HAW HAMBURG

BACHELORTHESIS
Leonard Vincent Simon Pahlke

Automatisierungsstrategien in der Hybrid Cloud

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Leonard Vincent Simon Pahlke

Automatisierungsstrategien in der Hybrid Cloud

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 20. September 2021

Titel	Automatisierungsstrategien in der Hybrid Cloud
Autor	Leonard Vincent Simon Pahlke
Stichworte	Cloud, IT-Infrastruktur, Hybrid-Cloud, Infrastructure as Code (IaC)

Kurzzusammenfassung

In dieser Arbeit wurde untersucht, wie Hybrid-Cloud-Infrastrukturen automatisiert werden können. Dafür wurden Hybrid-Cloud-Automatisierungsstrategien (HCA-Strategien) erarbeitet, die auf den Technologien Kubernetes, Service Mesh und VPN-Tunneling aufbauen. Erarbeitete Strategien beschreiben zudem, wie eine Automatisierung mit Infrastructure as Code (IaC) Tools aussehen kann. In insgesamt zwei Fallstudien wird eine HCA-Strategien mit mehreren Cloud-Anbieter und IaC-Tools entwickelt.

Title	Hybrid cloud automation strategies
Author	Leonard Vincent Simon Pahlke
Keywords	Cloud, IT-Infrastructure, Hybrid-Cloud, Infrastructure as Code (IaC)

Abstract

This thesis investigated how hybrid cloud infrastructures can be automated. For this purpose, hybrid cloud automation strategies (HCA strategies) were developed that build on the technologies Kubernetes, Service Mesh and VPN tunneling. Elaborated strategies also describe how automation with Infrastructure as Code (IaC) tools can look like. In a total of two case studies, HCA strategies are developed with several cloud providers and IaC tools.

Danksagung

Zu Beginn dieser Arbeit möchte ich mich bei allen Personen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Insbesondere möchte ich Herrn Prof. Dr. Olaf Zukunft für die inhaltlichen Diskussionen und wertvolle Kritik sowie Herrn Prof. Dr. Stefan Sarstedt für die Übernahme des Zweitgutachtens und beiden für die Unterstützung während meines Studiums danken. Außerdem möchte ich der Fakultät Informatik an der HAW Hamburg für die Möglichkeit des Bachelor Studiums danken. Des Weiteren bedanke ich mich bei meinen Arbeitgeber Storm Reply GmbH und der Reply AG für die Bereitstellung von Ressourcen während meines Studiums und für die praktische Arbeit. Speziell für diese Arbeit möchte ich mich bei meinen Arbeitskollegen Herrn Hossein Salahi und Herrn Alex Ferener-Vardi für die Diskussionen zu hybriden Kubernetes- und AWS-Architekturen bedanken. Schlussendlich bedanke ich mich herzlich bei meiner Familie für die Unterstützung während meines Studiums.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Arbeit untersucht, wie die IT-Infrastruktur verschiedener Cloud-Umgebungen in einem System kohärent automatisiert werden kann. Automatisierungen von verteilten Systemen sind mit dem Aufkommen der Cloud und beim Integrieren von DevOps Prinzipien in Unternehmen besonders in den Fokus gerückt[70][68][73][72]. Das heterogene Feld an IT-Infrastrukturen und dadurch verwendeten Technologien hat noch keinen Standard hervorgebracht, der eine leichte System-Integration zwischen Cloud-Umgebungen ermöglicht[1].

1.1 Motivation

Einen wichtigen Teil der digitalen Transformation von Unternehmen stellt die Umstellung der selbstverwalteten IT-Infrastruktur auf eine Cloud-basierende extern verwaltete IT-Infrastruktur dar[70][71][69]. Diese Transformation ist nicht neu, sondern seit einigen Jahren im vollen Gang[71][65]. Es stellt sich jedoch mehr und mehr heraus, dass nicht alle Prozesse in die extern verwaltende Cloud verlagert werden können und weiterhin selbst verwaltet werden müssen[67]. Mehrere Cloud-Infrastrukturen in einem System zu nutzen, ist durch den hohen System-Integrations-Aufwand eine große Herausforderung. Eine Hybride-Cloud öffnet den Rahmen, mehr Open Source Technologien einbauen zu können und die Abhängigkeit gegenüber den Public-Cloud-Anbietern zu reduzieren. Die Gestaltungsfreiheit ist in einer Hybrid-Cloud daher höher und ermöglicht eine höhere Flexibilität der IT-Systeme. Unternehmen wünschen demnach vermehrt Cloud-Umgebungen zu nutzen[53].

Der Aspekt der Automatisierung hat mit IaC-Tools Einzug in den Aufbau komplexer verteilter Systeme gefunden. IaC-Tools könnten daher ein geeignetes Vehikel sein, um

[1] Dies kann an der großen Anzahl heterogener IaC-Tools erkannt werden (siehe Kapitel 3)

den komplexen Aufbau einer Hybrid-Cloud zu vereinfachen und praktikabel zu machen. Diese beiden Aspekte wurden in Forschungsarbeiten noch nicht zusammen untersucht.

1.2 Ziele

In dieser Arbeit soll untersucht werden, ob ein Hybrid-Cloud-System über Automatisierungen praktikabler und einfacher aufzusetzen ist. Es wird erörtert, ob die Kombination mehrerer Infrastruktur-Automatisierungstools (IaC-Tools) eine sinnvolle Strategie ist, Hybrid-Cloud-Systeme zu verwalten. Um diese Fragen beantworten zu können, wird die aktuelle Cloud-Landschaft beleuchtet. Ebenso muss ein Überblick über derzeitig relevante IaC-Tools geschaffen werden, die für Infrastruktur-Automatisierungen infrage kommen. Dafür werden schrittweise Grundlagen und Konzepte herausgearbeitet und in einer praktischen Anwendung untersucht. Weil Projekte und Anforderungen stark schwanken, sollen Technologien und damit verbundene Strategien aufgezeigt werden, anhand denen eine Hybrid-Cloud aufgebaut werden kann.

1.3 Gliederung

Diese Arbeit ist in acht Kapiteln strukturiert. Nach der Einleitung folgt das zweite Kapitel mit den Grundlagen. Nach den Grundlagen werden IaC-Tools im dritten Kapitel detaillierter behandelt. In dem Kapitel werden mehrere IaC-Tools vorgestellt, Unterschiede aufgezeigt und in Kategorien zugeordnet. Im vierten Kapitel werden folgend Hybrid-Cloud Automatisierungsstrategien entwickelt. Eine Hybrid-Cloud-Automatisierungsstrategie wird als Fallstudie im fünften Kapitel entwickelt, implementiert und im sechsten Kapitel an Umfang ergänzt. Im siebten Kapitel wird die praktische Untersuchung analysiert. Es folgt in Kapitel acht die Zusammenfassung mit einer kritischen Bewertung der Technologie, welche daraus einen Ausblick für weitere Forschungsarbeiten ermöglicht.

2 Grundlagen

In diesem Kapitel werden die Grundlagen dieser Arbeit besprochen. Es wird primär auf die Cloud und damit verbundene Themen eingegangen.

Um sich die Cloud besser vor Augen zu führen, wird in diesem Abschnitt die Applikation Dropbox als Beispiel gebracht. In der Öffentlichkeit wird oftmals der Begriff der Cloud nur mit Cloud-Speicher in Verbindung gebracht, der von Dropbox u. a. angeboten wird.

Es fällt jedoch mehr unter dem Begriff 'Cloud' als nur das Speichern von Daten. Dropbox-Services werden in der Cloud verwaltet, allerdings können auch komplett andere Anwendungsfälle in der Cloud laufen. Es wird hier also nur ein Teil des Cloud-Funktionsumfangs mit der Cloud assoziiert, nämlich der des verteilten Speicherns von Daten. Um ein weiteres Bild zu fassen, hilft der Blick auf den Dropbox Service Stack (siehe Abbildung 2.1). In der Abbildung 2.1 sind mehrere unterschiedliche Services zu erkennen, die nicht für das Speichern der Daten zuständig sind, allerdings für das Produkt wichtig sind. Services in den Bereichen Maschinelles Lernern, Sicherheit, Analyse, Speicher, Migration und Übertragung und Weitere finden sich wieder. Anhand der Bandbreite unterschiedlicher Services sowie anhand Dropbox wird deutlich, dass mithilfe der Cloud vielseitige und komplexe IT-Lösungen entwickelt werden können.

In den nächsten Abschnitten Public-Cloud-Anbieter, IT-Ressourcen, IT-Infrastruktur, Virtualisierung werden die Grundlagen der Cloud erörtert, bevor wir weiter auf die Cloud und Cloud-Computing zu sprechen kommen wird. Nachdem die Cloud definiert ist, wird in den Abschnitten Container-Orchestrierung, Serverless Computing und Service Mesh Technologien beschrieben, die im Laufe der Arbeit zurückgegriffen wird. Zum Ende des Kapitels wird im Abschnitt Automatisierung DevOps und IaC-Tools eingeführt.

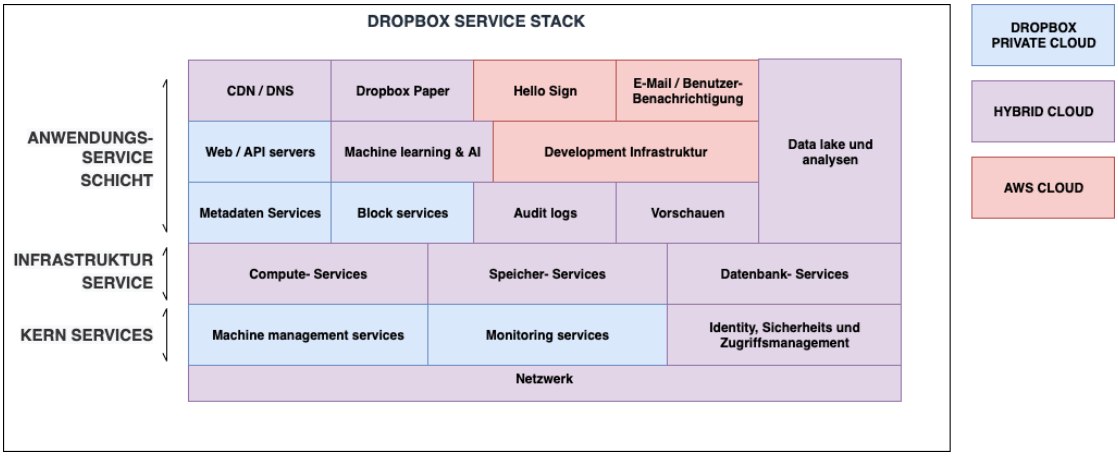


Abbildung 2.1: Dropbox Service Stack[11].

2.1 Public-Cloud Anbieter

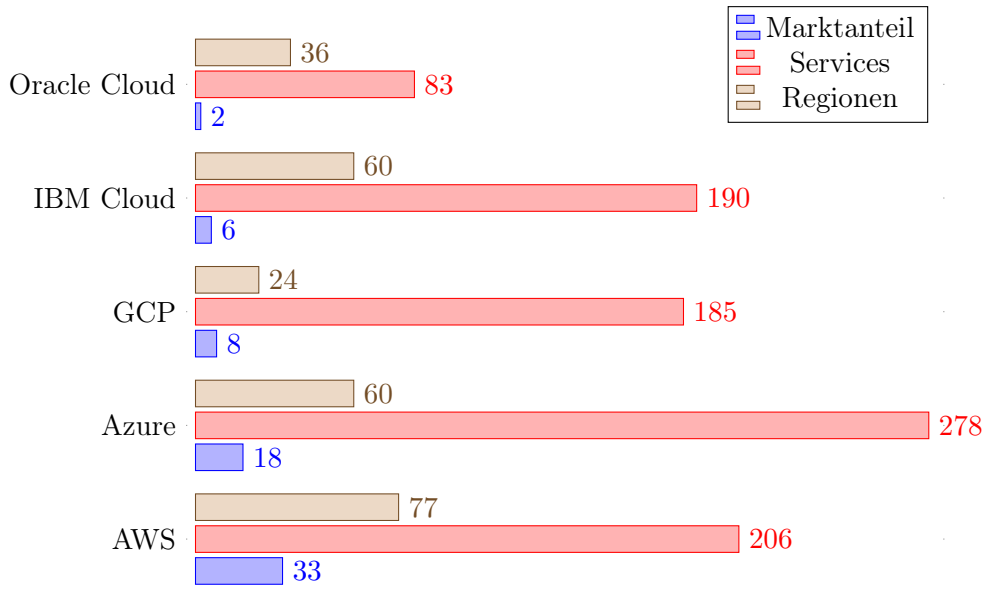
Es gibt eine Reihe von öffentlichen Cloud-Anbietern, welche eine IT-Infrastruktur für Unternehmen anbieten. Unter den Anbietern sind viele der großen amerikanischen Techkonzerne zu finden. 2006 Hat Amazon.com mit seinem Tochterunternehmen Amazon-Web-Services (AWS) den Startschuss gesetzt. 2008 folgte Google mit der Google Cloud Plattform (GCP) und 2010 Microsoft Azure (Azure). Auch Oracle, IBM und Alibaba haben eine Public-Cloud.

AWS ist einer der Public-Cloud-Anbieter, der Services in 26 Kategorien ordnet[1]. Darunter finden sich u. a. Datenbanken, Speicher, Netzwerk, maschinelles Lernen, Sicherheit, Analyse, Internet of Things (IoT), Blockchain, Container, Serverless und weitere. Einige dieser Kategorien sind auch im Dropbox-Service-Stack wieder zu finden. Der Dropbox Service-Stack beinhaltet unterschiedliche Arten von Services, die nicht alle primär Daten abspeichern.

In der Abschnitt 2.1 werden zur Einordnung relevante Eckdaten der Cloud-Anbieter gegeben.

[1] AWS hat im September 2021 26 Service-Kategorien.

Weltweiter Marktanteil von Cloud-Infrastruktur Dienstleistern in Q4 2019[66]



Die gelisteten Anbieter aus Abschnitt 2.1 stammen aus den USA. Es finden sich aber auch Cloud-Anbieter aus China. Ein europäischer Anbieter, welcher eine echte Alternative darstellt, ist schwer auszumachen. Dieser Markt ist klar Amerikanisch dominiert.

In Europa gib es eine Initiative, die das ändern soll. Gaia-X ist ein deutsch-französisches Projekt zur Entwicklung der europäischen IT-Infrastruktur[17]. Die Initiative soll europäische Unternehmen unabhängiger und souveräner vom Amerikanischen und Chinesischen Anbietern machen. Aus dem KPMG-Cloud-Monitor 2020 geht hervor, dass nach einem europäischen Cloud-Anbieter nachgefragt wird[53]. Gaia-X könnte den Weg dafür ebnen.

2.2 IT-Ressourcen

Im Gabler Wirtschaftslexikon werden IT-Ressourcen in den Kategorien Software-, Hardware- und Plattform-IT-Ressourcen definiert.

1. **Software-IT-Ressourcen**: “IT-Ressourcen, mit denen ein Anwender direkt interagiert. Solche Ressourcen bieten komplette Anwendungen über ein Nutzerinterface an.“[44]

2. **Plattform-IT-Ressourcen**: “IT-Ressourcen, die von anderen Anwendungen genutzt werden. Plattform-IT-Ressourcen stellen dabei entweder Betriebsumgebungen, [...] bereit oder bieten Anwendungsfunktionalität an, die von kundenspezifischen Anwendungen genutzt wird. “[44]

3. **Hardware-IT-Ressourcen**: “physikalische oder virtualisierte“[44]

IT-Ressourcen werden in der Cloud als Service nach einem Anwendungszweck angeboten. Im Anhang findet sich ein Beispiel einer AWS IT-Ressource (siehe Abschnitt A.3). Die unterschiedlichen Arten von Services werden im Laufe des Kapitels definiert.

2.3 IT-Infrastruktur

Um sich dem Begriff IT-Infrastruktur zu nähern, wird zunächst eine allgemeinere Definition in den Blick gefasst. 1966 wurde von Reimut Jochimsen die Infrastruktur folgendermaßen definiert.

> “[...] die Gesamtheit der materiellen, institutionellen und personalen Anlagen, Einrichtungen und Gegebenheiten, die den Wirtschaftseinheiten im Rahmen einer arbeitsteiligen Wirtschaft zur Verfügung stehen.“[40]

Wenn Infrastruktur auf IT bezogen wird, bedeutet das im übertragenen Sinne, dass die IT-Infrastruktur bereitgestellt wird, um Anwendungssoftware zu ermöglichen. Dabei kann die IT-Infrastruktur unter zwei Sichten gefasst werden[59].

1. Zum einen aus technischer Sicht, worunter Hardware, Software sowie baulichen Einrichtungen für den Betrieb von Software fallen.

2. Zum anderen aus Sicht des Informationsmanagements, die auch personelle und institutionelle Aspekte unter IT-Infrastruktur fassen.

IT-Infrastrukturen beinhalten IT-Ressourcen und ordnen diese an, wodurch komplexere- und Verteilte-Systeme realisiert werden können. In der Abbildung 2.2 werden die zuvor beschrieben Sichten der IT-Infrastruktur gezeigt.

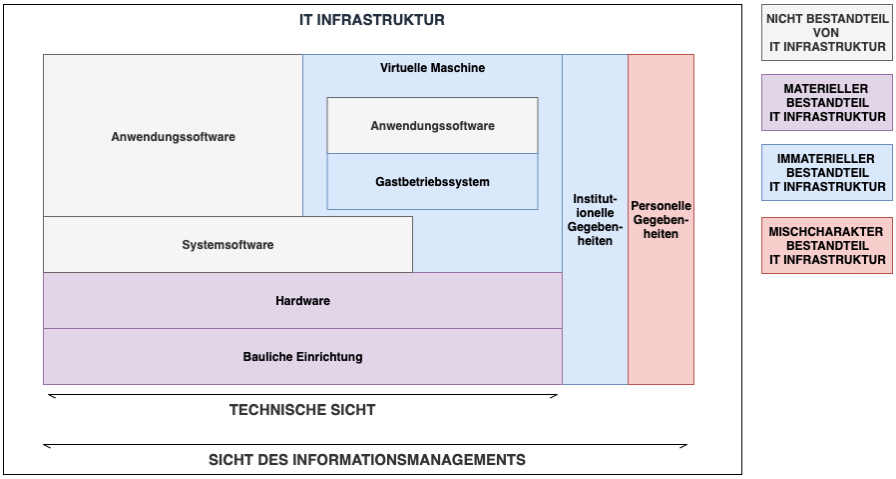


Abbildung 2.2: Bestandteile einer IT-Infrastruktur[59].

2.4 Rechenzentrum

Cloud-Anbieter verwalten Rechenzentren (RZ), um IT-Ressourcen anbieten zu können. Bei einem Rechenzentrum handelt es sich nicht nur um Server, die in einem Raum stehen. Die gesamte Struktur, die nötig ist, um Hardware zu betreiben, zu warten und sicher zu stellen, gehört ebenfalls dazu.

> "Unter 'IT-Betriebs-Bereich' sind Räume zu verstehen, in denen die Hardware aufgebaut ist und betrieben wird, die der Bereitstellung von Diensten und Daten dient. Das RZ umfasst neben dem IT-Betriebs-Bereich alle weiteren technischen Supportbereiche (Stromversorgung, Kälteversorgung, Löschtechnik, Sicherheitstechnik etc.), die dem bestimmungsgemäßen Betrieb und der Sicherheit des IT-Betriebsbereichs dienen."[18]

Ein IT-System kann auch in mehreren Rechenzentren als verteiltes System laufen.

> "Ist die IT-nutzende Organisation an mehreren, räumlich voneinander getrennten Standorten angesiedelt, und sind diese durch andere als hauseigene LAN-Verbindungen miteinander gekoppelt, ist jeder der Standorte entsprechend separat zu betrachten und zu behandeln."[18]

Im Anhang wird beschrieben, wie AWS Rechenzentren in Regionen und Verfügbarkeitszonen unterteilt werden (siehe Abbildung A.1). Außerdem wird im Anhang beschrieben,

inwiefern die Verfügbarkeit der IT-Infrastruktur ein kritischer Wert des Rechenzentrums ist (siehe Abschnitt A.2).

2.5 Virtualisierung

Wenn eine Anwendung in der Cloud ausgeführt wird, geschieht dies auf virtuellen Maschinen (VM) (indirekt oder direkt). Welcher Server genutzt wird, ist abstrahiert und dem Nutzer i. d. R. nicht bekannt. Ermöglicht wird dies durch die Virtualisierung, die ein Kernbestandteil der Cloud ist.

> "Virtualisierung ist eine Herangehensweise in der Informationstechnik, die Ressourcen so in eine logische Sicht zusammenfasst, dass ihre Auslastung optimiert werden kann. Das Schlagwort Virtualisierung umfasst mehrere grundsätzlich verschiedene Konzepte und Technologien."[16]

Das Konzept der Virtualisierung wurde bereits in den 1970er-Jahren entwickelt, um die Kosten für die Anschaffung von Hardware zu senken und die Produktivität zu steigern, indem mehr Benutzer gleichzeitig daran arbeiten können[55].

> "Die Hauptanwendung von VMs besteht mittlerweile darin, die Ausführung einer Reihe von Anwendungen, die ursprünglich für unterschiedliche Hardware und Betriebssysteme gedacht waren, auf einem bestimmten Rechner zu ermöglichen."[21]

Die verschiedenen Konzepte der Virtualisierung umfassen Speichervirtualisierung, Betriebssystemvirtualisierung, Datenvirtualisierung und Netzwerkvirtualisierung.

> "Eine Virtuelle Maschine (VM) läuft in einer abgeschotteten Umgebung auf einer physischen Hardware und verhält sich wie ein vollwertiger Computer mit eigenen Komponenten. In einer VM kann ein Betriebssystem mit Anwendungen genau wie auf einem realen Computer laufen. Die Anwendungen, die in einer VM laufen, bemerken diesen Umstand nicht. Anforderungen der Betriebssystem-Instanzen werden von der Virtualisierungssoftware transparent abgefangen und auf die real vorhandene oder emulierte Hardware umgesetzt."[16]

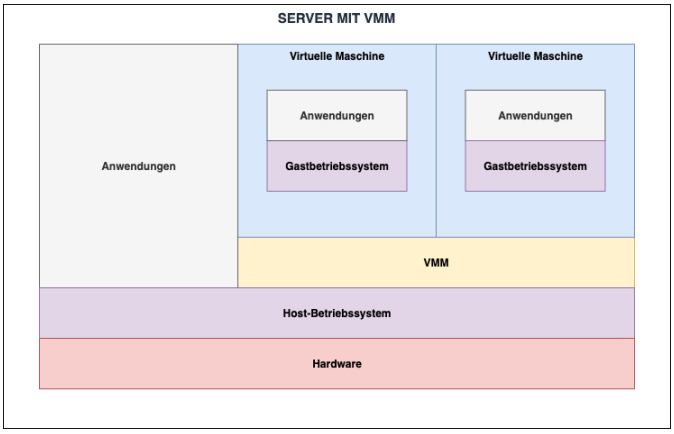


Abbildung 2.3: Vollständiger Virtualisierung über VMM[16]

2.5.1 Vollständige Virtualisierung mit Virtual Machine Monitoren (VMM)

"Vollständige Virtualisierungslösungen bieten einer virtuellen Maschine eine vollständige, virtuelle PC-Umgebung inklusive eigenem BIOS. Jedes Gastbetriebssystem erhält eine eigene virtuelle Maschine mit virtuellen Ressourcen wie Prozessor(en), Hauptspeicher, Laufwerke, Netzwerkkarten, etc. Kern der Lösung ist ein sogenannter Virtueller Maschinen-Monitor (VMM), der als Anwendung im Host-Betriebssystem läuft (siehe Abbildung 2.3)."[16]

"Der Virtual Machine Monitor (VMM) ist als dünne Softwareschicht eine Abstraktion, welche virtuelle Maschinen exportiert. Die Abstraktion ähnelt der Hardware, so dass jede Software, die für diese Hardware geschrieben wurde, in der virtuellen Maschine ausgeführt werden kann (siehe Abbildung 2.4)."[48]

"Die Aufgabe des VMM ist die Zuweisung der Hardwareressourcen an die virtuellen Maschinen. Teilweise emuliert er auch Hardwarekomponenten, die nicht für den gleichzeitigen Zugriff mehrerer Betriebssysteme ausgelegt ist, wie zum Beispiel Netzwerkkarten."[16]

"Jede dieser Schnittstellen (virtuelle Maschinen) ist eine effiziente Nachbildung des ursprünglichen Computersystems, komplett mit allen Prozessoranweisungen (d. h. sowohl privilegierten als auch nicht-privilegierten Anweisungen) und Systemressourcen (d. h. Speicher und I/O-Geräte). Durch die

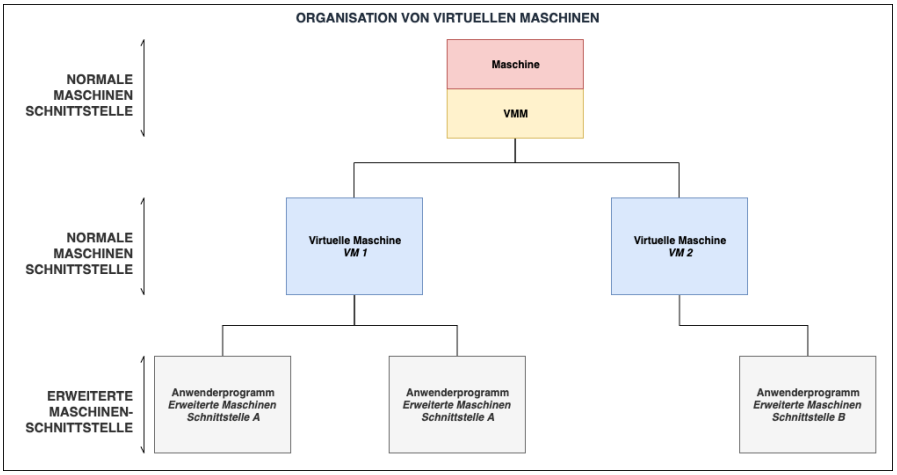


Abbildung 2.4: Organisation virtueller Maschinen[25]

Ausführung jedes Betriebssystems auf einer eigenen virtuellen Maschine wird es möglich, mehrere verschiedene Betriebssysteme (privilegierte Softwarekerne) gleichzeitig auszuführen (siehe Abbildung 2.4).“[25]

“Ein VMM bietet eine einheitliche Sicht auf die zugrundeliegende Hardware, sodass Maschinen verschiedener Hersteller mit unterschiedlichen I/O-Subsystemen gleich aussehen, was bedeutet, dass virtuelle Maschinen auf jedem verfügbaren Computer laufen können. Anstatt sich also um einzelne Maschinen mit eng gekoppelten Hardware- und Software-Abhängigkeiten zu kümmern, können Administratoren die Hardware einfach als einen Pool von Ressourcen betrachten, auf dem bei Bedarf beliebige Dienste ausgeführt werden können.

Da der VMM auch eine vollständige Kapselung des Softwarezustands einer virtuellen Maschine bietet, kann die VMM-Schicht virtuelle Maschinen nach Belieben auf verfügbare Hardwareressourcen zuordnen und sogar virtuelle Maschinen zwischen Maschinen migrieren. Der Lastausgleich zwischen einer Sammlung von Maschinen wird dadurch trivial, und es gibt ein robustes Modell für den Umgang mit Hardwareausfällen oder für die Skalierung von Systemen. Wenn ein Computer ausfällt und offline gehen muss oder wenn eine neue Maschine online geht, kann die VMM-Schicht die virtuellen Maschinen einfach entsprechend neu zuordnen. Virtuelle Maschinen lassen sich auch leicht replizieren, so dass Administratoren bei Bedarf neue Dienste online stellen können.“[48]

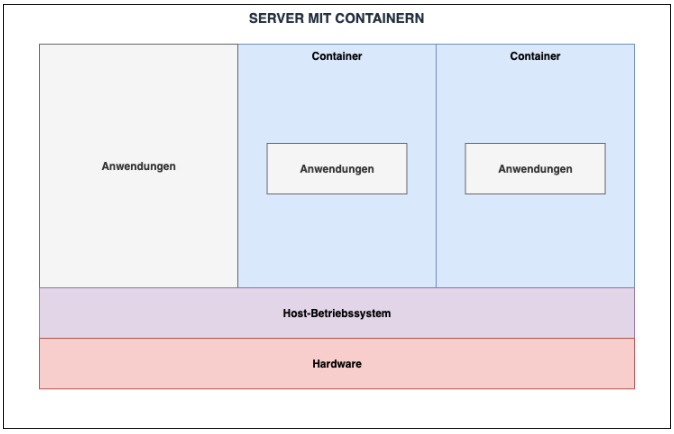


Abbildung 2.5: Betriebssystem-Virtualisierung[16]

Nur mit Virtualisierung und VMM ist die heutige Cloud-Landschaft möglich. Virtualisierung kommt auch bei der Containerisierung ins Spiel, die im nächsten Abschnitt vorgestellt wird.

2.5.2 Betriebssystem Virtualisierung und Container

Bei der Betriebssystemvirtualisierung laufen unter ein und demselben Betriebssystem mehrere identische Systemumgebungen, die gegeneinander abgeschottet sind und von den verschiedenen Lösungen meist als Container bezeichnet werden.

Im Gegensatz zu VMs stellt ein Container kein komplettes Betriebssystem dar, was bedeutet, dass Container nicht booten müssen, etwas leichter in der Dateigröße sind und mit weniger Ressourcen intensiv arbeiten (siehe Abbildung 2.5).

> "»A virtual Machine is an operating system that thinks it's running on its own computer. A container on the other hand is an application that thinks it's an operating system.« - *David Clinton*"[23]

2013 hat Docker den heutigen Standard in der Container-Verwaltung gesetzt. Container sind heutzutage weit verbreitet und werden in den meisten IT-Systemen von Unternehmen bereits verwendet.

> "Docker definiert Container als eine 'standardisierte Einheit von Software'. Die 'Softwareeinheit' - oder, der Dienst oder die Anwendung, die innerhalb

> eines Containers läuft - hat vollen, privaten Zugriff auf ihre eigene, isolierte Sicht auf die Betriebssystemkonstrukte.“[15]

Container im großen Stiel zu verwalten, wird im Abschnitt 'Container-Orchestrierung' beschrieben (siehe Abschnitt 2.7). Mit den eingeführten Begriffen wird im nächsten Abschnitt nun das Cloud-Computing definiert.

2.6 Cloud-Computing

Cloud-Computing kann als verteiltes Computersystem neben Cluster-Computing und Grid-Computing angesiedelt werden[2]. Cloud-Computing wurde 2011 vom National Institute of Standards and Technology (NIST) der USA definiert. Diese Standardisierung liegt nun einige Jahre zurück, findet aber bis jetzt große Beachtung von allen Cloud-Anbietern. In dem herausgebrachten Paper wird Cloud-Computing folgendermaßen definiert:

> “Cloud-Computing ist ein Modell zur Ermöglichung eines allgegenwärtigen, bequemen, bedarfsgerechten Netzwerkzugriffs auf einen gemeinsamen Pool konfigurierbarer Computing-Ressourcen (z. B. Netzwerke, Server, Speicher, Anwendungen und Dienste), die mit minimalem Verwaltungsaufwand oder Interaktion mit dem Dienstanbieter schnell bereitgestellt und freigegeben werden können. Dieses Cloud-Modell setzt sich aus fünf wesentlichen Merkmalen, drei Dienstmodellen und vier Bereitstellungsmodellen zusammen.“[54]

Die Merkmale, Dienstmodelle und Bereitschaftsmodelle des Cloud-Computings nach NIST sind nun nacheinander aufgelistet.

2.6.1 Merkmale des Cloud-Computing

In diesem Abschnitt werden die Merkmale des Cloud-Computing aufgelistet, die von NIST definiert worden sind[54].

- **On-Demand-Selbstbedienung**: “Ein Verbraucher kann Rechenkapazitäten, wie z. B. Serverzeit und Netzwerkspeicher, einseitig und automatisch nach Bedarf bereitstellen, ohne dass eine menschliche Interaktion mit dem jeweiligen Dienstanbieter erforderlich ist.“[54]

[2] Siehe [1] zu mehr Informationen zu Cluster-Computing und Grid-Computing

- **Breitband Netzzugang**: "Die Funktionen sind über das Netzwerk verfügbar und der Zugriff erfolgt über Standardmechanismen, die die Nutzung durch heterogene Thin- oder Thick-Client-Plattformen (z. B. Mobiltelefone, Tablets, Laptops und Workstations) fördern."[54]
- **Ressourcen-Pooling**: "Die Rechenressourcen des Anbieters werden gepoolt, um mehrere Verbraucher mit einem Multi-Tenant-Modell zu bedienen, wobei verschiedene physische und virtuelle Ressourcen dynamisch zugewiesen und je nach Verbrauchernachfrage neu zugewiesen werden. Es besteht eine gewisse Standortunabhängigkeit [...]. Beispiele für Ressourcen sind Speicher, Verarbeitung, Arbeitsspeicher und Netzwerkbandbreite."[54]
- **Rapide Elastizität**: "Services können elastisch bereitgestellt und freigegeben werden, in einigen Fällen automatisch, um schnell nach außen und innen zu skalieren, je nach Bedarf. Für den Kunden erscheinen die für die Bereitstellung verfügbaren Kapazitäten oft unbegrenzt und können in beliebiger Menge und zu jeder Zeit angeeignet werden."[54]
- **Gemessener Service**: "Cloud-Systeme steuern und optimieren automatisch die Ressourcennutzung, indem sie eine Messfunktion auf einer für die Art des Services geeigneten Abstraktionsebene nutzen (z. B. Speicher, Verarbeitung, Bandbreite und aktive Benutzerkonten). Die Ressourcennutzung kann überwacht, gesteuert und gemeldet werden [...]."[54]

2.6.2 Bereitstellungsmodelle in der Cloud

NIST hat vier verschiedene Formen der Bereitstellung definiert[54].

- **Private-Cloud**: "Die Cloud-Infrastruktur wird für die ausschließliche Nutzung durch eine einzelne Organisation mit mehreren Verbrauchern (z. B. Geschäftseinheiten) bereitgestellt. [...]"[54]
- **Community-Cloud**: "Die Cloud-Infrastruktur wird für die ausschließliche Nutzung durch eine bestimmte Gemeinschaft von Kunden aus Organisationen bereitgestellt, die gemeinsame Anliegen haben [...]."[54]
- **Public-Cloud**: "Die Cloud-Infrastruktur wird für die offene Nutzung durch die Allgemeinheit bereitgestellt. [...]"[54]

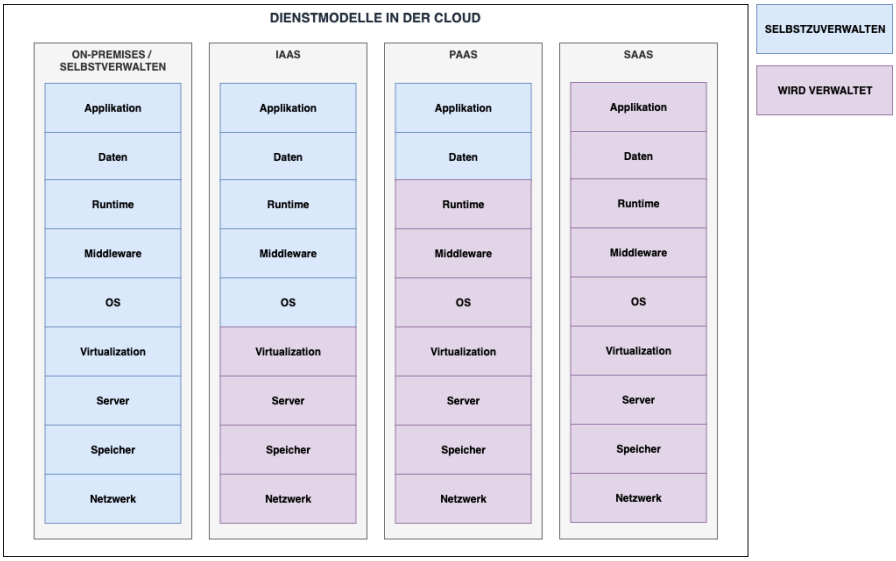


Abbildung 2.6: Dienstmodelle in der Cloud (nach NIST)[37]

- **Hybrid-Cloud**: "Die Cloud-Infrastruktur ist eine Zusammensetzung aus zwei oder mehr verschiedenen Cloud-Infrastrukturen (privat, gemeinschaftlich oder öffentlich), die eigenständige Einheiten bleiben, aber durch standardisierte oder proprietäre Technologie miteinander verbunden sind [...]."[54]

Im Anhang wird auch der AWS Service AWS-Outpost beschrieben, der einen Hybrid-Cloud Anwendungsfall verfolgt (siehe Abschnitt A.4).

2.6.3 Dienstmodelle in der Cloud

Die im NIST Paper eingeführten Cloud-Dienstmodelle werden in diesem Abschnitt eingeführt[54]. In der Abbildung 2.6 sind vier Service-Arten nebeneinander dargestellt[3].

In der Grafik Abbildung 2.6 wird visuell veranschaulicht, welche Komponenten vom jeweiligen Servicemodel verwaltet werden. Die Komponenten bauen aufeinander auf und bieten, wenn sie für den Anwender verwaltet werden, keinen direkten Zugriff auf die Konfiguration der jeweiligen Komponente. In der nachfolgenden Liste werden die Servicemodelle aus dem NIST-Paper aufgeführt[54].

[3]Diese ursprüngliche Definition von vier Servicetypen wurde inzwischen aufgeweicht, wie später in dem Kapitel beschrieben, und Servicetypen wie *FaaS* wurden hinzugefügt.

- **SaaS**: "Software as a Service (*SaaS*). Die dem Verbraucher zur Verfügung gestellte Möglichkeit besteht darin, die auf einer Cloud-Infrastruktur laufenden Anwendungen des Anbieters zu nutzen. Die Anwendungen sind von verschiedenen Client-Geräten aus zugänglich, entweder über eine Thin-Client-Oberfläche, wie z.B. einen Web-Browser (z.B. webbasierte E-Mail), oder über eine Programmoberfläche. Der Verbraucher verwaltet oder kontrolliert nicht die zugrundeliegende Cloud-Infrastruktur einschließlich Netzwerk, Server, Betriebssysteme, Speicher oder sogar einzelne Anwendungsfunktionen, mit der möglichen Ausnahme von begrenzten benutzerspezifischen Anwendungskonfigurationseinstellungen."[54]

- **PaaS**: "Plattform as a Service (*PaaS*). Die Fähigkeit, die dem Verbraucher zur Verfügung gestellt wird, besteht darin, auf der Cloud-Infrastruktur vom Verbraucher erstellte oder erworbene Anwendungen bereitzustellen, die mit Hilfe von Programmiersprachen, Bibliotheken, Diensten und Tools erstellt wurden, die vom Anbieter unterstützt werden. Der Verbraucher verwaltet oder kontrolliert nicht die zugrunde liegende Cloud-Infrastruktur einschließlich Netzwerk, Server, Betriebssysteme oder Speicher, sondern hat die Kontrolle über die bereitgestellten Anwendungen und möglicherweise Konfigurationseinstellungen für die Anwendungs-Hosting-Umgebung."[54]

- **IaaS**: "Infrastructure as a Service (*IaaS*). Die Fähigkeit, die dem Verbraucher zur Verfügung gestellt wird, besteht in der Bereitstellung von Verarbeitungs-, Speicher-, Netzwerk- und anderen grundlegenden Computerressourcen, wobei der Verbraucher in der Lage ist, beliebige Software, zu der Betriebssysteme und Anwendungen gehören können, einzusetzen und auszuführen. Der Verbraucher verwaltet oder kontrolliert nicht die zugrundeliegende Cloud-Infrastruktur, hat aber die Kontrolle über Betriebssysteme, Speicher und eingesetzte Anwendungen sowie möglicherweise eine begrenzte Kontrolle über ausgewählte Netzwerkkomponenten (z.B. Host-Firewalls)."[54]

Der nächste Abschnitt befasst sich mit Serverless Computing und stellt mit *FaaS* und *BaaS* zwei weitere Dienstmodelle vor, die im NIST-Papier nicht definiert wurden.

2.7 Container-Orchestrierung

In größeren Anwendungen werden Container mit Container-Orchestrierungstools wie Kubernetes verwaltet.

Zu den Aufgaben dieser Orchestrierungstools zählen[15]:

- Die Bereitstellung und das Deployment von Containern auf den Clusterknoten.
- Ressourcenmanagement von Containern, d. h. Platzierung von Containern auf Knoten, die ausreichend Ressourcen bereitstellen, oder Verschieben von Containern auf andere Knoten, wenn die Ressourcengrenzen eines Knotens erreicht sind.
- Zustandsüberwachung der Container und der Knoten, um bei Ausfällen auf Container- oder Knotenebene einen Neustart einzuleiten.
- Ein- oder aus-skalieren von Containern innerhalb eines Clusters.
- Bereitstellung von Netzwerkanbindungen für Container.
- Internes Load-Balancing zwischen Containern.

Kubernetes ist ein häufig eingesetztes Tool, wenn Container orchestriert werden. Kubernetes wird in dieser Arbeit genutzt, und daher separat nochmal.

> "Kubernetes wurde von Google auf der Grundlage der Erfahrungen aus dem Betrieb von Containern im großen Maßstab entwickelt. Es läuft in jeder Cloud oder in lokalen Rechenzentren und abstrahiert die zugrunde liegende Infrastruktur. Dies ermöglicht den Aufbau von Hybrid-Clouds sowie die Migration in und aus der Cloud und zwischen verschiedenen Clouds. Es ist Open-Source und ist innerhalb der Cloud Native Foundation (CNCF) angesiedelt."[56]

In der Abbildung 2.7 ist eine Übersicht eines Kubernetes Cluster gegeben Die Kubernetes spezifischen Komponenten werden nun nacheinander besprochen.

- **Control Plane**: "Die Container-Orchestrierungsschicht, die die API und Schnittstellen zur Definition, Bereitstellung und Verwaltung des Lebenszyklus von Containern bereitstellt."[42]

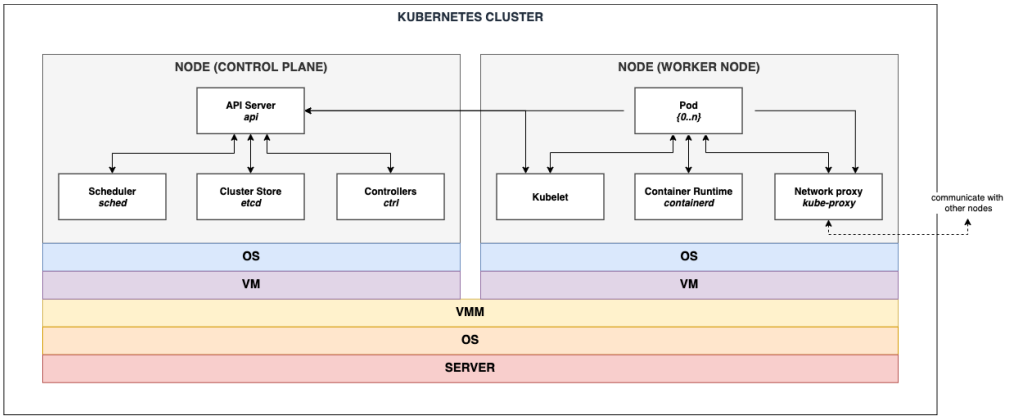


Abbildung 2.7: Kubernetes Cluster mit zwei Nodes[56]

  - **API Server**: "Der API-Server ist eine Komponente der Kubernetes-Kontrollebene, die die Kubernetes-API offenlegt. Der API-Server ist das Frontend für die Kubernetes-Kontrollebene."[42]
  - **Scheduler**: "Control-Plane-Komponente, die auf neu erstellte Pods ohne zugewiesenen Node achtet und einen Node auswählt, auf dem der neue Pod dann laufen sollen."[42]
  - **Cluster Store (etcd)**: "Konsistenter und hochverfügbarer Key-Value-Speicher, der als Backup-Speicher von Kubernetes für alle Cluster-Daten verwendet wird."[42]
  - **Controllers**: "In Kubernetes sind Controller Kontrollschleifen, die den Zustand des Clusters beobachten und dann bei Bedarf Änderungen vornehmen oder anfordern. Jeder Controller versucht, den aktuellen Cluster-Zustand näher an den gewünschten Zustand zu bringen."[42]
- **Worker-Node / Node**: "Ein Node ist ein Arbeitsrechner in Kubernetes."[42]
  - **Kubelet**: "Ein Agent, der auf jedem Knoten des Clusters läuft. Er stellt sicher, dass Container in einem Pod ausgeführt werden."[42]
  - **Container Runtime**: "In jeden Knoten des Clusters muss eine Container-Runtime installieren werden, damit dort Pods ausgeführt werden können."[42]
  - **Network proxy**: "kube-proxy ist ein Netzwerk-Proxy, der auf jedem Knoten des Cluster läuft und einen Teil des Kubernetes-Service-Konzepts implementiert."[42]

– **Pod**: “Das kleinste und einfachste Kubernetes-Objekt. Ein Pod repräsentiert eine Gruppe von laufenden Containern in dem Cluster.“[42]

In Kubernetes gibt es noch weitere Komponenten, die hier allerdings nicht weiter relevant sind[4].

2.8 Serverless Computing

Die ursprüngliche Einteilung der Servicemodelle in SaaS, PaaS, IaaS wurde in den letzten Jahren von Cloud-Anbietern zunehmend aufgeweicht und neue Servicemodelle wurden lose definiert. Serverless Services werden unter neuen Servicemodellen gefasst, die zwischen SaaS und PaaS einzuordnen sind[57]. Serverless Services werden oft mit Backend as a Service (*BaaS*) und Function as a Service (*FaaS*) in Verbindung gebracht. Serverless ist als Name von Services irreführend, da es impliziert, dass Rechenleistung ohne Server bzw. ohne Computer angeboten wird - dies ist natürlich nicht richtig[5].

> “Serverless Architecture ist ein neuer Ansatz für die Erstellung von Systemen in der Cloud. Sie umfasst Backends-as-a-Service (BaaS) - von Anbietern gehostete, hoch skalierbare Daten- und Logikkomponenten, die unsere Anforderungen an Datenbanken, Messaging-Plattformen, Benutzerverwaltung und mehr erfüllen. Zusätzlich umfasst die Serverless-Architektur Functions-as-a-Service (FaaS) - die Möglichkeit, benutzerdefinierte serverseitige Software in kleinen, ereignisgesteuerten Funktionen zu schreiben, die auf einer vollständig verwalteten Plattform bereitgestellt werden. FaaS lagert die gesamte Bereitstellung, Ressourcenzuweisung und -bereitstellung, Skalierung, Betriebssystemwartung und Prozessüberwachung aus.“[57]

Mit dem Service AWS-Lambda hat AWS im Dezember 2014 einen Service im Bereich Serverless-Computing gestartet. AWS Lambda ist ein Service, der unter FaaS einzuordnen ist und darauf abzielt, kleinere Berechnungen schnell durchzuführen, ohne dabei manuell einen Server zu starten.

[4] In dem Buch 'The Kubernetes Book' sind die übrigen Komponenten und die Funktionsweise anschaulich erklärt[56].

[5] Das 'less' in 'Serverless' bezieht sich auf die Abstraktion der operativen Tätigkeiten, die der Benutzer in Bezug auf die Verwaltung der Services durchführen muss.

> "AWS Lambda ist ein Serverless-Service, mit dem Code ausgeführt werden kann, ohne dass Server bereitgestellt oder verwaltet werden müssen, ohne dass eine Workload-basierte Cluster-Skalierungslogik verwendet wird, ohne dass Ereignisintegrationen verwaltet werden müssen und ohne dass Laufzeiten verwaltet werden müssen. Mit Lambda kann Code für praktisch jede Art von Anwendung oder Backend-Service ohne Verwaltungsaufwand ausgeführt werden. [...]"[8]

Der Serverless-Computing ist auch aus Sicht der agilen Projektentwicklung interessant. Im Anhang finden sich mehr Informationen zum Vergleich traditioneller- und Serverless-Services (siehe Abschnitt A.5).

2.9 Service Mesh

Ein Service Mesh kommt zum Einsatz, um typische Probleme einer verteilten Anwendung zu lösen. Darunter zählen u. a. Skalierbarkeit, Sicherheit, Load Balancing, Offenheit und Erweiterbarkeit und Ressourcen-Identifikation[41].

> "Ein Service-Mesh ist eine konfigurierbare Infrastrukturebene mit niedriger Latenz, die für ein hohes Volumen an netzwerkbasierter Interprozesskommunikation zwischen Anwendungsinfrastrukturdiensten unter Verwendung von Anwendungsprogrammierschnittstellen (APIs) ausgelegt ist. Ein Service-Mesh stellt eine Kommunikation zwischen containerisierten und Anwendungsinfrastrukturdiensten schnell, zuverlässig und sicher dar. Das Mesh bietet wichtige Funktionen wie Service Discovery, Load Balancing, Verschlüsselung, Beobachtbarkeit, Nachvollziehbarkeit, Authentifizierung und Autorisierung sowie Unterstützung für das Circuit-Breaker-Pattern."[50]

In der Abbildung 2.8 wird der Aufbau eines Service Meshes gezeigt. Die Instanzen A, B und C können in der Public- oder Private-Cloud liegen. Die logische Anbindung und Kommunikation läuft über den Sidecar-Proxy[6].

[6] Ein Sidecar-Proxy übernimmt Aufgaben der Plattformabstraktion, Proxy zu Remote-Diensten, Protokollierung und Konfiguration.[49]

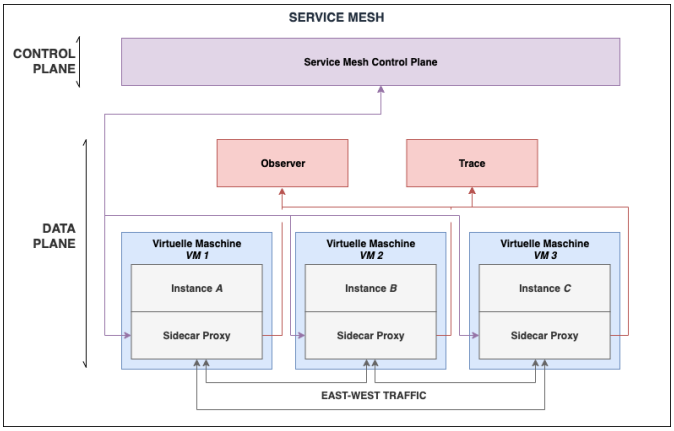


Abbildung 2.8: Aufbau eines Service Meshs[50]

2.10 Automatisierung

Automatisierung spielte schon immer eine wichtige Rolle in der Informatik[7]. In den letzten Jahren wurde mit DevOps eine neue Perspektive auf Automatisierung geworfen.

2.10.1 DevOps

DevOps kann als ein Paradigma gesehen werden, das neben der technischen Seite auch andere Aspekte in der Projektentwicklung berücksichtigt. DevOps ist eine Wortkomposition, die sich aus 'Development' und 'Operations' zusammensetzt. Nach dem DevOps-Ansatz sollen diese beiden traditionell getrennten Aufgabenbereiche zusammenwachsen. Diese Maßnahme soll zu einer besseren Kommunikation innerhalb des Unternehmens führen. DevOps umfasst aber noch mehr, was durch das CALMS-Modell beschrieben wird. CALMS ist ein Akronym für Culture-Automation-Lean-Measurement-Sharing (siehe Abbildung 2.9). Die Unternehmenskultur 'Culture' stellt das Fundament für die tragenden Säulen 'Automation', 'Lean', 'Measurement' und 'Sharing'. Darauf liegt das Dach - *DevOps*.

Die Säule 'Automation' steht im Fokus und wird daher detaillierter erläutert. Die 'Automation' bezieht sich auf einen automatisierten Software-Zyklus, um schnellere Software-Releases zu ermöglichen. Dies kann z. B. mit CI/CD (Continuous Integration / Con-

[7]Es gibt bspw. eine Konferenz mit diesem Fokus: IEEE International Conference on Computer Science and Automation Engineering (CSAE)

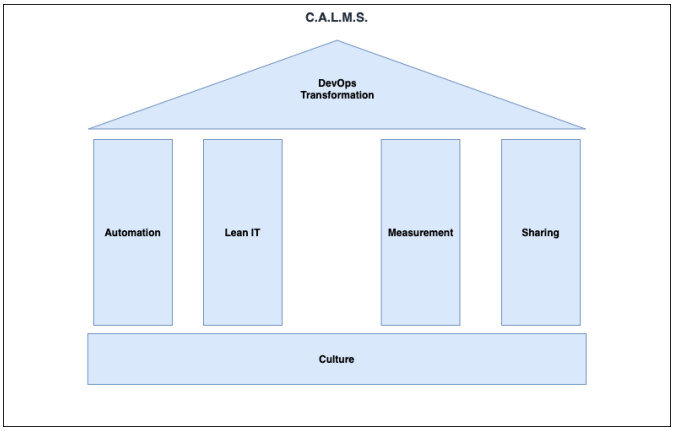


Abbildung 2.9: DevOps Model C.A.L.M.S.[26]

tinuous Deployment) oder mit Infrastructure as Code (IaC) erreicht werden[8]. Auf IaC wird nun näher eingegangen.

2.10.2 Infrastructure as Code (IaC)

> "Infrastructure-as-Code (IaC) ist ein Verfahren für den Umgang mit dynamischen Infrastrukturen wie IaaS-Diensten in der Public-Cloud. Dabei werden Ressourcen wie virtuelle Server, Storage und Netzwerke vollständig in Templates (im Sinne von maschinenlesbaren Steuerdateien) für IaC-Tools definiert und damit gesteuert. Ein Ziel von IaC ist die möglichst umfassende Automatisierung von administrativen Prozeduren, um die manuelle Konfiguration von Servern vollständig zu überwinden."[74]

IaC-Skripte und -Konfigurationen sind Software und daher Softwareentwicklern vertrauter und können z. B. Versionsverwaltungs-Tools (z. B. GIT) nutzen. AWS bietet mit AWS CloudFormation ein IaC-Tool, womit AWS Services verwaltet werden können.

```
1 AWSTemplateFormatVersion: 2010-09-09
2 Description: Creates a EC2 instance
3
4 Parameters:
5   EC2ImageId:
6     Type: AWS::EC2::Image::Id
```

[8] Auf CI/CD Pipelines wird an dieser Stelle nicht weiter eingegangen, da CI/CD-Pipelines nicht im weiteren Fokus dieser Arbeit stehen.

```
7     Default: ami-0803e21a2ec22f953
8
9 Resources:
10   EC2Instance:
11     Type: AWS::EC2::Instance
12     Properties:
13       InstanceType: t2.micro
14       ImageId: !Ref EC2ImageId
15       Tags:
16         - Key: Name
17           Value: "TestEC2Instance"
```

Listing 2.1: Beispiel für eine IaC-Datei, die mit dem IaC-Tool AWS CloudFormation interpretiert werden kann

Das Codebeispiel Listing 2.1 beschreibt eine primitive AWS Infrastruktur im YAML Dateiformat. AWS CloudFormation gibt auf Dokumentationsseiten von AWS den Syntax für diese Templates vor. Der Beispielcode erstellt bei dem Ausführen eine virtuelle Maschine in AWS und startet diese automatisch.

Domain Specific Languages (DSL) ist ein weiterer Begriff, der mit IaC in Zusammenhang gebracht wird. DSL gibt es in drei Ausprägungen, external DSL, internal DSL und language workbench.

1. "Eine externe DSL ist eine Sprache, die in der Anwendung von der Hauptsprache getrennt wird. Das Skript des externen DSL wird normalerweise in der Host-Anwendung geparst. Beispiele für externe DSLs, sind reguläre Ausdrücke, SQL, Awk und XML-Konfigurationsdateien für Systeme wie Struts und Hibernate."[47]

2. "Eine interne DSL ist eine besondere Art der Verwendung einer Allzwecksprache. Ein Skript in einer internen DSL ist gültiger Code in seiner Allzwecksprache, verwendet aber nur eine Teilmenge der Funktionen der Sprache in einem bestimmten Stil, um einen kleinen Aspekt des Gesamtsystems zu behandeln. Das Ergebnis sollte sich eher wie eine benutzerdefinierte Sprache anfühlen, als wie die Host-Sprache. Das klassische Beispiel für diesen Stil ist Lisp. Auch Ruby hat eine starke DSL-Kultur entwickelt."[47]

3. “Eine Language Workbench ist eine spezialisierte IDE zur Definition und Erstellung von DSLs. Insbesondere wird eine Language Workbench nicht nur dazu verwendet, die Struktur einer DSL festzulegen, sondern auch als eigene Editier Umgebung für das Schreiben von DSL-Skripten. Die resultierenden Skripte verbinden die Editierumgebung und die Sprache eng miteinander.“[47]

Im nächsten Kapitel werden weitere IaC-Tools beschrieben und unterschiedliche Ausprägungen herausgearbeitet.

3 Vorstellung der Infrastructure as Code Tools

Dieser Abschnitt befasst sich mit Infrastruktur as Code Tools (IaC-Tools). Zunächst wird eine Übersicht relevanter IaC-Tools gegeben. Dabei wird nur eine Auswahl von verfügbaren IaC Tools angegeben. Eine Liste aller IaC-Tools erstellen zu wollen würde diese Arbeit sprengen und dient nicht dem Ziel dieser Arbeit. Nachdem eine Übersicht der IaC-Tools gegeben wurde, werden im Anschluss die IaC-Tools nacheinander kurz beschrieben. Aus den Beschreibungen sollen sich Kategorien von IaC-Tools herauskristallisieren.

3.1 IaC-Tool Übersicht

Die Auswahl der IaC-Tools wird von drei Anbieter-Typen abgeleitet.

- Public-Cloud-Anbieter spezifische IaC-Tools.
- Unternehmen[1], die IaC-Tools erstellen und Cloud-unabhängig sind.
- Open-Source IaC-Tools.

Die Kombination aus herstellerspezifischen IaC-Tools, Cloud-übergreifenden IaC-Tools und Open-Source-Lösungen sollte einen guten Mix ergeben. Dieser Mix sollte alle Tools enthalten, die für die Definition von Hybrid-Cloud-Automatisierungsstrategien im vierten Kapitel benötigt werden. Im nächsten Abschnitt werden die IaC-Tools Terraform, Pulumi, AWS CloudFormation, AWS CDK, Ansible, Chef, Puppet, Packer, Cloud-Init, CfEngine, Vagrant, NixOs, Salt beschrieben.

[1] Es werden IaC-Tools der Unternehmen RedHat und HashiCorp betrachtet, die zu diesem Anbietertyp gehören

3.2 Beschreibungen der IaC-Tools

Terraform

Terraform beschreibt IT-Ressourcen, stellt diese zur Verfügung und updated diese. Durch Anbieter-Plugins, wie das von AWS, können Anbieter-spezifische Services genutzt werden [29]. Bspw. ermöglicht das AWS Plugin AWS Services konfigurieren zu können. Terraform nutzt 'tf' Dateien mit HCL Syntax. HCL ist eine externe Domain Specific Language (DSL) (siehe 2.10.2), die neben Terraform auch bei Packer[2] zum Einsatz kommen.

In dem Listing 3.1 ist ein Terraform Beispielcode gegeben.

- Zunächst wird in Zeile 1-6 AWS als Anbieter spezifiziert. Über 'var.X' wird auf zuvor gesetzte Variablen zurückgegriffen.
- Ab Zeile 8 wird der Service AWS VPC beschrieben. Dieser Service wird mit dem parameter 'cidr-block' spezifiziert. Es sind noch weitere Konfigurationsmöglichkeiten innerhalb der VPC Ressource möglich, die mit Defaultwerten belegt sind und optional angepasst werden.

```
1  provider "aws" {
2      version = "~> 0.1"
3      region = "${var.region}"
4      access_key = "${var.access_key}"
5      secret_key =  "${var.secret_key}"
6  }
7
8  resource "aws_vpc" "vpc" {
9      cidr_block = "10.0.0.0/16"
10 }
```

Listing 3.1: Beispielcode einer Terraform-Datei. Beim Ausführen wird eine VPC-Service in AWS erstellt.

Terraform bietet eine Schnittstelle zur Erstellung von Plugins. Damit wird eine breite Palette zur Verwaltung von IT-Ressourcen angeboten. Terraform unterstützt Multi-Cloud Systeme[30][45].

Pulumi

[2] Packer ist ein weiteres IaC-Tool, das noch diskutiert wird.

Pulumi ist ähnlich zu Terraform und kann ebenso IT-Ressourcen zur Verfügung stellen. Im Gegensatz zu Terraform nutzt Pulumi höhere Programmiersprachen zum Definieren der IT-Ressourcen und verzichtet damit auf eine DSL. Services werden mithilfe von Libraries spezifiziert. Pulumi bietet eine Bibliothek mit 'best practice' Lösungen an, die Pulumi Crosswalk[3] heißt. Methoden in der Bibliothek abstrahieren mehrere Services und vereinfachen somit die Konfigurierung.

AWS CloudFormation

AWS CloudFormation ist ein Service von AWS zum Definieren, Erstellen und Provisionieren von AWS Services[13]. AWS CloudFormation arbeitet mit CloudFormation Stacks, in dem eine Sammlung von Services verwalten werden. Der CloudFormation Stack nutzt JSON oder YAML zur Angabe der Services. In Unterabschnitt 2.10.2 ist ein Beispiel einer AWS CloudFormation Datei gegeben.

AWS CDK

AWS CDK bietet die Möglichkeit mit höheren Programmiersprachen AWS CloudFormation-Stacks zu erstellen. AWS CDK bietet ein höheres Abstraktionslevel im Vergleich zu AWS CloudFormation, das durch Kapselung erzielt wird. Häufig wiederkehrende Pattern sind mit Methoden abstrahiert. AWS CDK setzt dies konsequenter um als Pulumi und verfolgt dieses Konzept durchgängig bei jedem Service. In dem Code-Schnipsel Listing 3.2 ist ein Beispiel gegeben. Es ist ein Ausschnitt einer Typescript Datei eines CDK Projektes gegeben, welches das höhere Abstraktionslevel im Vergleich zu AWS CloudFormation veranschaulicht. Der Ausschnitt gibt wie bei dem Terraform-Beispiel (siehe Listing 3.1) ein AWS-VPC an. Mit dem Parameter 'subnetType' wird implizit angegeben, wie das Netzwerk virtualisiert wird. Liegt die 'Public'-Einstellung vor, so wird das Subnet mit dem Internet verbunden. Wird das Subnet mit der 'Private'-Einstellung erstellt, so können IT-Ressourcen in diesem Netz über ein Proxy (NAT-Gateway) zugreifen[4]. Diese Subnet-Einstellungen sind in AWS-CloudFormation etwas umständlicher zu konfigurieren.

```
1 new ec2.Vpc(this, "VPC", {
2   subnetConfiguration: [
3     {
4       cidrMask: 24,
5       name: "ingress",
6       subnetType: ec2.SubnetType.PUBLIC,
```

[3] Pulumi Crosswalk Bibliothek: https://www.pulumi.com/docs/guides/crosswalk/aws/

[4] Es gibt noch eine weitere Einstellungsmöglichkeit, die 'Isolated'-Einstellung, mit dieser kann das Internet nicht erreicht werden.

```
    },
    {
      cidrMask: 24,
      name: "application",
      subnetType: ec2.SubnetType.PRIVATE,
    },
  ],
});
```

Listing 3.2: AWS CDK Beispiel

Ansible

Ansible wird für die Bereitstellung und Konfiguration an Maschinen verwendet. In einer YAML-Datei wird ein Ansible-Playbook mit Tasks definiert. Tasks werden nacheinander auf einem oder mehreren Zielmaschinen ausgeführt[5]. In einer Task kann ein Befehl oder ein Skript ausgeführt, Bibliotheken installiert oder Systemaufrufe getätigt werden.

In dem Listing 3.3 ist ein Ansible-Playbook als Beispiel gegeben. In Zeile 3 wird die Zielmaschine angeben ('Hosts'). Ab Zeile 4 werden 'Tasks' angeben, die sequenziell ausgeführt werden. In Zeile 8 wird die UFX library genutzt, um Firewall Einstellungen auf der Zielmaschine vorzunehmen.

```
-
  name: Play1
  hosts: localhost
  tasks:
    - name: Execute script on server
      script: test.sh

    - name: UFW - Allow SSH connections
      ufw:
        rule: allow
        name: OpenSSH
```

Listing 3.3: Beispiel YAML-Datei eines Ansible-Playbooks.

Chef

Chef ist ein Konfigurationsmanagement-Tool, das Ruby nutzt. In sogenannten 'Receipts' werden wie bei Ansible Schritte zur Konfiguration zum Ausführen beschrieben. Im Ge-

[5] Ansible muss auf den Zielmaschinen keine Applikation ausführen, um Änderungen vornehmen zu können. Änderungen werden 'Agentless' über SSH vorgenommen

gensatz zu Ansible nutzt Chef einen zentralen 'Chef Infra Server' zum Konfigurieren von Servern.

> "Der Chef Infra Server fungiert als Drehscheibe für Konfigurationsdaten. Der Chef Infra Server speichert Kochbücher ('cookbooks'), die Richtlinien, die auf Knoten angewendet werden, und Metadaten, die jeden registrierten Knoten beschreiben, der von Chef Infra Client verwaltet wird. Knoten verwenden Chef Infra Client, um den Chef Infra Server nach Konfigurationsdetails zu fragen, wie Rezepte, Vorlagen und Dateiverteilungen. Chef Infra Client erledigt dann so viel der Konfigurationsarbeit wie möglich auf den Knoten selbst (und nicht auf dem Chef Infra Server). Dieser skalierbare Ansatz verteilt den Konfigurationsaufwand in der gesamten Organisation."[20]

Puppet

Puppet ist ein weiteres Konfigurationsmanagement-Tool, das Server über eine zentrale, in Ruby geschriebene Spezifikation konfiguriert. Puppet arbeite in dem Client Server Model. In der push- oder pull-Methode wird bei dem gewünschten Server der Status angepasst und aufrecht erhalten.

Packer

> "Packer automatisiert die Erstellung von Maschinen-Images. Es umfasst ein Konfigurationsmanagement, welches das Installieren und Konfigurieren von Maschinen-Images ermöglicht."[32]

Wird in AWS eine EC2 Instanz erstellt, wird ein Machine Image[6] vom Amazon Machine Images (AMI) Service ausgewählt[12]. Packer kann auf Public-Cloud VM Services wie AWS EC2 Maschinen Images bereitstellen. Public-Cloud-Anbieter bieten Anwendern Maschinen Images an, mit denen VM's gestartet werden können. In der Regel werden 'Base-Images' angeboten, die nicht unbedingt anwenderspezifische Anforderungen erfüllen. Packer kann genutzt werden, um Maschinen Images zu optimieren und anwenderspezifische Anforderungen zu erfüllen[7].

Cloud-Init

[6] Ein Maschinen-Image ist ein Blueprint einer VM (siehe Abschnitt 2.5).

[7] Unternehmen haben oft rechtliche Einschränkungen, die eingehalten werden müssen. Die Konfiguration von Maschinen-Images ist eine sichere Option, um sicherzustellen, dass diese Anforderungen erfüllt werden.

Cloud-Init wird zum initialen Konfigurieren einer VM genutzt[8]. Cloud-Init wird u.a. bei AWS EC2 Instanzen verwendet, um Konfigurationen initial auszuführen[31].

CfEngine

CFEngine ist ein weiteres Konfigurationstool. CFEngine verwendet dezentrale Agenten, die den Serverstatus überwachen und den definierten, gewünschten Serverstatus herstellen können. Die Agenten laufen auf jedem Server als zusätzliche Applikation[19].

Vagrant

Vagrant wird zum Konfigurieren, Bereitstellen und Erstellen von VM-Images verwendet[33]. In der in Ruby geschriebenen Vagrantfile wird der Server angegeben, die für die Anwendung genutzt wird[34]. Auf dem Server werden die Maschinen Images ausgeführt. In dem Listing 3.4 ist ein Beispiel einer Vagrantfile gegeben.Es wird eine VM mit der VirtualBox Provider[9] lokal erstellt. In Zeile 9 und 10 wird der VM Hardware Resourcen zugewiesen Vagrant bietet eine Schnittstelle, um sich mit der von Vagrant erstellten VM zu verbinden und Anweisungen auszuführen.

```
1  Vagrant.configure("2") do |config|
2      config.vm.box = "precise32"
3      config.vm.hostname = "myprecise.box"
4      config.vm.network :private_network, ip: "192.168.0.42"
5
6      config.vm.provider :virtualbox do |vb|
7        vb.customize [
8          "modifyvm", :id,
9          "--cpuexecutioncap", "50",
10         "--memory", "256",
11       ]
12     end
13   end
```

Listing 3.4: Beispiel einer Vagrantfile.

NixOs

[8] Es wäre auch möglich ein Maschinen Image zu erstellen, welches nötige Installationen beinhaltet. Maschinen-Images sind allerdings statisch und müssen bei einem update eines der Tools neu erstellt werden. Es ist daher sinnvoll, bei einer 'frisch' erstellten VM alle installierten Pakete initial up-zu-daten.

[9] Es zahlreiche Anbieter siehe 'www.vagrantup.com/docs/providers'

NixOS basiert auf Nix, einem Paketverwaltungssystem. NixOS erweitert diese Funktionalität, indem es Nix auf die Verwaltung von Konfigurationsdateien ausweitet. Durch die Erstellung von Systemkonfigurationen aus Nix-Expressions stellt NixOS sicher, dass solche Konfigurationen sich u.a. nicht gegenseitig überschreiben und zurückgerollt werden können. Diese Konfigurationen können auf verschiedenen Systemen angewandt werden[52].

Salt

Salt ist ein Konfigurationsmanagementsystem. Salt ist dafür gemacht, um den Status von VMs abzufragen und Schritte einzuleiten, sobald der gewünschte Status nicht zutreffend ist. Salt kann z. B. sicherstellen, dass Pakete installiert und geupdated sind und dass Applikationen ausgeführt werden. Salt kann Befehle entweder auf einzelnen Knoten oder unter Verwendung beliebiger Auswahlkriterien abfragen und ausgeführt werden[60].

3.3 IaC-Tool-Kategorien

In diesem Abschnitt werden die Iac-Tools in Kategorien eingeordnet. Eine Übersicht ist in der Tabelle 3.1 geben.

Aus den Beschreibungen der IaC-Tools geht hervor, dass die Iac-Tools Ansible, Chef, Puppet, Cloud-Init, Salt und CF-Engine zum Konfigurieren von IT-Ressourcen geschaffen sind. Diese Iac-Tools können unter Konfigurationstools gefasst werden. Die Konfigurationstool-Kategorie ist folgendermaßen definiert: “Ein IaC-Tool versteht man als Konfigurationstool, wenn es erstellte IT-Ressourcen in der Konfiguration anpassen kann.“

Des weiteren gibt es die IaC-Tools Terraform, AWS CDK, AWS CloudFormation und Pulumi, die Services verwalten. Diese IaC-Tools sind unter Managementtools gefasst. Die Managementtool-Kategorie ist folgendermaßen definiert: “Ein IaC-Tool versteht man als Managementtool, wenn es IT-Ressourcen erstellen, beschreiben, updaten und löschen kann.“

Packer spezifiziert eine VM und könnte als Packagingtool gefasst werden. Die Packagingtool-Kategorie ist folgendermaßen definiert: “Ein IaC-Tool wird als Packaging-Tool bezeichnet, wenn eine Initialkonfiguration einer IT-Ressource erstellt und wiederverwendet werden kann, die zudem plattformunabhängig ist.“

Vagrant und NixOs sind Mischtools, die Packagingtool und Konfigurationstool Aspekte kombinieren. In der Tabelle 3.1 sind die hier besprochenen IaC-Tools aufgelistet. Die

Tools Azure Resource Manager (ARM) und GCP DM wurden in der Beschreibung weggelassen, sind aber in der Tabelle enthalten, um die Produkte der Public-Cloud-Anbieter Azure und GCP zu erwähnen[10].

Tabelle 3.1: Übersicht der IaC Tools

Name	Konfig-urationstool	Manage-menttool	Packaging-tool	Datei Format	Public Cloud gebunden
Terraform	-	+	-	HCL	-
Pulumi	-	+	-	PY, TS, GO, C#	-
AWS CF	-	+	-	YAML, JSON	AWS
AWS CDK	-	+	-	PY, TS, JAVA, C#	AWS
ARM	-	+	-	JSON	Azure
GCP DM	-	+	-	YAML	GCP
Ansible	+	-	-	YAML	-
Chef	+	-	-	RB	-
Puppet	+	-	-	PP	-
Packer	-	-	+	HCL, JSON	-
Cloud-init	+	-	-	TXT	-
CFEngine	+	-	-	CF	-
Salt	+	-	-	YAML	-
Vagrant	+	-	+	RB	-
NixOs	+	-	+	PY & NIX	-

In dem nächsten Kapitel werden Hybrid-Cloud Strategien eingeführt, die IaC-Tools verwenden.

[10] Die IaC-Tools von Azure und GCP unterscheiden sich nur in Nuancen von den zuvor beschriebenen Management-Tools

4 Hybrid Cloud Automatisierungsstrategien

In diesem Abschnitt werden entwickelte Hybrid-Cloud Automatisierungsstrategien (HCA-Strategien) beschrieben. Eine HCA-Strategie ist die Beschreibung, um IT-Infrastruktur in einer Hybrid-Cloud automatisiert und Anwendungsfall-unabhängig zu verwalten. Bevor HCA-Strategien betrachtet werden, werden Herausforderungen einer Hybrid-Cloud in den Fokus gerückt. Die Herausforderungen müssen von einer HCA-Strategie überwunden werden.

4.1 Herausforderungen in der Hybrid Cloud

Die Hybrid-Cloud zeichnet sich durch physisch getrennten Netze aus, die erst überbrückt werden müssen, um ein kohärentes System bilden zu können. Es muss eine Netzwerkverbindung zwischen dem Public-Cloud-Netzwerk und dem Private-Cloud-Netzwerk hergestellt werden. Da eine Verlegung von Kabeln zwischen Public- und Private-Cloud wegen offensichtlichen Gründen nicht verhältnismäßig ist, wird das Problem logisch über das Internet gelöst. Wird das Internet genutzt, gibt es die Herausforderung der Service-Discovery. Neben dem Problem der Service Discovery gibt es auch eine Sicherheitsherausforderung. Die Kommunikation zwischen den beiden Netzwerken muss verschlüsselt über das (öffentlich zugängliche) Internet abgehalten werden. HCA-Strategien gehen auf die beiden Punkte ein. Neben diesen Herausforderungen gibt es noch eine Reihe von weiteren Herausforderungen, die oft vorkommen, aber Unternehmen- oder Projekt-abhängig sind, z. B. rechtliche Anforderungen oder Anforderungen bei der Migration von Daten[35]. In der folgenden Auflistung werden die beiden wichtigsten Herausforderungen eines hybriden Cloud-Systems noch einmal prägnant genannt.

1. Kommunikation über getrennte Netze (Private- und Public-Cloud)

2. Sichere Kommunikation in einer unsicheren Umgebung (Internet)

Andere angegebene Herausforderungen sind nicht unbedingt in jedem Hybrid-Cloud-System vorhanden und werden daher vernachlässigt. Nach der Verdeutlichung der Herausforderungen werden Ansätze skizziert, die der Ausgangspunkt für HCA-Strategien sein sollen.

4.2 Ansätze

Wie beschrieben, müssen HCA-Strategien getrennte IT-Infrastrukturen verbinden. Um einen Ansatzpunkt finden zu können, wird das OSI-Modell[75] herangezogen. HCA-Strategien setzen an Schicht-3 (Netzwerkschicht) oder Schicht-7 (Anwendungsschicht) des OSI-Modells an.

- Wird eine HCA-Strategie basierend auf der Schicht 7 entwickelt, wird die Kommunikation über ein in Schicht 7 liegendes Protokoll (z.B. HTTP) abgewickelt. Zum Beispiel könnte an jeder VM eine Schnittstelle veröffentlicht werden. Damit wäre eine Kommunikation praktisch möglich.
- Wird eine HCA-Strategie basierend auf der Schicht 3 entwickelt, wird die Kommunikation über ein in Schicht 3 liegendes Protokoll (z.B. TCP) abgewickelt. So kann beispielsweise eine VPN-Verbindung zwischen dem Netzwerk der Public-Cloud und der Private-Cloud aufgebaut werden.

Im nächsten Abschnitt werden HCA-Strategien beschrieben, die an 'Layer-3' oder 'Layer-7' des OSI-Modells anknüpfen.

4.3 Hybrid-Cloud Automatisierungsstrategien

Wie zu Beginn des Kapitels angeschnitten, umfasst eine hier vorgestellte Hybrid-Cloud Automatisierungsstrategie (HCA-Strategie) die folgenden Aspekte: 1. Eine Einführung, 2. Eine Übersicht der Infrastruktur, 3. Eine Herangehensweise, wie eine Applikation entwickelt und verwaltet werden kann mit Berücksichtigung von IaC Tools.

In der Abbildung 4.1 ist ein Baum gegeben. Die Blätter stellen HCA-Strategien dar, von denen im nächsten Abschnitt einige in Detail beschrieben werden. Von der Wurzel

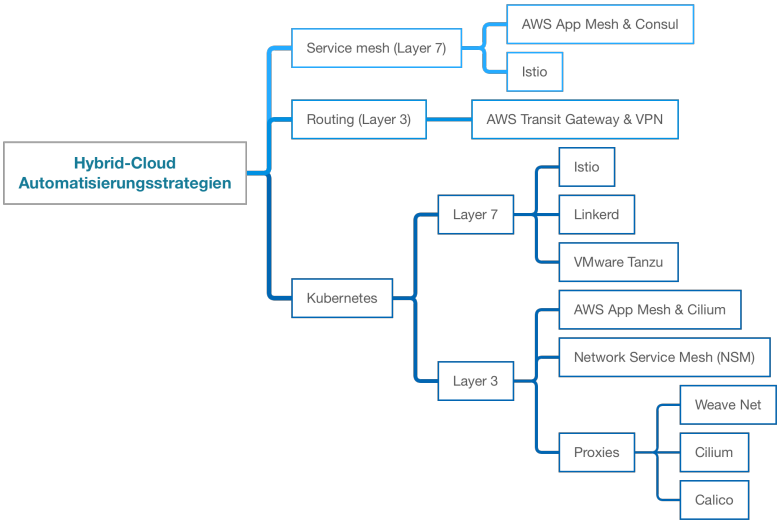


Abbildung 4.1: Übersicht der Hybrid-Cloud Automatisierungsstrategien (HCA-Strategien)

gehen die Zweige 'Routing' (Layer-3), 'Service-Mesh' (Layer-7) und 'Kubernetes' ab. Der 'Kubernetes'-Zweig fasst deutlich mehr Blätter als die beiden anderen Knoten und ist zur besseren Übersicht in zusätzliche innere-Knoten 'Layer-3' und 'Layer-7' aufgeteilt.

4.3.1 Routing

Der Zweig Routing in der Abbildung 4.1 umfasst nur einen Kindsknoten - 'AWS Transit Gateway'.

AWS Transit Gateway

AWS Transit Gateway ist ein AWS Service zur zentralen Verwaltung von AWS VPC- und VPN-Verbindungen zwischen innerhalb von AWS oder zwischen AWS und selbst verwaltenen Servern[5]. Server in Private-Cloud's können über VPN-Tunnel beim Transit Gateway Service registriert werden. Die VPN Verbindungen werden in einer Stern-Topologie aufgebaut. Der Transit Gateway Service steht im Zentrum dieser Topologie[1].

[1]Das Transit Gateway reduziert die Anzahl an nötigen VPN Verbindungen, da nur eine Verbindung per VM zum Transit Gateway aufgebaut werden muss, und so alle VMs über das Transit Gateway miteinander kommunizieren können.

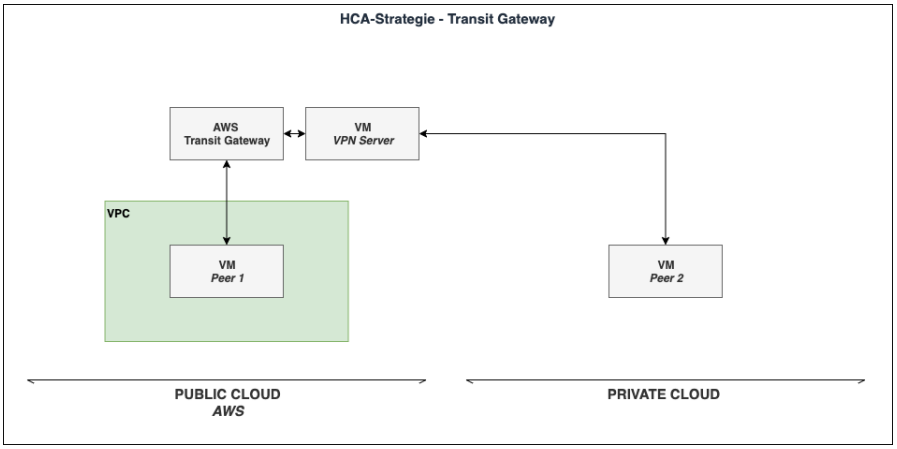


Abbildung 4.2: Infrastruktur der Transit Gateway HCA-Strategie

In der Abbildung 4.2 ist die Infrastruktur dieser HCA-Strategie abgebildet. In der Public Cloud wird der AWS Transit Gateway Service erstellt, der mit einem VPC verbunden ist, von dem aus Anfragen an die Private Cloud gehen. In der Private Cloud wird eine VM erstellt, die sich mit dem Transit Gateway verbindet und dann mit IT-Ressourcen im Public-Cloud VPC kommunizieren kann.

Wenn diese HCA-Strategie gewählt wird, müssen AWS-Services und virtuelle Maschinen verwaltet werden. Die erstellten Ressourcen müssen einen VPN-Tunnel einrichten, um eine Verbindung herzustellen. Die folgende Liste zeigt die Aktivitäten, die durchgeführt werden, um das Deployment der HCA-Strategie zu automatisieren.

1. Private Cloud-VMs werden mit Vagrant erstellt und Informationen über die VMs wie IP-Adressen werden zwischengespeichert.
2. Public-Cloud Services werden mit einem IaC-Managementtool erstellt und Informationen über die erstellten IT-Ressourcen werden zwischengespeichert.
3. Der VPN-Tunnel zwischen den VM's und dem Transit-Gateway wird mit Ansible-Playbook's initalisiert.

Die Verwaltung von AWS Services wird über ein IaC-Managementtool realisiert. Zum verwalten der VM's in der Private-Cloud kann 'Vagrant' verwendet werden. Um eine Verbindung zwischen den erstellten IT-Ressourcen aufzubauen, kann Ansible verwendet werden.

4.3.2 Service Mesh

Das Konzept des Service-Meshes wurde in Abschnitt 2.9 beschrieben und wird hier aufgegriffen. Der Zweig Service-Mesh in der Abbildung 4.1 umfasst zwei HCA-Strategien.

1. Die erste HCA-Strategie verbindet zwei Service-Meshes und nutzt dafür AWS App-Mesh & Consul.
2. In der zweiten HCA-Strategie wird mit Istio ein Service-Mesh genutzt, welches die Hybrid-Cloud überspannt.

Die zweite HCA-Strategie wird nicht ausgeführt, da dieser unter dem Kubernetes Abschnitt Istio in Kombination mit Kubernetes beleuchtet wird. Beide HCA-Strategien werden sich voraussichtlich ähneln. Es wird die Kubernetes Istio-HCA-Strategie beschrieben, da diese komplexer als die Service-Mesh Istio-HCA-Strategie ist[2].

App Mesh & Consul

Die Idee dieser HCA-Strategie ist, ein Service-Mesh in der Private-Cloud und ein weiteres in der Public-Cloud laufen zu lassen[3]. Es gibt somit zwei Service-Meshes mit zwei unabhängigen Control- und Data-Planes. Ziel ist, diese zunächst unabhängigen Service-Meshes miteinander zu verbinden. Dafür müssen beide Service-Meshes synchronisiert werden. Dies wird über eine Schnittstelle gelöst, welche die Service Erreichbarkeit vermittelt. Wird eine VM in Service-Mesh 'A' registriert, so wird das Service-Mesh 'B' darüber informiert. Services in beiden Cloud-Umgebungen können so einander entdecken und erreichen.

In der Abbildung 4.3 ist die Infrastruktur der HCA-Strategie gezeigt. In der Public-Cloud läuft AWS App-Mesh zusammen mit dem Service Cloud-Map[4] und dem Certificate-

[2] Die Kubernetes-Istio-HCA-Strategie verwendet neben dem Istio-Service-Mesh auch Kubernetes, was eine weitere Technologie in der HCA-Strategie bedeutet.

[3] Zwei unterschiedliche Service-Meshes zu nutzen könnte bei der Integration Cloud-Umgebung-spezifischer Anwendungen unterstützen. (AWS App-Mesh ist von Vorteil, wenn andere AWS-Services genutzt werden.)

[4] AWS Cloud Map ist ein Service zur Ermittlung von Cloud-Ressourcen. Mit Cloud Map können benutzerdefinierte Namen für Anwendungsressourcen definiert werden (siehe 'https://aws.amazon.com/de/cloud-map').

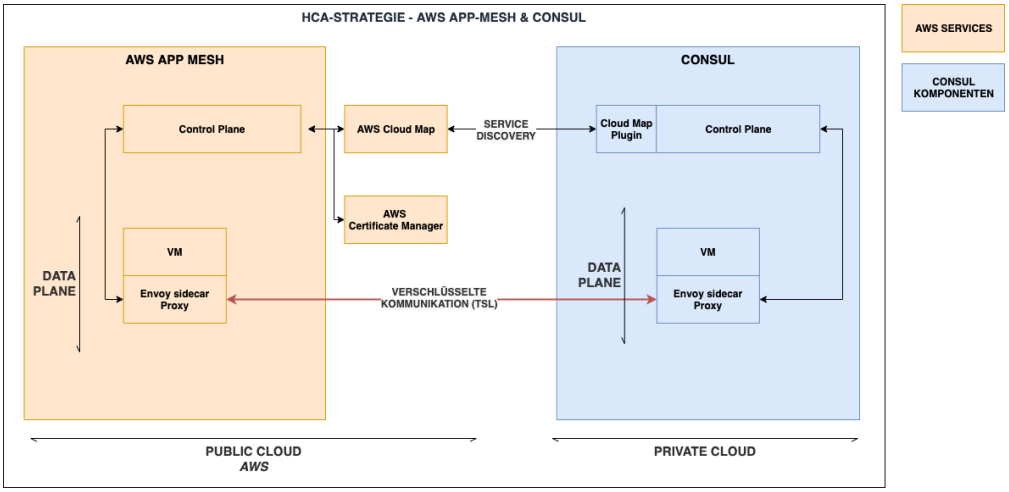


Abbildung 4.3: Infrastruktur der AWS App Mesh & Consul HCA-Strategie

Manager Service[5]. In der Private Cloud läuft Consul[6]. AWS Cloud-Map bietet Integrationsmöglichkeiten für Consul und App-Mesh, so dass eine Synchronisation der registrierten Services hergestellt werden kann[27]. Cloud-Map stellt somit die zuvor angesprochene Schnittstelle zur Verfügung, die die Service Discovery zwischen beiden Service-Meshes erlaubt. Die Kommunikation zwischen den beiden Service-Meshes wird über TLS verschlüsselt. Die für TLS nötigen Zertifikate werden im Certificate-Manager Service verwaltet.

Wird diese HCA-Strategie ausgewählt, so müssen AWS Services in der AWS Public-Cloud und Consul sowie VM's in der Private-Cloud verwaltet werden. Consul kann mit Ansible konfiguriert werden oder über Terraform, das ein Consul Plugin besitzt[28]. Die Verwaltung von AWS-Services wird über ein IaC-Managementtool realisiert. 'Terraform' ist empfohlen, da es auch für Consul verwendet werden kann. Zum Verwalten der VM's in der Private-Cloud kann 'Vagrant' verwendet werden.

4.3.3 Kubernetes

Der Zweig 'Kubernetes' in der Abbildung 4.1 umfasst acht HCA-Strategien. In diesem Abschnitt wird der Fokus jedoch nur auf zwei der acht aufgeführten HCA-Strategien

[5] AWS Certificate Manager ist ein Service, der öffentliche und private SSL/TLS-Zertifikate für die Verwendung mit AWS-Services und internen verbundenen Ressourcen bereitstellen, verwalten und einsetzen kann. (siehe 'https://aws.amazon.com/de/certificate-manager/')

[6] Consul ist eine Service-Mesh-Service, die eine vollwertige Control Plane mit Service Discovery-, Konfigurations- und Segmentierungsfunktionen bietet. Consul ist nicht an einen Public-Cloud-Anbieter gebunden (siehe 'https://www.consul.io/docs/intro')

gelegt. Dies liegt an dem großen Umfang, der damit verbunden ist, für alle acht Anwendungen HCA-Strategien zu beschreiben. Warum eine HCA-Strategie zur Beschreibung ausgewählt wurde oder nicht, wird nun der Reihe nach besprochen.

1. Das Service-Mesh Istio[7] ist unter den Service-Mesh HCA-Strategien aufgeführt und wird an dieser Stelle ein weiteres Mal in Kombination mit Kubernetes verwendet. Im Service-Mesh Abschnitt wurde Istio nicht beleuchtet und auf die Istio Kubernetes HCA-Strategie verwiesen.
   Die Kubernetes HCA-Strategie von Istio wird hier hervorgehoben und gibt ebenfalls eine Idee, wie eine Istio Service-Mesh HCA-Strategie aussieht.

2. Linkerd wird implizit bereits mit der Istio Kubernetes HCA-Strategie konzeptionell betrachtet. Es handelt sich hierbei ebenso um ein Service-Mesh, welches in Verbindung mit Kubernetes auf einen Hybrid Cloud System angepasst werden kann.
   Diese HCA-Strategie wird an dieser Stelle nicht genauer betrachtet.

3. Tanzu ist ein Service von vmWare welches vmWare-VM's voraussetzt. Diese HCA-Strategie ist daher durch diese Abhängigkeit weniger attraktiv gegenüber HCA-Strategien, die diese Einschränkung nicht haben.
   Tanzu wird hier nicht weiter beleuchtet.

4. Die AWS App Mesh & Cilium HCA-Strategie ist wieder mit einem Service Mesh aufgebaut, das mit Kubernetes arbeitet. App Mesh nutzt dafür den Cilium Proxy. AWS App-Mesh wurde im Service-Mesh Abschnitt besprochen und wird daher hier nicht noch einmal behandelt.

5. Network Service Mesh (NSM) wird implizit bereits mit der Istio Kubernetes HCA-Strategie konzeptionell betrachtet. Es handelt sich hierbei ebenso um ein Service Mesh, welches in Verbindung mit Kubernetes auf einen Hybrid Cloud System angepasst werden kann.
   Diese HCA-Strategie wird an dieser Stelle nicht genauer betrachtet.

6. Cilium wird unter dem Proxy-Knoten aufgeführt. Cilium wird noch in der HCA-Strategie AWS App Mesh & Cilium verwendet, die, wie oben erwähnt, an dieser Stelle nicht näher beschrieben wird. Cilium ist ein neues hochinteressantes Projekt und daher hervorzuheben.

[7] Istio ist für Hybrid-Cloud-Systeme interessant, weil sich Istio flexibel zusammenstellen lässt, breit Verwendung findet und somit komplexe Hybrid-Cloud-Systeme realisieren kann (siehe 'https://istio.io/')

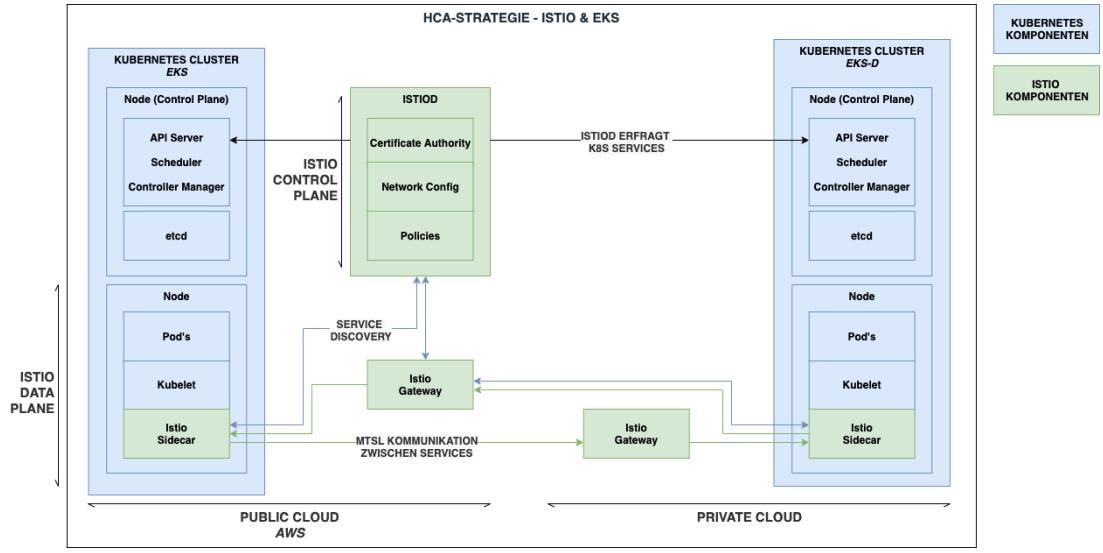


Abbildung 4.4: Infrastruktur der Istio EKS HCA-Strategie[62]

7. Calico ist unter dem Proxy Knoten aufgeführt. Calico ist ein populärer Proxy, der allerdings auf direkte Route Table manipulation setzt. Cilium zum Vergleich nutzt dafür eBPF[24], eine elegantere Form, dies zu erreichen.
   Calico wird nicht betrachtet.

8. WeaveNet ist unter dem Proxy Knoten aufgeführt.
   WeaveNet ist vergleichbar mit Calico und wird nicht beleuchtet.

Istio mit EKS

Die Idee dieser HCA-Strategie ist es, ein Service Mesh mit Kubernetes für beide Cloud-Umgebungen zu nutzen. Es gibt somit ein Control- und Data-Plane, welches beide Cloud-Umgebungen überspannt. Istio wird an dieser Stelle ausgewählt, um das Service Mesh zu verwalten. Istio erlaubt verschlüsselte Kommunikation mit mTSL[39]. Neben Istio werden zwei Kubernetes Cluster in beiden Cloud Umgebungen gestartet. Das Kubernetes Cluster kann mit dem Service AWS Elastic Kubernetes Service (AWS EKS)[7] verwaltet werden. AWS EKS ist ein von AWS verwaltener Kubernetes Infrastruktur Service. Kubernetes kann auch komplett manuell aufgesetzt werden. AWS EKS ist an dieser Stelle allerdings besonders interessant, da es seit Ende 2020 eine EKS-Distro gibt. Über die EKS-Distro (EKS-D)[6] kann AWS EKS auch auf Private Cloud Instanzen installiert werden. AWS EKS unterstützt somit einen Hybrid Cloud Anwendungszweck[10].

In der Abbildung 4.4 ist die Infrastruktur der HCA-Strategie gezeigt. Es sind zwei Kubernetes Cluster sowie das Control- & Data-Plane von Istio zu erkennen. Die Kubernetes Cluster sind getrennt voneinander und beinhalten jeweils einen Master-Node. Wie zuvor erwähnt, kann AWS EKS in der Public Cloud und EKS-D in der Private Cloud verwendet werden. Die Istio Control Plane wird in einer der beiden Cloud-Umgebungen erstellt und kommuniziert über Istio Gateways, die in jeder der beiden Cloud-Umgebungen einmal implementiert werden. Es lassen sich mehrere Kommunikationspfade erkennen, die über Pfeile dargestellt sind und nun nacheinander betrachtet werden.

- Der schwarze Pfeil zweigt, dass Istiod über die Kubernetes API Endpoints der Worker-Nodes anfragt.
- Der blauer Pfeil zeigt, wie Services den Endpoint eines anderen Services erhalten.
- Der gelbe Pfeil zeigt, wie die Kommunikation zwischen Services abläuft. Die Kommunikation zwischen Worker Nodes / Services wird über mTLS verschlüsselt[39]. Beide Cloud-Umgebungen verwalten ein Istio-Gateway, das Anfragen an die Services weiterleitet.

Wird diese HCA-Strategie ausgewählt, so müssen AWS Infrastruktur, die Istio Infrastruktur und VM Instanzen verwaltet werden. Zur Umsetzung werden die IaC Tools AWS CDK, Ansible und Packer ausgewählt. Ansible ist in der Istio-Dokumentation zum Konfigurieren und Deployment ausgiebig dokumentiert und bietet sich daher an[38]. Packer wird Vagrant vorgezogen, da Ansible mit Packer zusammen verwendet werden kann. Ansible übernimmt das Deployment der VM-Images, das von Packer nicht unterstützt wird. Vagrant könnte auch verwendet werden, bietet aber mehr Funktionalität, als hier gebraucht wird. AWS CDK wird zum Konfigurieren der AWS Services genutzt. Es könnte an stelle von AWS CDK auch jedes andere Managementtool ausgewählt werden (ausgenommen ARM und GCP DM). AWS EKS-D kann nicht mit AWS CDK oder einem anderen Managementtool verwaltet werden. Der AWS EKS-D Cluster wird mit einem weiteren Ansible Playbook konfiguriert. Die nachfolgenden Punkte werden sequenziell nacheinander aufgerufen und zeigen, wie diese HCA-Strategie automatisiert werden kann.

1. Erstelle VM-Image mit Packer
2. Erstelle Instanzen in Public- und Private-Cloud mit VM-Image
3. Erstelle AWS Services mit AWS CDK in der Public-Cloud und registriere Public-Cloud VMs

4. Erstelle AWS EKS-D mit Ansible in der Private-Cloud und registriere Private-Cloud VMs
5. Konfiguriere Istio Infrastruktur in der Public- und Private-Cloud mit Ansible
6. Starte Kubernetes Pods mit VM-Image über Ansible in Public- und Private-Cloud

Diese Konfiguration ist sehr komplex und es ist zu erwarten, dass weitere Schritte, wie das Ausführen von Ansible-Playbooks oder Shell-Skripten, um Konfigurationen vorzunehmen, erforderlich sind, nachdem der Anwendungsfall festgelegt wurde.

Kubernetes Multi Cluster mit Cilium Proxies

Die Idee dieser HCA-Strategie ist es einen Cilium Multi-Cluster mit Kubernetes aufzusetzen. Cilium ist als Proxy auf Kubernetes Worker- und Master-Node eingesetzt, was im Kern den Linux Kernel über eBPFs anpasst, um u.a. darüber effizient Routing zu steuern. eBPF ist eine Technologie, die Programme im Linux-Kernel ausführen kann, ohne den Kernel-Quellcode zu verändern oder Kernel-Module zu laden. [24]

> "Der Berkeley Packet Filter (BPF) ist eine in-Kernel virtuelle Maschine für Paketfilterung, die ab 2013 tiefgreifend überarbeitet wurde und nun als erweiterter BPF (eBPF) bekannt ist. Zusätzlich zu einigen architektonischen Verbesserungen führt eBPF die Fähigkeit ein, generische Ereignisverarbeitung im Kernel zu handhaben, JIT-Kompilierung für höhere Leistung, stateful processing unter Verwendung von Maps und Bibliotheken (helpers) zur Handhabung komplexerer Aufgaben, die im Kernel verfügbar sind[58]" .

In der Abbildung 4.5 ist die Infrastruktur dieser HCA-Strategie gezeigt. Um die Komponenten der Architektur einordnen zu können, sind Kubernetes Kenntnisse vorausgesetzt. Die beiden Kubernetes Cluster erstellen etcd Proxies die über TSL verschlüsselt erreicht werden können. Die etcd proxies ermöglichen Lesezugriff auf den Kubernetes etcd Key-Value Store. Die Cilium Agents, die als Sidecar in jeder Kubernetes Worker-Node laufen, beobachten diesen Proxy des jeweils anderen Kubernetes Clusters. Agents im Private Cloud Kubernetes Cluster schauen auf den etcd Proxy des Public Cloud Kubernetes Clusters und anders herum. Änderungen im etcd werden somit abgefangen und im eigenen etcd nachgetragen. Beide etcd Datenbanken sind gespiegelt. Die Cilium Agents ermöglichen verschlüsselte Kommunikation zwischen den Clustern.

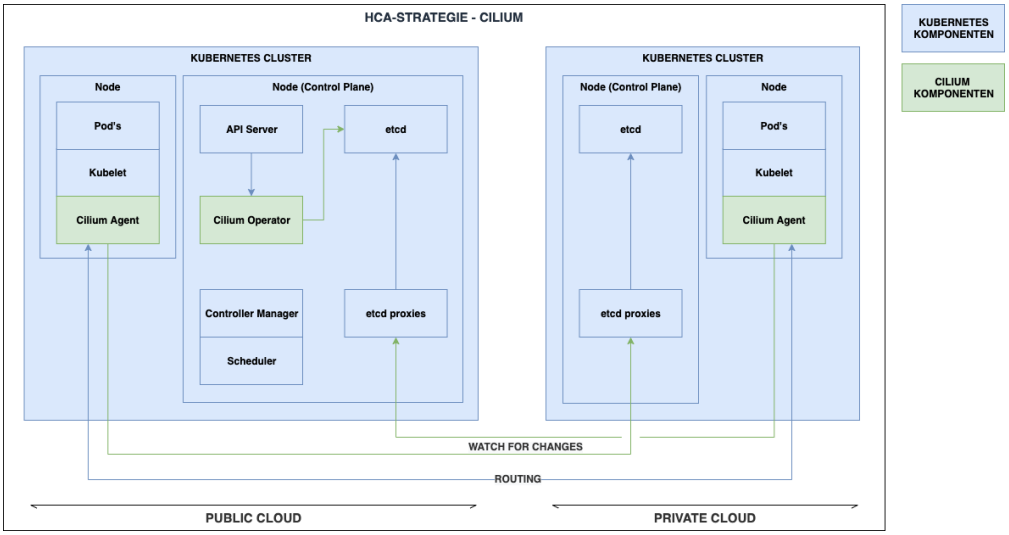


Abbildung 4.5: Infrastruktur der Kubernetes Cilium HCA-Strategie[22]

Wird diese HCA-Strategie ausgewählt, so müssen VM Images sowie Kubernetes konfiguriert werden. Für VM Images kann zwischen Packer und Vagrant entschieden werden. Zum Konfigurieren von Kubernetes bietet es sich an, Ansible zu nutzen. Zum vereinfachten Erstellen von Kubernetes Clustern kann Kubespray genutzt werden[61], das mit Ansible Playbooks ausgeführt wird.

5 Fallstudie HCS-DIRECT-CONNECT

In diesem Kapitel wird eine Fallstudie erarbeitet. Gegenstand der Fallstudie ist die Implementierung und Beobachtung einer Hybrid-Cloud-Automatisierungsstrategie. Der Projektentwicklungsprozess folgt dem Wasserfallmodell und ist in die Abschnitte: Anforderungen, Design, Implementierung, Verifizierung und Wartung unterteilt. Auf die letzteren Beiden kann in dieser Fallstudie verzichtet werden[1].

5.1 Anforderungen

In diesem Abschnitt werden die Anforderungen an das komplexe Softwareprojekt eruiert. Die Unterabschnitte folgen dem Schema einer Software Requirements Specification (SRS)[36] folgen.

5.1.1 Einführung

Überblick der Anforderungen an die Fallstudie

Die Fallstudie beinhaltet, den Entwurf einer Hybrid-Cloud, welche sich an einer der erarbeiteten Hybrid-Cloud-Automatisierungsstrategien orientiert (siehe Kapitel 4). Um die Hybrid-Cloud-Automatisierungsstrategie bewerten zu können, wird eine Beobachtungskomponente in die Fallstudie integriert. Diese beiden Komponenten werden als innere-Komponente und als äußere-Komponente aufgefasst.

1. Das innere-System implementiert eine Hybrid-Cloud-Automationsstrategie (Kontext wird noch definiert).

[1] Die Verifikation der erstellten Infrastruktur ist nicht Gegenstand der Arbeit. Im letzten Kapitel dieser Arbeit wird die Verifikation des IaC-Codes als Ausgangspunkt für weitere Untersuchungen zur Disposition gestellt.
Zu einer Wartung des Projektes kommt es nicht und daher wird diese Phase nicht erreicht.

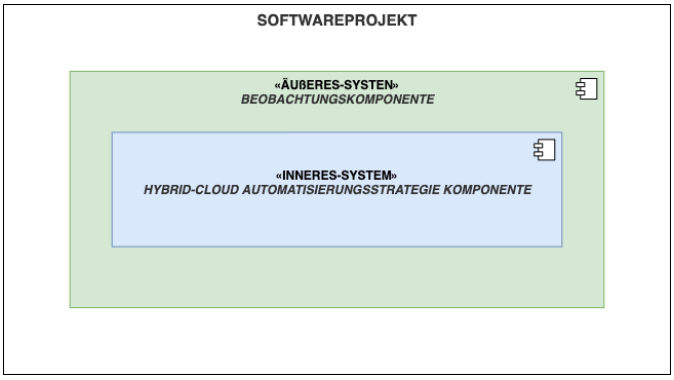


Abbildung 5.1: Projektübersicht

2. Das äußere-System beobachtet das innere-System und protokolliert Ergebnisse.

In der Abbildung 5.1 ist eine Übersicht des Projektes gezeigt. Im nächsten Abschnitt werden Zweck sowie die Ziele der Fallstudie besprochen.

Zweck der Fallstudie

Dieses Softwareprojekte verfolgt das Ziel, eine Hybrid-Cloud-Automatisierungsstrategie umzusetzen. Folgende Punkte sollen dabei bewertet werden können.

1. Unterstützen IaC-Tools bei der Automatisierung von Softwareprojekten?
   a) Entwicklungsprozess
   b) Verwaltung von IT-Ressourcen / IT-Infrastruktur
2. Sind die zuvor beschriebenen Herausforderungen einer Hybrid-Cloud überwunden worden? (siehe Abschnitt 4.1)
3. Welche Probleme sind bei der gewählten Automatisierungsstrategie aufgetreten oder absehbar und für welche Probleme könnten andere Automatisierungsstrategien eine Lösung bieten?
4. Wie ausgereift sind die verwendeten Public-Cloud-Services?
   a) Wie lassen sich IaC-Tools integrieren?
   b) Wird ein Hybrid-Cloud-System unterstützt?

c) Wie nutzerfreundlich ist die Public-Cloud?

Diese Fragen werden in der Analyse der Fallstudie aufgegriffen und anhand des entwickelten Systems ausgewertet.

5.1.2 Allgemeine Beschreibung

Wie in der Einleitung erwähnt, sind zwei Systeme zu implementieren - das innere- und das äußere-System (siehe Abbildung 5.1). Das äußere-System beobachtet und zeichnet die Aktivitäten des inneren-Systems auf. Das innere-System bildet ein System ab, das in einem anschließend definierten Anwendungsfall von einem Nutzer verwendet wird.

Anwendungsfall

Der Anwendungsfall des inneren-Systems ist im Umfang überschaubar zu halten, da eine komplexere Logik in der Anwendung nicht zwangsläufig eine Verbesserung in der Beobachtung zur Folge hat. Es werden nun drei Anwendungsfälle beschrieben, wovon eine als Anwendungsfall des inneren-Systems ausgewählt wird. Da eine Hybrid-Cloud-Automatisierungsstrategie implementiert wird, muss die Anwendung verteilt in der Private- und Public-Cloud ausgeführt werden[2].

1. Der erste Anwendungsfall implementiert einen verteilten Algorithmus. Es gibt eine Vielzahl von detailliert definierten Algorithmen[43][46]. Dieser Anwendungsfall setzt den Lamports Bakery-Algorithmus[43] um. Die Private-Cloud wird als kritischer-Bereich definiert, worauf Klienten, die in der Public-Cloud angesiedelt sind, zugreifen.

2. Der zweite Anwendungsfall bildet eine Cloud-Anbindung eines Legacy Systems ab. Das Legacy-System ist in der Private-Cloud angesiedelt und wird über Services in der Public-Cloud angesprochen.

[2] Es hätten auch weitere Anwendungsfälle beschrieben werden können. Je mehr Anwendungsfälle beschrieben und verglichen werden, desto weniger wird voraussichtlich jeder einzelne Anwendungsfall an neuen Aspekten zur Diskussion beitragen und umso aufwändiger fällt die Analyse im nach hinein aus - drei Anwendungsfälle sollten interessante Aspekte beleuchten, ohne zu ausführlich an dieser Stelle zu werden.

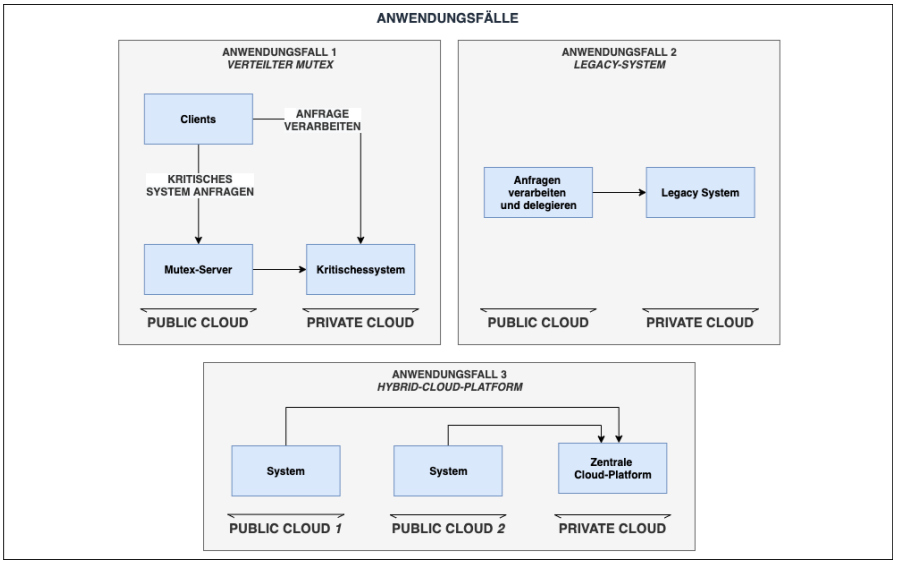


Abbildung 5.2: Anwendungsfälle

3. Der dritte und letzte Anwendungsfall zielt darauf ab, mehrere Public-Cloud-Anbieter, wie in einer Multikernel-Architektur als 'Plug-ins' an eine zentrale Komponente anzubinden. Wird dieser Anwendungsfall gewählt, werden in der Privat-Cloud zentral Log-Einträge verwaltet, die von der Public-Cloud aus geschickt werden. In der Public-Cloud wird ein System, bestehend aus Frontend, Backend verwaltet, um Log-Einträge zu erzeugen.

In der Abbildung 5.2 sind die drei Anwendungsfälle grafisch veranschaulicht. Um einen Anwendungsfall für diese Fallstudie auszuwählen, werden Vor- und Nachteile der aufgelisteten Optionen durchgegangen.

1. Verteilter Mutex

- (+): Der erste Anwendungsfall implementiert einen verteilten Algorithmus, der definiert ist. Es liegen Implementierungen vor, die als Referenz herangezogen werden können. Unvorhersehbare Herausforderungen werden dadurch minimiert.
- (-): Der erste Anwendungsfall führt eine zusätzliche Logik ein, die bei der Beobachtung des Systems praktisch keinen Vorteil mit sich bringt, da bei der Beobachtung die Kommunikation Gegenstand der Bewertung ist. Ob ein Algorithmus korrekt kommuniziert und nach Definition abläuft, ist nicht relevant. Daher ist es

redundant, einen Algorithmus im Anwendungsfall zu implementieren. Die Implementierung des Algorithmus verlangsamt den Entwicklungsprozess und stellt eine zusätzliche Fehlerquelle in dem System dar.

2. Legacy-System

- (+): Der zweite Anwendungsfall ist bei der Betrachtung der Komponenten übersichtlich gestaltet. Der Anwendungsfall verfolgt das Ziel, Authentifizierungsanfragen verarbeiten zu können. Da der Anwendungsfall frei definiert und nicht von einem Kunden vorgegeben ist, kann die Komplexität auf ein Minimum beschränkt werden.
- (-): -

3. Hybrid-Cloud Platform 2. Legacy-System

- (+): Der dritte Anwendungsfall verbindet mehrere Public-Cloud-Anbieter miteinander. Es wäre daher einfach, das innere-System zu erweitern, um eine weitere Public-Cloud zu testen.
- (-): Der dritte Anwendungsfall ist kompliziert zu planen, da generische Schnittstellen der Kernkomponente zu entwickeln sind, die mit jeder Public-Cloud-Umgebung arbeiten können. Um die Schnittstellen generisch zu gestalten, sind tiefe Kenntnisse aller untersuchten Public-Cloud-Anbieter notwendig. Die parallele Betrachtung von zwei Public-Cloud-Umgebungen ist nur dann interessant, wenn beide Public-Cloud-Umgebungen in einem System parallel genutzt werden[3]. Ist dies nicht der Fall, ist eine Anbindung mehrerer Public-Cloud Umgebungen für Beobachtungen nicht vorteilhaft. Es könnte in diesem Fall auch die erste Public-Cloud getestet werden und dann in einem anderen System die nächste.

Nachdem die Vor- und Nachteile aufgezeigt wurden, sticht der zweite Anwendungsfall wegen des geringen Umfangs und keinen aufgeführten Nachteilen positiv heraus und wird daher ausgewählt. In den nächsten Abschnitten wird der zweiter Anwendungsfall detaillierter beschrieben.

[3]Das wäre der Fall, wenn beispielsweise eine Anfrage in Public-Cloud 1 gestellt wird, dann eine Komponente in der Public-Cloud 2 anfragt und dann die log Einträge an die Private-Cloud schickt.

Benutzermerkmale

Die Unterteilung des Projektes in inneres- und äußeres-System legt nahe, dass zwei Nutzer diese beiden Systeme nutzen. Das Projekt wird von zwei von Benutzern verwendet.

- Der Benutzer des inneren-Systems will das System nutzen, um sich zu Authentifizieren und wird als *User* angesprochen.
- Der Benutzer des äußeren-Systems will die Funktionalität des inneren-System verifizieren und wird als *Dev* angesprochen.

Eine genauere Betrachtung der beiden Benutzer mit einer Persona[51] wird hier nicht durchgeführt. Bei einem komplexeren Projekt wäre dies möglicherweise angebracht.

Produktfunktionen

In der nachfolgenden Liste werden die Produktfunktionen des äußeren- und inneren-Systems aufgelistet. Hiermit wird gezeigt, welche Funktionen die Akteure Dev und User im System anwenden können.

- Produktfunktionen des äußeren-Systems
    1. Der Dev kann eine Datei aller Beobachtungen generieren lassen.
    2. Der Dev kann die Beobachtung des inneren-Systems starten.
    3. Der Dev kann die Beobachtung des inneren-Systems stoppen.
    4. Der Dev kann den aktuellen Status der Beobachtung anzeigen lassen.
- Produktfunktionen des inneren-Systems
    1. Die User können ein User-Profil registrieren.
    2. Die User können sich mit dem zuvor registrierten User-Profil anmelden.
    3. Die User können sich nach erfolgreicher Anmeldung das User-Profil anzeigen lassen.

Im nächsten Abschnitt wird beschrieben, welche übergreifenden Einschränkungen definiert werden.

Allgemeine Beschränkungen

Folgende Punkte zeigen auf, wie das Softwareprojekt vom Umfang begrenzt wird. Diese Begrenzung im Softwareentwicklungsprozess soll dabei helfen den Fokus auf die zuvor abgesteckten Ziele zu halten. Sollte das Projekt zu einem späteren Zeitpunkt weiterentwickelt werden, könnten dies die Ausgangspunkte für ein verbessertes Produkt sein.

1. Auf eine UI wird verzichtet.
2. Das innere-System hat eine öffentliche Schnittstelle zum äußeren-System.
3. Es wird eine Hybrid-Cloud- Automatisierungsstrategie aus vorherigen Abschnitten herangezogen und ausgearbeitet.
4. Beobachtungsergebnisse müssen manuell ausgewertet werden.
5. Es wird angenommen, dass der User und Dev die Schnittstellen kennt und auf eine ausführliche Dokumentation verzichtet werden kann.

Diese Einschränkungen werden nun detaillierter beschrieben.

5.1.3 Detailliertere Anforderungen

Architektonische Einschränkungen

Zu Beginn des Abschnitts wurde beschrieben, dass in diesem Kapitel eine Hybrid-Cloud-Automatisierungsstrategie ausgewählt und implementiert werden soll. Im vorherigen Kapitel wurden vier HCA-Strategien beschrieben (siehe Kapitel 4). Aus den Beschreibungen ist ersichtlich, dass die HCA-Strategie Transit Gateway den geringsten Umfang hat (siehe Abschnitt 4.3.1). Die anderen Hybrid-Cloud Automatisierungsstrategien bieten Vorteile, die bei dieser minimalistisch angelegten Fallstudie nicht zum Tragen kommen.

Funktionale und Nicht-funktionale Anforderungen

In der nachfolgenden Liste sind die funktionalen und nicht-funktionalen Anforderungen aufgelistet.

Funktionale Anforderungen

1. Das innere-System soll in der Lage sein, zwischen Public- und Private-Cloud zu kommunizieren.
2. Das innere-System soll die Kommunikation zwischen der Public- und Private-Cloud verschlüsseln.
3. Das innere-System soll Log-Einträge zentral speichern.
4. Das innere-System muss über eine öffentliche Schnittstelle in der Public-Cloud zugänglich sein.
5. Das äußere-System sollte in der Lage sein, die Funktionalität des inneren-Systems zu verifizieren

Nicht-funktionale Anforderungen

1. Das System sollte ein Budget von 50 Euro pro Monat nicht überschreiten.
2. Das System soll die Verarbeitung über Log-Einträge transparent abbilden.
3. Das äußere-System soll in unter einer Minute die Funktionalität des inneren-Systems verifizieren können.
4. Das innere-System soll IaC-Tools verwenden, um IT-Ressourcen zu erstellen.

Die Spezifizierung der Abhängigkeiten ist hiermit abgeschlossen. Im nächsten Abschnitt 'Design' wird die nächste Phase des Projektentwicklungsprozesses begonnen.

5.2 Design

In diesem Abschnitt wird die Software-Architektur der Fallstudie HCS-DIRECT-CONNECT entwickelt. Die Software-Architektur folgt dem arc42[63][64] Software-Architektur-Schema[4]. Das Projekt heißt HCS-Direct-Connect, was sich zusammensetzten lässt aus Hybrid-Cloud-System (HCS) und Direct-Connect, was von der Transit-Gateway HCA-Strategie stammt, die auf direkte VPN-Tunnelverbindungen setzt.

In diesem Abschnitt werden zwei Designs ausgearbeitet. Zunächst wurde der erste Entwurf entwickelt, der zu einer unnötig komplexen Softwarearchitektur führte. Das zweite

[4] Annahmen und Abhängigkeiten wurden im vorherigen Abschnitt beschrieben und werden daher nicht erneut aufgeführt

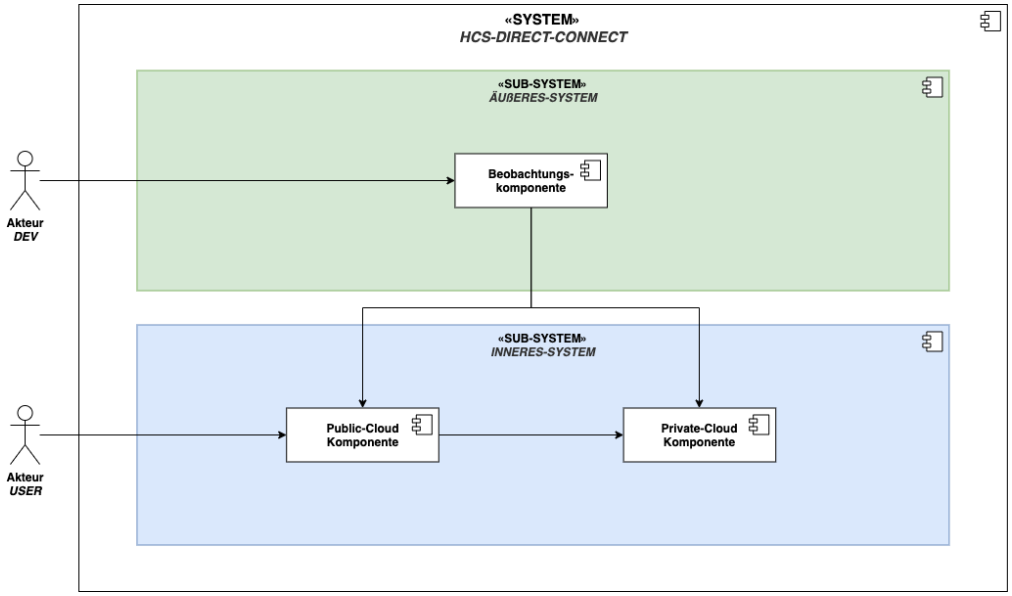


Abbildung 5.3: Kontextsicht

Design baut auf dem ersten Design auf und passt es an.
Der Hauptunterschied zwischen den beiden Designs besteht in der Art wie die Beobachtung des Hybrid-Cloud-Systems durchgeführt wird. Das erste Design beobachtet das System über Log-Einträge. Das zweite Design beobachtet das System über Health Checks.

5.2.1 Design 1: Verifikation über log-Einträge

Kontext und Umfang

In der Abbildung 5.3 ist die Kontextsicht des Softwareprojektes gezeigt. Es sind zwei Sub-Systeme zu sehen, die von zwei verschiedenen Akteuren, dem Dev und dem User, verwendet werden.
Die Beobachtungskomponente prüft, ob das Innere-System als Hybrid-Cloud funktioniert und hat daher zwei Abhängigkeiten zu den Komponenten. Die Public-Cloud-Komponente fragt die Private-Cloud-Komponente an und hat daher eine Abhängigkeit[5].

[5]Wenn das System erstellt wird, besteht keine zyklische Abhängigkeit zwischen den Komponenten, sodass die Erstellung von Ressourcen in den Cloud-Umgebungen iterativ ausgerollt werden kann

Lösungsstrategie

Die Lösungsstrategie wird in zwei Abschnitten vorgestellt, die folgende zwei Fragen behandeln:

1. Wie wird die Hybrid-Cloud-Automatisierungsstrategie verwendet?
2. Wie wird die Hybrid-Cloud beobachtet?

Hybrid-Cloud-Automatisierungsstrategie

Die für diese Fallstudie ausgewählte Transit Gateway Hybrid-Cloud- Automatisierungsstrategie gibt einen Rahmen vor, wie ein Projekt strukturiert werden soll (siehe Abschnitt 4.3.1). Für diesen Anwendungsfall muss die Automatisierungsstrategie angepasst und detaillierter beschrieben werden. Damit die Automatisierungsstrategie weiter ausgearbeitet werden kann, wird zunächst ein Blick auf die Kommunikation zwischen Public- und Private-Cloud geworfen.

Die Kommunikation zwischen Public- und Private-Cloud wird über VPN-Tunnel geleitet. Um dies zu ermöglichen, werden die AWS-Services 'Transit-Gateway' und 'Direct-Connect' eingesetzt. Diese beiden Services müssen auf die Tauglichkeit in diesem Projekt geprüft werden.

- AWS Transit-Gateway Service: Der Transit Gateway Service erlaubt mehrere AWS Virtual Private Networks (VPC) und VPN Verbindungen zu verwalten. Im System sind drei Netzwerke miteinander zu verbinden, Public-Cloud, Private-Cloud und das Netzwerk der Beobachtungs-Systems. In unserem Anwendungsfall könnte der Transit Gateway Service verwendet werden, würde aber wegen der geringen Anzahl von Netzwerken nicht viel Sinn ergeben und übermäßige Kosten verursachen.
- AWS Direct Connect: Dieser Service ist einer von AWS verwalteter VPN Service, der hier genutzt werden kann. Wie erwähnt bilden VPN Verbindungen das Fundament dieser Hybrid-Cloud-Automatisierungsstrategie. Um Flexibilität und erhöhte Transparenz zu erlangen, wäre eine Open-Source Lösung wie WireGuard vorzuziehen.

Teil der Lösungsstrategie ist es die Kommunikation über VPN-Tunnel zu leiten und dabei auf das Open-Source Tool WireGuard zu setzen. Der AWS Transit-Gateway Service und der AWS Direct-Connect Service wird nicht eingesetzt.

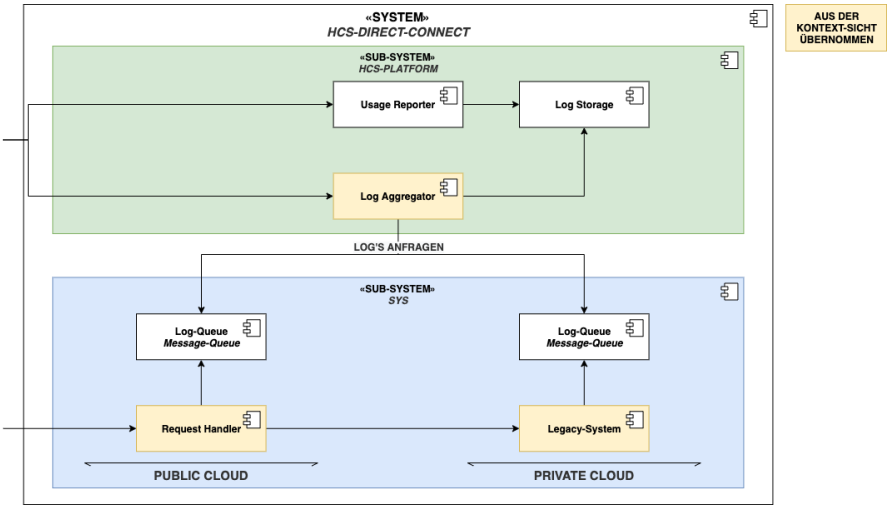


Abbildung 5.4: Bausteinsicht - Blackbox

Beobachtung der Hybrid-Cloud-Systeme

Nachdem die Hybrid-Cloud-Automatisierungsstrategie besprochen wurde, wird der Blick von der inneren-Komponente auf die äußere-Komponente des Projektes gerichtet (siehe Kontextansicht Abbildung 5.3). Die Beobachtung der Hybrid-Cloud wird über Schnittstellen in Private- und Public-Cloud durchgeführt. Beide Cloud-Umgebungen produzieren log-Einträge, die das Beobachtungssystem anfragt und auswertet. Teil der Lösungsstrategie ist die Beobachtung der Hybrid-Cloud über log-Einträge.

Bausteinsicht

In der Abbildung 5.4 ist die Bausteinsicht als Blackbox-Ansicht gegeben. Die gelben Komponenten sind direkt aus der Kontextansicht übernommen. Die weißen Komponenten sind noch nicht behandelt worden und werden nun nacheinander beschrieben.

- Request-Handler: Der ‘Request-Handler’ verarbeitet Authentifizierungsanfragen und greift dabei auf das ‘Legacy-System’ zurück.
- Legacy-System: Das ‘Legacy-System’ verwaltet User-Informationen und bietet eine Schnittstelle auf diese zuzugreifen.
- Log-Aggregator: Der ‘Log-Aggregator’ aggregiert logs, die in ‘Log-Queues’ abgelegt sind und speichert diese logs in der ‘Log-Storage’ Komponente.

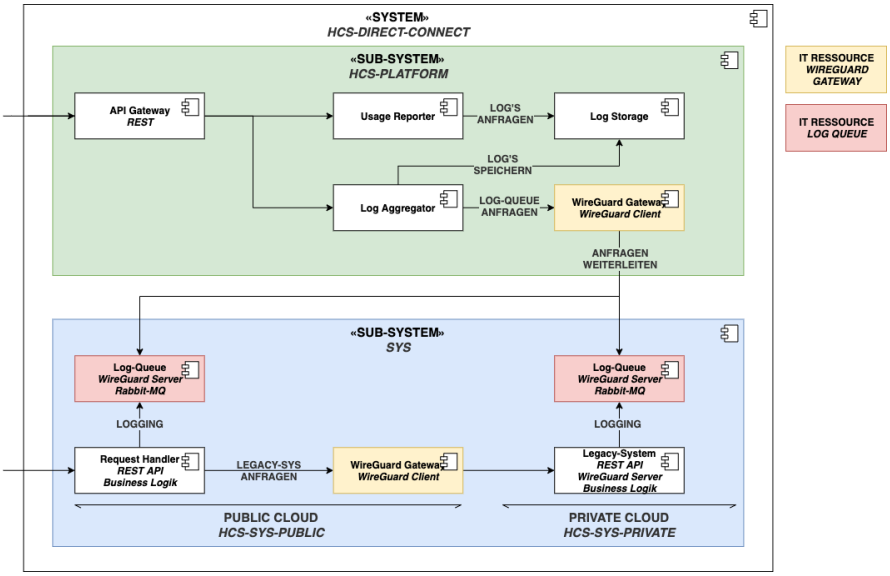


Abbildung 5.5: Bausteinsicht - Whitebox

- Usage-Reporter: Diese Komponente erzeugt aus den Daten im 'Log-Store' einen Report[6].
- Log-Storage: Diese Komponente speichert log-Einträge und bietet Schnittstellen zur Interaktion.
- Log-Queue: Diese Komponente puffert Log-Einträge, die vom 'Log-Aggregator' abgerufen werden. Die Komponente wird zweimal verwendet[7].

In der Abbildung 5.5 ist die Bausteinsicht als Whitebox-Ansicht gegeben. Die neuen Komponenten werden nacheinander durchgegangen. Wiederverwendete Komponenten sind farblich hervorgehoben.

- HCS-Platform: Die beiden Komponenten 'Usage-Reporter' und 'Log-Aggregator' werden über ein 'API-Gateway' mit einer öffentlichen Schnittstelle angesprochen[8]. Der 'Log-Aggregator' kommuniziert über das 'WireGuard-Gateway' mit den beiden

[6] Es ist sinnvoll, an dieser Stelle eine weitere Komponente einzuführen und die Aufgabe vom 'Log-Aggregator' zu trennen, der sonst auch diese Aufgabe übernehmen könnte, da so eine klarere Aufgabenteilung vorgenommen wird (Architektur Prinzip: *Seperation of Concerns*[64]).

[7] Eine Queue erhöht die Resilienz des Systems und ermöglicht es, Log-Einträge in Intervallen anzufordern. Da die Komponente doppelt verwendet wird, muss der 'Log-Aggregator' nur eine Schnittstellendefinition ansprechen können.

[8] Das 'API-Gateway' ist eine Fassade für beide Komponenten, welches ein weiteres Architektur Designpattern ist[64]

'Log-Queues', die mit dem 'WireGuard-Server' angebunden sind. Das 'WireGuard-Gateway' steuert Traffic zur HCS-Sys-Komponente.

- HCS-Sys-Public-Cloud: Der 'Request-Handler enthält eine REST-API und die damit verknüpfte Business-Logik. Wird das 'Legacy-System' angefragt, wird der Traffic über das 'WireGuard-Gateway' geroutet. Log-Einträge werden an die RabbitMQ gesendet.
- HCS-Sys-Private-Cloud: Das 'Legacy-System' ist ähnlich aufgebaut wie der 'Request-Handler' und verfügt ebenfalls über eine REST-API und die dazugehörige Business-Logik und speichert ebenfalls Log-Einträge in der 'Log-Queue'. Das Legacy-System enthält einen 'WireGuard-Server', der mit dem 'WireGuard-Client' im 'WireGuard-Gateway' verbunden ist.

Der Zusammenhang der verwendeten Komponenten wird im nächsten Abschnitt als Laufzeitansicht aufgegriffen.

Laufzeitsicht

In der Abbildung 5.6 ist die Laufzeitsicht gegeben, in der das in den Anforderungen angesprochene Test-Szenario ausgeführt wird. Das Szenario ist in drei Phasen unterteilt.

1. In der ersten Phase des Szenarios werden Anfragen an den 'Request-Handler' geschickt. Die Funktionen des 'Request-Handlers' werden im Schema Sign-up, Sign-in, Session wiederholt aufgerufen.
   Der 'Request-Handler' greift dabei auf das 'Legacy-System' zurück. Der 'Request-Handler' wie auch das 'Legacy-System' schreibt log-Einträge währenddessen in die 'Log-Queue'.
2. In der zweiten Phase des Szenarios werden die log-Einträge aus den beiden 'Log-Queues' über den 'Log-Aggregator' aggregiert abgespeichert.
3. In der dritten und letzten Phase werden log-Einträge aus dem 'Log-Storage' ausgelesen und als Datei ausgegeben. Das Szenario ist abgeschlossen und die Datei kann manuell ausgewertet werden.

Das Szenario soll, nachdem es gestartet wurde, voll automatisiert ablaufen. Im nächsten Abschnitt wird beschrieben, wie die Komponenten in den Cloud-Umgebungen angeordnet werden.

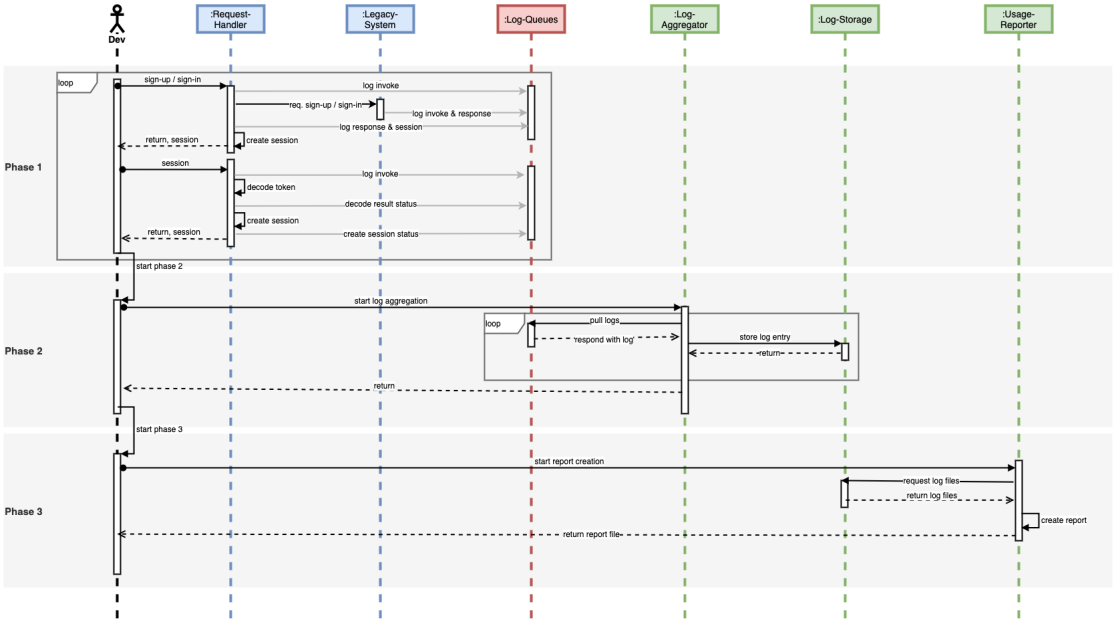


Abbildung 5.6: Laufzeitsicht

Verteilungssicht

Die zuvor gezeigten Architektursichten ziehen noch nicht in Betracht, dass es sich um ein verteiltes System handelt, das entwickelt wird. In der Abbildung 5.7 ist die Verteilungssicht des gesamten-Systems gegeben. Die eingeführten Komponenten der Bausteinansicht sind in Verteilungssicht wieder zu finden. Komponenten, die in der Public-Cloud verwaltet werden, sind mit AWS-Services Referenzen dargestellt. In der folgenden Liste werden die in der Public Cloud verwalteten Services, einer nach dem anderen, knapp besprochen[9].

- HCS-Sys-Platform:
  - Der AWS API-Gateway Service ist ein Gateway, welches in dieser Konfiguration HTTP-Anfragen verarbeitet. Anfragen werden an die beiden Lambda-Funktionen ‘Log-Aggregator’ und ‘Reporter’ weitergeleitet, welche HTTP-Anfragen dann bearbeiten.
  - AWS Lambda ist ein Serverless-Service, der Code ausführt (siehe Abschnitt 2.8). AWS Lambda wird in diesem Aufbau zum Verarbeiten von HTTP-Anfragen verwendet.

[9] AWS-Services sind sehr vielschichtig und werden hier nicht in vollem Umfang vorgestellt.

  - Amazon Simple Storage Service (Amazon S3) ist ein Objektspeicher-Service, der gespeicherte Datein in S3-Buckets ordnet.
  - AWS Elastic Compute Cloud (EC2) verwaltet Virtuelle Maschinen. Es können Amazon Maschinen Image (AMI) zum Starten einer EC2-Instanz ausgewählt werden.

- HCS-Sys-Public-Cloud:
  - Elastic Load Balancing (ELB) verteilt den eingehenden Traffic automatisch auf mehrere Ziele, wie EC2-Instanzen, Container und IP-Adressen. Der Application Load Balancer (ALB) arbeitet auf der siebten-Ebene des OSI Models und nutzt dafür HTTP.
  - Elastic Container Service (ECS) registriert einen Service von Fargate und stellt sicher, dass die gewünschte Konfiguration, wie Anzahl von laufenden Containern, zutrifft.
  - Fargate ist eine Serverless Container-Runtime, die eine Service Konfiguration ausführen kann. In der Service-Konfiguration werden Angaben wie Memory, CPU, Container-Image angeben.
  - Elastic Container Registry (ECR) bietet ein Container-Repository, welches genutzt werden kann, um Container-Images abzulegen.

Neben den ausgewählten Services liegt ein weiteres Augenmerk auf der Netzwerktopologie. Das Netzwerk wird in beiden Public-Cloud-Umgebungen gleichermaßen mit drei Subnetzen eingerichtet; der Datenverkehr sollte in dieser Konfiguration leichter zu kontrollieren sein.

- Subnetz Ingress Processing: In diesem Subnetz haben Services Zugriff auf das Internet und können öffentliche API's anbieten.
- Subnetz Processing: In diesem Subnetz haben Services keinen Zugang zum Internet und können nur vom Ingress-Processing- Subnetz aus erreicht werden. Services in diesem Subnetz können angeschlossene Services in anderen Ingress-Processing-Subnetzen über das Ingress-Gateway-Subnetz erreichen.

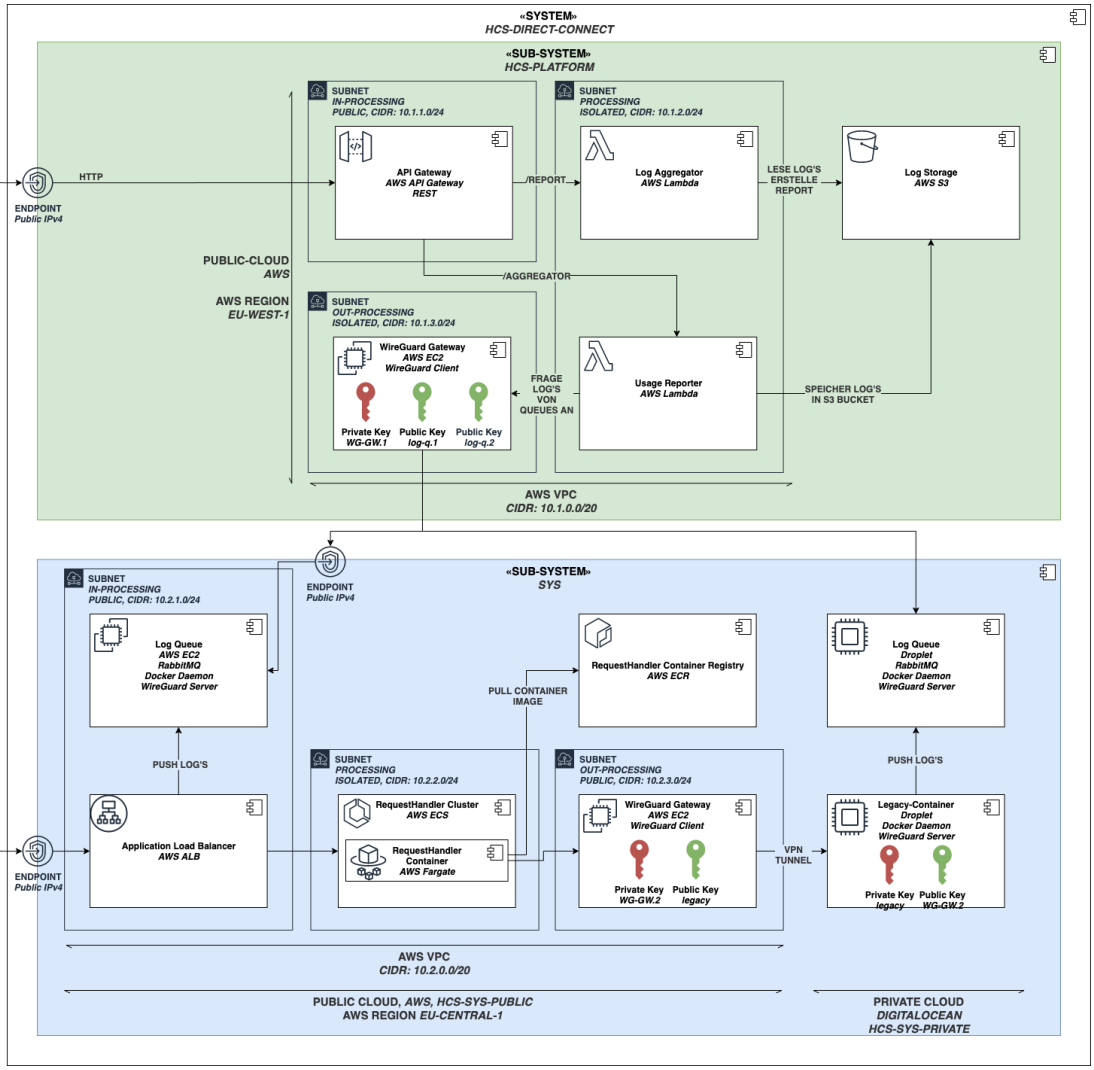


Abbildung 5.7: Verteilungssicht

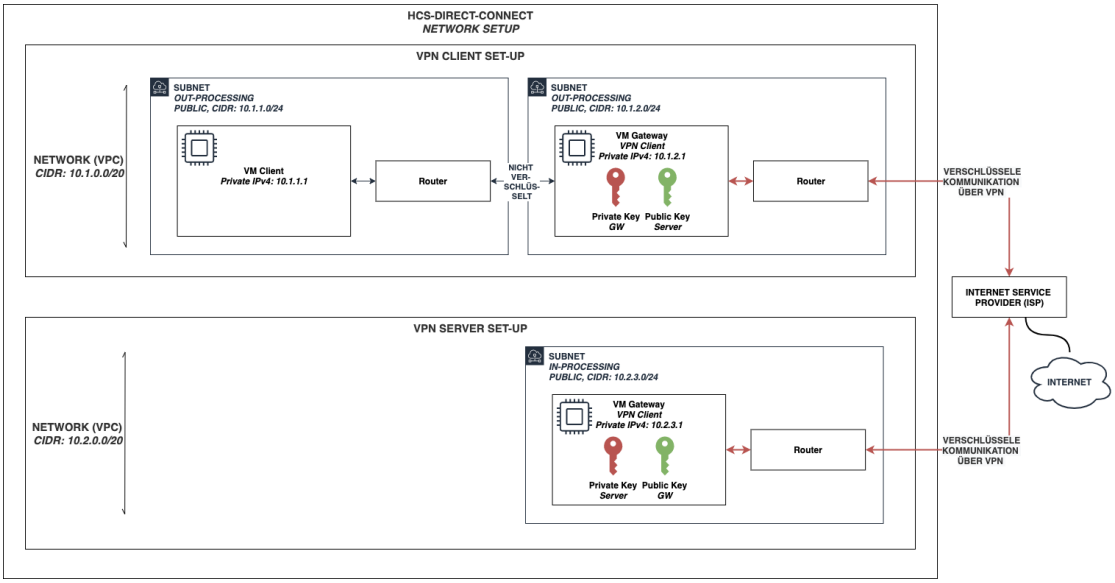


Abbildung 5.8: Verteilungssicht - Netzwerk-Topologie

- Subnetz Egress Gateway: Dieses Subnetz leitet Traffic zu Ingress-Processing- Subnetzen außerhalb einer anderen Cloud-Umgebung weiter[10].

In der Abbildung 5.8 ist die Netzwerktopologie abstrahiert, um die Kommunikation zwischen Public- und Private-Cloud zu veranschaulichen. Im Diagramm finden sich die zuvor besprochenen Subnets wieder, ebenso wie VM-Instanzen, die über einen VPN-Tunnel kommunizieren. Die VM-Gateway-Instanz fungiert als VPN-Client, der einen VPN-Tunnel mit der VM-Server-Instanz, dem VPN-Server, aufbaut. Nach dem Aufbau des VPN-Tunnels ist das Gegenstück des VPN-Tunnels über die in der Schnittstellenkonfiguration hinterlegte private IP-Adresse zu erreichen.

Architektonische Entscheidungen

In diesem Abschnitt werden Optionen in der Architektur besprochen, die im Zuge der Planung vorgestellt wurden.

- CloudWach & log-Einträge: In der AWS Public-Cloud gibt es einen AWS Service, AWS CloudWatch, der nativ log-Einträge sammelt und verarbeitet. Dieser Service

[10] Das Gateway, welches in diesem Subnet liegt, muss zunächst einen VPN-Tunnel aufbauen, um Services in einem Ingress-Processing-Subnetz erreichen zu können.

wird in der Planung nicht verwendet.
Statt der log-Queue hätte CloudWatch die log-Einträge auch regions- & konto-übergreifend direkt in AWS-S3 über AWS-FireHose speichern können (siehe Abbildung B.1). Diese Verbindung würde es ermöglichen, dass die log-Einträge nahezu in Echtzeit in S3 erscheinen.
Dieses Setup wurde nicht gewählt, da es von den oben genannten AWS-Services abhängt und die log-Einträge nicht in nahezu Echtzeit verarbeitet werden müssen. Für das aktuelle System muss die 'Log-Queue' einmal entwickelt werden und kann dann in zwei Umgebungen verwendet werden.

- Container-Images & andere Artefakte: Container-Images werden aktuell in ECR (Public-Cloud) und Docker-Hub (Private-Cloud) gespeichert. In einer hypothetischen Komponente 'HCS-Sys-Repository' könnten Artefakte verwaltet werden. Diese neue Komponente hat keine Abhängigkeiten zu den bisherigen Komponenten (HCS-Sys-Platform, HCS-Sys-Public-Cloud, HCS-Sys-Private-Cloud). In der Komponente 'HCS-Sys-Repository' könnten Artefakte wie Maschinen-Images, Container-Images und Code-Repositories verwaltet werden.

- Skalierung: Verwendete AWS-Services wie API-Gateway, AWS Lambda und AWS Fargate skalieren nativ, werden aber durch die Flaschenhälse wie die 'Legacy-Komponente' und das EC2-Gateway ausgebremst. Das System skaliert daher nicht durchgehend und wird bei steigenden Anfragen Probleme verursachen.

- VPN-Tunnel / Kommunikation: Der 'Request-Handler' ist eng mit der 'Legacy-Komponente' verbunden; wenn die Kommunikation zwischen den Komponenten gestört ist, ist das System nicht resilient, fällt aus und kann nicht mehr auf Anfragen reagieren. Die Ausfallsicherheit des Systems könnte erhöht werden, wenn VPN-Tunnel automatisch aufgebaut werden, sobald diese gestört sind.

Es könnten noch weitere Risiken und architektonische Feinheiten besprochen werden, die allerdings an Relevanz verlieren. Die wichtigsten Punkte wurden hiermit behandelt.

5.2.2 Design 2: Health-Checks

Dieser Abschnitt knüpft an das erste Design an und überarbeitet Teile davon. Die Komplexität der entwickelten Software-Architektur im ersten Design ist für die schmal konzipierte Anwendung zu hoch. In der Abbildung 5.9 ist die im ersten Design erstelle

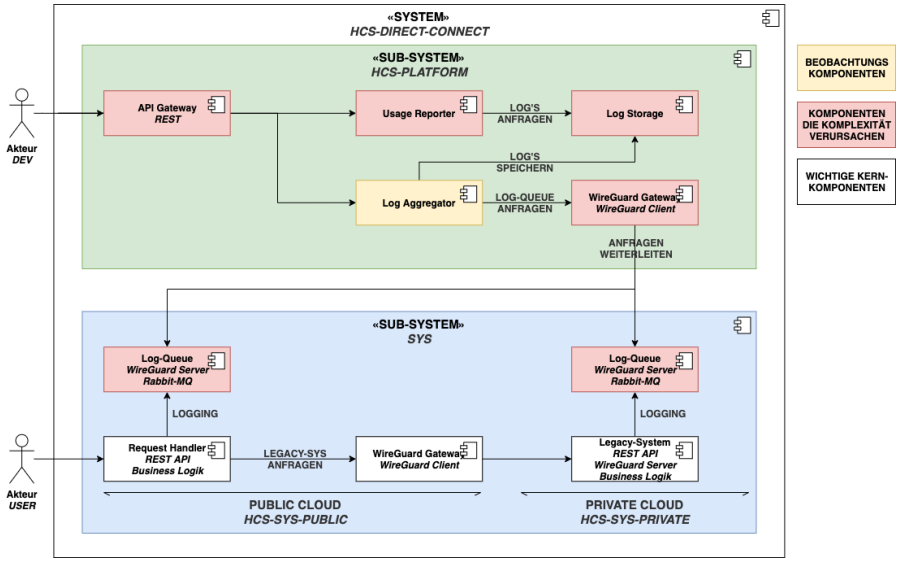


Abbildung 5.9: Komplexität der Bausteinsicht

Bausteinsicht zu sehen. Die weißen Bausteine sind ein fester Bestandteil des Hybrid-Cloud-Systems. Der gelbe Baustein ist für die Beobachtung des Hybrid-Cloud-Systems zuständig. Die roten Bausteine ermöglichen und vereinfachen die Arbeit des gelben Bausteins und sollten weggekürzt werden.

Der Grund für die hohe Komplexität der Software-Architektur des ersten Entwurfs ist die Annahme, dass über Log-Einträge sichergestellt werden muss, dass das entwickelte System als Hybrid-Cloud funktioniert. Die Beobachtung ist über Log-Einträge möglich, es besteht aber auch die Möglichkeit, über Health-Checks eine intakte VPN-Verbindung zu testen. Dieses Design greift diese Idee auf und modelliert die Architekturansichten entsprechend um.

Architekturansichten

In der Abbildung 5.10 ist die überarbeite Version der Bausteinsicht gezeigt. Die zuvor rot markierten Komponenten in der Abbildung 5.9 wurden in der zweiten Version der Bausteinsicht größtenteils entfernt. Die Komponenten ‘Request-Handler‘, ‘WireGuard Gateway‘, ‘Legacy-System‘ hängen wie zuvor gleichermaßen zusammen. Die ‘Health-Checker-Lambda‘ ersetzt den ‘Log-Aggregator‘ und wird über ein API-Gateway ausgeführt. In der Abbildung 5.11 ist die zweite Version der Laufzeitsicht zu sehen.

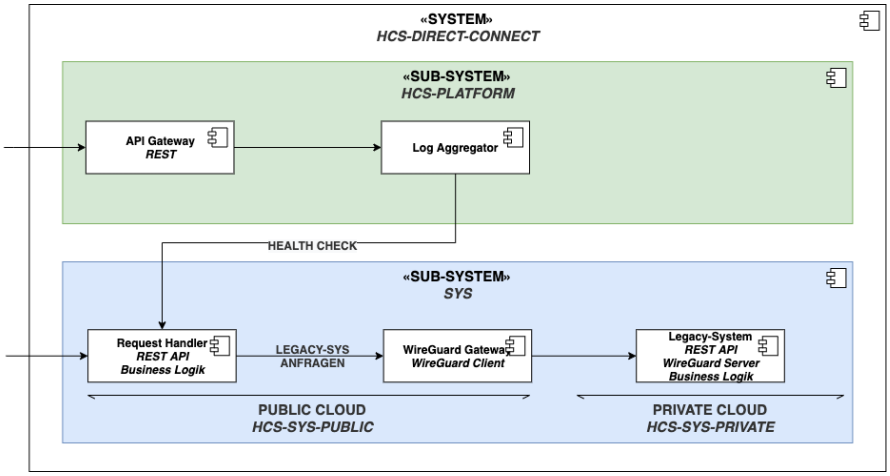


Abbildung 5.10: Bausteinsicht - Version 2

1. Der Akteur ruft über den Endpunkt des 'API-Gateways' den 'Health-Checker' auf.
2. Der 'Health-Checker' ruft einen Health-Check Endpunkt des 'Request-Handlers' auf
3. Der 'Request-Handler' leitet diese Anfrage an das 'Legacy-System' weiter
4. Antwortet das 'Legacy-System' so wird das in der Antwort vermerkt
5. Antwortet der 'Request-Handler', so wird das in der Antwort vermerkt
6. Antwort wird an den Akteur zurück gegeben.

Diese Laufzeitsicht der Abbildung 5.11 ist deutlich simpler im Ablauf und vermerkt gleichermaßen, ob die Kommunikation zwischen dem 'Request-Handler' und dem 'Legacy-System' funktioniert. Die Health-Checks können als Sign-in / Sign-up interpretiert werden, was in den Anforderungen formuliert wurde, um das System zu testen.

In der Abbildung 5.12 ist die zweite Version der Verteilungssicht gegeben. Die Verteilungssicht nutzt die zuvor eingeführten AWS-Services[11]. In der 'HCS-Sys-Platform' ist kein Subnet aufgezeigt, implizit wird das Default-VPC im AWS Account dadurch referenziert.

[11] Auf die Container Registry (AWS ECR) wird verzichtet, da der Container direkt zu AWS Fargate gepused wird und dadurch keine Zwischenspeicherung nötig ist.

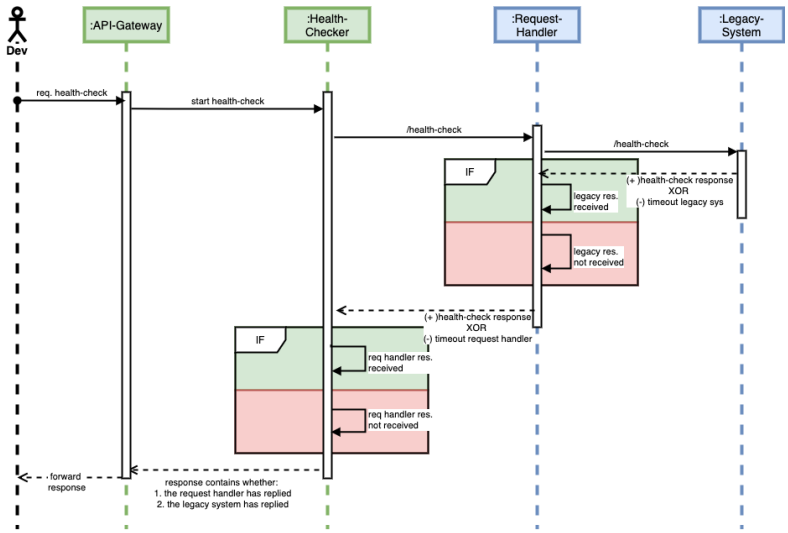


Abbildung 5.11: Laufzeitsicht - Version 2

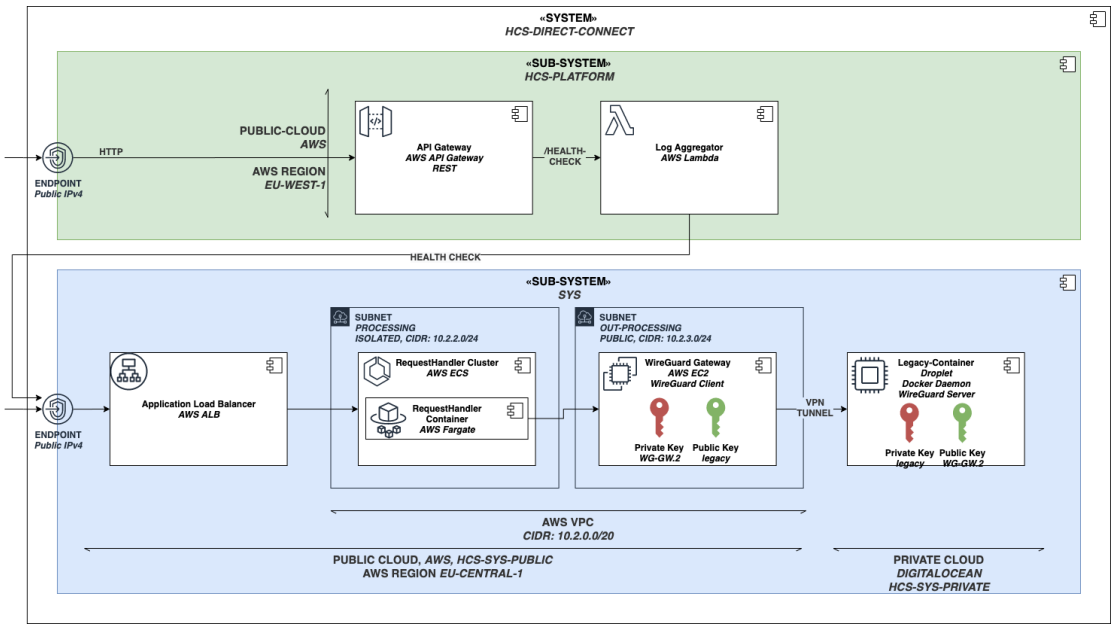


Abbildung 5.12: Verteilungssicht - Version 2

IaC-Deployment

Im ersten Entwurf wurden die IaC-Tools nicht in dem Umfang berücksichtigt, wie es für ein IaC-Tool-basiertes Projekt notwendig ist. Im zweiten Entwurf wird nun an dieser Stelle nachgebessert. In der Hybrid-Cloud-Automatisierungsstrategie wurden Pulumi, Vagrant und Ansible genannt, um das System zu automatisieren.

Um den Einsatz der Tools genauer herauszuarbeiten, ist es nötig die einzelnen Tätigkeiten des Deployments zu identifizieren. Die einzelnen Tätigkeiten sind in nachfolgender Liste zunächst unsortiert gegeben.

1. Push Container Image Request-Handler-Container zu Dockerhub (Container Registry)
2. Push Container Image Legacy-Container zu Dockerhub (Container Registry)
3. Verwalte AWS Services von hcs-sys-platform
4. Verwalte AWS Services von hcs-sys-public-cloud
5. Erstelle Maschinen Image für hcs-sys-private-cloud
6. Installiere WireGuard Gateway EC2
7. Installiere Legacy-Komponente
8. Erstelle Private-Key in WireGuard Gateway EC2
9. Erstelle Private-Key in Legacy-Komponente
10. Erstelle Public-Key - Key-exchange von WireGuard Gateway EC2
11. Erstelle Public-Key - Key-exchange von Legacy-Komponente
12. Baue eine VPN-Tunnel-Verbindung zwischen WireGuard Gateway EC2 und der Legacy-Komponente auf
13. Erstelle Legacy-Server VM

Diese Tätigkeiten müssen in eine zeitliche Abfolge gebracht werden. Damit diese Abfolge zustande kommt, müssen Abhängigkeiten zwischen den einzelnen Tätigkeiten identifiziert werden. In der Abbildung 5.13 ist der Deployment-Prozess gezeigt.

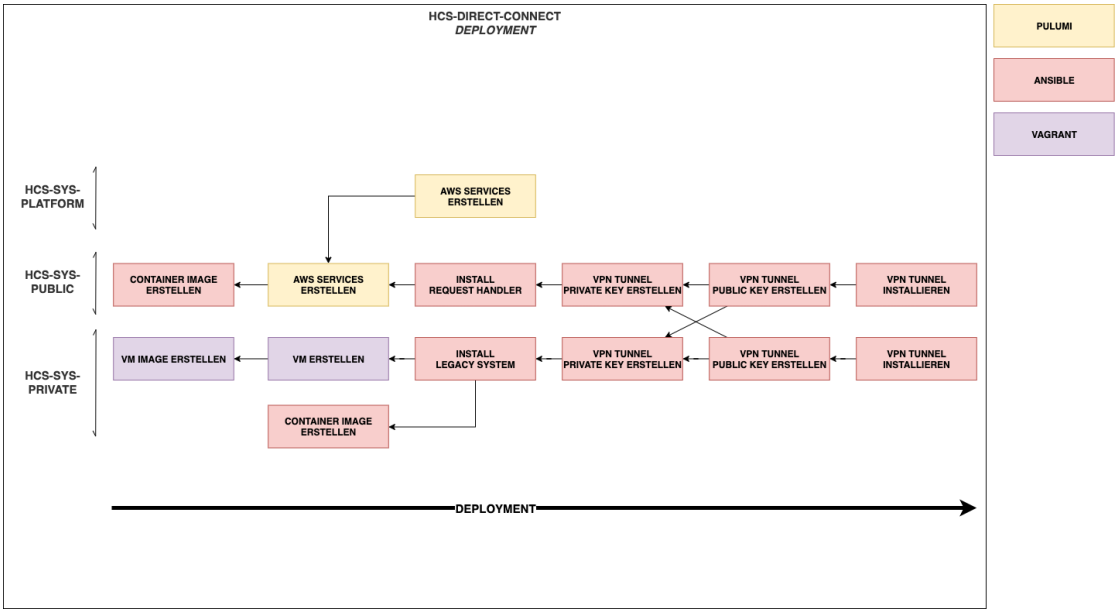


Abbildung 5.13: Projekt Deployment

5.3 Implementierung

In diesem Abschnitt wird beschrieben, wie das Projekt implementiert wurde. Im Abschnitt B.3 ist das deployment gezeigt. Die Code-Repository-Struktur ist in einem Versions-Control-System angelegt und wird über GIT verwaltet. Die Implementierung verlief in drei Phasen[12]. Damit die Phasen im Code nachvollzogen werden können, wird jede Phase im Code-Repository über einen Branch geforkt, der den Status der Phase dokumentiert.

1. Lokale Entwicklung: Container erstellen und lokal testen.
2. Cloud-Deployment: Cloud-Umgebungen getrennt aufsetzen und automatisiert verwalten.
3. Hybrid-Cloud-Deployment: VPN-Tunnel aufbauen.

In der Abschnitt B.4 ist die Projekt-Struktur als Baum dargestellt. Die Äste ‘hcs-sys-private‘, ‘hcs-sys-platform‘ und ‘hcs-sys-public‘ beinhalten Komponenten-spezifische- Konfigurationen. Der Zweig ‘hcs-sys-private‘ enthält Komponenten, die wiederverwendet wer-

[12]Die Phasen sind vom Umfang her nicht ausgeglichen. Die erste Phase hat einen geringen Umfang, die Phasen zwei und drei einen höheren. Nachdem die drei Phasen abgeschlossen sind, wird das System getestet.

den können. Der Kindknoten ‘deploy-project.sh‘ der Wurzel stellt eine Shell-Datei da und soll die gesamte Konfiguration am Ende des Projekts zentral anstoßen können. In der README.md-Datei des Projektes wird kurz und knapp das Projekt eingeführt. README.md-Dateien finden sich auch bei den meisten Ästen des Projektes, um weitere Details zu dokumentieren.

5.3.1 Phase 1: Lokale Entwicklung

In diesem Abschnitt wird beschrieben, wie die beiden Container ‘request-handler‘ und ‘legacy-container‘ entwickelt wurden. Aus der Verteilungssicht zu schließen, werden Docker-Container verwendet, die eine REST-API anbieten. Lokal wurden beide Container im selben Netzwerk getestet.

Sind die Container in der Hybrid-Cloud-Umgebung, so spricht der ‘request-handler-container’ die Private-IP des ‘legacy-containers’ an. Die Private-IP des ‘legacy-containers’ ist zunächst unbekannt und muss beim ‘request-handler-container’ dynamisch gesetzt werden, was über einen Endpunkt ‘/set-config‘ in der REST-API möglich gemacht wird. Neben dem ‘/set-config‘ Endpunkt werden die Endpunkte ‘/health-check‘ erstellt, die sich aus der Laufzeitsicht ergeben. Der ‘/health-check‘ Endpunkt des ‘request-handler-containers’ ruft die korrespondierenden Endpunkte des ‘legacy-containers’ auf.

Damit die Container vereinfacht getestet werden können, ist zusätzlich eine interaktive Swagger Dokumentation unter ‘/docs‘ verfügbar. Der Root-Endpunkt ‘/‘ beider container gibt generelle Konfigurations-Informationen aus, die als Container-Health-Check dienen und verifizieren, dass der Container ansprechbar ist[13].

5.3.2 Phase 2: Cloud-Deployment

In dieser Phase werden die beiden Cloud-Umgebungen aufgesetzt. Zunächst werden die beiden Public-Cloud-Umgebung ‘hcs-sys-platform‘ und ‘hcs-sys-public‘ erstellt und danach die Private-Cloud-Umgebung ‘hcs-sys-private‘.

[13] Ein Container-Health-Check Endpunkt (‘/‘) kann bei dem Fargate-Service angeben werden und startet den container erneut, sobald der Container-health-check keine Antwort zurück gibt.

Entwicklung der HCS-Platform

Die HCS-Sys-Platform wird in der AWS Public-Cloud verwaltet. Die eingesetzten AWS-Services werden mit dem IaC-Tool Pulumi definiert. Im Anhang findet sich der Pulumi-Stack, der erstellt wurde (siehe Listing B.4).

Dynamische Konfigurationsvariablen (wie Hostname, Pfad, Port), werden über eine Pulumi-Konfigurationsdatei übergeben. Durch dynamisch gestezte Variablen ist es möglich, das Hybrid-Cloud-System automatisiert auszurollen. Es folgen zwei Code-Schnipsel, welche diese Dynamische Übergabe von Variablen zeigen.

```
echo "Set pulumi configuration..."
pulumi config set --path 'data.reqHandlerHostname' '10.0.0.1'
pulumi config set --path 'data.reqHandlerPath' '/health-check-connection'
pulumi config set --path 'data.reqHandlerPort' '8000'
```

Listing 5.1: Skript setzt Pulumi Konfigurationsvariablen.

```
// Get pulumi configuration to enable dynamic deployments
let config = new pulumi.Config();

// Structured configuration input
interface Data {
  reqHandlerHostname: string;
  reqHandlerPath: string;
  reqHandlerPort: number;
}

// Create config object
let configData = config.requireObject<Data>("data");

let reqHandlerHostname = configData.reqHandlerHostname;
let reqHandlerPath = configData.reqHandlerPath;
let reqHandlerPort = configData.reqHandlerPort;
```

Listing 5.2: Pulumi Code fragt Konfigurationsvariablen an.

Entwicklung des HCS-SYS-Public-Cloud-Systems

Die HCS-Sys-Public beinhaltet AWS-Services, die verwaltet werden müssen. Die AWS Services werden mit dem IaC-Tool Pulumi definiert. Im Anhang findet sich der Pulumi-Stack, der erstellt wird und die HCS-Sys-Public AWS-Services abdeckt (siehe Listing B.3).

Bei der Implementierung hilft Pulumi Crosswalk, best-practices zu abstrahieren. Die Pulumi Crosswalk-Bibliothek wird verwendet, um einen Anwendungs-Load-Balancer und einen Listener zusammen zu bauen, was normalerweise in zwei Schritten mit zwei Klassen abgebildet wird (siehe Listing 5.3). Dieses AWS-Service-Setup wird häufig verwendet, muss aber ohne die Verwendung der Crosswalk-Bibliothek umständlich definiert werden.

```
// Create an application load balancer (ALB) & listener
//  that has a publicly accessible endpoint
//  needed to gain access to the ecs cluster
const albListenerReqHandler = new awsx.lb.ApplicationListener(
  "${clusterReqHandlerName}-alb",
  { port: albClusterReqHandlerPort }
);
```

Listing 5.3: Pulumi Crosswalk Bibliothek wird eingesetzt.

Entwicklung der HCS-SYS-Private-Cloud-Systems

Die HCS-Sys-Private beinhaltet eine VM, die verwaltet werden muss. Diese wird wie im Design-Abschnitt beschrieben, mit Vagrant spezifiziert und verwaltet. Im Anhang findet sich in gekürzter Fassung die Konsolenausgaben, die bei der Ausführung der Vagrantfile erzeugt werden (siehe Listing B.2). Private-Cloud-Anbieter wird an dieser Stelle von einem Public-Cloud-Anbieter simuliert. Zum Simulieren des Private-Cloud-Anbieters wurde DigitalOcean ausgewählt.
Im Anhang findet sich die Vagrantfile, die zum Verwalten der Private-Cloud VM genutzt wird (siehe Listing B.1). Im Vergleich zum AWS-Setup ist der DigitalOcean-Server nicht in einer 'speziellen' Netzwerktopologie untergebracht, die konfiguriert wird, sondern hat ohne Firewall-Konfigurationen, Zugang zum Internet. In der Vagrantfile werden auch VM Härtungsschritte, wie z. B. die Installation von Docker, beschrieben.

5.3.3 Phase 3: Hybrid-Cloud-Deployment

In dieser Phase wird beschrieben, wie die Public-Cloud und die Private-Cloud über einen VPN-Tunnel verbunden werden. Der VPN-Tunnel wird mit WireGuard aufgebaut. Die Kommunikation über den VPN-Tunnel ist bidirektional. Die Schritte, um den VPN-Tunnel aufzubauen, wurden bereits in einer IaC-Deployment-Ansicht gezeigt (siehe Abbildung 5.13). Der Prozess ist im Code-Repository in der deploy.sh dokumentiert.

In dem Code-Schnipsel Listing 5.4 ist die WireGuard Interface-Konfiguration gegeben. Dieses Interface wird in leicht abgewandelter Form in beiden Cloud-Umgebungen aktiviert. In Zeile 3 wird der Private-Key der Maschine referenziert, der zur Validierung von Verbindungen verwendet wird, die einen Public-Key enthalten.

```
1  # Server Konfigurationen
2  [Interface]
3  PrivateKey = <OWN_PRIVATE.KEY.VALUE>
4  Address = 10.50.0.1/24
5  ListenPort = 51280
6
7  # Konfigurationen fuer die Clients.
8  [Peer]
9  PublicKey = <TARGET_MACHINE_PUBLIC.KEY.VALUE>
10 AllowedIPs = 10.50.0.2/32
```

Listing 5.4: WireGuard Interface set-up

Die Fallstudie ist hiermit vorerst abgeschlossen. Ergebnisse werden in Kapitel 7 analysiert. Im nächsten Kapitel wird die Fallstudie kompakt ergänzt.

6 Fallstudie HCS-DIRECT-CONNECT-2 - eine Ergänzung

Dieses Kapitel ergänzt die HCS-DIRECT-CONNECT Fallstudie (siehe Kapitel 5). In dieser Anwendung sollen Untersuchungen aus der ersten Fallstudie auf weitere Public-Cloud-Anbieter und IaC-Tools vertieft und ausgedehnt werden. Die erste Fallstudie HCS-DIRECT-CONNECT wird im weiteren Verlauf dieses Kapitels HCS-1 und die Ergänzung wird HCS-2 genannt. In HCS-2 werden folgende vier Änderungen umgesetzt.

- Die Public-Cloud wird auf Google Cloud Platform (GCP) verwaltet.
  Die Umstellung von AWS auf GCP ermöglicht die Beobachtung, wie Pulumi in einer anderen Public-Cloud-Umgebung mit einer anderen Programmiersprache funktioniert.
- Die Private-Cloud Services werden genutzt (Public-Cloud-Anbieter Digital Ocean wird weiterhin als pseudo Private-Cloud verwendet).
  Services der Private-Cloud wurden in der ersten Fallstudie nicht betrachtet und bieten die Möglichkeit ein anderes IaC-Tool, wie Terraform, zu beleuchten.
- Die Automatisierung wird von Shell-Skripten auf Python-Skripte umgestellt.
  Durch die Automatisierung der IaC-Tool Prozesse über Python Skripte wird sich eine mächtigere Automatisierung erhofft.
- Der VPN-Tunnel wird mit einem Service in der Public-Cloud-Umgebung verwaltet.
  Diese Umstellung wird bei der Bewertung der ausgewählten HCA-Strategie helfen.

Mit dieser Ergänzung kann in der Analyse aus einer breiter gefächerten Applikation aussagekräftiger argumentiert werden. Da das Projekt weitgehend unverändert bleibt, wird in diesem Fall auf einen ausformulierten Projektentwicklungsprozess verzichtet. Dieses Kapitel gliedert sich in zwei Abschnitte, 1. 'Design' und 2. 'Implementierung'. Die Analyse der HCS-1 Fallstudie, einschließlich der HCS-2 Ergänzung, folgt im nächsten Kapitel.

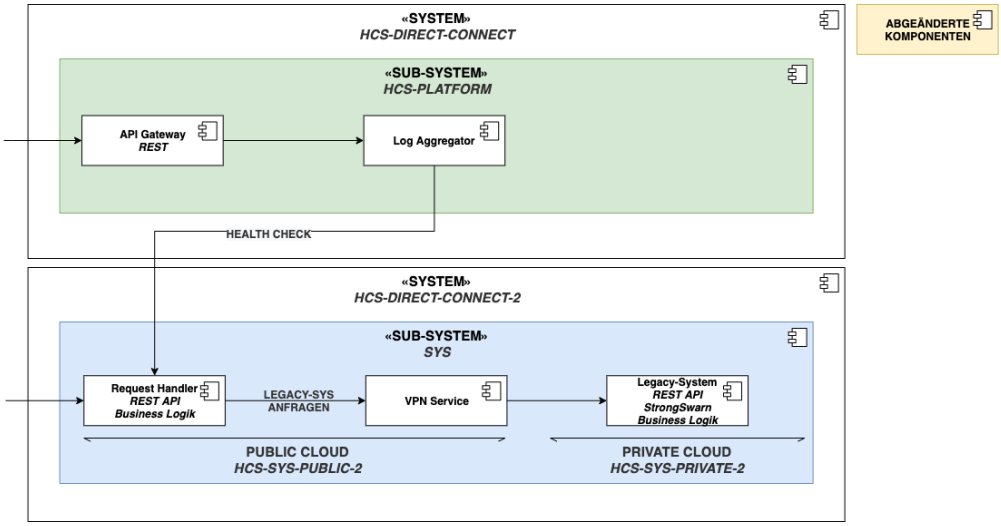


Abbildung 6.1: Bausteinsicht

6.1 Design

In diesem Abschnitt wird beschrieben, welche Änderungen im Design, das HCS-2 System gegenüber des HCS-1 Systems hat. Weil der Anwendungsfall, die Anforderungen sowie die Laufzeit des HCS-2 Systems, nach der Einleitung zufolge, unverändert bleibt, wird der Fokus auf die beiden Architekturansichten - Bausteinsicht und Verteilungssicht gelegt. Die angesprochene Automatisierung der IaC-Tools mittels Python Skripten wird im Anhang ausgeführt (siehe Anhang Abschnitt C.1). Das Deployment des Systems findet sich ebenfalls, über Konsolen-Ausgaben veranschaulicht, im Anhang wieder (siehe Abschnitt C.2).

In der Abbildung 6.1 ist die Bausteinsicht des HCS-2 Projektes zu sehen. Die Darstellung weicht nur leicht von der zuvor entwickelten Bausteinsicht der Fallstudie HCS-1 ab. Veränderte Bausteine sind im Diagramm gelb hervorgehoben.

- Es wird der 'GCP VPN Service', 'Cloud VPN', verwendet und nicht wie zuvor ein eigenes VPN Gateway verwaltet.
- Anstelle von WireGuard wird StrongSwarn verwendet, ein vergleichbares Tool, da es von GCP vorgeschlagen wird.

In der Abbildung 6.2 ist die Verteilungssicht des HCS-2 Projektes zu sehen. Die zuvor genutzten AWS-Services stehen auf GCP nicht zur Verfügung und müssen deshalb er-

setzt werden. Da mit dem VPN-Service von GCP gearbeitet wird, unterscheidet sich der Aufbau dahingegen zum zuvor entwickelten Aufbau in AWS, welches keinen AWS VPN Service nutzte. Im Folgenden werden die Services durchgegangen, die zur Entwicklung ausgewählt wurden.

1. **Firewall Rules**: Mit diesem Service wird konfiguriert, welche Ports und Protokolle zur Kommunikation zu einem GCP-Service hin oder von einem GCP-Service aus, geöffnet sind. Soll beispielsweise eine VM über SSH erreicht werden, so müssen folgende Parameter einer Firewall Rule hinterlegt sein: 'Action:allow, Traffic:ingress, Port:22, Protocol:TCP'.

2. **Cloud Routes**: Dieser Service konfiguriert eine Routeing-Tabelle, um so anwendungsspezifische Subnets aufbauen zu können. In dieser Anwendung wird z.B. eine Route zum Private-Cloud VPC-CIDR angegeben.

3. **Compute Engine**: Dieser Service wird genutzt, um VM's zu verwalten. Dieser Service ist vergleichbar zu AWS EC2.

4. **Cloud VPN**: Dieser Service konfiguriert ein VPN-Gateway und ein VPN-Tunnel. An einem VPN-Gateway können mehrere VPN-Tunnel registriert werden. Der VPN-Tunnel stellt die Verbindung zwischen den Cloud-Umgebungen her, benötigt dafür jedoch eine erstelltes VPN-Gateway auf Seiten der Private-Cloud.

Die Funktionsweise der Applikation hat sich nicht geändert. Im nächsten Abschnitt wird knapp besprochen, wie die Implementierung mit Pulumi gestaltet wurde.

6.2 Implementierung

In diesem Abschnitt wird ein Blick auf die abgeschlossene Implementierung der GCP-Migration geworfen. Im Zuge dieser wurde ein neuer Ordner 'HCS-2' im Code-Repository angelegt. In diesem Ordner findet sich der Code in den Projekten 'HCS-SYS-PUBLIC-2' und 'HCS-SYS-PRIVATE-2' wieder. Das Private-Cloud System wird mit Terraform definiert. Das Public-Cloud System nutzt weiterhin Pulumi, es wird jedoch die Programmiersprache von Typescript auf GO gewechselt. Im Anhang finden sich die erstellten IaC-Tool-Stack Bäume (siehe für GCP: Unterabschnitt C.3.1 & für DigitalOcean: Unterabschnitt C.3.2) Im Nachfolgenden ist ein Ausschnitt aus dem HCS-SYS-PUBLIC

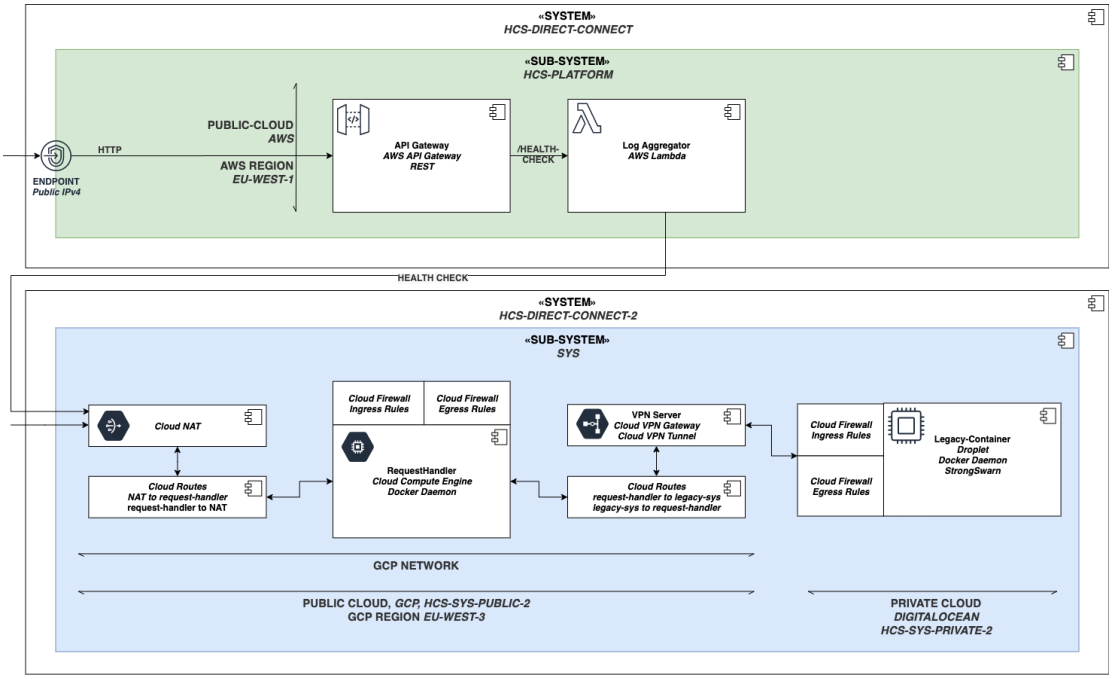


Abbildung 6.2: Verteilungssicht

Projekt gegeben, der verdeutlicht, das bei der Erstellung von Infrastruktur die logischen Bausteine von Programmiersprachen äußerst hilfreich sind.

```
// NETWORK FIREWALL
for _, e := range []FirewallConfig{
  // Allow inbound VPN UDP, TCP and ICMP Traffic
  {
    name: "ig-vpn",
    firewallAllowAry: compute.FirewallAllowArray{
      compute.FirewallAllowArgs{Protocol: pulumi.String(_TCP)},
      compute.FirewallAllowArgs{Protocol: pulumi.String(_UDP)},
      compute.FirewallAllowArgs{Protocol: pulumi.String(_ICMP)},
    },
    direction:   _DIRECTION_INGRESS,
    description: "Allow inbound TCP, UDP and ICMP traffic from any source",
  }
} {
  _, err = compute.NewFirewall(ctx, createName(e.name), &compute.FirewallArgs
    {
    Project:           pulumi.String(PROJECT_NAME_GCP),
    Network:           hcsVpc.Name,
    Description:       pulumi.String(e.description),
```

```
19      Allows:            e.firewallAllowAry,
20      DestinationRanges: e.destinationRanges,
21      SourceRanges:      e.sourceRanges,
22      Direction:         pulumi.String(e.direction),
23    })
24    if err != nil {
25      return err
26    }
27  }
```

Listing 6.1: Ausschnitt aus der 'main.go Datei des Projektes HCS-SYS-PUBLIC-2, das zeigt, mit Loops, Konstanten und definieren Interfaces Infrastruktur vereinfacht erstellt werden kann

In dem Listing 6.1 ist gezeigt, wie eine Firewall rule innerhalb des HCS-SYS-PUBLIC-2 Projektes erstellt wird. Es ist zu erkennen, dass in einem Loop, 'Firewall Rules' erstellt werden. Die Firewall Rules werden über eine 'FirewallConfig' definiert (siehe Zeile 2)[1]. Alle angegebenen FirewallConfig's erstellen neue Firewall's in GCP (siehe Zeile 14).

[1]Das Struct 'FirewallConfig' ist in diesem Code Ausschnitt nicht definiert, jedoch sollen sich die Parameter aus der Implementierung erahnen (siehe Zeile 4-13).

7 Evaluierung der Fallstudie

In diesem Abschnitt wird die Fallstudie HCS-SYS-DIRECT-CONNECT, welche in den vorherigen beiden Kapiteln entwickelt wurde, analysiert. Zunächst wird evaluiert, ob die Ziele, die für die Fallstudie definiert wurden, erreicht worden sind. Zum Schluss werden Erweiterungen und Verbesserungsvorschläge aufgelistet. Im Anhang finden sich zwei Abschnitte, die Abweichungen zwischen Design und Implementierung identifizieren. Siehe Abschnitt D.1 zu Unterschieden in der Projektstruktur und Abschnitt D.2 zu Unterschieden im AWS VPC set-up.

7.1 Analyse und Bewertung der Fallstudie

Unter den Anforderungen der Fallstudie wurden die Ziele der Fallstudie definiert, die nachfolgend nacheinander bewertet werden (siehe Abschnitt 5.1.1). Beide Systeme, HCS-1 (HCS-DIRECT-CONNECT) und HCS-2 (HCS-DIRECT-CONNECT-2), werden nun gemeinsam evaluiert.

7.1.1 Automatisierung der Infrastruktur mit IaC-Tools

In diesem Abschnitt wird der Entwicklungsprozess mit IaC-Tools und die Verwaltung von Infrastruktur mit IaC-Tools analysiert.

Im Entwicklungsprozess wurden die IaC-Tools Pulumi, Terraform, Ansible und Vagrant genutzt. IaC-Tools wurden daher in verschiedenen Ausprägungen im Entwicklungsprozess genutzt. Alle verwendeten IaC-Tools haben dazu beigetragen, den Anwendungsfall abzubilden. Die IaC-Tools werden nun in Stichpunkten durchgegangen.

- **Pulumi**: Die verwendeten Pulumi Bibliotheken sind gut dokumentiert und bieten meist hilfreiche Beispiele. Pulumi ähnelt in der Dokumentation und in den Beispielen stark Terraform. Beide Dokumentationen sind nahezu identisch. Wegen dieser Ähnlichkeit können auch Beispiele von Terraform herangezogen werden, die dann leicht auf Pulumi Bibliotheken übertragt werden können. Pulumi ist meist in der Anwendung[1].

- **Terraform**: Terraform ist gut gut dokumentiert. Die Dokumentationen sind Anbieter abhängig und daher variiert die Qualität etwas[2]. Da Terraform mit HCL arbeitet, kann nicht mit bekannten höheren Programmiersprachen gearbeitet werden, was bei kleinen Projekten, wie dieses, allerdings keine Rolle spielt. In HCL sind keine logischen Bausteine verfügbar, die allerdings auch nicht notwendig waren. Auch wenn HCL neue Konventionen einführt, die Funktionsweise gut zu verstehen.

- **Ansible**: Die Dokumentation in Ansible ist etwas undurchsichtiger aufgebaut. Die Abbildung des Deployment-Prozesses in der Playbook YAML-Datei ist gut zu verstehen. In Ansible gibt es zusätzlich zu den Terminalbefehlen einer Bibliothek oft Integrationen, die verwendet werden können, aber für diese Anwendung nicht relevant waren. In Ansible werden die Rechner in einer HOST file angegeben, das im Hybrid-Cloud System gepflegt werden muss.

- **Vagrant**: Die Dokumentation in Vagrant ist wie bei Terraform Anbieter-abhängig. Die Vagrant CLI ist für Automatisierungen nicht unbedingt geeignet[3], Terraform ist meist die bessere Wahl und wird in Hybrid-Cloud Systemen wohl nicht gebraucht[4].

Mit den IaC-Tools Pulumi, Terraform und Vagrant wurden Services in der Cloud erstellt. Die erstellten IT-Infrastrukturen enthielten letztlich oft mehr Services als geplant[5]. Oft werden IT-Ressourcen im Hintergrund automatisch erstellt. Dies ist meist kein Problem,

[1] In HCS-2 wurde, die Go Bibliothek von Pulumi verwendet. In dieser Bibliothek werden mit Wrappern gearbeitet, die den Status der assoziierten IT-Ressource nachverfolgen (z.B. 'pulumi.String(' '), pulumi.StringOutput(' '), pulumi.StringInput(' '). Es ist nicht möglich den nativen String ('str') aus pulumi.String auszulesen. Ich würde daher Typescript empfehlen

[2] Bei der DigitalOcean integration gab es ein paar tage lang einen 'Floating IP delete Authentication Bug', der manuelle intervention benötigt hat

[3] Die Vagrant-CLI eignet sich nur bedingt zur Automatisierung, da Befehle wie "vagrant sshäuf Zwischenstände in .vagrant angewiesen sind, in denen neue SSH-Keys gespeichert werden. Um eine Verbindung mit der erstellten VM über den normalen OpenSSH client aufbauen zu können muss die Host Konfiguration in .vagrant genutzt werden.

[4] Wird 'nur' eine VM 'schnell' benötigt, dann ist Vagrant gut geeignet

[5] Im Anhang findet sich ein konkretes Beispiel, das den AWS VPC Aufbau betrachtet (siehe Abschnitt D.2)

da Standardkonfigurationen ausreichen. Werden allerdings spezielle Konfigurationen benötigt, wie es bei dieser Fallstudie der Fall war, ist der Standardaufbau nicht zielführend und muss angepasst werden. Vom Standardaufbau kann abgewichen werden allerdings ist dann tieferes Wissen der Services nötig, um die nötigen Konfigurationen vorzunehmen[6]. IaC-Tools sind bei anwenderspezifischen Konfigurationen interessant, da über Bibliotheken diese vorab konfiguriert und dann ausgerollt werden können[7]. Im großen und ganzen erleichtern IaC-Tools den Umgang mit Infrastruktur, wenn es um den Aufbau und die Weiterenticklung geht.

Im Entwicklungsprozess ist die Verwendung von IaC-Tools in der Kombination mit anderen Schnittstellen, wie die CLI (*aws*, *doctl*, *gcloud*) oder das Cloud-Dashboard sind für andere Arbeiten hilfreich und sinnvoll. Die CLI ist schneller und direkter Weg mit dem System zu interagieren und Optionen durchzugehen. Das Dashboard, das bei Cloud Anbietern und bei einigen open source tools verfügbar ist, bietet eine bessere visuelle Übersicht, was bei komplexeren Aufgaben hilfreich sein kann.

7.1.2 Herausforderungen in der Hybrid-Cloud

Die zuvor definierten Herausforderungen in der Hybrid-Cloud beziehen sich auf 'sichere Kommunikation' und 'Etablieren eines Kommunikationskanals' (siehe Abschnitt 4.1).

1. **Sichere Kommunikation**: Bei der ausgewählten HCA-Strategie wurde auf VPN-Tunnel gesetzt, die eine verschlüsselte Kommunikation aufbauen. Die Verschlüsselung wird von der jeweiligen Bibliothek durchgeführt, die somit keine Herausforderung darstellt.

2. **Erstellen eines Kommunikationskanals**: Die Verbindung mit dem VPN-Tunnel war eine Herausforderung, da selbst konfiguriert und voll automatisiert ausgerollt werden soll. Es kann auf keinen vor-konfigurierten Service zurückgegriffen werden, der die Kommunikation übernimmt. Das Automatisieren der HCA-Strategie wird im nächsten anschließenden Abschnitt besprochen.

Neben den beiden vorab definierten Herausforderungen sind zwei weitere Herausforderungen aufgekommen.

[6] Oft ist es hilfreich das System einmal ohne IaC-Tool aufzubauen und dann nachträglich die Konfiguration zu übertragen.

[7] Bei dieser Fallstudie ist das über die Pulumi Bibliothek Crosswalk geschehen, die open source best practices enthält.

1. **Orchestrierung der IaC-Tools**: Wie beschrieben, sind die verwendeten IaC-Tools nicht für zusammenhängende Hybrid-Cloud Deployments ausgelegt. Um das Hybrid-Cloud Deployment zu automatisieren, wurden Shell und Python Skripte genutzt, die das Orchestrieren der IaC-Tools umsetzten. Diese Orchestrierung hat ein Großteil der Entwicklungszeit beansprucht[8].

2. **Cloud-Anbieter spezifisches Wissen**: Hybrid-Cloud Systeme nutzen mit der Public-Cloud die angebotenen Services. Vor allem bei AWS sind die Services in einem 'Ecosystem' miteinander verbunden. Um mit den Services zu arbeiten, ist ein tiefes Verständnis anderer Services nötig. Im nächsten Abschnitt findet sich ein Erfahrungsbericht zu den Public-Cloud Anbietern.

Im nächsten Abschnitt werden Erweiterungen und Verbesserungsvorschläge des HCS-1 und HCS-2 Systems präsentiert.

7.2 Erfahrungsbericht Public-Cloud Anbieter und Services

In diesem Abschnitt soll dargelegt werden, wie die Arbeit mit den Public-Cloud-Anbietern AWS, GCP und DigitalOcean im Rahmen der Fallstudie HCS-DIRECT-CONNECT sich dargestellt hat. Als Erster wird ein Blick auf die angebotenen Services gelegt und wo unterschiede zwischen den Public-Cloud-Anbietern festzustellen waren. Danach wird beleuchtet, wie IaC-Tools mit der Public-Cloud harmonieren.

In der nachfolgenden Liste werden die Public-Cloud Anbieter nacheinander besprochen.

- **AWS**: AWS ist ein komplexer Public-Cloud-Anbieter, der viel AWS-spezifisches Wissen erfordert, um sich zurechtzufinden. Die Benutzererfahrung von AWS im Bezug auf das Cloud-Dashboard und der Dokumentation ist nicht konsistent und variiert in der Qualität und Nutzerfreundlichkeit von Service zu Service. Es muss in das AWS-Ecosystem einstückweit eingetaucht werden und ein Überblick über mehrere Services gemacht werden, bevor effektiv mit AWS gearbeitet werden kann. Ein AWS-Service nutzt oft andere AWS-Services. Ist die richtige Konfiguration gewählt, so harmonieren AWS-Services miteinander gut.

[8] Nur durch Orchestrierung ist eine Automatisierung möglich. Durch Orchestrierung können IaC-Tools gemeinsam einen Emergenzeffekt erzeugen und ein System schaffen, das die IaC-Tools einzeln nicht hätten realisieren können.

AWS bietet viele Services, die breit gefächert kategorisiert sind (Computing, Serverless, Security, ...). Die meisten Services sind Nischen Produkte. Es gibt aber auch, wie angesprochen Services, die immer wieder verwendet werden und das Rückgrat von AWS bilden. Eine weitere Gruppe an Services sind Kopien, die meist von Open-Source Projekten inspiriert wurden. Die Services von AWS könnte demnach in die Gruppen Backbone-Service, Edge-Case-Service und Mock-Service kategorisiert werden.

- **GCP**: GCP hat einige Parallelen zu AWS. GCP hat ebenso wie AWS eine breite Auswahl an Services. Auch hier finden sich die Kategorien Edge-Case-, Backbone- und Mock-Service wieder. Im Gegensatz zu AWS ist die Benutzererfahrung des GCP-Dashboards und auch die Qualität der Dokumentation sehr viel besser.
- **DigitalOcean**: DigitalOcean hat im Vergleich zu den anderen beiden Public-Cloud-Anbietern nur wenige Services im Angebot. DigitalOcean setzt wenig Anbieter spezifisches Wissen voraus und hat eine gute Dokumentation. Deswegen ist DigitalOcean wohl für den Einstieg der einfachste Public-Cloud-Anbieter. DigitalOcean setzt stark auf Open-Source-Integrationen. Open-Source Integrationen wurden in der Fallstudie nicht verwendet.

Die verwendeten IaC-Tools haben mit den Public-Cloud-Anbietern gut harmoniert. Die breite Masse an Serviceangeboten bei den Cloud-Anbietern AWS und GCP erschweren Navigation in Dokumentationen. Der deutlich überschaubarere Service-Katalog von DigitalOcean ist für diese Anwendung ausreichend und lenkt den Fokus nicht so sehr ab. Da DigitalOcean ein kleiner Public-Cloud-Anbieter ist, sind IaC-Plug-ins oft von Drittanbietern erstellt, was in der Qualität der Dokumentation sich wiederspiegelt. Es finden sich deutlich mehr Beispiele für AWS und GCP.

7.3 Erweiterungen und Verbesserungsvorschläge

In diesem Abschnitt werden Erweiterungen und Verbesserungsvorschläge aufgeführt, die bei der Verwendung oder Weiterführung der Fallstudie berücksichtigt werden sollten.

- IP Adressen reservieren, um stabile VPN-Tunnel aufbauen zu können[9].

[9] In HCS-2 sind Floating IP's bei DigitalOcean zum Einsatz gekommen, die genau diesen Punkt umsetzten. Bei dem HCS-1 System werden keine Public IPv4 Adressen reserviert.

- Subnet CIDR ranges testen, bevor das System erstellt wird, um Private IPv4 Überschnitt zu vermeiden.
- Redundante VPN-Tunnel erstellen, um einen Ausfall abzufangen (High-Availability VPN).
- Konfiguriere Firewall Einstellungen in der Private-Cloud. Aktuell werden Ports in der VM direkt konfiguriert[10].
- Erstelle ein Maschinen Image, um die VM Konfiguration zu automatisieren. Aktuell werden initiale Installationen mit einem Shellskript automatisiert.
- Container-Images in einer Container-Registry veröffentlichen, um eine 'Single Source of Truth' für Container-Images zu schaffen. Zudem kann aktuell das Container-Image nicht ohne die gesamte Infrastruktur ge-updated werden. An eine Container registry könnte zudem mit einer CI/CD Pipeline weiter automatisiert werden.
- Status des Hybrid-Cloud Deployments in einer Datei speichern, um nicht Idempotente-Anweisungen zwischen zu speichern[11].
- Deployments von einem Container ausführen, der alle nötigen Bibliotheken installiert und Weiteres konfiguriert, um so Platform-spezifische Fehler zu vermeiden.
- Prozesse im Deployment parallelisieren, sofern das möglich ist (siehe Abbildung 5.13 als Anknüpfungspunkt).
- Infrastruktur-Tests implementieren, um das System automatisiert verifizieren zu können.

Im nächsten Kapitel wird ein Blick zurück auf die Arbeit geworfen und besprochen, an welchen Punkten weitere Untersuchen anknüpfen können.

[10] Firewall Einstellungen wurden nur dem Private-Cloud System in HCS-1 direkt auf der Maschine konfiguriert.

[11] In dem HCS-2 System wird das VPN Secret in einer JSON-Datei temporär abgespeichert. Dies könnte noch ausgeweitet werden.

8 Zusammenfassung und Ausblick

In diesem Abschnitt wird ein Blick zurück auf die Arbeit geworfen und ein Ausblick gegeben. Zu Beginn der Arbeit wurden Ziele aufgestellt, die jetzt explizit besprochen werden (siehe Abschnitt 1.2). Im Kern waren zwei Fragen aufgestellt worden.

- Ist ein Hybrid-Cloud-System mit Automatisierungen praktikabler aufzusetzen?
- Ist die Kombination mehrerer Infrastruktur-Automatisierungstools (IaC-Tools) eine sinnvolle Strategie, um Hybrid-Cloud-Systeme zu verwalten?

In den beiden Fallstudien wurden mehrere IaC-Tools verwendet, um die IT-Infrastruktur automatisiert zu verwalten. Dabei muss aktuell auf ein IaC-Tool-Mix gesetzt werden, da kein IaC-Tool alleine die gesamte IT-Infrastruktur verwalten hätte können. Die IaC-Tools sind für die jeweiligen Anwendungsfälle[1] geeignet, funktionieren allerdings im Zusammenspiel mit anderen IaC-Tools nicht automatisch. Um ein kohärentes System mit Verwendung mehrerer IaC-Tools automatisiert ausrollen zu können, muss mit eigenen Skripten die Prozesse angestoßen werden (in der Abbildung 5.13 sind die Prozesse der Fallstudie gezeigt). Es fehlt derzeit an einem IaC-Tool 'Orchestrierer-Service', der als 'wrapper' fungiert. In den Fallstudien wurden Shell- und Python-Skripte erstellt, um diese IaC-Tool-Orchestrierung umzusetzen.

Die zweite Frage, ob eine Kombination mehrerer IaC-Tools eine sinnvolle Strategie sei, stellt sich aktuell noch nicht da, wie beschrieben kein IaC-Tool die gesamte IT-Infrastruktur hätte verwalten können. Allgemein ist ein all umfassendes IaC-Tool kaum praktikabel und erstrebenswert (siehe Kapitel 3). IaC-Tools einzusetzen ist definitiv sinnvoll, allerdings ist es fraglich, ob eine komplette Automatisierung aller Prozesse zum aktuellen Zeitpunkt sinnvoll ist.

Alles in allem ist eine hybride Cloud IT-Infrastruktur komplex. Die Verwendung mehrerer Cloud-Umgebungen verkompliziert den Entwicklungsprozess erheblich. Technische

[1] Wie z.B. AWS Services verwalten, VM Images erstellen, usw. (siehe Abschnitt 3.3)

Herausforderungen einer Hybrid-Cloud können jedoch durch den Gebrauch von Technologien wie Kubernetes, VPN-Tunnel, Service-Meshs bewältigt werden. Welche dieser Technologien eingesetzt werden, hängt von den Projekt-Anforderungen ab und beeinflusst den weiteren Entwicklungsprozess maßgeblich. Das Arbeiten mit mehreren Cloud-Umgebungen erfordert ein fundiertes Cloud-spezifisches Wissen. IaC-Tools können als weitere Art und Weise Schnittstellen der Cloud-Anbieter anzusprechen, hilfreich sein. Bei der Entwicklung eines Hybrid-Cloud-Systems finden neben IaC-Tools andere Schnittstellen zur Cloud-Umgebung wie die CLI oder das Cloud-Dashboard weiterhin sinnvoll Verwendung.

Diese Arbeit zeigte zwischen den Kapiteln auf, wo weitere Untersuchungen unternommen werden könnten:

- Kategorisierung und Bewertung von IaC-Tools (siehe Abschnitt 3.3)
- systematische Entwicklung und Realisierung von HCA-Strategien (siehe Abschnitt 4.3)
- Untersuchung einzelner HCA-Strategien (siehe Abschnitt 4.3)
- Weiterentwicklung der Fallstudie (siehe Abschnitt 7.3)
- realitätsnäheren Anwendungsfall mit einer HCA-Strategie implementieren (siehe Abschnitt 5.1.2)

In Zukunft werden die diskutierten Technologien voraussichtlich weiter an Dynamik gewinnen. Auch die Hybrid-Cloud steht zunehmend im Fokus der Public-Cloud-Anbieter, was sich an den neuen Services wie AWS-Outpost oder der Open Source AWS-EKS Distro ablesen lässt. Diese Systemintegrationen und Überbrückungen von Public- zu Private-Cloud können durch IT-Infrastruktur-unabhängige IaC-Tools vorangetrieben werden. Die angesprochenen Technologien Kubernetes und Service Mesh werden den Umgang mit Hybrid-Clouds weiter ausbauen und praktikabler gestalten.

Literaturverzeichnis

[1] A. S. Tannenbaum, M. .: *Verteilte Systeme*. Pearson, 2008. – URL https://link.springer.com/book/10.1007/978-3-662-53143-3. – ISBN 9783827372932

[2] A.Tayal, D. .: *Determining the Total Cost of Ownership of Serverless Technologies when compared to Traditional Cloud*. https://pages.awscloud.com/NAMER-field-GC-Deloitte-TCO-whitepaper-2019-learn-ty.html. 2019. – Zugegriffen am 20.09.2021

[3] Avidor, R.: *Murphy's laws origin*. 2021. – URL http://www.murphys-laws.com/murphy/murphy-true.html. – Zugegriffen am 20.09.2021

[4] AWS: *Amazon EC2 Instance Types*. https://aws.amazon.com/ec2/instance-types/. – Zugegriffen am 20.09.2021

[5] AWS: *Amazon Virtual Private Cloud, Transit Gateways*. https://docs.aws.amazon.com/vpc/latest/tgw/vpc-tgw.pdf#. – Zugegriffen am 20.09.2021

[6] AWS: *AWS EKS Distro*. https://distro.eks.amazonaws.com. – Zugegriffen am 20.09.2021

[7] AWS: *AWS EKS, User Guide*. https://docs.aws.amazon.com/eks/latest/userguide/eks-ug.pdf. – Zugegriffen am 20.09.2021

[8] AWS: *AWS Lambda, Developer Guide*. https://docs.aws.amazon.com/lambda/latest/dg/lambda-dg.pdf. – Zugegriffen am 20.09.2021

[9] AWS: *How can I push Amazon CloudWatch Logs cross-account to Kinesis Data Firehose?* https://aws.amazon.com/premiumsupport/knowledge-center/kinesis-firehose-cloudwatch-logs/. – Zugegriffen am 20.09.2021

[10] AWS: *AWS EKS-Distro Accouncement*. https://aws.amazon.com/blogs/opensource/introducing-amazon-eks-distro/. 2020. – Zugegriffen am 20.09.2021

[11] AWS: *Hybrid Cloud with AWS*. November 2020. – URL https://d1.awsstatic.com/whitepapers/hybrid-cloud-with-aws.pdf?did=wp_card&trk=wp_card. – Zugegriffen am 20.09.2021

[12] AWS: *Amazon Machine Images (AMI)*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-ug.pdf#AMIs. 2021. – Zugegriffen am 20.09.2021

[13] AWS: *AWS CloudFormation concepts*. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-whatis-concepts.html. 2021. – Zugegriffen am 20.09.2021

[14] AWS: *AWS Outpost*. https://docs.aws.amazon.com/outposts/latest/userguide/outposts.pdf#what-is-outposts. 2021. – Zugegriffen am 20.09.2021

[15] B. Scholl, P. .: *Cloud Native*. O'Reilly Media, Inc., 2019. – URL https://www.oreilly.com/library/view/terraform-up/9781492046899/. – ISBN 9781492053828

[16] Baun, C.: *Betriebssysteme Kompakt*. Springer Vieweg, 2017. – URL https://link.springer.com/book/10.1007/978-3-662-53143-3. – ISBN 9783662531426

[17] BMWI: *GAIA-X*. https://www.bmwi.de/Redaktion/DE/Dossier/gaia-x.html. – Zugegriffen am 20.09.2021

[18] BSI: *Rechenzentrums-Definition*. https://www.bsi.bund.de/DE/Themen/Sicherheitsberatung/Hochverfuegbarkeit/Rechenzentren/Rechenzentren_node.html. – Zugegriffen am 20.09.2021

[19] CFEngine: *CFEngine Guide*. https://docs.cfengine.com/docs/3.12/guide.html. – Zugegriffen am 20.09.2021

[20] Chef: *Chef Infra Server Overview*. https://docs.chef.io/server/. – Zugegriffen am 20.09.2021

[21] Chiueh, S.: A Survey on Virtualization Technologies. (2005), S. 1–42

[22] Cilium: *Deep Dive into Cilium Multi-cluster*. https://cilium.io/blog/2019/03/12/clustermesh. – Zugegriffen am 20.09.2021

[23] Clinton, D.: *Teach Yourself Linux Virtualization and High Availability: prepare for the LPIC-3 304 certification exam*. Bootstrap IT, 2017. – URL https://bootstrap-it.com/books/. – ISBN 9781365854866

[24] Ebpf: *Ebpf Homepage*. https://ebpf.io. – Zugegriffen am 20.09.2021

[25] Goldberg, R. P.: Survey of virtual machine research. In: *Computer* 7 (1974), Nr. 6, S. 34–45

[26] Group, DevOps: *The CALMS Model of DevOps*. https://www.devopsgroup.com/insights/resources/diagrams/all/calms-model-of-devops/. – Zugegriffen am 20.09.2021

[27] HashiCorp: *Consul-AWS*. https://learn.hashicorp.com/tutorials/consul/sync-aws-services. – Zugegriffen am 20.09.2021

[28] HashiCorp: *Consul Provider*. https://registry.terraform.io/providers/hashicorp/consul/latest/docs. – Zugegriffen am 20.09.2021

[29] HashiCorp: *The imperative for hybrid IT: cloud and innovation in the modern era*. https://www.terraform.io. – Zugegriffen am 20.09.2021

[30] HashiCorp: *Unlocking the Cloud operating model: Cloud compliance and management*. https://www.hashicorp.com/resources/unlocking-the-cloud-operating-model-provisioning?utm_source=terraformcomplianceCTA. – Zugegriffen am 20.09.2021

[31] HashiCorp: *Cloud-Init: The Good Parts*. https://www.hashicorp.com/resources/cloudinit-the-good-parts. 2019. – Zugegriffen am 20.09.2021

[32] HashiCorp: *Packer Build Automated Machine Images*. https://www.packer.io. 2021. – Zugegriffen am 20.09.2021

[33] HasiCorp: *Introduction to Vagrant*. https://www.vagrantup.com/intro. – Zugegriffen am 20.09.2021

[34] HasiCorp: *Vagrantfile*. https://www.vagrantup.com/docs/vagrantfile. – Zugegriffen am 20.09.2021

[35] Hill, S.: *Implementing hybrid cloud: 5 major challenges*. https://sandhill.com/article/implementing-hybrid-cloud-5-major-challenges/. – Zugegriffen am 20.09.2021

[36] IEEE: IEEE Guide for Software Requirements Specifications. In: *IEEE Std 830-1984* (1984), S. 1–26

[37] IJCSIS: Hybrid Cloud: An Enterprise Solution. (2017), S. 12

[38] Istio: *Installation with Ansible*. https://istio.io/v1.0/docs/setup/kubernetes/ansible-install/. – Zugegriffen am 20.09.2021

[39] Istio: *Istio Concepts Security*. https://istio.io/latest/docs/concepts/security/. – Zugegriffen am 20.09.2021

[40] Jochimsen, R.: *Theorie der Infrastruktur. Grundlagen der marktwirtschaftlichen Entwicklung*. Mohr (Siebeck), 1966. – ISBN B0000BRSV7

[41] K. S. Mishra, A. K. .: Some Issues, Challenges and Problems of Distributed Software System. In: *IJCSIT* 5 (2014), Nr. 0975-9646, S. 4922–4925

[42] Kubernetes: *Kubernetes Glossar*. https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane. – Zugegriffen am 20.09.2021

[43] Lamport, L.: *Leslie Lamport - My Writings*. http://lamport.azurewebsites.net/pubs/pubs.html#bakery. – Zugegriffen am 20.09.2021

[44] Leymann, F.: *IT-Ressource Defintion*. https://wirtschaftslexikon.gabler.de/definition/it-ressource-53361. 2020. – Zugegriffen am 20.09.2021

[45] Lounsbury, J.: *The imperative for hybrid IT: cloud and innovation in the modern era*. https://www.silect.is/blog/multi-cloud-vpn-terraform/. 2020. – Zugegriffen am 20.09.2021

[46] Lynch, N. A.: *Distributed Algorithms*. 1996. – ISBN 978-1-55860-348-6

[47] M. Fowler, R. .: *Domain Specific Languages*. Addison-Wesley Professional, 2010. – URL https://www.martinfowler.com/books/dsl.html. – ISBN 9780321712943

[48] M. Rosenblum, T. .: Virtual machine monitors: current technology and future trends. In: *Computer* 38 (2005), Nr. 5, S. 39–47

[49] Microsoft: *Sidecar Pattern*. https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar. – Zugegriffen am 20.09.2021

[50] NGINX: *What's a service mesh?* https://www.nginx.com/blog/what-is-a-service-mesh/. – Zugegriffen am 20.09.2021

[51] Nielsen, L.: *30. Personas*. https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/personas. – Zugegriffen am 20.09.2021

[52] NixOs: *How Nix works*. https://nixos.org/guides/how-nix-works.html. – Zugegriffen am 20.09.2021

[53] P. Heidkamp, A. .: *Cloud-Monitor 2020: Gut eingerichtet in der Cloud*. https://hub.kpmg.de/studie-cloud-monitor-2020?utm_campaign=Cloud-Monitor%202020&utm_source=AEM. 2020. – Zugegriffen am 20.09.2021

[54] P. Mell, T. .: *The NIST Definition of Cloud Computing*. https://csrc.nist.gov/publications/detail/sp/800-145/final. 2011. – Zugegriffen am 20.09.2021

[55] Popek, Gerald J. ; Goldberg, Robert P.: Formal Requirements for Virtualizable Third Generation Architectures. In: *Commun. ACM* 17 (1974), Juli, Nr. 7, S. 412–421. – URL https://doi.org/10.1145/361011.361073. – ISSN 0001-0782

[56] Poulton, N.: *The Kubernetes Book*. Independently published (April 17, 2021), 2021. – URL https://nigelpoulton.com/books/. – ISBN 9798703756065

[57] Roberts, M.: *Defining Serverless, Part 1*. https://blog.symphonia.io/posts/2017-06-22_defining-serverless-part-1. 2017. – Zugegriffen am 20.09.2021

[58] S. Miano, F. Risso M. .: Creating Complex Network Services with eBPF: Experience and Lessons Learned. In: *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, S. 1–8

[59] S. Patig, S. .: *IT-Infrastruktur Definition*. https://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Informationsmanagement/IT-Infrastruktur/. 2019. – Zugegriffen am 20.09.2021

[60] Saltstack: *Salt Documentation*. https://docs.saltproject.io/en/latest/topics/index.html#the-30-second-summary. – Zugegriffen am 20.09.2021

[61] Sigs, Kubernetes: *Deploy a Production Ready Kubernetes Cluster*. https://github.com/kubernetes-sigs/kubespray. – Zugegriffen am 20.09.2021

[62] Song, J.: *Hybrid Cloud with Istio and AWS EKS-Distro*. https://thenewstack.io/ensure-consistency-in-hybrid-clouds-with-amazon-web-services-eks-d-and-istio/. 2021

[63] Starke, G.: *arc42: Better Software Architectures*. https://arc42.org/. – Zugegriffen am 20.09.2021

[64] Starke, G.: *Effektive Softwarearchitekturen, Ein Praktischer Leitfaden, 9., ueberarbeitete Auflage*. Carl Hanser Verlag Muenchen, 2020. – URL https://www.hanser-fachbuch.de/buch/Effektive+Softwarearchitekturen/9783446463769. – ISBN 9783446463769

[65] Statista: *Adoption rate of emerging technologies in organizations worldwide as of 2020, by scale*. https://www.statista.com/statistics/661164/worldwide-cio-survey-operational-priorities/. Dezember 2020. – Zugegriffen am 20.09.2021

[66] Statista: *Cloud Anbieter Marktanteil*. https://de.statista.com/infografik/20802/weltweiter-marktanteil-von-cloud-infrastruktur-dienstleistern/. 2020. – Zugegriffen am 20.09.2021

[67] Statista: *Usage of private cloud computing in companies in Germany from 2011 to 2019*. https://www.statista.com/statistics/489144/private-cloud-computing-usage-in-companies-germany/. Oktober 2020. – Zugegriffen am 20.09.2021

[68] Statista: *Anteil der Unternehmen in der DACH-Region, die folgende Technologien nutzen oder derzeit implementieren, im Jahr 2020*. https://de.statista.com/statistik/daten/studie/818025/umfrage/nutzung-verschiedener-

technologien-in-unternehmen-in-der-dach-region/. Maerz 2021. – Zugegriffen am 20.09.2021

[69] Statista: *Information technology (IT) worldwide spending from 2005 to 2022*. https://www.statista.com/statistics/203935/overall-it-spending-worldwide/. Juli 2021. – Zugegriffen am 20.09.2021

[70] Statista: *Public IT cloud services market revenue worldwide from 2016 to 2020, by segment*. https://www.statista.com/statistics/370305/global-public-it-cloud-services-spending-by-segment/. Mai 2021. – Zugegriffen am 20.09.2021

[71] Statista: *Umsatz mit Cloud Computing** weltweit von 2010 bis 2020 und Prognose bis 2022*. https://de.statista.com/statistik/daten/studie/195760/umfrage/umsatz-mit-cloud-computing-weltweit/. August 2021. – Zugegriffen am 20.09.2021

[72] Statista: *Usage of cloud computing in companies in Germany from 2011 to 2020*. https://www.statista.com/statistics/459998/usage-of-cloud-computing-by-companies-in-germany/. August 2021. – Zugegriffen am 20.09.2021

[73] Statista: *Welche IT-Projekte haben in Ihrem Unternehmen die höchste Priorität?* https://de.statista.com/statistik/daten/studie/1131250/umfrage/umfrage-zu-den-wichtigsten-it-projekte-in-anwenderunternehmen-in-der-schweiz/. Juni 2021. – Zugegriffen am 20.09.2021

[74] Stender, D.: *Cloud-Infrastruktur, Das Handbuch fuer DevOps-Teams und Administratoren*. Rheinwerk Verlag, 2020. – ISBN 9783836269483

[75] technology, Technical Committee : ISO/IEC JTC 1 I.: ISO/IEC 7498-1:1994 Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model. In: *ISO/IEC JTC 1 Information Technology* (1994), S. 59

[76] W. Heinhaus, U. .: *Was sichert Rechenzentren von Tier I bis IV?* https://www.datacenter-insider.de/was-sichert-rechenzentren-von-tier-i-bis-iv-a-619349/. 2017. – Zugegriffen am 20.09.2021

A Anhang, Kapitel 2

Dieser Anhang gehört zum Kapitel 2. Innerhalb des Kapitels wird auf den Inhalt dieses Anhangs verwiesen.

A.1 Regionen und Verfügbarkeitszonen in AWS

Bei AWS wird in Regionen und Verfügbarkeitszonen unterschieden. Regionen sind hierbei Standorte mit mehreren Rechenzentren. Ein einzelnes Rechenzentrum bildet dabei eine Verfügbarkeitszone[1]. AWS unterhält bspw. die Region Frankfurt benannt als 'eu-central-1'. Innerhalb der Region gibt es drei Verfügbarkeitszonen 'eu-central-1a', 'eu-central-1b' und 'eu-central-1c'.

In der Abbildung A.1) ist der Aufbau von Regionen und Verfügbarkeitszonen des Public-Cloud Anbieters AWS visualisiert. In dem Diagram ist die Unterscheidung in Regionen und Verfügbarkeitszonen gezeigt.

- **Regionen** beziehen sich auf einen Standort in dem AWS mehrere Rechenzentren verwaltet. In dem Diagram ist eu-west-1 und eu-central-1 als Regionen genannt. eu-west-1 referenziert die Region Irland und eu-central-1 referenziert die Region Frankfurt.
- **Verfügbarkeitszonen** beziehen sich auf Rechenzentren, die innerhalb einer Region zur verfügung stehen. AWS bietet in jeder Region zwei oder drei Verfügbar-

[1] Rechenzentren sind Standortgebunden. Eine Geografische Unterscheidung in Regionen und Verfügbarkeitszonen macht deshalb Sinn.

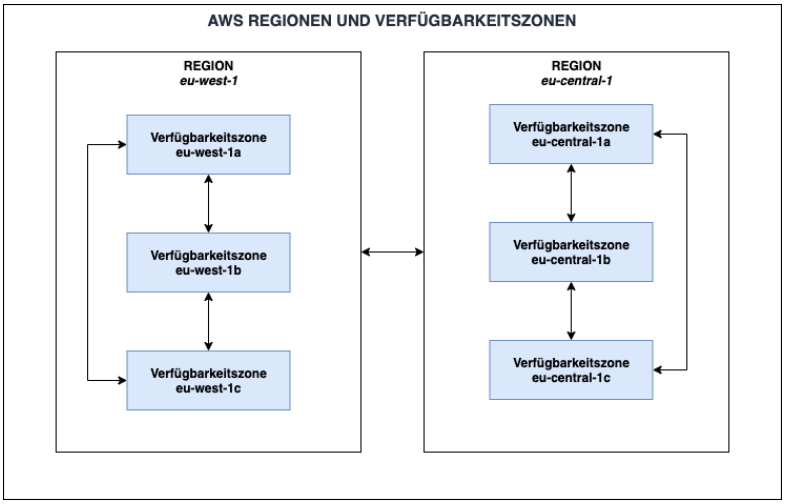


Abbildung A.1: AWS Regionen und Verfügbarkeitszonen

keitszonen an, damit bei ausfällen in Rechenzentren ausgewichen werden kann[2]. Die Region Irland hat drei Verfügbarkeitszonen: eu-west-1a, eu-west-1b und eu-west-1c.

Andere Public-Cloud-Anbieter wie GCP, Azure oder DigitalOcean folgen der gleichen Struktur von Regionen und Verfügbarkeitszonen / Zonen.

A.2 Die Verfügbarkeit eines Rechenzentrums

Ein kritischer Wert eines Rechenzentrums stell die Verfügbarkeit dar. Verfügbarkeit eines Rechenzentrums bedeutet, ob die vorliegende IT-Infrastruktur einwandfrei genutzt werden kann. Die Verfügbarkeit eines Rechenzentrums kann in eine Vier-Tier Topologie gegliedert werden. In Tabelle A.1 sind die einzelnen Verfügbarkeitsklassen aufgelistet.

Ein Ausfall der IT-Systeme kann den Supergau eines Unternehmens bedeuten. Cloud-Anbieter müssen sich selbst den Anspruch stellen 100% verfügbar zu sein, um kritische Anwendungsapplikationen verwalten zu dürfen. Cloudharmony.com bietet eine Auflistung von Servern einzelner Cloud-Anbieter und zeichnet deren Erreichbarkeit auf. Die

[2]Natürlich ist es auch möglich, überregional zu arbeiten, aber das erhöht oft die Kosten, da die Rechenzentren innerhalb einer Region über Kabel direkt miteinander Verbunden sind und dadurch effizienter kommunizieren können. Des weiteren haben große Unternehmen manchmal Einschränkungen und dürfen z. B. nur Daten innerhalb der EU oder Deutschlands verarbeiten, wenn es keine weitere Region gibt, die diese Anforderung erfüllt, kann der Cloud-Anbieter nicht ausgewählt werden. Dass eine Region mehrere Verfügbarkeitszonen hat, ist daher relevant.

Tabelle A.1: Verfügbarkeitseinordnung von Rechenzentren[76]

Verfügbarkeitsklasse	Redundanz	Verfügbarkeit	Ausfall-Wahrscheinlichkeit p. Jahr
I	N* keine Redundanz	99.67%	28,8 Stunden
II	N+1	99.75%	22,0 Stunden
III	2 N System	99.98%	1,6 Stunden
IV	(2(N+1)	99,99%	0,8 Stunden

großen, bereits eingeführten Cloud-Anbieter sind so gut wie immer erreichbar. Wie zuvor beschrieben verwalten Cloud-Anbieter deswegen mehrere Rechenzentren in der selben Region. Unter den Cloud-Anbietern kommt es nur bei Katastrophen zu einem Regionalen Total-Ausfall. Eine Katastrophe kann zum Beispiel eine Cyber-Attacke oder eine Natur-Katastrophe sein. Unternemen replizieren deshalb oft extra Daten in mehrere Regionen.

"»Alles, was schiefgehen kann, wird auch schiefgehen.«"(Murphys Gesetz)[3]

A.3 IT-Ressource - AWS EC2

AWS bietet mit dem Service EC2 10 Typen von virtuelle Maschinen an[4]. Zwei dieser Typen sind 'A'-Instanzen, die einen ARM-Prozessor und 'T'-Instanzen, die einen Burst-Modus haben, welcher temporär die Leistung erhöhen kann. Instanzen bekommen Hardware-Ressourcen zugewiesen, wie in der Tabelle anhand der 'A'-Instanz gezeigt ist (siehe Tabelle A.2).

Tabelle A.2: Übersicht der AWS EC2 A1-Instanzen[4]

Instance	vCPU	Mem (GiB)	Network Performance (Gbps)
a1.medium	1	2	Bis zu 10
a1.large	2	4	Bis zu 10
a1.xlarge	4	8	Bis zu 10
a1.2xlarge	8	16	Bis zu 10
a1.4xlarge	16	32	Bis zu 10
a1.metal	16*	32	Bis zu 10

A.4 AWS Outpost als Hybrid-Cloud enabler

Services wie AWS Outpost werden von Public-Cloud-Anbietern wie AWS angeboten, um private Cloud-Systeme mit der Public Cloud zu verbinden. Bei AWS Outpost wird ein Container mit einer eingebauten IT-Infrastruktur vor Ort beim Unternehmen installiert, um private Systeme mit der Public-Cloud zu verbinden.

> "AWS Outposts ist ein vollständig verwalteter Service, der die AWS-Infrastruktur, -Services, -APIs und -Tools auf Kundengelände erweitert. Durch die Bereitstellung eines lokalen Zugriffs auf die von AWS verwaltete Infrastruktur ermöglicht AWS Outposts den Kunden die Erstellung und Ausführung von Anwendungen vor Ort mit denselben Programmierschnittstellen wie in AWS Regions, während lokale Rechen- und Speicherressourcen für geringere Latenzzeiten und lokale Datenverarbeitungsanforderungen genutzt werden."[14]

A.5 Serverless Services im Vergleich zu Traditionellen Services

In einem Whitepaper wurde ein traditioneller Ansatz mit dem Serverless-Ansatz verglichen[2]. Im traditionellen Cloud-Projekt wurden Dienste wie AWS-EC2 (AWS VM Service) verwendet und im Serverless Cloud-Projekt Dienste wie AWS-Lambda. In der Abbildung A.2 ist gezeigt, dass der Serverless Ansatz in Bereich der Projektkosten, Applikations-Skalierbarkeit und Flexibilität, Kapazität und Veröffentlichungszeit von neuen Applikations-Versionen besser oder attraktiver abschneidet.

> "Zusammenfassend lässt sich sagen, dass der Zeitaufwand für die Wartung und den laufenden Betrieb bei Serverless-Anwendungen sinkt, da diese vollständig vom Cloud-Anbieter verwaltet werden. Daher werden sich die Rollen eines dezidierten Betriebsteams weiterentwickeln müssen. Serverless-Architekturen bieten außerdem unbegrenzte Skalierbarkeit und integrierte Hochverfügbarkeit, die in einem traditionellen Umfeld zusätzliche Anstrengungen und Kosten bedeuten. Wenn wir nur die Infrastrukturkosten zwischen den Plattformen vergleichen, können wir feststellen, dass ein traditionelles Modell kosteneffektiv ist. Wenn wir jedoch die zusätzlichen Vorteile und Kosteneinsparungen eines Serverless-Modells aufschichten, können Unternehmen durch den

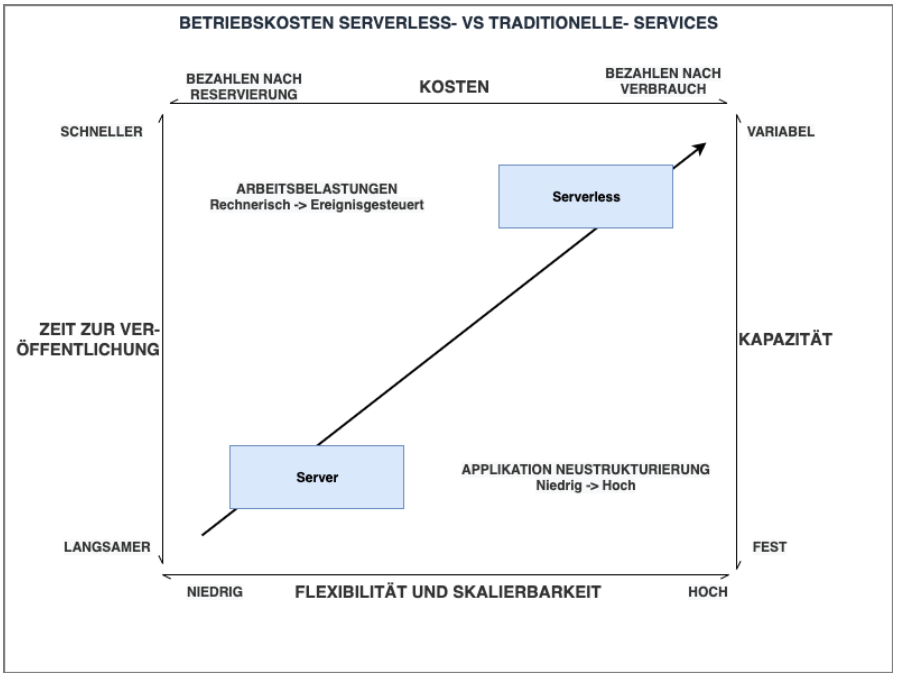


Abbildung A.2: Gesamtbetriebskosten von Serverless-Services und traditionellen Services[2]

> Aufbau von Anwendungen und allgemeinen Organisationsstrukturen zur effektiven Nutzung einer Serverless-Architektur erhebliche Einsparungen erzielen.“[2]

Der traditionelle Cloud-Ansatz kann nicht durch den Serverless-Ansatz ersetzt werden, sondern stellt eine Alternative für Anwendungen da. Es gibt Anwendungsfälle für beide Ansätze.

B Anhang, Kapitel 5

Dieser Anhang gehört zum Kapitel 5. Innerhalb des Kapitels wird auf den Inhalt dieses Anhangs verwiesen.

B.1 AWS CloudWatch als Alternative zum zentralen aggregierten von log Einträgen in AWS

In der Abbildung B.1 ist eine Alternative zum aggregieren von log Einträgen innerhalb der Fallstudie HCS-DIRECT-CONNECT gezeigt. Diese Alternative nutzt den AWS Service CloudWatch zum dezentralen verwalten der log Einträge. Neben AWS CloudWatch kommen auch AWS FireHose und AWS EventBridge-Events zum Einsatz. Idee dieses Aufbaus ist es ein zentralen Anlaufpunkt im HCS-SYS-PLATFORM System für log Einträge zu schaffen, der von mehreren AWS Accounts aus angesprochen werden kann[9]. In diesem Aufbau wird das zentrale System von nur einem Account aus gespeist.

Im dem AWS-Account HCS-SYS-PUBLIC wird bei allen AWS-Services die automatische integration mit AWS CloudWatch zum aggregieren von log Einträgen aktiviert.[1] Um log Einträge an den HCS-PLATFORM Account zu übertragen, muss eine Subscription zwischen einer log Gruppe und einem AWS EventBridge-Event in der HCS-SYS-PLATFORM erstellt werden. Die erstellten Events werden mit dem AWS FireHose-Service verknüpft, um diese Events, die Log-Einträge enthalten, schlussendlich 'batchweise' in AWS S3 zu speichern.

[1] Innerhalb von AWS. Oft gibt es eine Option, diese Integration selbst zu gestalten. Dienste, die keine sinnvollen Protokolleinträge erzeugen, wie AWS-interne API-Aufrufe, sollten ebenfalls nicht gespeichert werden.

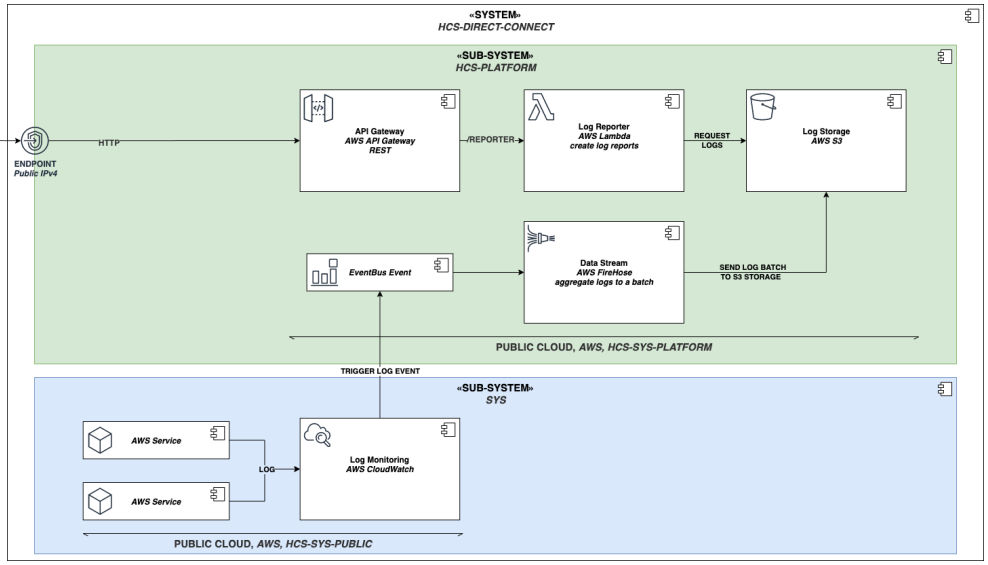


Abbildung B.1: Verteilungssicht - Alternative mit nativen AWS-Services

B.2 HCS-SYS-PRIVATE Vagrantfile

In dem nachfolgenden Ausschnitt ist die Vagrantfile gegeben, die ausgeführt wird, um das HCS-SYS-PRIVATE System in der (pseudo) Private-Cloud, DigitalOcean, zu erstellen.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
require 'yaml'

# Check plugin installations needed to build the VM
unless Vagrant.has_plugin?("vagrant-docker-compose")
  system("vagrant plugin install vagrant-docker-compose")
  puts "Dependencies installed, please try the command again."
  exit
end

unless Vagrant.has_plugin?("vagrant-digitalocean")
  system("vagrant plugin install vagrant-digitalocean")
  puts "Dependencies installed, please try the command again."
  exit
end

# Configure Virtual Machine
Vagrant.configure("2") do |config|
  # Variables
```

```
21 LEGACY_CONTAINER_PORT = 8050
22 PROJECT_NAME = "hcs-sys-private-cloud"
23 PRIVATE_KEY_PATH = '~/.ssh/id_ed25519'
24 DO_BOX_URL = "https://github.com/devopsgroup-io/vagrant-digitalocean/raw/
     master/box/digital_ocean.box"
25 TOKEN = begin
26   YAML.load_file('token.yaml').fetch('digital_ocean_token')
27 rescue Errno::ENOENT, KeyError
28   '.'
29 end
30
31 config.vm.define "droplet-hcs-sys-private-cloud" do |config|
32   config.vm.provider :digital_ocean do |provider, override|
33     override.vm.box = 'digital_ocean'
34     override.vm.box_url = DO_BOX_URL
35     override.vm.allowed_synced_folder_types = :rsync
36
37     # Digital Ocean Access Token
38     provider.token = TOKEN
39     # SSH File to access the VM
40     override.ssh.private_key_path = PRIVATE_KEY_PATH
41     provider.ssh_key_name = "hcssh"
42
43     # OS
44     provider.image = 'ubuntu-20-04-x64'
45     # Hardware specifications
46     provider.size = 's-1vcpu-1gb'
47     # Datacenter Region
48     provider.region = 'fra1'
49   end
50
51   # Copy file to VM
52   config.vm.synced_folder "legacy-container/", "/vagrant"
53
54   # Make port accessible
55   config.vm.network(:forwarded_port, guest: LEGACY_CONTAINER_PORT, host:
     LEGACY_CONTAINER_PORT)
56
57   # Run inline shell - update system and install wireguard, generate
     private key
58   config.vm.provision :shell, inline: "sudo apt-get update -y"
59   config.vm.provision :shell, inline: "sudo apt-get -y install wireguard"
60   config.vm.provision :shell, inline: "sudo apt -y install resolvconf"
```

```
61     config.vm.provision :shell, inline: "wg genkey | sudo tee /etc/wireguard/
       client_private.key | wg pubkey | sudo tee /etc/wireguard/client_public.
       key"
62     config.vm.provision :shell, inline: "sudo chmod 600 /etc/wireguard/
       client_private.key"
63
64     # Install docker and docker-compose
65     config.vm.provision :docker
66     config.vm.provision :docker_compose, env: { "PORT" => "#{
       LEGACY_CONTAINER_PORT}" }, yml: ["/vagrant/docker-compose.yaml"], rebuild
       : true, project_name: "#{PROJECT_NAME}", run: "always"
67   end
68 end
```

Listing B.1: Vagrantfile des HCS-SYS-PRIVATE Projektes

B.3 Deployment des HCS-SYS-DIRECT-CONNECT Systems

In diesem Abschnitt wird anhand der Konsolen-Einträgen gezeigt, wie das HCS-SYS-DIRECT-CONNECT System deployed wird. Da die Konsolen-Einträge sehr lang sind, werden diese zwischendurch gekürzt.

In dem Unteren Code Ausschnitt ist der erste Teil des HCS-DIRECT-CONNECT deployments gegeben. In diesem Abschnitt wird das Private-Cloud System mit Vagrant erstellt. Der Prozess wird von der 'Vagrantfile' abgebildet.

```
1  DEPLOY-MAIN: START WITH HYBRID-CLOUD-DEPLOYMENT...
2    CHECK Installations
3  DEPLOY-MAIN: 1. create hcs-sys-private-cloud setup useing vagrant
4    START HCS-SYS-PRIVATE - DEPLOY.SH ...
5      Deploy legacy system...
6      Bringing machine up with 'digital_ocean' provider...
7      ==> droplet-hcs-sys-private-cloud: Using existing SSH key
8      ==> droplet-hcs-sys-private-cloud: Creating a new droplet...
9      ==> droplet-hcs-sys-private-cloud: Assigned IP address
10     ==> droplet-hcs-sys-private-cloud: Private IP address
11     ==> droplet-hcs-sys-private-cloud: Rsyncing folder
12     ==> droplet-hcs-sys-private-cloud: Running provisioner: shell...
13     ==> droplet-hcs-sys-private-cloud: Running provisioner: docker...
14     ==> droplet-hcs-sys-private-cloud: Running provisioner: docker_compose...
```

```
    ==> droplet-hcs-sys-private-cloud: Building app
    ==> droplet-hcs-sys-private-cloud: Creating network
    ==> droplet-hcs-sys-private-cloud: Creating app ...
    Creating hcs-sys-private-cloud_app_1 ... done
  FINISHED HCS-SYS-PRIVATE - DEPLOY.SH
```

Listing B.2: HCS-DIRECT-CONNECT Deployment, Private-Cloud (1/4)

In dem Unteren Code Ausschnitt ist der zweite Teil des HCS-DIRECT-CONNECT deployments gegeben. In diesem Abschnitt wird das Public-Cloud System mit Pulumi erstellt. Der Prozess wird von der 'index.ts' in Typescript geschriebenen Pulumi-Datei abgebildet.

```
DEPLOY-MAIN: 2. create hcs-sys-public-cloud setup useing pulumi
  START HCS-SYS-PUBLIC - DEPLOY.SH ...
    Set pulumi configuration...
    Deploy Pulumi project...
    Resources:
     + 50 to create
     +   pulumi:pulumi:Stack
     +   - awsx:x:ec2:SecurityGroup
     +      - aws:ec2:SecurityGroup
     +   - aws:lb:ApplicationLoadBalancer
     +      - awsx:lb:ApplicationTargetGroup
     +         - aws:lb:TargetGroup
     +      - awsx:lb:ApplicationListener
     +         - awsx:x:ec2:IngressSecurityGroupRule
     +            - aws:ec2:SecurityGroupRule
     +         - awsx:x:ec2:EgressSecurityGroupRule
     +            - aws:ec2:SecurityGroupRule
     +         - aws:lb:Listener
     +      - aws:lb:LoadBalancer
     +   - awsx:x:ecs:FargateTaskDefinition
     +      - aws:iam:Role
     +      - aws:ecr:Repository
     +      - aws:iam:Role
     +      - aws:ecr:LifecyclePolicy
     +      - aws:iam:RolePolicyAttachment
     +      - aws:iam:RolePolicyAttachment
     +      - aws:iam:RolePolicyAttachment
     +      - aws:iam:RolePolicyAttachment
     +      - aws:ecs:TaskDefinition
     +   - awsx:x:ecs:Cluster
     +      - awsx:x:ec2:SecurityGroup
```

```
32      +          - awsx:x:ec2:EgressSecurityGroupRule
33      +             - aws:ec2:SecurityGroupRule
34      +          - awsx:x:ec2:IngressSecurityGroupRule
35      +             - aws:ec2:SecurityGroupRule
36      +          - awsx:x:ec2:IngressSecurityGroupRule
37      +             - aws:ec2:SecurityGroupRule
38      +          - aws:ec2:SecurityGroup
39      +       - aws:ecs:Cluster
40      +    - awsx:x:ecs:FargateService
41      +       - aws:ecs:Service
42      +    - aws:cloudwatch:LogGroup
43      +    - aws:ec2:Vpc
44      +    - awsx:x:ec2:Vpc
45      +       - awsx:x:ec2:Subnet
46      +       - awsx:x:ec2:Subnet
47      +    - aws:ec2:Subnet
48      +    - aws:ec2:InternetGateway
49      +    - aws:ec2:Subnet
50      +    - aws:ec2:SecurityGroup
51      +    - aws:ec2:RouteTable
52      +    - aws:ec2:NetworkInterface
53      +    - aws:ec2:RouteTableAssociation
54      +    - aws:ec2:Instance
55      +    - aws:ec2:RouteTable
56      +    - aws:ec2:RouteTableAssociation
57        Resources: 50 created
58    FINISHED HCS-SYS-PUBLIC - DEPLOY.SH
```

Listing B.3: HCS-DIRECT-CONNECT Deployment, Public-Cloud (2/4)

In dem Unteren Code Ausschnitt ist der dritte Teil des HCS-DIRECT-CONNECT deployments gegeben. In diesem Abschnitt wird das HCS-SYS-PLATFORM System mit Pulumi erstellt. Der Prozess wird von der 'index.ts' in Typescript geschriebenen Pulumi-Datei abgebildet.

```
1  DEPLOY-MAIN: 3. create hcs-sys-platform-cloud setup useing pulumi
2    START HCS-SYS-PLATFORM - DEPLOY.SH ...
3      Set pulumi configuration...
4      Deploy Pulumi project...
5           Type
6       +   pulumi:pulumi:Stack
7       +   - aws:iam:Role
8       +   - aws:lambda:Function
9       +   - aws:iam:RolePolicy
```

```
10      +   - aws:apigateway:x:API
11      +       - aws:apigateway:RestApi
12      +       - aws:apigateway:Deployment
13      +       - aws:lambda:Permission
14      +       - aws:apigateway:Stage
15      Resources: 9 created
16    FINISHED HCS-SYS-PLATFORM - DEPLOY.SH
```

Listing B.4: HCS-DIRECT-CONNECT Deployment, Platform (3/4)

In dem Unteren Code Ausschnitt ist der vierte Teil des HCS-DIRECT-CONNECT deployments gegeben. In diesem Abschnitt werden die erstellten IT-Ressourcen installiert und eine VPN-Verbindung aufgebaut. Der Prozess wird in dem Shell Skript 'deploy-hcs-1' beschrieben und greift auf die Ansible Playbooks 'playbook-hcs-public-install' und 'playbook-hcs-wg' zurück.

```
1  DEPLOY-MAIN: 4.1. get vm ip adresses hcs-sys-public-cloud
2  DEPLOY-MAIN: 4.2. get vm ip adresses hcs-sys-private-cloud
3    Set hosts for ansible playbook
4  DEPLOY-MAIN: 5. install hcs-sys-public-cloud gateway
5    PLAY [reqhandler]
6      TASK [Gathering Facts]
7      TASK [Install Prerequisites]
8      TASK [Disable password authentication for root]
9      TASK [Update apt]
10     TASK [Install required system packages]
11     TASK [Allow redirection of network packets at kernel level]
12     TASK [Apply changes]
13     TASK [UFW - Allow SSH connections]
14     TASK [UFW - Allow WireGuard connections]
15     TASK [Enable UFW]
16     TASK [Set the permissions for the directory]
17     TASK [Create public- and private-key for vpn connection]
18     TASK [Update read/write settings for private-key file]
19   PLAY RECAP
20     reqhandler : ok=13   changed=12
21 DEPLOY-MAIN: 6.1. get private and public keys hcs-sys-public-cloud
22 DEPLOY-MAIN: 6.2. get private and public keys hcs-sys-private-cloud
23 DEPLOY-MAIN: 7.1. create vpn-tunnel private-cloud
24 DEPLOY-MAIN: 7.2. create vpn-tunnel public-cloud (server)
25   PLAY [reqhandler]
26     TASK [Gathering Facts]
27     TASK [Disable password authentication for root]
28     TASK [copy]
```

```
29      TASK [Start WireGuard interface]
30      TASK [Start WireGuard interface by system boot]
31    PLAY RECAP
32      reqhandler : ok=5
```

Listing B.5: HCS-DIRECT-CONNECT Deployment, VPN-Connection set up (4/4)

B.4 HCS-DIRECT-CONNECT Project-Struktur

In der Abbildung B.2 ist die geplante Projektstruktur gezeigt. Dateien sind nach Dateiformat farblich hervorgehoben, Ordner sind grau. Die Projektstruktur spiegelt sich im Code Repository wieder. Das Projekt ist in vier Unterordnern gegliedert, welche jeweils ein System in einer Cloud-Umgebung spezifiziert. Im root Verzeichnis, des Projektes, wird das Deployment der IaC-Tools in den Sub-Projekten über Shell-Skripte orchestriert.

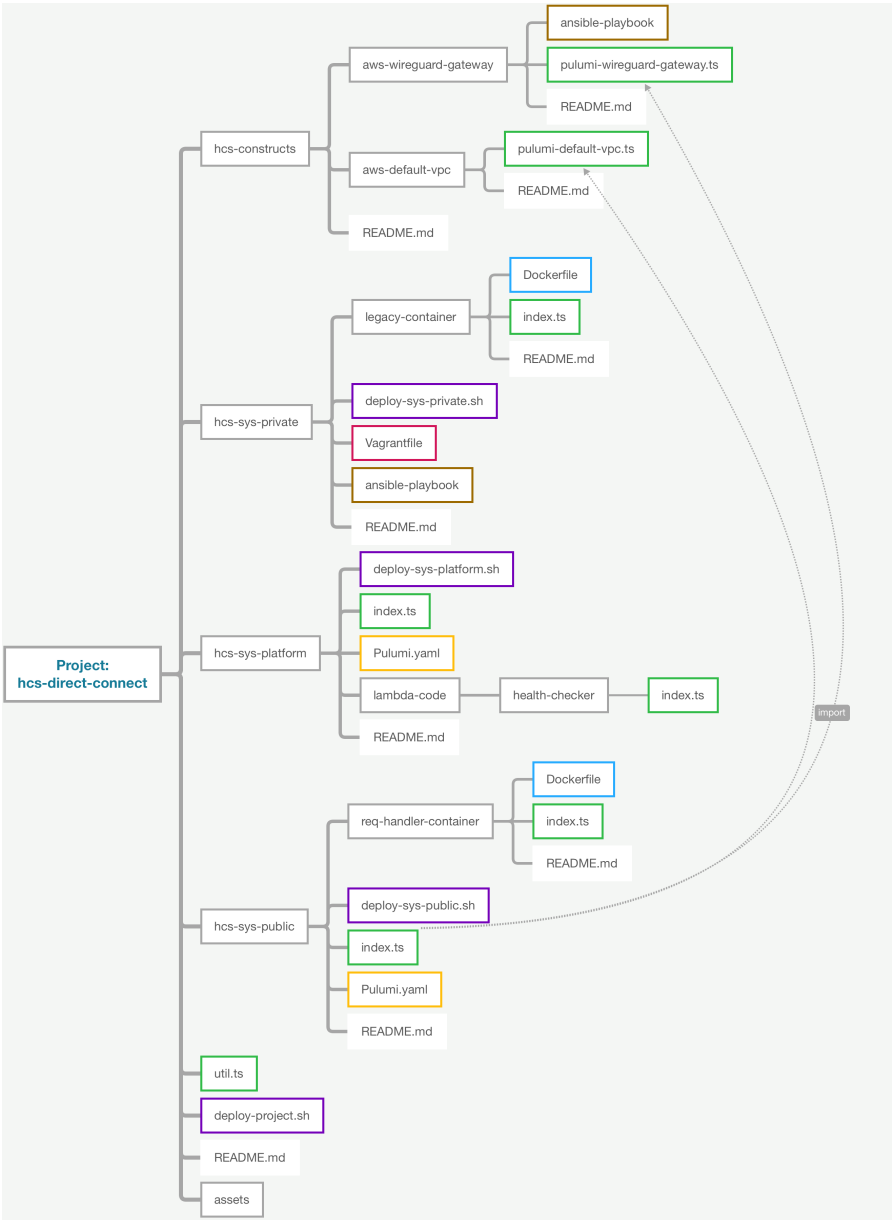


Abbildung B.2: Projekt-Struktur

C Anhang, Kapitel 6

Dieser Anhang gehört zum Kapitel 6. Innerhalb des Kapitels wird auf den Inhalt dieses Anhangs verwiesen.

C.1 Automatisierung der IaC-Tools mit Python-Skripten

In diesem Abschnitt ist beschrieben, wie eine Automatisierung von IaC-Tool Prozessen in der HCS-DIRECT-CONNECT-2 Fallstudie mit Python Skripten implementiert wurde. In der Abbildung C.1 ist eine Übersicht der dafür implementierten Klassen gegeben. In der nachfolgenden Liste werden die einzelnen Klassen beschrieben.

- **Provisioner**: Eine Abstrakte Klasse, die beschreibt, welche Instanz Variablen und Methoden benötigt werden, um ein Projekt zu verwalten. Die beiden Methoden *deploy()* und *destroy()* werden beim deployment / destruction Prozess der HCA-Strategie verwendet[1].
- **InProvisionerPriv2** & **InProvisionerPub2**: Diese Klasse definiert Projekt spezifisch Inputs[2].
- **PrivisionerPriv2** & **PrivisionerPub2**: Diese Klasse implementiert die Provisioner Klasse und implementiert daher die abstrakten Methoden.

In der Abbildung C.2 ist ein Aktivitätsdiagramm gegeben, dass den 'deployment'- und 'destruction'-Prozess des HCS-DIRECT-CONNECT-2 Systems beschreibt. Der Prozess wird von einem Terminal aus gestartet und nutzt User-Input.

[1] In dem Aktivitätsdiagramm Abbildung C.2 ist der Prozess dargetsellt.
[2] Zur Information: In der Programmiersprache 'Go' wäre dies ein Struct, in 'Python' eine herkömmliche Klasse und in 'Typescript' ein Interface

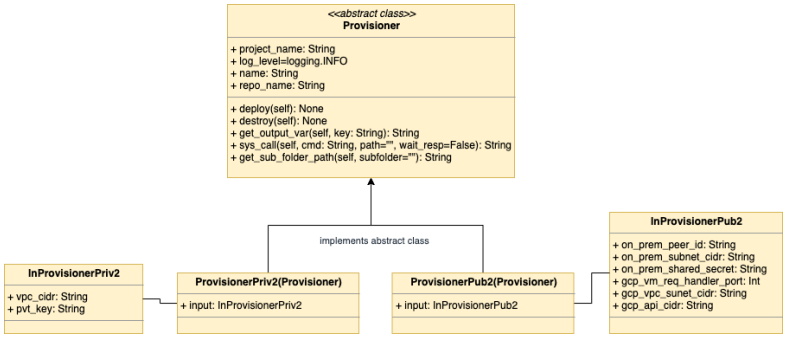


Abbildung C.1: Klassendiagramm

1. Zunächst wird der User aufgefordert auszuwählen, ob das System 'deployed' oder 'destroyed' werden soll. Beim deployment wird ein bestehendes System auch geupdated.

2. Wird das System 'Destroyed':

   - Es werden die beiden Provisioner Klassen 'ProvisionerPriv2' und ProvisionerPub2 mit leeren Instanzvariablen[3] erstellt.
   - Bei den Objekten wird die 'destroy()' Methode aufgerufen.
   - Weil HCS-SYS-PLATFORM auch teil der Applikation ist aber keinen Provisioner-Wrapper hat[4], wird das destroy.sh Skript ebenfalls aufgerufen.

3. Wird das System Deployed:

   - Zunächst werden Default-Variablen gesetzt, die von mehreren der drei Sub-Projekte genutzt werden. Nachdem teile des Gesamten-Systems erstellt wurden, werden dynamische Variablen, wie öffentliche IPv4-Adressen angefragt.
   - Das HCS-SYS-PRIVATE-2 System wird mit der 'ProvisionerPriv2' Klasse deployed.
   - Das HCS-SYS-PUBLIC-2 System wird mit der 'ProvisionerPub2' Klasse deployed.
   - Das HCS-SYS-PLATFORM System wird mit dem 'deploy.sh' Skript deployed.

[3] Einige Instanzvariablen haben einen Default-Wert, die übrigen werden mit einem leeren String initialisiert.
[4] Für dieses System könnte natürlich ebenfalls ein Provisioner Wrapper erstellt werden.

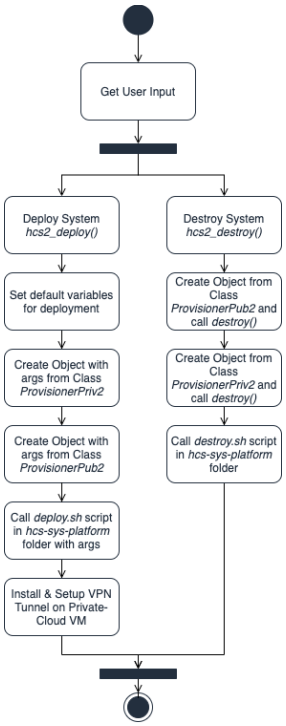


Abbildung C.2: Aktivitätsdiagramm

- Der VPN-Tunnel wird auf der Legacy-VM installiert.

Dieser Prozess ist im Code-Repository[5] in den Datein *./provision-hcs-2*, *./hcs-2/abc-provision*, *./hcs-2/provision-hcs-priv* und *./hcs-2/provision-hcs-pub* zu finden.

C.2 Deployment des HCS-SYS-DIRECT-CONNECT-2 Systems

In diesem Abschnitt wird anhand der Konsolen-Einträgen gezeigt, wie das HCS-SYS-DIRECT-CONNECT-2 System deployed wird. Da die Konsolen-Einträge sehr lang sind, werden diese zwischendurch gekürzt.

```
1 -> python provision_hcs_2.py
2
3 RUN HCS-2-PROVISION
```

[5]Das Code-Repository kann, mit einem Browser über die URL *https://github.com/leonardpahlke/hcs-direct-connect* erreicht werden

```
Select method [a, b]
- [a]: deploy the system
- [b]: destroy the system
>> a
Security file does not exists... creating a new secret

START WITH HCS-2-PRIVATE
INFO:root:START DEPLOYMENT HCS-2-PRIVATE ...
INFO:root:Set terraform config...
INFO:root:Terraform config set
INFO:root:Deploy terraform project ...

Terraform used the selected providers to generate the following execution
    plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:
  # digitalocean_droplet.legacy_vm will be created
  + resource "digitalocean_droplet" "legacy_vm" {
      + ...
  # digitalocean_firewall.legacy_vm_firewall will be created
  + resource "digitalocean_firewall" "legacy_vm_firewall" {
      + ...
  # digitalocean_floating_ip.hcs_floating_ip will be created
  + resource "digitalocean_floating_ip" "hcs_floating_ip" {
      + ...
  # digitalocean_floating_ip_assignment.hcs_floating_ip_asignment will be
    created
  + resource "digitalocean_floating_ip_assignment" "hcs_floating_ip_asignment
    " {
      + ...
  # digitalocean_vpc.vpc will be created
  + resource "digitalocean_vpc" "vpc" {
      + ...

Plan: 5 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + floating_ipv4 = (known after apply)
  + instance_ipv4 = (known after apply)
  + vpc_cidr      = "10.194.0.0/20"
digitalocean_floating_ip.hcs_floating_ip: Creating...
digitalocean_vpc.vpc: Creating...
```

```
digitalocean_vpc.vpc: Creation complete after 2s [id=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXX]
digitalocean_floating_ip.hcs_floating_ip: Creation complete after 2s [id=XXX.XX.XXX.XXX]
digitalocean_droplet.legacy_vm: Creating...
digitalocean_droplet.legacy_vm: Still creating... [10s elapsed]
digitalocean_droplet.legacy_vm: Still creating... [20s elapsed]
digitalocean_droplet.legacy_vm: Still creating... [30s elapsed]
digitalocean_droplet.legacy_vm: Creation complete after 34s [id=XXXXXXXX]
digitalocean_floating_ip_assignment.hcs_floating_ip_asignment: Creating...
digitalocean_firewall.legacy_vm_firewall: Creating...
digitalocean_firewall.legacy_vm_firewall: Creation complete after 2s [id=XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXX]
digitalocean_floating_ip_assignment.hcs_floating_ip_asignment: Still creating ... [10s elapsed]
digitalocean_floating_ip_assignment.hcs_floating_ip_asignment: Creation complete after 12s [id=XXXXXXX-XXX.XX.XXX.XXX]

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.

Outputs:

floating_ipv4 = "XXX.XX.XXX.XXX"
instance_ipv4 = "XXX.XX.XXX.XXX"
vpc_cidr = "10.194.0.0/20"
INFO:root:Terraform project deployed
INFO:root:FINISHED DEPLOYMENT HCS-2-PRIVATE
on_prem_peer_ip: XXX.XX.XXX.XXX
```

Listing C.1: "HCS-DIRECT-CONNECT-2 Deployment (1/2)"

In dem zweiten Konsolen-Auszug ist das Public-Cloud Deployment nachzuvollziehen.

```
START WITH HCS-2-PUBLIC
INFO:root:START HCS-2-PUBLIC ...
INFO:root:Set pulumi config...
INFO:root:Pulumi config set
INFO:root:Deploy pulumi project ...
Previewing update (dev)
Updating (dev)
     Type                          Name                                    Status
 pulumi:pulumi:Stack           hcs-sys-public-gcp-dev
 + gcp:compute:Address         hcs-dev-europe-west3-a-addr-vm-static
 + gcp:compute:Address         vpnstaticip
```

```
 + gcp:compute:Network          hcs-dev-europe-west3-a-vpc
 + gcp:compute:Route            hcs-dev-europe-west3-a-route-igw
 + gcp:compute:VPNGateway       hcs-dev-europe-west3-a-vpn-gw
 + gcp:compute:Firewall         hcs-dev-europe-west3-a-ig-vpn
 + gcp:compute:Subnetwork       hcs-dev-europe-west3-a-sn-vpn
 + gcp:compute:Firewall         hcs-dev-europe-west3-a-eg-all
 + gcp:compute:ForwardingRule   hcs-dev-europe-west3-a-fr-esp
 + gcp:compute:ForwardingRule   hcs-dev-europe-west3-a-fr-udp-500
 + gcp:compute:ForwardingRule   hcs-dev-europe-west3-a-fr-udp-4500
 + gcp:compute:Instance         hcs-dev-europe-west3-a-vm-req-han
 + gcp:compute:VPNTunnel        hcs-dev-europe-west3-a-vpn-tunnel
 + gcp:compute:Route            hcs-dev-europe-west3-a-route-onprem
Outputs:
    instance_name: "hcs-dev-europe-west3-a-vm-req-han"
    instance_zone: "europe-west3-a"
    project_name : "hcs-sys-direct-connect"
    tunnel_ip    : "XX.XXX.XXX.XX"
Resources:
    + 15 created

INFO:root:Pulumi project deployed
INFO:root:FINISHED HCS-2-PUBLIC
```

Listing C.2: "HCS-DIRECT-CONNECT-2 Deployment (2/2)"

C.3 Erstellte IT-Ressourcen HCS-2

C.3.1 Pulumi-Stack des GCP Projektes HCS-SYS-PUBLIC-2

In dem nachfolgenden Ausschnitt ist der Pulumi-Stack Baum des HCS-SYS-PUBLIC-2 Systems zu sehen. Der Pulumi-Stack zeigt die Services, die erstellt werden, um das HCS-SYS-PUBLIC-2 System in GCP zu erstellen. Der Code zum Pulumi-Stack findet sich im Code Repository[6] in der Datei *hcs/hcs-2/hcs-sys-public-2/main* wieder.

```
pulumi:pulumi:Stack
- gcp:compute:Network
- gcp:compute:Address
- gcp:compute:Route
- gcp:compute:VPNGateway
```

[6]Das Code Repository kann, mit einem Browser über die URL *https://github.com/leonardpahlke/hcs-direct-connect* erreicht werden

```
- gcp:compute:Subnetwork
- gcp:compute:Firewall
- gcp:compute:Firewall
- gcp:compute:ForwardingRule
- gcp:compute:ForwardingRule
- gcp:compute:ForwardingRule
- gcp:compute:Instance
- gcp:compute:VPNTunnel
- gcp:compute:Route
```

Listing C.3: HCS-DIRECT-CONNECT-2 GCP Services des Pulumi-Stacks

C.3.2 Terraform-Stack des DO Projektes HCS-SYS-PRIVATE-2

In dem nachfolgenden Ausschnitt ist der Terraform-Stack Baum des HCS-SYS-PRIVATE-2 Systems zu sehen. Der Terraform-Stack zeigt die Services, die erstellt werden, um das HCS-SYS-PRIVATE-2 System in DigitalOcean zu erstellen. Der Code zum Terraform-Stack findet sich im Code Repository[7] in der Datei *hcs/hcs-2/hcs-sys-private-2/main* wieder.

```
Terraform-Resource-Stack
- digitalocean_droplet
- digitalocean_firewall
- digitalocean_floating_ip
- digitalocean_floating_ip_assignment
- digitalocean_vpc
```

Listing C.4: HCS-DIRECT-CONNECT-2 DigitalOcean Services des Terraform-Stacks

[7] Das Code Repository kann, mit einem Browser über die URL *https://github.com/leonardpahlke/hcs-direct-connect* erreicht werden

D Anhang, Kapitel 7

Dieser Anhang gehört zum Kapitel 7. Innerhalb des Kapitels wird auf den Inhalt dieses Anhangs verwiesen.

D.1 Project-Struktur - HCS-1

Wird die geplante Projektstruktur mit der erstellten Projektstruktur verglichen, so sind leichte Abweichung zu erkennen (siehe geplante Projektstruktur Abbildung B.2). Grüne Knoten wurden hinzugefügt und waren nicht geplant, graue Knoten waren geplant und rote Knoten waren geplant, wurden aber nicht umgesetzt. In der Abbildung D.1 ist die Projektstruktur zu sehen, die nach dem Abschluss des Projektes erstellt wurde. Die hier gezeigte Projekt-Struktur spiegelt nicht eins-zu-eins das Code-Repository wieder, da in diesem auch die Ergänzung der Fallstudie untergekommen ist, hier aber nicht aufgenommen wurde, um den Vergleich sauberer darzustellen. In der nachfolgenden Liste sind die wichtigsten Unterschiede aufgefasst.

- 'hcs-constructs' war als Ordner eingeplant, der Einstellungen und Konfigurationen beinhaltet, die wiederverwendet werden können. Wie sich herausstellt, sind keine Bestandteile wiederverwertet worden. Wäre das erste Design implementiert worden, so wäre vermutlich dieses genutzt.
- Neben dem 'deploy.sh' Skript, welches in jedem Subprojekt geplant war, ist ein 'destroy.sh' Konter hinzugefügt worden, der das deployment rückgängig macht.
- Die Logik der beiden erstellten Container in mehreren Dateien gefächert, um eine bessere Code-Qualität zu erzeugen.
- Im 'Root-Verzeichnis' sind weitere Shell-Skripte zum Automatisieren hinzugekommen.

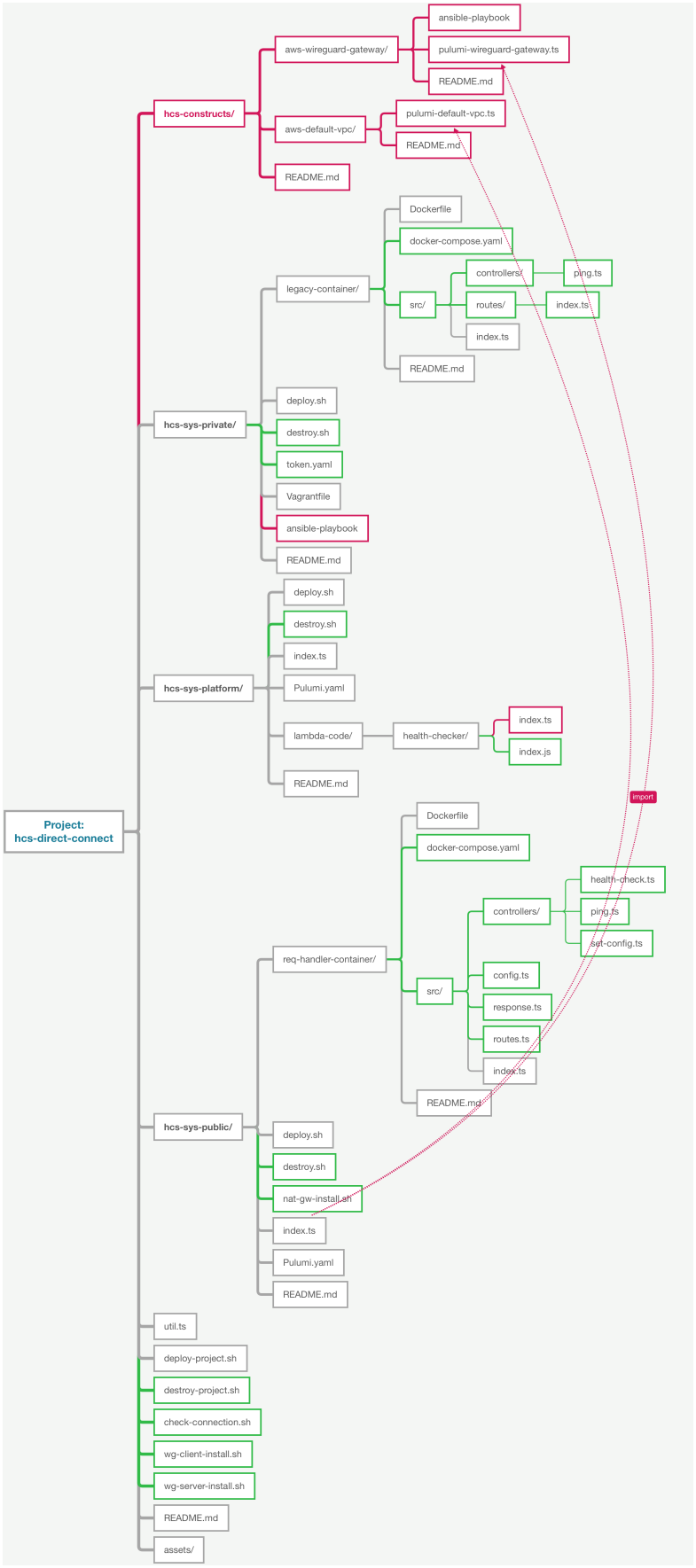


Abbildung D.1: Abweichungen in der Projektstruktur

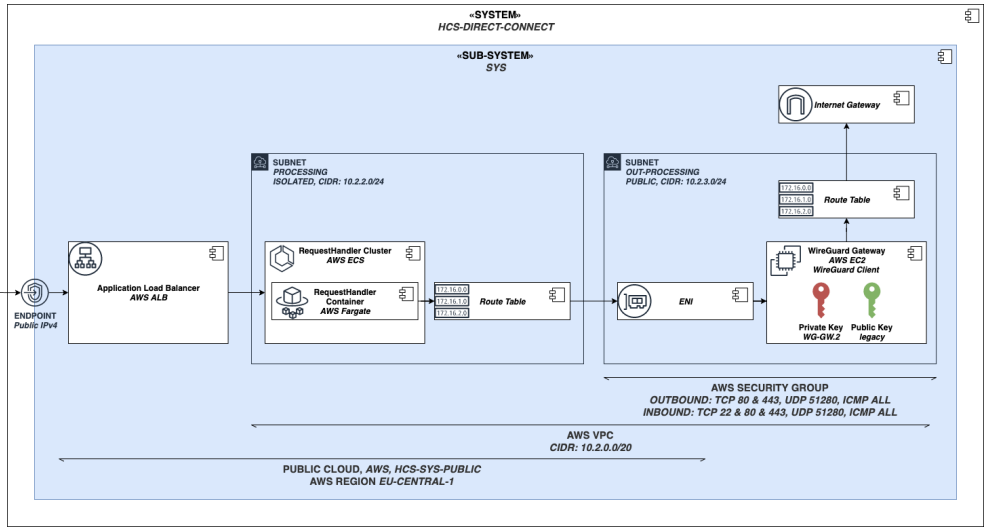


Abbildung D.2: Verteilungssicht - AWS Netzwerk

- Installiere Public-Cloud VM
- Konfiguriere WireGuard auf der Public-Cloud VM
- Konfiguriere WireGuard auf der Private-Cloud VM

Es ist zu erkennen, dass die geplante Projekt-Struktur oberflächlich eingehalten wurde. Allerdings ist in den Punkten 'Automatisierung zwischen den IaC-Tools' und 'Projekt-Struktur Konventionen' von bspw. ExpressJS-Containern einige Abweichungen zu beobachten. Diese Abweichungen führt sich auf die Beschreibungsoffenen Formulierungen im Anforderungen- und Design-Abschnitt zurück. Abweichungen in diesem Ausmaß stellen die Regel dar.

D.2 AWS VPC Aufbau - HCS-1

Im HCS-1 Projekt musste die Public-Cloud Konfiguration umfangreicher als geplant aufgesetzt werden. Der VPC Netzwerk Aufbau ist in der Typescript-Datei ('hcs-1/hcs-sys-public/index.ts') beschrieben und nutzt dafür Pulumi als IaC-Tool. Die Unterschiede sind in der Abbildung D.2 grafisch dargestellt und werden in der nachfolgenden Liste durchgegangen.

- Route-Table: Dieser Service wurde in beiden Subnets einmal konfiguriert.

1. Private-Subnet (Fargate): Outbound Traffic muss an das Public-Subnet geschickt werden. Konfiguration: 'cidrBlock: {"0.0.0.0/0"→ natInstance.id}'.
2. Public-Subnet (NAT-Gateway): Outbound Traffic wird an das Internet Gateway weitergeleitet. Die Instanz arbeitet demnach als NAT-Gateway (Network-Address-Translation Gateway). Konfiguration: 'cidrBlock: {"0.0.0.0/0"→ internetGateway.id}'.

- Internet-Gateway: Dieser Service musste konfiguriert werden, damit die NAT-Gateway-VM eine Public-Ip zugewiesen bekommen kann.
- Security-Group: Dieser Service konfiguriert, welche Ports und Protokolle für hinterlegte Services freigeschalten werden.

Beim Design ist immer eine Abweichung zu treffen, welches Level an Detail beschrieben wird. Da dieses Projekt viele IT-Ressourcen in den Blick nimmt, die meist nur leicht konfiguriert werden müssen, wird meist relativ abstrakt die IT-Infrastruktur beschrieben. Die hier ausgeblendeten Services, werden automatisch erstellt, wenn 'normale' Konfigurationen getroffen werden. Da dieser Anwendungsfall jedoch vom normalen VPC set-up abweicht, müssen diese Services anwendungsspezifisch konfiguriert werden. Mehr Detail an dieser Stelle ist daher sinnvoll.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Automatisierungsstrategien in der Hybrid Cloud

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original