können.

2.2.2 Vorteile von IaC und durch IaC gelöste Probleme des manuellen Provisionings

Kein Configuration Drift durch einheitliche Codebase: 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 oftmals auch leicht verschiedene Anforderungen auf die dann mit Optimierungen, etwa in Form spezifischer Konfigurationsdetails, reagiert werden kann. Wird nun das ursprüngliche System geupdated werden individuelle, oft undokumentierte Anpassungen nicht berücksichtigt und ein Update kann unbequeme Konsequenze nach sich ziehen. Werden alle Veränderungen in einer einheitlichen Codebase verwaltet und Updates häufig vorgenommen verhindert man starken Configuration Drift der über Zeit stattfindet

Wiederverwendbarkeit durch einheitlichen Code: Ein weiterer Vorteil der durch die Verwendung einer einheitlichen Codebase entsteht ist die Wiederverwendbarkeit und Repoduzierbarkeit eines Systems. Wenn ein identisches System an einer anderen Stelle aufgebaut werden soll oder sollte das System aus unvorhergesehen Gründen in seiner Gesamtheit verloren gehen 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.

Schnelleres Provisioning durch Cloud: Ein bereits häufig angesprochener Vorteil ist die schnelle Bereitstellung, auf tiefere Erklärung kann daher hier verzichtet werden.

Schnellerer Profit: Schnellere Bereitstellung der Hardware und schnellere Auslieferung von Software die durch DevOps Methoden begünstigt werden erzeugen entsprechend auch schneller einen Mehrwert.

Einheitliches Tooling in Dev, Ops und weiteren Beteiligten Teams: Verwendung von IaC fördert die einheitlicheres Tooling in allen Bereichen die mit einem Softwareprodukt in Zusammenhang stehen. Cloud Technologien fördern und ermöglichen diese Ver-

einheitlichung, manuelles Provisioning ohne ein Cloud Modell erschwert dies oder macht es sogar unmöglich.

Stärkere Automatisierung im Arbeitsablauf: Automatisierung bedeutet immer einen gewissen upfront Overhead, jedoch kann auf längere Sicht deutlich von automatisierten Abläufen profitiert weren. Ein Beispiel hierfür sind automatisierte und in eine CI/CD Pipeline integrierte Tests.

Höhere Zuverlässigkeit und Sicherheit durch schnelle Updates: Von eine Struktur die schnelle und häufige Veränderungen fördert kann auch die Sicherheit und Zuverlässigkeit eines Systems profitieren. Auf Sicherheitsrisiken kann in kurzer Zeit und ohne viel Aufwand reagiert werden, eine unsichere Funktion kann etwa schnell durch eine sichere Variante ausgetauscht werden, mögliche damit verbundene Probleme können dann in automatisierten Tests erkannt werden wodurch das System zuverlässig bleibt.

Schnellere Fehlersuche und -behebung:

Fehler die dennoch auftreten können dann durch den Einsatz kleinerer Komponenten schneller isoliert, gefunden und behoben werden. Monolithische Systeme erschweren dies durch dadurch dass weniger klar isolierte Komponenten vorhanden sind die häufig mehr Abhängigkeiten aufweisen und daher schwerer als einzelne Einheit getestet werden können.

2.2.3 Herausforderungen und Argumente gegen den Einsatz von IaC

Kief Morris bennent drei Argumente die gegen die Einführung von IaC genannt werden, diese sollen hier Im Kontext der gennanten Vorteile betrachtet werden.

Veränderugen werden nicht oft genug durchgeführt um Automatisierung zu rechtfertigen.

Die Idee dass ein System einmal erstellt und dann "fertigïst und Automatisierung der Veränderungen daher überflüssig ist entspricht sehr selten der tatsächlichen Realität. IT-Systeme und damit auch IT-Infrastruktur wird während ihrem gesamten Lebenszyklus mehr oder weniger kontinuierlich verändert und erweitert.
Sicherheislücken in alten Versionen von Softwarepackages oder Betriebssystemen sind keine Seltenheit und müssen gepatcht werden um einen sicheren und Zuverlässig Betrieb zu gewährleisten, neue Features in bestehender Software kann neue Infrastruktur, zum Beispiel in Form eines neuen Datnbankservers, notwendig machen oder eine veränderte Konfiguration ist nötig um die Performance zu steigern. 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 das System erfährt, daher sollte es möglich schnell und unkompliziert das System zu verändern indem mehr Resourcen zur Verfügung gestellt werden.

Ä fundamental truth of the Cloud Age is: Stablity comes from making changes."(IaC Buch S. 5)

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

Infrastruktur soll zuerst aufgebaut, danach automatisiert werden.

Die Umsetzung von Infrastructure as Code stellt eine durchaus große Herausforderung, das Erlernen eine ßteep Curve"(IaC Buch S.6) dar, deren Nutzen nicht unbedingt direkt ersichtlich ist. Dadurch kommt es zu Situationen in denen es attraktiv erscheint Infrastruktur zuerst aufzubauen sich erst später um die Automatisierung zu kümmern. Mit diesem Ansatz werden viele der Vorteile von IaC jedoch verwirkt und die Schwierigkeit der Umsetzung von IaC fällt deutlich größer aus, verglichen mit einem Projekt das von Beginn auf IaC ausgelegt ist da Äutomatisierung Teil des Designs und der Implementierungeines Systems ist"(IaC Buch S.6).
Automatisierung mit IaC vereinfacht auch die Implementierung autoamtisierter Tests eines Systems was es wiederum erlaubt Fehler schneller zu beheben. Ist dies bereits von Beginn an Teil des Arbeitsprozesses verbessert dies die Qualität der Infrastruktur.

TODO: Wenn erfolgreich werden neue Features gefordert, ohne Automatisierung gerät Management der wachsenden Infrastruktur schnell außer Kontrolle Deliver steady stream of value , build system incrementally

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. Die Äccelerate State of DevOps ReportSStudie kam zu dem Schluss das Organisationen dazu tendieren entweder gut oder schlecht in beiden Kriterien abzuschneiden, wird eines vernachlässigt führt es am Ende in der Regel zu einem "fragile Mess"wie die untenstehende Tabelle erläutert.

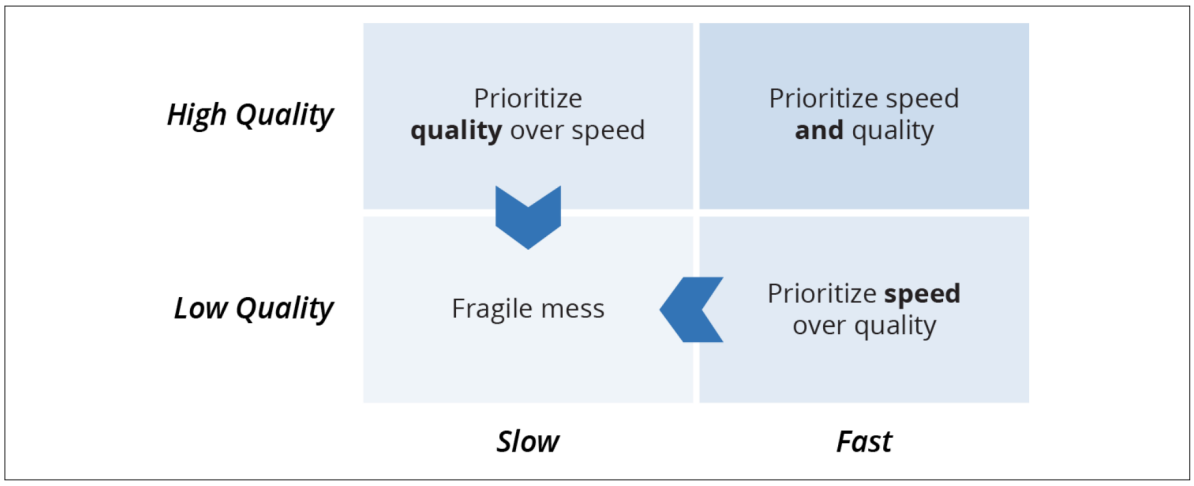

Abb. 2.5: Iron Age vs Cloud Age IaC Busch S.6

Wird Geschwindgkeit über Qualität gestellt (Quadrant links oben) entstehen mit der Zeit chaotische und instabiele Systeme an denen Veränderungen mit der Zeit nur noch erschwert und deshalb langsam durchgeführt werden können.
Wird Geschwindigkeit zu niedrig priorisiert führt es allerdings auch dazu dass letzten Endes durch Druck von Deadlines und schnellen workarounds technische Schulden aufgebaut werden die ebenfalls zu einem qualitativ schlechten System führen da notwendige Veränderungen nicht schnell genug eingearbeitet werden können.

Aufgrund dieser Probleme ist es wichtig Geschwindgkeit und Qualität gleichermaßen zu priorisieren, moderne Entwicklungsansätze DevOps haben das Ziel dies zu erreichen, der Erfolg wird von der Accelerate Studie belegt.

2.2.4 Die drei Kernverfahren von IaC

Kief Morris identifiziert drei Kernverfahren ("Core Practices") zur Implementierung von Infrastructure as Code:

- **Define everything as Code:** Alle Teile eines System in Form von Code zu definieren bringt mehrere Vorteile mit sich.
  Code kann mehrfach ausgeführt werden, daher ist eine als Code definieres System Wiederverwendebar. 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 vohersehbar, 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 des Systems entspricht und damit auch dokumentiert.

- **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 ëinzutestenßondern von Beginn and und kontnuierlich einzubauen (The idea is to build quality in rather than trying to test quality in. (vgl IaC Buch S.10)). Nach der Accelerate Studie fördert es die Qualität der Arbeit eines Teams wenn jedes Mitglied mindestens einmal am Tag den eigenen Fortschritt integriert.
- **Build Small, Simple Pieces That You Can Change Independently:** Systeme die aus mehreren kleineren voneinander unabhängigen Komponenten bestehen sind generell stabieler als große Monolithen. Eine Änderung die einen Fehler verursacht betrifft nur diese 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 restliche System auf eine ältere Version zurückgesetzt werden muss um den Betrieb wieder herzustellen.

2.2.5 Die 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.
In der Einrichtung wird die Infrastruktur bereitgestellt und konfiguriert, genauso wird auch Software intalliert und eingerichtet.
Nachdem das System dann in Betrieb genommen wurde können Anpassung notwendig werden, Server werden hinzugefügt und abgebaut, Software wird aktualisiert und neu konfiguriert.

Überblick über die wichtigsten Infrastructure as Code Tools

Infrastructre 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 ein Beispiel für ein spezialisierteres Tool. Die untenstehende Abbildung soll einen Überblick über die aktuell relevantesten Tools 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.

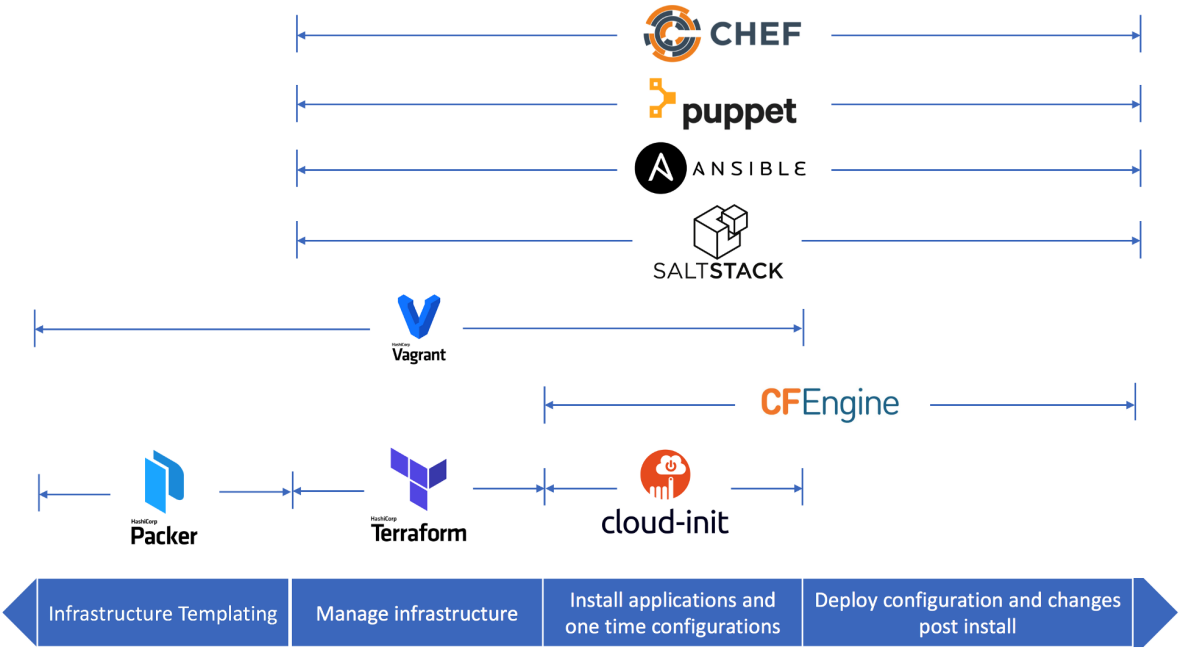

Abb. 2.6: Überblick IaC Tools https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde

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 bietet sich der Einsatz eines darauf spezialisierten Tools an.
Hier gibt es zusätzlich zu Terraform eine weitere prominente Alternative: Pulumi.

Pulumi bietet im Vergleich zu Terraform mehrere Vorteile, es gibt allerdings auch diverse Nachteile die, gegen den Einsatz von Pulumi sprechen. Die folgenden Vorteile bietet Pulumi gegenüber Terraform:

- **Sprache:** Im Gegensatz zu Terraforms Domänen-spezifischer HashiCorp Configuration Language (HCL) setzt Pulumi auf den Einsatz bekannter General-Purpose Programmiersprachen, Python, TypeScript, JavaScript, Go, C# und F#, werden aktuell unterstützt.
  Durch den Einsatz dieser Sprachen können Schleifen, Funktionen und weitere bekannte Konstrukte verwendet werden, Terraform bietet diese Möglichkeiten nicht, ähnliche Effekte werden nur mit Workarounds erreicht.
- **Testing:** Um Terraform-code testen zu können werden Third-party Tools benötigt, Terraform selbst bietet kein Test-Framework. Durch den Einsatz bekannter General-Purpose Programmiersprachen in Pulumi ist es auch möglich die dort zum Einsatz

kommenden Frameworks zu verwenden, Integrationstest sind allerdings nur in Go unterstützt (https://phoenixnap.com/blog/pulumi-vs-terraform).

- **Pulumi mit Terraform:** Pulumi kann kann sowohl lokale als auch remote Terraform State Files lesen und verarbeiten. Dies erlaubt es Pulumi und Terraform nebeneinander einzusetzen um verschiedene Teile der Infrastruktur mit dem jeweils anderen Tool zu managen, soll zum Beispiel einer Umstellung von Terraform auf Pulumi erfolgen ist dies möglich ohne das gesamte System auf einmal auser Betrieb nehmen zu müssen.

Terraform wiederum bietet die folgenden Vorteile:

- **Modularität:** Terraform Module erlauben es ein System in mehrere klar definierte Komponenten zu strukturieren, dadurch wird die Wiederverwendbarkeit dieser Komponenten ermöglicht und gefördert. Die daraus resultierenden Vorteile wurden bereits in den vorangegangenen Kapiteln dargelegt, daher soll hier nicht weiter darauf eingegangen werden. Pulumi strukturiert Infrastruktur Code entweder in einem großen monolithischen Projekt oder vielen kleinen Mikroprojekten, beide Optionen sind weniger flexibel als die Lösung die Terraform bietet.
- **State Debugging:** Es ist beinahe unvermeidbar dass es während der Arbeit an einem Infrastrukturprojekt über einen längeren Zeitraum irgendwann einmal zu einem Fehler im State kommt. Sowohl Terraform als auch Pulumi bieten hier CLI Befehle die bei der Fehlersuche und -behebung unterstützen, Terraform bietet hier jedoch aktuell noch mehr Optione mit denen Resourcen aus dem aktuellen State gelöscht oder hinzugefügt werden können, dadurch wird weniger händische, und potentiell fehleranfälligere Arbeit direkt im State File notwendig.
- **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 Lernresourcen, mehr Codebeispiele, größere Bekanntheit und mehr Jobs für die Arbeit mit Terraform.
- **Dokumentation:** Einen weiteren Voteil von Terraform stellt dessen umfangreiche und ausgereifte Dokumentation sowie auch die Dokumentation der einzelnen Provider dar. Der genaue Aufbau der einzelnen Resourcen 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 erwartet wird.

-Terraform unterm Strich aktuell interessanter für große Projekte/Unternehmen => daher Wahl für diese Arbeit

- Ein wirklich fundierter Vergleich zwischen Terraform und Pulumi benötigt deutlich mehr Zeit und Anspruch als im Rahmen dieser Arbeit möglich, für die in diesem Fall und

zu diesem Zeitpunkt zu treffende Entscheidung soll der grobe Vergleich hier ausreichen. Ein umfangreicherer Vergleich zwischen Terraform und Pulumi könnte in zwei Jahren ein anderes Ergebnis liefern als heute, beide Technologien und insbesondere Pulumi sind noch relativ neu und nicht vollständig ausgereift.

Weitere zu beachtendende Aspekte

Ein Aspekt über Terraform der häufig falsch interpretiert und verbreitet wird ist dass Terrafom als Cloud agnostisch beschrieben ist. Dies ist NICHT der Fall. Die einzelnen Terraform Provider welche jeweils die Abstraktion der Cloud Service Provider tatsächlich vornehmen sind nicht austauschbar das heißt der Terraform Code muss für jeden Cloud Service Provider individuell geschrieben werden, Terraform Code ist daher nicht Cloud agnostisch.

(- Secrets Management? Terraform Enterprise?)

2.2.6 Terraform Funktionsprinzip

Terraform ist ein Infrastructure as Code Tool das es ermöglicht Infrastruktur sicher und effizient aufzubauen, zu verändern und zu versionieren.
Terraform verwendet eine High-level DSL die es erlaubt die Infrastruktur in deklarativer und menschenlesbaren Konfigurationsdateien zu beschreiben die versioniert, geteilt und wiederverwendet werden können.
Bevor Veränderungen vorgenommen werden erzeugt Terraform einen Execution Plan der beschreibt welche Veränderungen durchgeführt werden sollen und überprüft werden kann bevor er ausgeführt wird.
Ein Resourcengraph der Abhängigkeiten erfasst erlaubt es voneinander unabhängige Resourcen parallel und dadurch effizient zu erzeugen und gibt einen besseren Einblick in den Aufbau des beschriebenen Systems.
Automatisierte Änderungen erlauben es komplexe Veränderungen an der Infrastruktur mit minimaler menschlicher Interkation durchzuführen. Terraform beachtet bestehende Abhängigkeiten und nimmt Veränderungen durch Execution Plans schrittweise vor bis der definierte Zustand erreicht ist. (Alles obere vgl https://www.terraform.io/intro/index.html)

Die folgende Abbildung stellt das Funktionsprinzip von Terraform mit den wichtigsten beteiligten Komponenten dar.

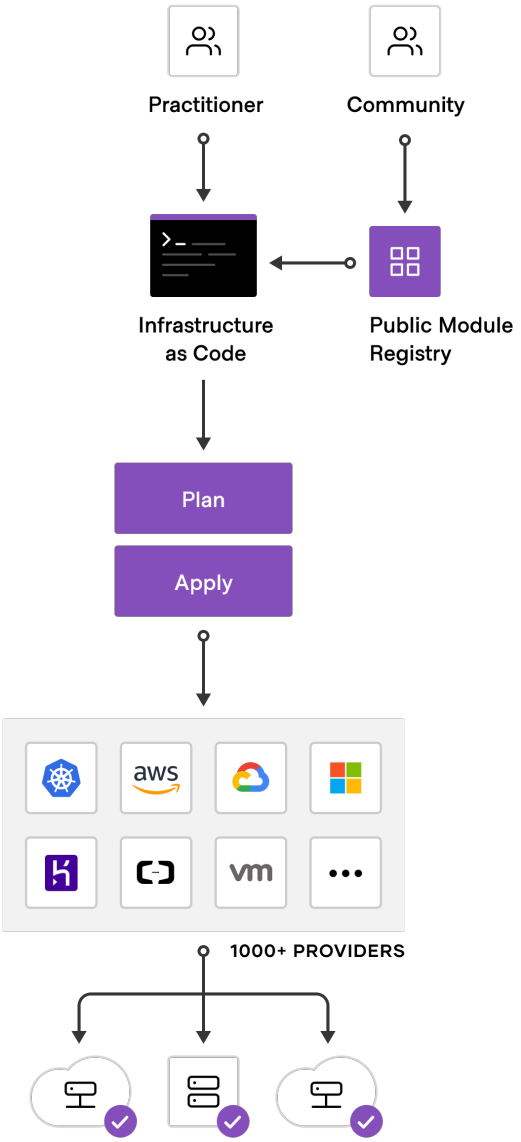

Abb. 2.7: Terraform Funktionsprinzip

Der Terraform Anwender schreibt den Infrastruktur Code in HCL. Dabei kann er auf vorhandene Module aus der Public Module Registry zurückgreifen um typische Strukturen wie Netzwerke aufzubauen. Die Terraform Core-Software nutzt dann Plugins, die Terraform Provider, um die jeweiligen Cloud Plattform API's anzusprechen die dann die Infrastrktur aufbauen. Die aktuell existierenden Resourcen sind dabei in der terraform.tfstate Datei festgehalten, gemeinsam mit dem aktuellen State und dem Infrastruktur Code ermittelt Terraform welche Änderungen vorgenommen werden müssen um den im Code beschriebenen Zustand zu erreichen.

Hashicorp Configuration Language

Die von Terraform verwendete Konfigurationssprache wurde mit dem Ziel entwickelt einen Kompromiss zwischen Maschinenfreundlichkeit und Menschenlesbarkeit zu erzielen. 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 zum Einsatz.

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

TODO Codebeispiel

Blöcke stellen für gewöhnlich ein Objekt, im Fall von Infrastrkturcode meistens eine Computing Resource, 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 der einem 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 For eines Ausdrucks.

Input und Output Variablen

Input Variables 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 Inputvariablen enthält angepasst werden, alles andere kann ohne Veränderungen wiederverwendet werden.

TODO Codebeispiel

Inputvariablen sind einfache Key-Value Paare die einen Standardwert und eine optionale Beschreibung besitzen.

Outputvariablen werden in der Regel verwendet um auf spezifische Parameter einfachen und schnellen Zugriff zu ermöglichen.

TODO Codebeispiel

Werte wie die IP einer Virtuellen Maschine auf einer Public Cloud Plattform sind vor der Bereitstellung dieser nicht bekannt, werden aber häufig benötigt weshalb es nützlich ist eine Output Variable für diese zu deklarieren.

Terraform Module

Module sind Container für mehrere Resourcen die gemeinsam verwendet werden. Jedes Terraform Projekt besitzt ein Root Module das weitere Module verwenden kann. Die Terraform Registry stellt eine Vielzahl an veröffentlichten Modulen zur öfentlichen Verwendung bereit.

Terraform Provider

Terraform benötigt Plugins, die Provider, um mit Cloud Providern, SaaS Providern und anderen API's interagieren zu können. (vgl https://www.terraform.io/docs/language/providers/index.h Provider ermöglichen es Terraform für gewöhnlich einen bestimmten Provider zu konfigurieren, es gibt zum Beispiel einen Provider für Google Cloud Platform, es gibt aber auch Provider die eine lokales Utility Tool für die Nutzung durch Terraform konfigurieren. Öffentliche Provider werden primär auf der Terraform Registry bereitgestellt und dokumentiert.

Terraform Workflow

Der grundlegende Terraform Workflow besteht aus vier Schritten:

TODO Code terraform init

Das init Kommando installiert und konfiguriert die notwendigen Provider und Module und konfiguriert ein Backend (Fußnote Backend) falls angegeben.

TODO Code terraform plan

Mit terraform plan wird ein Execution Plan erstellt. Dazu gehört den die aktuell existierenden Resourcen zu erfassen, Veränderungen zwischen diesem Zustand und dem im Code konfigurierten Zustand zu erfassen und auf Grundlage dessen einen Plan zu erstellen welche Änderungen vorgenommen werden können um den Soll-Zustand zu erreichen.

Todo Code terraform apply

Durch terraform apply wird ein solcher Execution Plan ausgeführt und die darin enthaltenen Änderungen an der Infrastruktur vorgenommen.

Todo Code terraform destroy

Um den umkomplizierten Abbau eines mit Terraform definierten und aufgebauten Systems bewerkstelligen zu können wird das terraform destroy Kommando verwendet. Dies ist besonders nützlich um nicht mehr benötigte Testsysteme und andere temporäre Infrastrukturen zu entfernen.

3 Aufbau und Untersuchung

3.1 Evaluierungsanforderungen

3.1.1 Ziel der Evaluierung

Das Ziel der Evaluierung ist es einen Vergleich zwischen den Terraform Providern von Amazon AWS, Google Cloud Platform und Microsoft Azure zu erstellen. Der Hintergrund dazu ist es als erstes zu ermitteln ob alle drei der großen Cloud Plattformen Infrastructure as Code mit Terraform für ein gegebenes Szenario unterstützen. Weiter sollen mehrere Kriterien verglichen werden um zu ermitteln ob alle Plattformen vergleichbare Ergebnisse liefern, oder ob eine Plattform durch besonders gute oder schwache Ergebnisse heraussticht. Funktionieren bestimmte Aspekte (wie zum Beispiel die Containerimage-Registry) unterschiedliche soll dies ebenfalls erfasst und bewertet werden.
Die Untersuchung soll auch dazu dienen die Effektivität von Infrastructure as Code mit Terraform zu demonstrieren. Zusätzlich soll dargestellt werden dass die Technologie einsatzreif für Verwendung in Produktionssystemen ist.

3.1.2 Zu untersuchende Komponenten der Terraform Provider

Die Komponenten der Terraform Provider die Untersucht werden sollen hängen von dem System ab das als Untersuchungsobjekt genutzt werden soll. Dieses setzt sich aus einem Netzwerk, einer Container-Registry, einem Kubernetes-Cluster und einem Datenbankserver mit mehreren Datenbanken sowie den dazugehörenden Firewalls zusammen. Zusätzlich sollen einfache virtuelle Maschinen betrachtet werden um ein vollständigeres Bild zu erhalten.

3.1.3 Grenzen und Limitierungen der Evaluierung

Die Terraform Provider können weitaus mehr Computing Ressourcen erzeugen als in dieser Arbeit untersucht werden. Aufgrund des Umfangs können nicht alle Ressourcen verglichen werden, die hier verwendeten wurden ausgewählt um die grundlegendsten Szenarien zu untersuchen. Da die durch die verschiedenen Cloud Provider bereitgestellten Computing Ressourcen jedoch nicht immer identisch sind ist ein perfekter Vergleich nicht möglich.

Stellen die Cloud Provider nur Virtuelle Maschinen mit unterschiedlicher Leistungsfähigkeit bereit wird jedoch darauf geachtet möglichst ähnliche Konfigurationen zu verwenden um den Vergleich so identisch wie möglich zu gestalten. Das selbe gilt für die Größen von Festplatten und allen anderen vergleichbaren Objekten. Stehen nur sehr teure identische Maschinen bereit werden aus Kostengründen ebenfalls günstigere aber möglichst ähnliche Konfigurationen gewählt. Die genauen Daten der verglichenen Infrastrukturobjekte werden jeweils bei den Vergleichswerten aufgelistet.

3.2 Spezifizierug des Vergleichs

3.2.1 High-level Aufbau der Infrastruktur des Versuchsobjekts

Als Testobjekt dient ein Infrastruktursystem das typische Anwendungsfälle repräsentieren soll. Es besteht aus Netzwerk, mehreren Datenbanken und einem Kubernetes Cluster mit mehreren Worker Nodes. Das System ist eine leicht veränderte Variante des öffentlich einsehbaren Infrastruktur-Showcase der Novatec. Ergänzend zu diesem System sollen zusätzlich einige Tests mit einzelnen Ressourcen wie zum Beispiel Virtuellen Maschinen durchgeführt werden.
Weiterhin existiert ein bereits manuell erstellter Object Storage mithilfe dessen auf einer ebenfalls manuell erstellten Container Registry das Dockerimage für den Kubernetes Cluster bereitgestellt wird. Dieses Image bzw. die Registry wird dann im Terraform Projekt als Data Source erfasst.

Der originale Infrastrktur-Showcase besteht aufgrund der verhältnismäßig geringen Größe und Komplexität aus einem einzelnen Terraform Modul "Platform"das die gesamte Infrastruktur enthält. Für diese Arbeit wurden die Kubernetes- und Datenbank-Komponenten zu Demonstrationszwecken in jeweils eigene Module ausgelagert. Diese Module könnten in eine öffentliche oder auch private Terraform Module Registry hochgeladen werden um später in weiteren Projekten verwendet werden zu können.

TODO: Abbildung erstellen und einfügen.

3.2.2 Auswahl der zu untersuchenden Aspekte und Eigenschaften

Das Softwarequalitätsmodell der ISO 25010 wird als ein Grundstein für die Evaluierung von Software beschrieben (Vergleich https://iso25000.com/index.php/en/iso-25000-standards/iso-25010) Da die Terraform Provider statt einer ganzheitliche Software nur Plu-Ins und damit eine kleinere Komponente von Terraform darstellen kann nicht jeder der in der ISO 25010 aufgelistet Punkte sinnvoll angwendet werden. Einige Bereiche, zum Beispiel der Bereich Security mit all seinen untergeordneten Themen, benötigen eine deutlich genauere Betrachtung als es im Rahmen dieser Arbeit möglich ist um ein sinnvolles und vollständiges Ergebnis gewährleisten zu können.

Aus diesem Grund sollen nur ein kleinerer Teil der aufgeführten Merkmale betrachtet werden, interessant sind vor allem solche die eine schnelle Einführung von Terraform in den Entwicklungsprozess beeinflussen. Ebenfalls interessant sind die Flexibilität und Wartbarkeit da insbesondere letztere einen der großen Vorteile darstellt die man durch den Einsatz von Infrastructure as Code Tools- und Prinzipen erzielen kann.

Abb. 3.1: Überblick ISO 25010

Im Folgenden sollen die einzelnen Merkmale der ISO 25010 kurz zusammengefasst werden und erläutert werden ob diese in der Evaluierung der Terraform Provider sinnvoll betrachtet werden können und ob eine Betrachtung im Rahmen dieser Arbeit vorgenommen werden kann.

Functional Suitability

- **Functional completeness**: Die Vollständigkeit mit welcher die Aufgaben und Ziele die vom Nutzer an die Software gestellt werden erfüllt werden können.

  Anwendbarkeit: Eine zufriendenstellende Functional Completeness ist die grundlegende Vorraussetzung für den Einsatz jeder Software, dies gilt auch für den Einsatz von Terraform und die zu untersuchenden Provider.

- **Functional correctness**: Grad zu welchem die Software korrekte Ergebnisse mit einer ausreichenden Genauigkeit liefert.

  Anwendbarkeit: Da im Falle der Bereitstellung von Infrastruktur durch deklarativen Code wenig Spielraum zulässt und im Fall der hier untersuchten Systeme lediglich ein „richtig“ oder „falsch“ zulässt spielt dieser Punkt keine Rolle.

- **Functional appropriateness**: Grad zu welchem die Funktionalitäten der Software die Erfüllung der Aufgaben ermöglichen und unterstützen.

  Anwendbarkeit: Da die Terraform Provider im Prinzip nur eine Schnittstelle zu den jeweiligen Plattformen darstellen ist dieser Aspekt hier wenig relevant. Bei einer allgemeinen Betrachtung der Funktionalität von Terraform wäre dieser Punkt jedoch sehr interessant.

Performance Efficiency

- **Time behaviour**: Grad zu welchem die Anforderungen an Bearbeitungszeit und Durchsatz erfüllt werden.

  Anwendbarkeit: Die Dauer in welcher ein System von der Plattform durch den jeweiligen Terraform Provider bereitgestellt werden kann ist ein relevanter Aspekt der sehr gut untersucht und verglichen werden kann.

- **Resource utilization**: Beschreibt Umfang und Typ der Ressourcen die von der Software während des Betriebs benötigt werden.

  Anwendbarkeit: Da die Terraform Provider selbst nur die Schnittstelle darstellen ist dieser Aspekt nicht sinnvoll anwendbar. Bei einem Vergleich verschiedener Softwaretools wäre dieser Aspekt von größerer Bedeutung.

- **Capacity**: Grad zu welchem die maximalen Limits der Software die Anforderungen erfüllen.

  Anwendbarkeit: Auch dieser Aspekt ist nicht relevant, es trifft die die selbe Argumentation wie beim vorhergehenden Punkt zu.

Compatibility

- **Co-existence**: Grad zu welchem die Software ihre Funktion erfüllen kann während sie sich eine Umgebung mit anderen Programmen teilt ohne deren Funktionalität zu beeinträchtigen.

  Anwendbarkeit: Da die Terraform Provider Anforderungen an ihre Umgebung stellen um ihre Funktionalität erfüllen zu können kann dieser Aspekt durchaus sinnvoll betrachtet werden.

- **Interoperability**: Grad zu welchem die Software mit anderen Produkten Informationen austauschen und verarbeiten kann.

  Anwendbarkeit: Dieser Aspekt könnte im Hinblick auf Terraform mit Sicherheit untersucht werden und interessante Ergebnisse liefern. Im Falle der einzelnen Terraform Provider kann dieser Aspekt jedoch nur begrenzt sinnvoll betrachtet werden.

Usability

- **Appropriateness recognizability**: Beschreibt wie einfach ein Nutzer erkennen kann ob die Software eine angemessene Lösung für dessen Anwendungsfall darstellt.

  Anwendbarkeit: Dieser Aspekt kann zu einem gewissen Grad betrachtet und bewertet werden. Die Bezeichnungen der verschiedenen Computing Ressourcen können zum Beispiel ihre Funktion gut oder auch weniger genau beschreiben.

- **Learnability**: Grad zu welchem ein bestimmter User bestimmte Lernziele in einem definierten Kontext erreichen kann.

  Anwendbarkeit: Eine Untersuchung der Erlernbarkeit eines Terraform Providers könnte durchaus vorgenommen werden. Um ein aussagekräftiges Ergebnis erzielen zu können wären jedoch umfangreichere Untersuchungen mit mehreren Usern notwendig weshalb im Rahmen dieser Arbeit dieses Kriterium nicht betrachtet werden soll.

- **Operability**: Beschreibt Attribute die ein System besitzt das dessen Bedienung vereinfacht.

  Anwendbarkeit: Bei einer Betrachtung von Terraform als ganzes könnte dieser Punkt sinnvoll untersucht werden, die einzelnen Provider unterscheiden sich hier jedoch nicht.

- **User error protection**: Grad zu dem die Software den Nutzer gegen eigene Fehler (z.Bsp. während der Eingabe) schützt.

  Anwendbarkeit: Hier liegt die selbe Lage wie beim vorhergehenden Punkt vor, wird daher nicht betrachtet.

- **User interface aesthetics**: Beschreibt wie ansprechend und zufriedenstellend das Userinterface bewertet werden kann.

  Anwendbarkeit: Dieser Aspekt ist irrelevant da die Terraform Provider nicht direkt verwendet werden und kein User interface besitzen. Terraform selbst wird aus der Kommandozeile heraus bedient.

- **Accessibility**: Grad zu welchem die Software von verschiedenen Nutzern mit verschiedenen Fähigkeiten und Charakteristiken verwendet werden kann.

  Anwendbarkeit: Für den Vergleich der Provider nicht anwendbar. Die selbe Argumentation wie bei Operability und User error protection treffen auch hier zu.

3.2.3 Entscheidungskriterien für die Evaluierung

3.2.4 Konkreter Aufbau in Microsoft Azure

3.2.5 Analyse Azure

3.2.6 Konkreter Aufbau in Amazon AWS

3.2.7 Analyse AWS

3.2.8 Konkreter Aufbau in Google Cloud Platform

3.2.9 Analyse GCP

4 Ergebnisse

4.1 Gemessene Werte und Erkenntnisse

4.2 Auswertung der Ergenisse

5 Schluss

Ergebnis-Bewertung, Zusammenfassung und Ausblick

5.1 Gewonnenen Erkenntnisse

5.2 Zusammenfassung der Arbeit

5.3 Mögliche weitere untersuchenswerte Aspekte

5.4 Aktuelle und zukünftige Entwicklungen

A Kapitel im Anhang

Alles was den Hauptteil unnötig vergrößert hätte, z. B. HW-/SW-Dokumentationen, Bedienungsanleitungen, Code-Listings, Diagramme

Literaturverzeichnis

[1] Thomas Nonnenmacher, LaTeX Grundlagen - Setzen einer wissenschaftlichen Arbeit Skript, 2008,http://www.stz-softwaretechnik.de; *(Bei STZ Internetseite unter Publikationen - Skripte) [V. 2.0 26.02.08]*

[Gun04] Karsten Günther, LaTeX2 — Das umfassende Handbuch, Galileo Computing, 2004, http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-768; *1. Auflage*