

Bachelorthesis

im Studiengang

Informatik

zur Erlangung des akademischen Grads

Bachelor of Science (B.Sc.)

Thema:

Konzeption eines Prototypen zur automatisierten Abfrage der Preis-API von public Cloud-Anbietern als Grundlage für Sky Computing

eingereicht von:

Vito D'Elia

geb. am 27.07.2000 in Frankfurt am Main

Adalbertstrasse 37, 60486, Frankfurt am Main

Matrikel-Nr: 1320417

Im Sommersemester 2024

am Fachbereich 2: Informatik und Ingenieurwissenschaften

der Frankfurt University of Applied Sciences

Erstprüfer: Henry-Norbert Cocos

Zweitprüfer: Prof. Dr. Christian Baun

Abgabedatum: 13.08.2024

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher Form noch keiner Prüfbehörde vorgelegen.

Ort, Datum: Frankfurt am Main, 13.08.2024

Unterschrift:

A handwritten signature in blue ink, consisting of a large 'D' followed by a series of loops and a long horizontal stroke.

Danksagung

Hiermit möchte ich bei all denjenigen danken, die mir das Studium ermöglicht und mich in dieser Zeit begleitet haben.

Zunächst gilt mein Dank Herrn Henry-Norbert Cocos, der sich bereit erklärt hat, als Hauptreferent zur Verfügung zu stehen. Zusätzlich bedanke ich mich für die Unterstützung bei der Themenwahl sowie die Begleitung während der Ausarbeitung. Weiterhin möchte ich mich hiermit bei Herrn Prof. Dr. Christian Baun bedanken, der sich als Korreferent zur Verfügung gestellt hat.

Insbesondere möchte ich meiner Familie, meiner Freundin und meinen engsten Freunden danken, die mich auf diesem Weg begleitet und immer tatkräftig unterstützt haben.

Zusammenfassung

Konzeption eines Prototypen zur automatisierten Abfrage der Preis-API von public Cloud-Anbietern als Grundlage für Sky Computing

Diese Arbeit beschreibt den Prozess der Konzeption eines Prototypen zur automatisierten Abfrage der Preis-API von public Cloud-Anbietern. Die Preislisten der gewählten Cloud-Anbieter werden stündlich mittels einer Go-Applikation abgefragt. Die abgerufenen Datensätze werden anschließend verarbeitet, abgespeichert und in tabellarischer Form über eine lokale Web-Applikation zur Verfügung gestellt.

Abstract

Konzeption eines Prototypen zur automatisierten Abfrage der Preis-API von public Cloud-Anbietern als Grundlage für Sky Computing

This bachelor thesis describes the process of designing a prototype for the automated querying of public cloud providers pricing APIs. The pricing lists of the selected cloud providers are queried hourly using a Go application. The retrieved records are then processed, stored, and made available in tabular form through a local web application.

Inhaltsverzeichnis

1	Einleitung und Motivation	1
1.1	Fragestellung und Herausforderungen	2
2	Grundlagen	4
2.1	Representational State Transfer.....	4
2.2	Application Programming Interface	4
2.3	RESTful-API.....	5
2.4	JavaScript Object Notation.....	6
3	Stand der Technik	7
3.1	Existierende Lösungen	7
3.1.1	Cloudorado.....	7
3.1.2	Holori	9
3.2	Cloud Computing	10
3.2.1	Cloud-Bereitstellungsmodelle.....	11
3.2.2	Cloud-Service-Modelle.....	12
3.2.3	Multi-Cloud.....	14
3.3	Sky Computing.....	15
3.3.1	Forderungen	15
3.3.2	Kompatibilitätsschicht (Compatibility Layer)	16
3.3.3	Intercloud-Schicht (Intercloud-Layer)	16
3.3.4	Peering-Vereinbarung (Peer agreement).....	18
3.3.5	Offene Herausforderungen.....	19
4	Design	20
4.1	Anforderungsanalyse.....	20
4.2	Programmiersprache.....	22
4.3	Ausgewählte Cloud-Anbieter und Dienste.....	23
4.3.1	Cloud-Computing-Dienste.....	23
4.3.2	Cloud-Objektspeicherdienste.....	24
4.3.3	Cloud-Blockspeicherdienste	26
4.3.4	Cloud-Datenbankdienste.....	27
4.3.5	Vergleich zu Microsoft Azure und Google Cloud Platform	28

4.4	Preis-APIs.....	28
4.4.1	AWS Price List Bulk API.....	29
4.4.2	Azure Retail Prices REST API	30
4.4.3	Google Cloud Billing API	30
4.4.4	Preiskalkulator-API.....	31
4.5	Konzeption der Datenerlangung.....	32
4.6	Konzeption der Datenverarbeitung	33
4.7	Konzeption der Datenaufbereitung	35
5	Implementierung	36
5.1	Datenerlangung	36
5.1.1	AWS-Kommunikation	36
5.1.2	MSA-Kommunikation	42
5.1.3	GCP-Kommunikation	44
5.2	Datenverarbeitung	49
5.2.1	Einheitliche Datenstruktur	49
5.2.2	Speicherungsprozess	50
5.3	Datenaufbereitung	54
5.4	Automatisierung	57
6	Validierung	58
6.1	Web-Applikation	58
6.2	Testlauf.....	61
7	Fazit	64
7.1	Ausblick.....	65
	Literatur	66

Abbildungsverzeichnis

Abbildung 2.1: REST-API Model.....	5
Abbildung 3.1: Clouddorado Computing-Vergleich	8
Abbildung 3.2: Holori Storage-Vergleich	9
Abbildung 3.3: Cloud-Bereitstellungsmodelle	12
Abbildung 3.4: Cloud-Service-Modelle	13
Abbildung 3.5: Mögliche Sky Computing Architektur	18
Abbildung 4.1: S3 Bucket System.....	25
Abbildung 4.2: Ablauf der Datenerlangung	32
Abbildung 4.3: Ablauf der Datenverarbeitung	34
Abbildung 4.4: Entwurf der Benutzeroberfläche	35
Abbildung 5.1: Datenbankschema.....	50
Abbildung 5.2: Datenbankverbindung	52
Abbildung 5.3: Datenflussdiagramm.....	56
Abbildung 6.1: Filterung eines AWS-Speicherdienstes	58
Abbildung 6.2: Filterung einer SQL-Ressource der GCP	59
Abbildung 6.3: Zeitlicher Aufwand einer Aktualisierung.....	61

Tabellenverzeichnis

Tabelle 3.1: Internet und Sky im Vergleich	15
Tabelle 3.2: Kompatibilität durch freie Software.....	16
Tabelle 4.1: EC2-Instanz-Typen	24
Tabelle 4.2: Dienstnamen im Vergleich	28
Tabelle 5.1: Anbieter-Tabelle.....	50
Tabelle 5.2: Dienst-Tabelle	51
Tabelle 5.3: Dienst-Typ-Tabelle.....	51
Tabelle 6.1: Ergebnis des Testlaufs	63

Quellcodeverzeichnis

Listing 2.1: Beispiel einer JSON-Struktur.....	6
Listing 5.1: AWS URL-Map	36
Listing 5.2: Grundstruktur der AWS-Preisliste	37
Listing 5.3: Struktur des AWS-Produkt-Objekts.....	38
Listing 5.4: Struktur des AWS-Terms-Objekts	39
Listing 5.5: Go-Struktur für die AWS-Preis-API.....	40
Listing 5.6: Prozedur zur Einlesung der Daten.....	41
Listing 5.7: MSA-URL-Strings	42
Listing 5.8: Grundstruktur der MSA-Preisliste	42
Listing 5.9: Struktur des MSA-Items-Objekts.....	43
Listing 5.10: Go-Struktur für die MSA-Preis-API	44
Listing 5.11: GCP-URL-Strings	45
Listing 5.12: Grundstruktur der GCP-Preis-API	46
Listing 5.13: Struktur des GCP-PricingInfo-Objekts	47
Listing 5.14: Go-Struktur für die GCP-Preis-API	48
Listing 5.15: Vereinheitlichte Go-Struktur.....	49
Listing 5.16: Sammlung der existierenden Instanzen.....	52
Listing 5.17: Iteration zur Speicherung der Instanzen.....	53
Listing 5.18: Go-Struktur im Frontend.....	54
Listing 5.19: Aufbau der Tabelle.....	55
Listing 5.20: Automatisierungsprozess	57
Listing 6.1: AWS-Speicherdienst-URL.....	59
Listing 6.2: INSERT-Query	60

Abkürzungsverzeichnis

AMI	Amazon Machine Image
API.....	Application Programming Interface
AWS	Amazon Web Services
BSI.....	Bundesamt für Sicherheit in der Informationstechnik
CSS	Cascading Style Sheets
DNS	Domain Name System
GCP	Google Cloud Platform
GORM	Golang Object Relational Mapping
HDD	Hard Disk Drive
HPC	High Performance Computing
HTML.....	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS.....	Infrastructure as a Service
JSON.....	JavaScript Object Notation
KB.....	Kilobyte
MB.....	Megabyte
MSA	Microsoft Azure
PaaS	Platform as a Service
RDP	Remote-Desktop-Protokoll
REST	Representational State Transfer
SaaS	Software as a Service
SKU	Stock Keeping Unit
SSD.....	Solid State Drive
SSH.....	Secure Shell
URL	Uniform Resource Locator

1 Einleitung und Motivation

Cloud Computing erlaubt die Bereitstellung und Nutzung von IT-Infrastrukturen, Plattformen und jegliche Arten von Anwendungen, die über das Internet erreichbar sind [34]. Diese können unter dem Begriff Cloud-Ressourcen zusammengefasst werden. Diese Ressourcen sind in der Regel virtualisiert, wodurch problematische Aspekte wie etwa die Systemabhängigkeit einer Plattform keine Auswirkungen haben. Darüber hinaus sind diese Cloud-Dienste dynamisch skalierbar. Sollte eine Anwendung mehr Ressourcen benötigen, können diese ohne großen Aufwand automatisiert angefügt werden [34]. Dabei müssen nur die Ressourcen gezahlt werden, die tatsächlich genutzt werden. Diese flexible Bereitstellung kann zwar zur Kosteneinsparungen führen, birgt jedoch langfristig potenzielle Herausforderungen.

Ein zentrales Problem des Cloud Computing ist die zunehmende Unübersichtlichkeit der von verschiedenen Anbietern angebotenen Dienste. Von hoher Bedeutung hierbei ist die Analyse der anfallenden Kosten. Die Kosten der einzelnen Dienste müssen bislang konsolidiert und zentral analysiert werden, damit der Nutzer einen Überblick über die Gesamtkosten erhält. Darüber hinaus verändern sich die Preise der Dienste dynamisch. Diese Preisänderungen sind nur grob abschätzbar, wodurch wir zu einem schwerwiegenden Problem gelangen. Daher ist es für alle Nutzer von entscheidender Bedeutung, sowohl die aktuellen Preise als auch deren Entwicklung genau zu verfolgen. Diese Herausforderung könnte durch den nächsten Entwicklungsschritt im Bereich des Cloud Computing bewältigt werden.

Sky Computing ist ein neu erforschter Aspekt des Cloud Computing. Das Ziel ist eine Interoperabilität zwischen verschiedenen Cloud-Anbietern und deren Diensten zu ermöglichen. Daraus soll ein Netzwerk aus mehreren öffentlichen Clouds resultieren, die intern miteinander agieren, dem sogenannten Sky. Die variierenden Preise müssen daher erfasst und für eine Analyse bereitgestellt werden. Diese Analyse soll dem Nutzer einen präzisen Überblick auf die Preisdynamik der Dienste bieten und einen Vergleich zwischen den verschiedenen Anbietern ermöglichen. Die erforderlichen Preisinformationen müssen über geeignete Schnittstellen der verschiedenen Anbieter angefragt werden.

Dieser Abruf muss programmatisch erfolgen, um eine Automatisierung dieses Prozesses zu ermöglichen. Die erhaltenen Daten müssen durch mehrere Prozeduren in ein einheitliches Format gebracht und anschließend persistent in einer Datenbank abgespeichert werden.

1.1 Fragestellung und Herausforderungen

Die Anzahl der Anbieter für Cloud-Dienste ist groß und wächst kontinuierlich. Die Auswahl des passenden Anbieters ist ohne umfassende Recherche nur schwer zu treffen. Welcher Anbieter der richtige ist, hängt von den individuellen Anforderungen des Kunden ab. Zusätzlich sind die Kosten der Dienste variabel und insbesondere bei langfristiger Nutzung sind Preisänderungen unvermeidlich. Um die Benutzer bei ihrer Auswahl zu unterstützen, sollen daher die Preise der verschiedenen Anbieter für ausgewählte Dienste kontinuierlich abgefragt und für die zukünftige Nutzung weiterverarbeitet werden.

Derzeit ist noch unklar, ob bereits Preis-APIs für die verschiedenen Anbieter existieren. Es ist davon auszugehen, dass diese APIs unabhängig voneinander agieren, wodurch keine einheitliche Schnittstelle zur Verfügung steht. Sollte dies der Fall sein, ist eine gründliche Recherche der entsprechenden Dokumentationen erforderlich. Zudem kann die Implementierung der verschiedenen Schnittstellen äußerst zeitaufwändig sein. Der weitaus schwierigere Fall tritt ein, wenn überhaupt keine Schnittstellen vorhanden sind. In diesem Fall wäre die Entwicklung eines alternativen Konzepts erforderlich, um den Zugang zu den benötigten Daten für diesen Prototypen zu erhalten. Nun stellt sich die Frage, ob Sky Computing angesichts der grundlegenden Komplexität einen signifikanten Vorteil bieten kann. Denn die Sammlung und Verarbeitung der Preisinformationen beansprucht ebenfalls Ressourcen, die einkalkuliert werden müssen.

Bislang existieren nur wenige Internetseiten, die eine umfassende Vergleichbarkeit der verschiedenen Cloud-Anbieter ermöglicht. Die Sammlung der Preisinformationen stellt dabei den ersten Schritt dar und könnte in Zukunft als Grundlage für die Entwicklung des Sky Computing dienen. Nun stellt sich die Frage, ob ist die automatisierte Abfrage dieser Preisinformationen durch die Kommunikation mit den Preis-APIs der verschiedenen öffentlichen Cloud-Anbieter realisierbar ist.

Die Datensätze der Preis-APIs sollen in regelmäßigen Abständen automatisiert abgefragt und gespeichert werden. Darüber hinaus soll die Möglichkeit geschaffen werden, diese Preisinformationen über eine Web-Applikation zu analysieren.

Im Verlauf dieser Arbeit wird zunächst ein Überblick über die grundlegenden Technologien gegeben, die von zentraler Bedeutung für das Verständnis dieser Thematik sind. Anschließend wird der aktuelle Stand der Technik vorgestellt, der bereits bestehende Lösungen näher beschreibt und die Konzepte des Cloud Computing und Sky Computing erläutert. Im darauffolgenden Kapitel wird das Design der vorgestellten Konzeption detailliert dargestellt, gefolgt von einer Beschreibung der wesentlichen Entwicklungsschritte des Prototypen. Den Abschluss dieser Arbeit bildet eine Validierung dieser Implementierung, in der geprüft wird, ob die gestellten Anforderungen erfüllt wurden. Abschließend werden die Ergebnisse dieser Arbeit in einem Fazit zusammengefasst und ein Ausblick auf potenzielle zukünftige Entwicklungen gegeben.

2 Grundlagen

Um die Bedeutung dieser Thematik einordnen zu können, ist das grundlegende Verständnis einiger Technologien notwendig. Aus diesem Grund bietet dieses Kapitel eine Übersicht über die verwendeten Kommunikationssysteme, die eine automatische Abfrage der Preis-APIs ermöglichen.

2.1 Representational State Transfer

Representational State Transfer (REST) ist ein Architekturstil für die Kommunikation zwischen zwei Computersystemen, wobei diese die Rollen von Client und Server annehmen. Die REST-Kommunikation besteht auf der Verwendung von HTTP-Methoden (GET, POST, PUT, DELETE). Der Client sendet eine HTTP-Anfrage an die URL des Servers. Der Server verarbeitet diese Anfrage und sendet dem Client eine HTTP-Antwort zurück, die bei Bedarf zusätzliche Daten enthalten kann. Die HTTP-Antwort enthält einen Statuscode, der den Client über den Erfolg beziehungsweise Misserfolg seiner Anfrage informiert. Eine GET-Anfrage gilt als eine sichere HTTP-Methode, da sie lediglich Informationen anfordert. Das Versenden einer GET-Anfrage sollte denselben Einfluss auf eine Ressource haben, wie wenn keine Anfrage gesendet worden wäre. Das bedeutet, dass eine GET-Anfrage keine Veränderungen an der Ressource verursachen sollte [31].

2.2 Application Programming Interface

Ein Application Programming Interface (API) ist eine Schnittstelle, die eine Kommunikation und den Datentransfer zwischen unabhängigen Anwendungen ermöglicht. Eine API umfasst eine Sammlung von Definitionen und Protokollen, die die Interaktion zwischen den Systemen regeln. Die Entwickler solcher Schnittstellen bieten häufig Bibliotheken für verschiedene Programmiersprachen an. Diese Bibliotheken enthalten eine Reihe von Methoden, die aufgerufen werden können, um die Funktionalitäten der API vereinfacht zu nutzen.

2.3 RESTful-API

Eine RESTful-API ist eine API, die den Architekturstil von REST verwendet. Die Kommunikation zwischen Client und Server funktioniert über die Nutzung der standardisierten HTTP-Methoden. Diese Methoden werden genutzt, um Operationen auf spezifizierte Ressourcen innerhalb des Servers auszuführen:

- **GET:** Abrufen von Ressourcen
- **POST:** Erstellung einer neuen Ressource mit den gesendeten Daten
- **PUT:** Ersetze die Ressource mit den gesendeten Daten
- **DELETE:** Lösche die Ressource

Der Server empfängt die Anfrage des Clients und verarbeitet diese entsprechend. Abhängig von der Art der Anfrage sendet die REST API die angeforderten Informationen zurück an den Client. Abbildung 2.1 veranschaulicht dieses Verfahren. Der Client wählt eine der vier beschriebenen HTTP-Methoden. Die API verarbeitet die Anfrage und führt auf der Serverseite die entsprechende Operation aus. Nach Abschluss der Verarbeitung erhält der Client eine HTTP-Antwort zurück.

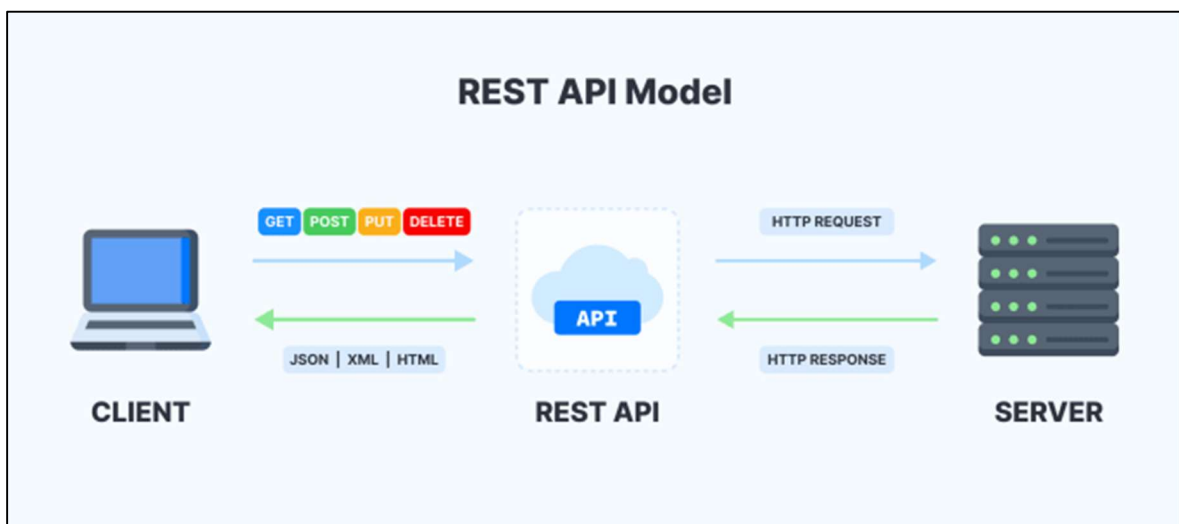


Abbildung 2.1: REST-API Model

Quelle: <https://hoanlk.com/en/http-rest-api-with-automation>

2.4 JavaScript Object Notation

Die JavaScript Object Notation (JSON) ist ein weit verbreitetes Datenaustauschformat. Es handelt sich um ein textbasiertes Format, das unabhängig von spezifischen Programmiersprachen verwendet werden kann. Diese Unabhängigkeit ermöglicht es, universelle Datenstrukturen zu erstellen, die von nahezu allen modernen Programmiersprachen unterstützt werden. Darüber hinaus eignet sich JSON auch für die Speicherung von Datensätzen. Dieses Austauschformat wird häufig in Web-Anwendungen und in API-Kommunikationen eingesetzt, wodurch es für diese Arbeit besondere Relevanz besitzt. JSON-Strukturen bestehen aus Schlüssel-Wert-Paaren, die in geschweiftem Klammern eingeschlossen sind. Die Schlüssel auf der linken Seite bestehen aus Strings, die den Namen des jeweiligen Datenfelds darstellen. Die Werte auf der rechten Seite repräsentieren die zugehörigen Daten. Diese können einfache Datentypen wie Strings, Zahlen oder Booleans sein, aber auch komplexe Typen wie Arrays oder verschachtelte Objekte. In Listing 2.1 ist eine beispielhafte Struktur dargestellt. Hier kann das erste Datenfeld über den Schlüssel `anbieter` angesprochen werden, um den Wert „Amazon Web Services“ abzurufen.

```
{
  "anbieter": "Amazon Web Services",
  "dienst": "Amazon EC2",
  "id": 123456789,
  "preisInformationen": [
    {
      "währung": "Euro",
      "einheit": "Stunde",
      "preis": 0.00123,
    }
  ]
}
```

Listing 2.1: Beispiel einer JSON-Struktur

3 Stand der Technik

In diesem Kapitel werden die Kriterien für die Umsetzung des angestrebten Prototyps beschrieben und bereits existierende Lösungen für vergleichbare Aufgaben untersucht. Anschließend wird das Konzept des Cloud Computing beschrieben, um darauf aufbauend die Notwendigkeit der Forschung im Bereich des Sky Computing zu begründen.

3.1 Existierende Lösungen

Dieser Abschnitt gibt einen Überblick über bereits existierende Lösungen, die einen Preisvergleich zwischen verschiedenen Cloud-Anbietern ermöglichen. Dabei wird untersucht, inwieweit diese Lösungen die Anforderungen an eine effektive Vergleichbarkeit der Cloud-Preise erfüllen.

3.1.1 Cloudorado

Cloudorado [1] ist ein Web-Portal, welches den Preisvergleich zwischen verschiedenen Cloud-Anbietern ermöglicht. Dabei werden die folgenden Cloud-Anbieter in den Vergleich miteinbezogen:

- Amazon Web Services (AWS)
- Microsoft Azure (MSA)
- Google Cloud Platform (GCP)
- Kamatera
- M5
- ZettaGrid
- Exoscale
- CloudWare
- e24cloud.com

Dabei spezialisiert sich Cloudorado auf den Preisvergleich von drei Diensten. An erster Stelle wird ein Vergleich von Cloud-Server-Diensten angeboten. Daraufhin können auch die Cloud-Hosting-Dienste der gelisteten Anbieter verglichen werden. Zuletzt ist auch der Vergleich von Cloud-Storage-Diensten möglich.

Beim Preisvergleich können die verschiedensten Konfigurationen angegeben werden, die dann einen neu berechneten Preis bestimmen. In Abbildung 3.1 sind die Konfigurationsmöglichkeiten für den Vergleich von Cloud-Server-Diensten erkennbar. So kann die CPU-Kern Anzahl, die Arbeitsspeichergröße, die Festplattenspeichergröße und das Betriebssystem gewählt werden. Auch für die restlichen Dienste sind unterschiedliche Konfigurationen einstellbar. Cloudorado bietet den Endnutzern einen kostenlosen Preisvergleich zwischen den verschiedenen Cloud-Anbietern.

The screenshot shows the Cloudorado configuration interface. At the top, there are four sliders for configuration: RAM (ranging from 512 to 64G, with 1G selected), Storage (ranging from 1GB to 1TB, with 50 selected), CPU cores (ranging from 1x to 64x, with 1x selected), and OS (with Linux selected and Windows as an option). A 'More options' button is located to the right of the OS slider. Below the sliders, an orange banner states '9 cloud server providers found'. Underneath is a table with three columns: 'Cloud Provider', 'Cloud Server Summary', and 'Price per month'.

Cloud Provider	Cloud Server Summary	Price per month
KAMATERA PERFORMANCE CLOUD	1 GB RAM / 1x CPU Instance show details	\$13 Go to Provider
M5	M5 Small (2 GB RAM, 1 CPU) show details	\$14 Go to Provider

Abbildung 3.1: Cloudorado Computing-Vergleich Quelle: [1]

Der Unterschied zwischen Cloudorado und diesem Prototypen ist die Preisdarstellung. Cloudorado gibt einen festen Betrag aus, welches den Benutzern eine simple Darstellung des Preises anbietet. Damit einhergehend ist jedoch die mangelnde Nachvollziehbarkeit der Preise. Zusätzlich führt der Knopf neben der Preisangabe zur Anmeldeseite des jeweiligen Anbieters. Im Umkehrschluss muss auf der Anbieterseite nach der genauen Ressource

gesucht werden, ohne dass eine Artikelnummer oder andere spezifische Informationen vorliegen. Ein weiterer Unterschied ist die Nutzbarkeit der Daten. Im Gegensatz zu Cloudorado soll der Prototyp den Preis jeder einzelnen Ressource angeben. Dies erfordert eine tiefere Untersuchung der Angebote, jedoch sind die gewählten Ressourcen durch die Angabe der Artikelnummer sofort auffindbar. Durch die automatisierte Abfrage der Preis-APIs wird sichergestellt, dass die Preise jederzeit aktuell sind. Diese Gewährleistung kann bei der Nutzung von Cloudorado aus externer Perspektive nicht in gleicher Weise gewährleistet werden.

3.1.2 Holori

Holori [41] ist ein weiteres Vergleichsportal, das Nutzern einen Preisvergleich zwischen verschiedenen Cloud-Anbietern und deren Diensten ermöglicht. Dabei unterstützt es eine Liste von insgesamt 12 öffentlichen Cloud-Anbietern. Ähnlich wie Cloudorado können hierbei drei Diensttypen miteinander verglichen werden: Computing-Dienste, Storage-Dienste und Netzwerk-Dienste. Die Daten können nach verschiedenen Attributen gefiltert werden. In der Abbildung 3.2 ist das Ergebnis einer Filterung von Storage-Diensten nach dem Anbieter Amazon Web Services zu sehen. Hierbei fehlen die Werte zu den Attributen Größe, Input-, Output-Operationen und Bandbreite. Zudem beträgt die Verarbeitungszeit nach einer einzelnen Filterung etwa 30 Sekunden. Dies kann bei komplexeren Datenanalysen sehr zeitaufwendig sein.







951 offers found						
Name		Size	IO	Bandwidth	Location	Price ↑
 standard		0 GB	0	0 MB/s		0,06545 \$
 standard		0 GB	0	0 MB/s		0,05 \$
 gp2		0 GB	0	0 MB/s		0,12 \$

Abbildung 3.2: Holori Storage-Vergleich Quelle: [41]

3.2 Cloud Computing

Cloud Computing beschreibt die Bereitstellung von IT-Ressourcen über das Internet, auch bekannt als Cloud. In der Regel werden diese Dienstleistungen von großen Unternehmen wie Amazon, Microsoft oder Google angeboten. Die Angebote der Cloud-Anbieter sind oft proprietär und auf Wettbewerbsvorteile ausgerichtet, wodurch eine standardisierte Lösung fehlt und ein schneller Wechsel zwischen den Anbietern erschwert wird [34]. Die Kunden greifen über ihren Internetbrowser auf die Ressourcen zu und können diese so nutzen und verwalten. Bei der Auswahl ihrer Dienste haben die Benutzer die Möglichkeit, diese eigenständig zu konfigurieren und anzupassen.

Im Laufe der Zeit konnten sich zwei Abrechnungsmodelle im Cloud Computing etablieren. Das erste Modell basiert auf einem Abonnement, bei dem zu Beginn eines Abrechnungszeitraums, meist monatlich, ein fixer Betrag gezahlt wird. Dieser Betrag deckt die Nutzung aller Ressourcen innerhalb dieses Zeitraums ab. Das zweite Abrechnungsmodell folgt dem Prinzip „On-Demand“ oder „Pay-per-Use“. Der Kunde zahlt nur für die Ressourcen, die tatsächlich genutzt wurden. Die Abrechnung erfolgt in der Regel auf Basis der stündlichen Nutzung und des aktuellen Ressourcenverbrauchs. Für diese Arbeit besitzt das zweite Abrechnungsmodell, das „On-Demand“-Prinzip, die größere Relevanz.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) erstellte die folgende Begriffsdefinition als einheitliche Grundlage für den Begriff „Cloud Computing“:

„Cloud Computing bezeichnet das dynamisch an den Bedarf angepasste Anbieten, Nutzen und Abrechnen von IT-Dienstleistungen über ein Netz. Angebot und Nutzung dieser Dienstleistungen erfolgen dabei ausschließlich über definierte technische Schnittstellen und Protokolle. Die Spannweite, der im Rahmen von Cloud Computing angebotenen Dienstleistungen umfasst das komplette Spektrum der Informationstechnik und beinhaltet unter anderem Infrastruktur (z. B. Rechenleistung, Speicherplatz), Plattformen und Software.“ [2]

3.2.1 Cloud-Bereitstellungsmodelle

Cloud Computing kann in verschiedene Bereitstellungsmodelle unterteilt werden, die sich hinsichtlich in der Nutzbarkeit und Verfügbarkeit der Dienste unterscheiden. Dieses Kapitel beschreibt die wesentlichen Modelle: die öffentliche Cloud, die private Cloud und die hybride Cloud.

Eine öffentliche Cloud wird von einem Drittanbieter bereitgestellt. Das bedeutet, dass der Anbieter und die potenziellen Nutzer nicht derselben organisatorischen Einheit angehören. Der Zugriff auf die virtuellen Ressourcen ist öffentlich und gelingt über die Nutzung eines Web-Portals. [34] Dies bringt den Vorteil, dass keine internen Wartungen durchgeführt werden müssen. Dieses Modell ist vor allem für Start-ups oder generell kleinere Unternehmen geeignet. Dies folgt aus der Skalierbarkeit und der daraus folgenden Kostensenkung. Bekannte Anbieter für dieses Modell sind beispielsweise Amazon Web Services (AWS) [5], Microsoft Azure (MSA) [6] und Google Cloud Platform (GCP) [7].

Eine private Cloud ist eine IT-Infrastruktur, die in einem privaten Netzwerk aufgesetzt wird. Dabei gehören der Anbieter und die Benutzerseite derselben organisatorischen Einheit an [34]. Durch die interne Instandhaltung der Infrastruktur erfolgt erhöhte Sicherheit und Datenschutz, welche wiederum auch mit zusätzlichen Kosten verbunden sind.

Die hybride Cloud ist eine Kombination aus einer privaten und einer öffentlichen Cloud. Die beiden Infrastrukturen interagieren gemeinsam. Dies erlaubt die Auslagerung verschiedener Workloads in beide Clouds [3]. Hierbei werden bestimmte Funktionalitäten in die öffentliche Cloud ausgelagert, während der Regelbetrieb über die privaten Ressourcen erfolgt [34]. Eine Web-Applikation kann über eine öffentliche Cloud gehostet werden. Dies spart die Kosten für die Wartung eines Servers. Kritische Daten, wie beispielsweise personenbezogene Kundeninformationen, können dann intern in der privaten Cloud abgespeichert werden, um zu gewährleisten, dass keine Drittanbieter Zugriff auf diese Informationen erhalten.

In Abbildung 3.3 sind alle bislang beschriebenen Cloud-Bereitstellungsmodelle erkennbar. Die Anbieter-Organisation A repräsentiert eine standardisierte öffentliche Cloud, wie sie beispielsweise von Amazon Web Services, Microsoft Azure oder Google Cloud Platform bereitgestellt wird. Die Benutzer-Organisation B visualisiert eine private Cloud, die daran erkennbar ist, dass sie nicht über das Internet erreichbar ist. Die Anbieter-Organisation C und Benutzer-Organisation D gemeinsam stellen eine hybride Cloud dar.

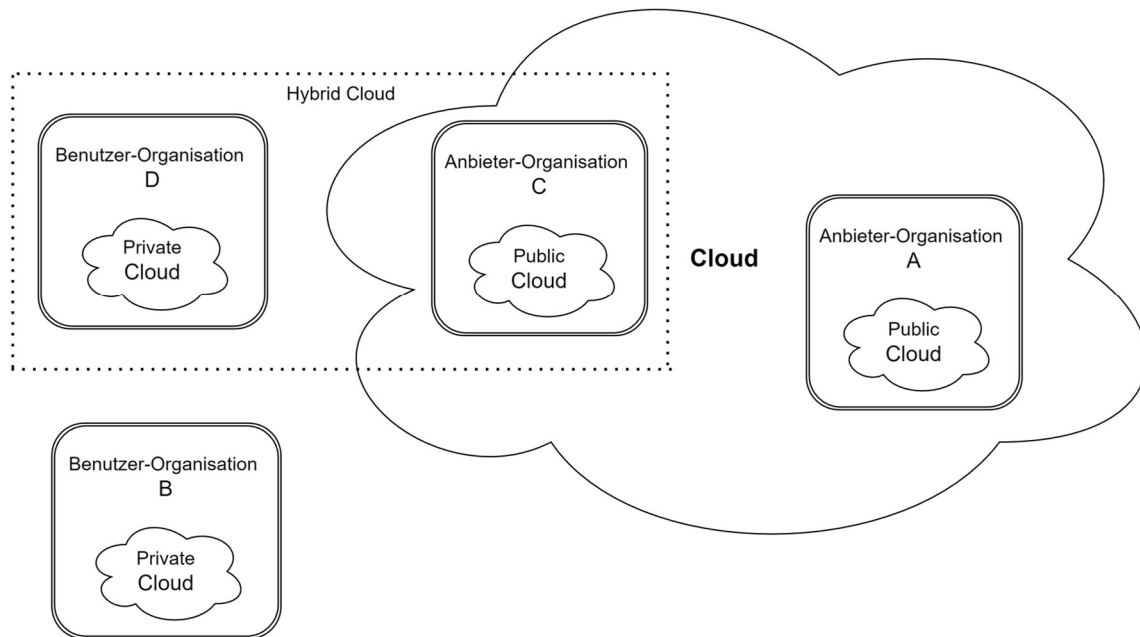


Abbildung 3.3: Cloud-Bereitstellungsmodelle

Quelle: In Anlehnung an [34]

3.2.2 Cloud-Service-Modelle

Cloud-Service-Modelle lassen sich nach ihrem Abstraktionsgrad unterscheiden. Je höher die Stufe der Abstraktion, desto mehr Infrastrukturen und Applikationen werden in die Cloud ausgelagert. Die bekanntesten Modelle sind Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS). Diese sind in Abbildung 3.4 innerhalb eines Schichtenmodells veranschaulicht.

IaaS umfasst die grundlegenden Bausteine für die Erstellung einer Cloud-Infrastruktur. Angeboten werden Netzwerkfunktionen, Datenspeicher und Rechenressourcen die virtuell oder dediziert bereitgestellt werden [4]. Für die Aufsetzung und Wartung der Anwendungen ist der Kunde selbst verantwortlich. Der Anbieter kümmert sich um die Instandhaltung der physischen Infrastruktur. Dieses Modell bietet dem Benutzer souveräne Kontrolle und Flexibilität über die vorhandenen Ressourcen.

PaaS ist ein weiteres Service-Modell und baut auf dem Grundgerüst von IaaS auf. In diesem Modell übernimmt der Anbieter zusätzlich die Verwaltung der zugrunde liegenden Infrastruktur, wie zum Beispiel Container und Betriebssysteme [4]. Dadurch entfallen Aufgaben wie Softwarewartungen und Patching für den Kunden, da diese in den Verantwortungsbereich des Anbieters fallen.

SaaS stellt die höchste Abstraktionsschicht dar. Hier stellt der Anbieter eine virtuelle Maschine bereit, die verschiedene Softwareanwendungen hostet und verwaltet. Der Benutzer benötigt lediglich einen Webbrowser, um auf diese Anwendungen zuzugreifen, was bedeutet, dass sie von überall aus erreichbar sind. Bekannte Beispiele für SaaS sind Microsoft 365, Google Workspace (vormals G Suite bzw. Google Apps), Dropbox [8] und viele weitere Dienste.

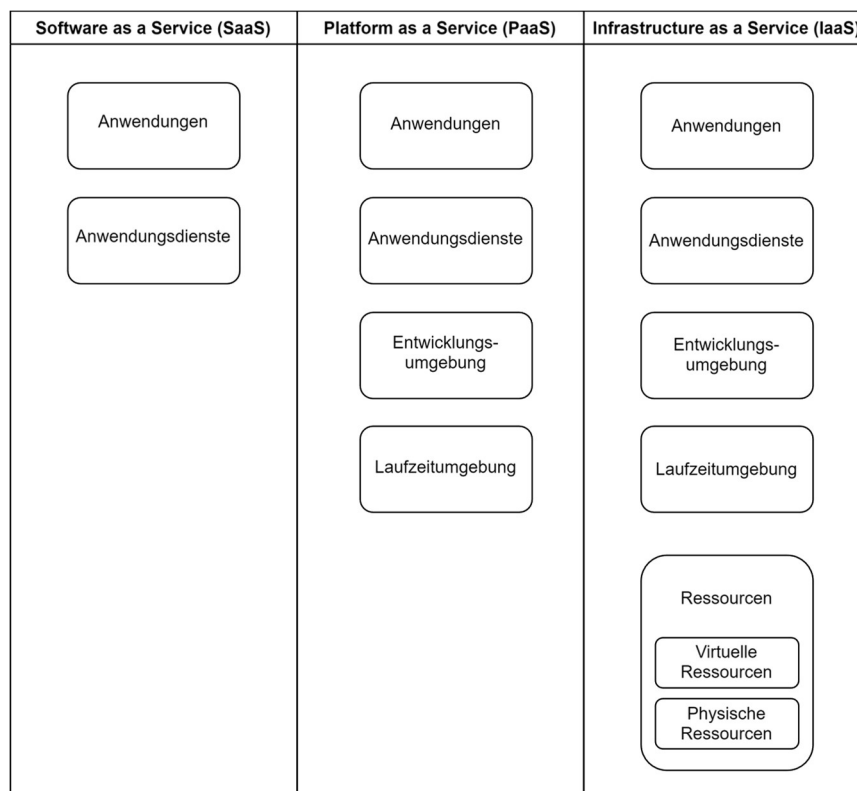


Abbildung 3.4: Cloud-Service-Modelle Quelle: In Anlehnung an [34]

3.2.3 Multi-Cloud

Dieser Abschnitt erläutert die grundlegende Idee hinter der Konstruktion einer Multi-Cloud-Umgebung. Dabei werden die bestehenden Herausforderungen aufgezeigt, die die Nutzung dieser Bereitstellungsart erschweren. Diese Analyse dient zur Verdeutlichung der Notwendigkeit von Sky Computing.

Eine Multi-Cloud-Umgebung besteht aus mindestens zwei öffentlichen Clouds und kann zusätzlich noch weitere private Clouds beinhalten. Multi-Cloud-Bereitstellungen zielen darauf ab, Redundanz im Falle von Hardware- oder Software-Ausfällen zu bieten und die Abhängigkeit von einem einzelnen Anbieter zu vermeiden [35]. Dieses Bereitstellungsmodell bietet die höchste Vergleichbarkeit mit der Idee des Sky Computing.

Zunächst sind generelle Aspekte des Cloud Computing wie die Datenverfügbarkeit und die dazugehörige Datensicherheit problematisch. Der Kunde ist vollständig auf den Anbieter angewiesen und muss dessen Sicherheitsmechanismen vertrauen, da keine eigene Kontrolle über die Sicherheit der Daten besteht [9]. Dasselbe gilt für die Datenverfügbarkeit. Der Kunde muss auf die kontinuierliche Verfügbarkeit der Cloud-Ressourcen vertrauen. Bei der Nutzung einer Multi-Cloud-Umgebung muss jedoch folgende Herausforderung beachtet werden. Lösungen, die über mehrere Cloud-Anbieter verteilt sind, benötigen eine effektive Überwachung [36]. Es müssen Mechanismen implementiert werden, um die Leistung, Verfügbarkeit, Sicherheit und die Kosten der verschiedenen Dienste zu überwachen. Dies ist notwendig, um sicherzustellen, dass die gesamte Infrastruktur effizient und ohne Unterbrechungen betrieben werden kann. Speziell in dieser Arbeit stellt sich die Frage, wie werden die Preise zukünftig bemessen. Darüber hinaus ist von einem Kostenanstieg im Verlauf der kommenden Jahre auszugehen [37]. Es erfordert bislang ein hohes Maß an Expertise und sorgfältiger Planung, um die einhergehenden Herausforderungen erfolgreich zu bewältigen.

Sky Computing geht einen Schritt weiter und strebt die nahtlose, anbieterübergreifende Nutzung von Cloud-Ressourcen an. Die Herstellung der Interoperabilität und Portabilität von Anwendungen und Daten zwischen den verschiedenen Cloud-Anbietern soll noch größere Flexibilität und Effizienz ermöglichen. Unter dem Aspekt der Effizienz findet auch das Preis-Leistungs-Verhältnis eine zentrale Rolle. Durch die erstellte Interoperabilität muss eine automatische Preisvergleichbarkeit und Analyse gewährleistet werden.

3.3 Sky Computing

Die Zukunft des Cloud Computing wird als Sky Computing bezeichnet. Im Gegensatz zur herkömmlichen Differenzierung zwischen verschiedenen Cloud-Anbietern zielt Sky Computing darauf ab, diese für den Nutzer zu vereinheitlichen. Eine der Grundideen besteht darin, dass die Anwendungen der Benutzer bei jedem Cloud-Anbieter ohne zusätzliche Konfigurationen identisch funktionsfähig sind. Entwickler sollen sich keine Sorgen um die Bereitstellung ihrer Anwendungen bei verschiedenen Anbietern machen müssen oder signifikante Einschränkungen bei der Migration zwischen Clouds erleben. Der grundlegendste Unterschied zur Multi-Cloud-Umgebung liegt in der vereinfachten Nutzung für die Anwender durch eine einheitliche Gestaltung in mehreren Aspekten und der daraus resultierenden Interoperabilität.

3.3.1 Forderungen

Um die Idee des Sky Computing umsetzen zu können sind grundsätzlich drei neue Module erforderlich. Diese würden die Interoperabilität zwischen den verschiedenen Cloud-Anbietern ermöglichen. Dabei handelt sich um eine Kompatibilitätsschicht, eine Intercloud-Schicht und eine Peering-Vereinbarung. Die Idee, Sky Computing durch das Hinzufügen neuer Schichten zu ermöglichen, basiert auf der Entwicklung des Internets und kann damit auch verglichen werden. [15]

Tabelle 3.1: Internet und Sky im Vergleich Quelle: [15]

Internet	Sky
Router	Server
Autonomous System	Datacenter / Availability Zone
Internet Service Provider	Cloud Provider
Enterprise Network	Private Cloud
Internet Protocol	Compatibility Layer
BGP	Intercloud Layer

3.3.2 Kompatibilitätsschicht (Compatibility Layer)

Die Kompatibilitätsschicht soll aus einer Sammlung verschiedene Schnittstellen bestehen. Die Schicht soll portabel sein und an jede beliebige Cloud angesetzt werden können. Durch die in der Schicht vorhandenen Schnittstellen sollen die Anwendungen ohne jegliche Veränderungen migriert werden können. Die Herausforderung liegt in der hohen Anzahl und der groben Definition der Dienste.

Diese Kompatibilitätsschicht könnte aus einer Kombination aus mehreren freien Software-Lösungen bestehen. Auf diese Weise könnte jede Problematik durch die Funktionen der entsprechenden Software-Lösung adressiert werden, was eine Interaktion mit den verschiedenen öffentlichen Cloud-Anbietern ermöglichen würde. [15]

Tabelle 3.2: Kompatibilität durch freie Software

Dienstschnittstelle	Freie Software-Lösung
Cluster-Ressourcen-Management	Kubernetes [10]
Anwendungspaketierung	Docker [11]
Datenbanken	MySQL [21]
Big Data Execution Engines	Apache Spark [12]
Streaming Engines	Apache Flink [13]
Bibliotheken für Maschinelles Lernen	PyTorch [14]

3.3.3 Intercloud-Schicht (Intercloud-Layer)

Die Intercloud-Schicht wird, wie in Abbildung 3.5 dargestellt, auf der Kompatibilitätsschicht angesetzt und soll eine Abstraktionsebene zwischen den verschiedenen Cloud-Anbietern schaffen. Dies verfolgt die Idee, das Management und die Konfiguration der verschiedenen Dienste zu vereinheitlichen und anbieterunabhängig zu gestalten. Das bedeutet, dass der Benutzer sich nicht darüber im Klaren sein muss, von welchem Anbieter ein beliebiger Dienst bereitgestellt wird. Stattdessen soll der Benutzer eine Präferenz auf der Grundlage von drei Kriterien angeben können:

Performanz, Verfügbarkeit und Kosten. Um die Intercloud-Schicht erfolgreich zu realisieren sind drei Funktionalitäten notwendig. [15]

An erster Stelle ist für die Dienste der freien Software-Lösungen ein einheitliches Namensschema notwendig. Um die benötigten Ressourcen zu finden, müssen diese nach einem standardisierten Namensschema benannt sein. Damit die benötigten Instanzen gefunden werden können, müssen sie nach einem einheitlichen Namensschema benannt sein. Hierfür könnte das Domain Name System (DNS) genutzt werden. Das DNS ist ein Protokoll zur Namensauflösung von Domainnamen in IP-Adressen, das auf einem hierarchischen Namensraum basiert [38]. Da das DNS bereits als etablierter Netzwerkdienst weit verbreitet ist, könnte dies die Implementierung erleichtern. Der neue Name einer Ressource setzt sich aus dem Dienst-Namen und zusätzlichen Metadaten zusammen. Diese Metadaten würden Informationen wie den Namen des Cloud-Anbieters, den Standort der kommunizierenden Datenzentrale und zusätzlich Daten wie den Versionsstatus der genutzten Schnittstellen beinhalten. Hinzukommend könnte dieses Namensschema auch Parameter wie den Preis und die Verfügbarkeit der Ressource abdecken. [15]

Die nächste Funktionalität ist ein Verzeichnissystem, welches die Dienste der Cloud-Anbieter mit den generierten Namen des zuvor beschriebenen Namensschemas verknüpft. In diesem System müsste jeder Cloud-Anbieter die Dienste anhand des Namens und der dynamischen Metadaten freigeben, sodass diese von außerhalb aufgerufen werden können. [15]

Um auf die Dienste zugreifen zu können, ist ein Account bei jedem Cloud-Anbieter erforderlich. Da die Wahl des Dienstes und somit auch des Anbieters variieren kann, muss für jeden Anbieter ein separates Konto erstellt werden. Dies ist nicht nur umständlich und zeitaufwendig, sondern erhöht auch das Risiko von Sicherheitsproblemen aufgrund der Vielzahl an Konten, die potenziellen Datenlecks ausgesetzt sind. Eine mögliche Lösung für dieses Problem wäre der Einsatz einer Drittanbieter-Anwendung. Eine solche Anwendung könnte Konten bei allen relevanten Cloud-Anbietern verwalten und die entstandenen Kosten an die Benutzer weiterleiten [15]. In Abbildung 3.5 ist die Verbindung zwischen den verschiedenen Intercloud-Schichten veranschaulicht. Alle Verbindungen führen zu einer zentralen Intercloud-Schicht, die die Kommunikation zwischen den Komponenten ermöglicht und dafür die zuvor beschriebenen Funktionalitäten verwendet.

3.3.4 Peering-Vereinbarung (Peer agreement)

Die Nutzung von Cloud-Diensten umfasst häufig große Datenmengen. Dabei besitzt jeder Cloud-Anbieter seinen eigenen Preis für die Verwendung ihrer Speicherlösungen. Typischerweise sind die Kosten für das Speichern von Daten geringer als die Kosten für deren Übertragung. Das Ziel der Peering-Vereinbarung ist es, diese Datensätze schnell und kostenlos zwischen den verschiedenen Cloud-Anbietern zu übertragen. Um dies zu realisieren, müssten die Cloud-Anbieter kooperieren und eine Vereinbarung treffen, die für alle beteiligten Parteien vorteilhaft ist. Diese Kooperationsmodelle sind auf dem Markt bereits zu beobachten. Ein Beispiel hierfür ist Google Anthos, eine Plattform, die derzeit sowohl auf der Google Cloud Platform als auch auf Amazon Web Services läuft.

Durch die Kombination der beschriebenen Schichten könnte die Idee des Sky Computing eine Architektur nutzen, wie sie in Abbildung 3.5 dargestellt ist, um zur Realisierung zu gelangen.

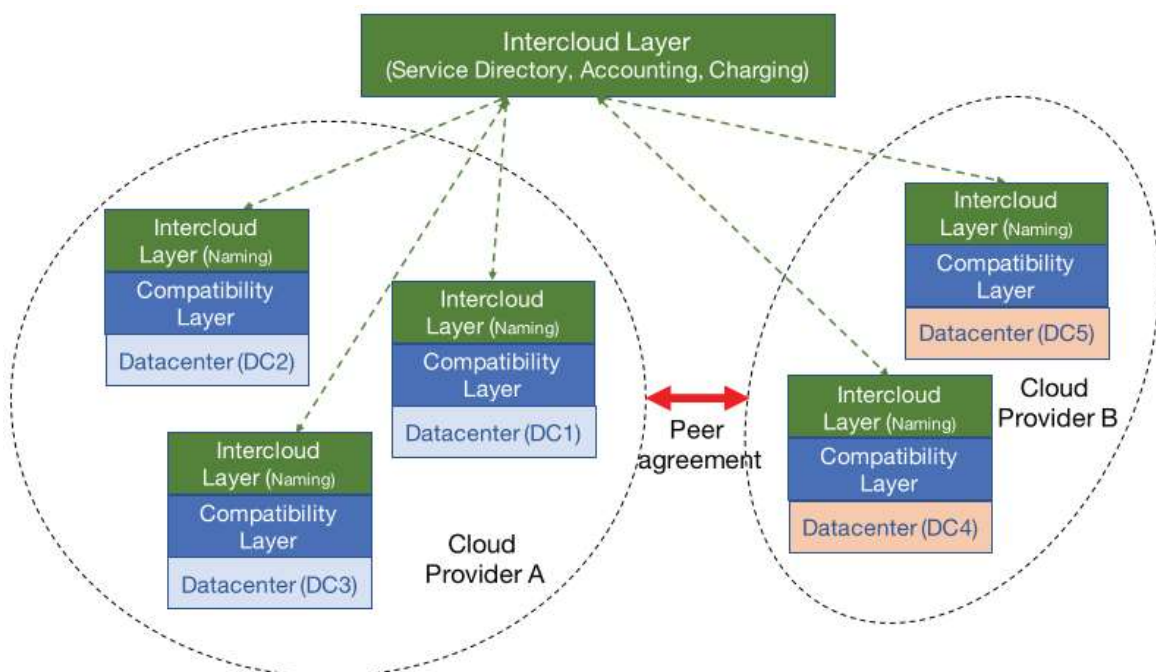


Abbildung 3.5: Mögliche Sky Computing Architektur

Quelle:[15]

3.3.5 Offene Herausforderungen

Die Realisierung der technischen Herausforderungen wäre durch die vorgeschlagene Architektur effektiv zu bewältigen. Eine offene Problematik bleibt jedoch die Datensicherheit. Je mehr Dienste der Benutzer verwendet, desto mehr potenzielle Angriffsstellen entstehen für mögliche Datenlecks [16]. Die Datensicherheit muss daher sowohl von der Intercloud-Schicht als auch von jedem einzelnen Cloud-Anbieter zu jederzeit gewährleistet werden.

Die weitaus zentralere Problematik für die Zukunft des Sky Computing liegt jedoch darin, ob dieses Konzept die Zustimmung der Cloud-Anbieter finden wird. Die Idee des Sky Computing kann nur erfolgreich umgesetzt werden, wenn sich die Cloud-Anbieter zusammenschließen und bereit sind, aktiv an der Entwicklung und Integration dieses Konzepts beizutragen. Das Problem hierbei ist, dass Sky Computing in erster Linie die Nutzer unterstützt und nicht Anbieter. Dadurch hätten kleinere Cloud-Anbieter den Zugang zu einem größeren Marktanteil und könnten mit besseren Angeboten in direkte Konkurrenz mit den etablierten Großunternehmen treten. Wie sich nun dieses neue Paradigma letztlich entwickeln wird, ist derzeit ungewiss.

Im speziellen Bezug auf diese Arbeit ist die Kosteneinschränkung, also der Preis der jeweiligen Ressource, ein hochrelevanter Faktor bei der Wahl eines konkreten Cloud-Anbieters, wie in Abschnitt 3.3.3 beschrieben. Diese Problematik kann mithilfe der Entwicklung des Sky Computing bewältigt werden. Um das Kriterium der Kosteneinschränkung anbieten zu können, müssen zunächst die Preisinformationen der verschiedenen Anbieter eingeholt und analysiert werden. Diese Informationen müssen im Rahmen des Sky Computing automatisiert in einer Kommunikation mit den verschiedenen Cloud-Anbietern angefragt werden. Anschließend müssen diese Daten für zukünftige Nutzungen standardisiert und vereinheitlicht werden. Dafür könnte der in dieser Arbeit konzipierte Prototyp als Grundlage genutzt werden.

4 Design

Dieses Kapitel dient als eine Beschreibung für das gewählte Design zur Konzeptionierung eines Prototypen zur automatisierten Abfrage der Preis-API von public Cloud-Anbietern. Im ersten Abschnitt werden die Anforderungen an den Prototypen beschrieben. Daraufhin folgt eine Beschreibung der gewählten Komponenten. Abschließend werden die wesentlichsten Entwicklungsschritte konzeptioniert.

4.1 Anforderungsanalyse

In diesem Abschnitt erfolgt eine detaillierte Anforderungsanalyse an den Prototypen. Diese beinhaltet eine Liste der erforderlichen Funktionalitäten sowie eine Beschreibung der nicht-funktionalen Metriken, die für das gewünschte Ergebnis erforderlich sind.

Die folgenden Funktionalitäten sind für den Prototypen erforderlich:

- Automatisierte Abfrage der Preis-APIs von drei public Cloud-Anbietern
- Abdeckung von drei Cloud-Diensten pro Anbieter
- Einlesen der aufgerufenen Datensätze
- Verarbeitung der Datensätze in ein einheitliches Format
- Speicherung der Daten in einer persistenten Datenbank
- Tabellarische Bereitstellung der Daten
- Filtermöglichkeiten innerhalb der Tabelle

Die Cloud-Anbieter aktualisieren ihre Preise nur sehr unregelmäßig und generell unvorhersehbar. Der Zeitpunkt, wann eine Aktualisierung durchgeführt wird, ist nicht festgesetzt. So sagt beispielsweise Amazon Web Services, dass sich ihre Preislisten zu jederzeit ändern könnten [32].

Ein weiteres Beispiel für die Unkalkulierbarkeit ist die letzte Änderung der Google Cloud Storage Preisliste. Diese liegt nach Angaben nun länger als ein Jahr zurück [33]. Dies erschwert die zeitliche Wahl der automatisierten Abfrage für diese Arbeit.

Die nicht-funktionalen Metriken beschreiben das Verhalten des Prototypen. In diesem System müssen einige Metriken gewährleistet werden, um eine dauerhaft erfolgreiche Datenaufbereitung zu ermöglichen. Dabei sind folgende nicht-funktionale Metriken von großer Wichtigkeit:

- Verfügbarkeit
- Zuverlässigkeit
- Interoperabilität
- Erweiterbarkeit
- Benutzerfreundlichkeit

Der Prototyp muss aufgrund der automatischen Aktualisierung der Preislisten und der Datenpopulation eine hohe Verfügbarkeit gewährleisten. Dabei sollen Wartungen und Systemausfälle die Betriebsbereitschaft des Prototypen nur minimal beeinträchtigen. Die im Prototyp bereitgestellten Daten müssen konsistent, präzise und vollständig sein. Dafür soll über einen festgelegten Zeitraum ein Testlauf durchgeführt werden, um die bislang beschriebenen Metriken zu prüfen. [40] Dabei sollen die erhaltenen Daten in eine einheitliche Form gebracht werden, sodass darauf aufbauend eine Anbieterunabhängigkeit entwickelt werden kann. Dies soll einen Grundstein der Interoperabilität zwischen den public Cloud-Anbietern legen. Das System sollte mit minimalem Aufwand weitere Cloud-Anbieter und Dienste hinzufügen können, um so eine hohe Erweiterbarkeit zu gewährleisten. Abschließend sollte der Prototyp eine leicht zu bedienende Benutzeroberfläche besitzen und geringe Wartezeiten aufweisen.

Durch die Implementierung der genannten Funktionalitäten und nicht-Funktionalen Metriken, soll es den Nutzern ermöglicht werden die Preise der verschiedenen Anbieter abzurufen und zu analysieren. Dies soll einen benutzerfreundlichen Preisvergleich ermöglichen.

4.2 Programmiersprache

Für die Implementierung des Prototypen wurde die Programmiersprache Go [22] gewählt. Projekte, die einen Automatisierungsprozess besitzen, sind meist in einer Skriptsprache wie Python geschrieben. Sie ist in ihrer Schreibweise effizienter da alle Funktionen in einer minimalen Anzahl von Zeilen verfasst werden können. Aufgrund ihrer hohen Beliebtheit sind unzählige Bibliotheken für die Skriptsprache Python verfügbar, wodurch gängige Funktionalitäten nicht von Grund auf neu entwickelt werden müssen.

Go eignet sich für alle bekannten Anwendungsfälle, da sie eine neue Programmiersprache ist und dadurch viele neue Funktionalitäten bietet, die ihre Vorgänger wie z.B. Java nicht besaßen. Sie bietet eine saubere Verarbeitung der unterschiedlichsten Datenstrukturen und ist dementsprechend eine gute Wahl für diese Aufgabenstellung.

Die Automatisierung wird dabei durch eine Go-Routine sichergestellt. Diese Go-Routine ist eine grundlegende Komponente der Programmiersprache und ermöglicht eine effiziente Ausführung verschiedener Funktionen und Methoden. Im Vergleich zu traditionellen Threads wie sie beispielsweise in Java vertreten sind, haben sie in Go einen geringeren Speicherverbrauch und produzieren auch einen geringeren Leistungsverlust. [23]

Für diesen Prototypen ist die Asynchronität der Go-Routine von großem Vorteil. Sie ermöglicht die tabellarische Darstellung in einer lokalen Web-Applikation und die automatische Abfrage der Preis-APIs, innerhalb einer einzelnen Anwendung.

Ein weiterer Grund liegt in der Nutzbarkeit des folgenden Frameworks. Golang Object Relational Mapping (GORM) ist eine Bibliothek, die Funktionalitäten bietet, um die Interaktion mit einer relationalen Datenbank in der Go-Programmiersprache zu vereinfachen. Beispielsweise werden Go-Strukturen automatisch in Datenbanktabellen konvertiert. Jedes Strukturfeld entspricht dabei einer Tabellenspalte. [24]

4.3 Ausgewählte Cloud-Anbieter und Dienste

Der Fokus dieser Arbeit liegt bei den drei größten public Cloud-Anbietern. Diese sind Amazon Web Services (32% Marktanteil), Microsoft Azure (23% Marktanteil) und Google Cloud Platform (10% Marktanteil) [17]. Die Sammlung der Preislisten beschränkt sich gemäß der Anforderungsanalyse auf drei spezifische Dienste pro Anbieter, die für einen Vergleich geeignet sind. Die Auswahl umfasst dabei einen Computing-Dienst, einen Speicherdienst und ein Datenbankdienst.

4.3.1 Cloud-Computing-Dienste

Alle gewählten Anbieter stellen ihren Benutzern einen Computing-Dienst zur Verfügung. Hierbei mietet der Benutzer einen konfigurierbaren virtuellen Server. Dieser wird als eine Instanz bezeichnet. Die Berechnung des Preises für diese Instanz ist abhängig von der Leistung und folgt dem On-Demand Prinzip. Sobald eine Instanz erstellt wurde, besitzt der Benutzer den Zugang auf diese Maschine. Sollte die Instanz mit einem Linux-Betriebssystem ausgestattet sein, so erfolgt eine Verbindung über das Secure Shell Protokoll (SSH). Bei einer Initialisierung mit einem Windows-Betriebssystem kommt für die Kommunikation das Remote-Desktop-Protokoll (RDP) zum Einsatz.

Der von Amazon Web Services bereitgestellte Computing-Dienst trägt den Namen EC2 (Elastic Compute Cloud). Dieser besitzt über 750 verschiedene Instanzen aus dem die Nutzer wählen können. Dies erlaubt den Benutzern, ihre Kapazitäten nach ihrem Vorhaben anzupassen. [19] AWS teilt ihre Instanz-Typen in sechs verschiedene Kategorien auf. Diese sind in Tabelle 4.1 dargestellt. Die Erstellung der Instanz startet mit der Auswahl einer Blaupause. Diese Blaupause wird Amazon Machine Image (AMI) genannt. Dabei handelt es sich um ein vorgefertigtes Image, dass sich in der Wahl des Betriebssystems und in der vorinstallierten Software unterscheidet [34]. Der Benutzer hat die Auswahl zu allen gängigen Systemen wie Ubuntu, Debian, Amazon Linux, SUSE-Linux, RedHat, Windows Server und Mac OS. Im Detail kann eine genaue Version und Architektur gewählt werden. Daraufhin folgt die Wahl der Instanz-Art. Die Ressourcenkonfigurationen unterscheiden sich in der Leistungsstärke des Prozessors und in der Größe des Arbeits- und

Festplattenspeichers. Für die Identifizierung des Benutzers auf dem virtuellen Server muss ein Schlüsselpaar generiert werden. Dieses funktioniert über das public-Key-Verfahren. Dabei wird vor der ersten Nutzung ein Schlüsselpaar bestehend aus einem öffentlichen und einem privaten Schlüssel generiert. Der öffentliche Schlüssel wird dem Amazon Benutzerkonto zugewiesen, während der private Schlüssel in der Arbeitsumgebung des Nutzers verweilt. Daraufhin ist der Nutzer in der Lage die erstellte Instanz zu starten. Diese erhält eine öffentliche und private IP-Adresse die dynamisch zugewiesen wird. Anhand der öffentlichen IP-Adresse kann auf den virtuellen Server über das Internet zugegriffen werden. Durch die private IP-Adresse ist der Rechner für andere Instanzen in der Cloud erkennbar. Dabei muss beachtet sein, dass die beschriebenen Adressen bei jedem Start der Instanz neu vergeben werden. [34]

Tabelle 4.1: EC2-Instanz-Typen

Priorisierung	Verwendungszweck
Allgemeine Zwecke	Gleichmäßige Verwendung von Rechen-, Speicher- und Netzwerkressourcen
Datenverarbeitung Optimierung	Hochleistungs-Webserver, wissenschaftliche Modellierung, dedizierte Spieleserver
Arbeitsspeicheroptimierung	Verarbeitung großer Datenmengen im Speicher
Beschleunigte Datenverarbeitung	Effizientere Ausführung für Gleitkomma-Berechnung, Grafikberechnung oder Datenmusterabgleich
Speicheroptimierung	Sequenzielle Lese- und Schreibzugriff Operationen auf sehr große Datenmengen
HPC-Optimierung	Komplexe Simulationen, Deep-Learning Workloads

4.3.2 Cloud-Objektspeicherdienste

Die Cloud-Anbieter stellen ebenfalls einen Objektspeicherdienst bereit. Dabei handelt sich um eine Speicherlösung für große Mengen von unstrukturierten Daten. Diese Daten können beispielsweise aus Audiodateien, Protokollen, Fotos, Videos, Webseiten oder auch Sensordaten bestehen. Jede Datei wird als einzelnes Objekt gespeichert und wird mit einem eindeutigen Namen gekennzeichnet. Zusätzlich werden der Datei Metadaten zugewiesen. [20].

Objektspeicherlösungen sind als kostengünstige Optionen bekannt und dementsprechend beliebter Dienst bei den Kunden der Cloud-Anbieter. Der Objektspeicherdienst von Amazon Web Services trägt den Namen S3 (Simple Storage Service). Die Infrastruktur ist darauf ausgelegt, dass die Zugriffe auf die Daten vom Kunden komplett kontrolliert werden können [39]. Amazon S3 nutzt für die Speicherung der Daten ein sogenanntes Bucket-System. Dieses System ist in Abbildung 4.1 dargestellt. Dabei werden alle Objekte in einen Bucket gespeichert. Der Name von jedem Bucket muss einzigartig sein. Die Objekte können hierbei eine Größe von 1 Byte bis hin zu 5 Terabyte besitzen. Darüber hinaus kann kein Bucket innerhalb eines anderen Bucket erstellt werden. Jeder Bucket kann vom Nutzer einzeln verwaltet werden, wobei auch der Zugriff auf einen Bucket nach bestimmten Regionen beschränkt werden kann. Dies ist vor allem beim Umgang mit Kundendaten sehr nützlich, aufgrund der Beachtung von regional gesetzlichen Grundlagen. [39]

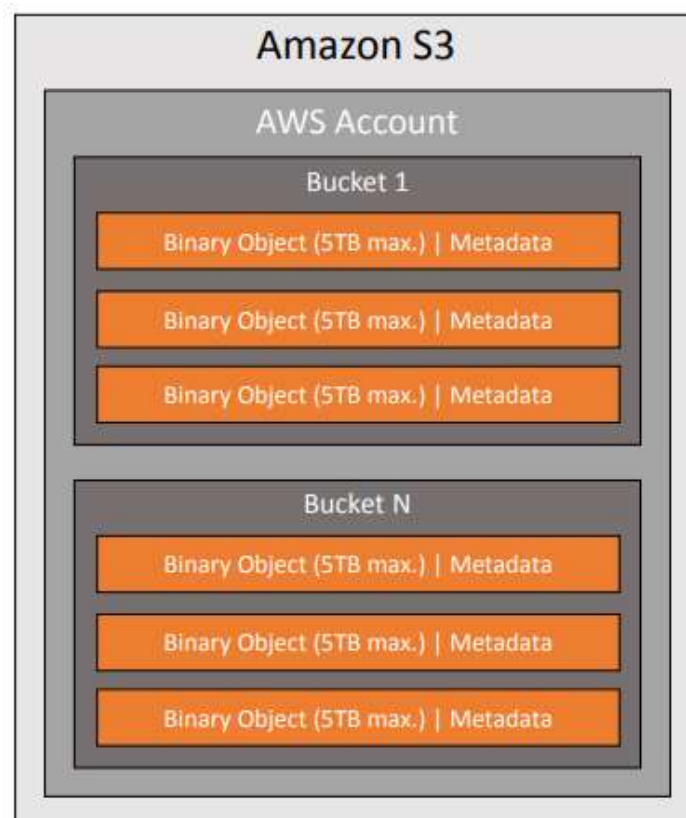


Abbildung 4.1: S3 Bucket System Quelle: [39]

Der S3-Speicherdienst ist in verschiedene Speicherklassen unterteilt. Die Wahl der Speicherklasse ist abhängig von den folgenden Kriterien [26]:

- Zugriffsrate
- Abrufzeit
- Kosten
- Haltbarkeit und Verfügbarkeit

4.3.3 Cloud-Blockspeicherdienste

Amazon Web Services, Microsoft Azure und Google Cloud Platform bieten neben dem Objektspeicherdienst ebenfalls einen Blockspeicherdienst an. Dieser Dienst wird für die Verwaltung von strukturierten, kleineren Datenmengen verwendet. Die Blockspeicherung teilt die Datensätze in identisch große Blöcke auf. Alle erstellten Blöcke erhalten eine eindeutige Kennung, sodass sie anhand dieser Kennung aufgerufen werden können. Diese Informationen werden in einer Datentabelle abgespeichert, welche als Daten-Lookup-Tabelle bezeichnet wird. Bei einem Aufruf der Daten werden die einzelnen Blöcke durch diese Tabelle aufgerufen und wieder zusammengefügt. Dadurch ist die Menge der gespeicherten Daten ein wesentlicher Faktor für die Blockspeicherung. Mit zunehmender Anzahl der Datensätze verlängert sich auch die Suchzeit nach den benötigten Informationen. [20]

Die Blockspeicherlösung von Amazon Web Services heißt EBS (Elastic Block Storage). Dieser Dienst existiert nicht als ein individueller Service, sondern ist nur in Verwendung mit anderen AWS-Diensten nutzbar, wie mit einer EC2-Instanz. Die virtuelle Maschine ist mit einer Hard Disk Drive (HDD) oder Solid State Drive (SSD) ausgestattet, welche nach dem Blockspeicherungsprinzip konfiguriert wurde. Es ist zu beachten, dass ein EBS-Volumen an nur genau einer Instanz angehängt werden kann. Des Weiteren muss sich die Instanz in derselben Verfügbarkeitszone wie das Volumen befinden. [34] Der Benutzer kann den EBS-Dienst nach seinem Vorhaben anpassen. Dazu zählt die Größe der Speichereinheit in Gigabyte und die Auswahl des Laufwerktyps. Die verschiedenen Typen priorisieren unterschiedliche Faktoren.

Beispiele für diese Faktoren sind die Schnelligkeit der Datenübertragung, die Kostensenkung oder die Anzahl von Input-, Output-Operationen pro Sekunde. Zusätzlich kann angegeben werden, ob die Daten verschlüsselt oder unverschlüsselt abgespeichert werden sollen.

4.3.4 Cloud-Datenbankdienste

Ein Cloud-Datenbankdienst stellt dem Nutzer eine relationale oder nicht-relationale Datenbank zur Verfügung. Durch die Nutzung einer Datenbank über die Cloud muss keine physische Infrastruktur verwaltet werden. Dabei übernimmt der Cloud-Anbieter die Bereitstellung, Aktualisierung und Wartung jeglicher Hardware- und Software-Komponenten. Dadurch ist hohe Skalierbarkeit gewährleistet, allerdings fallen auch einige negative Aspekte einher. Durch die Nutzung einer Cloud-Datenbank befindet sich der Kunde in einer Anbieterabhängigkeit. Hierbei können komplexe Migrationen und die Integration der Daten in anderweitige Systeme bestimmte Problematiken verursachen [27].

Der Cloud-Datenbankdienst von Amazon Web Services trägt den Namen RDS (Relational Database Service). Dieser Dienst erlaubt die Einrichtung, Inbetriebnahme und Skalierung einer relationalen Datenbank in der Cloud. Es handelt sich bei Amazon RDS um einen Plattformdienst. Dementsprechend ist die Verbindung über einen direkten Zugriff via SSH möglich. Dabei gewährt dieser Dienst den Zugriff auf sämtliche MySQL-Funktionalitäten, wodurch die Migration mit einer existierenden Anwendung vereinfacht wird. Ein weiterer Vorteil ist der zuverlässige Betrieb durch die automatisierte Erstellung von Backups und Snapshots. [34]

4.3.5 Vergleich zu Microsoft Azure und Google Cloud Platform

Alle Cloud-Dienste, die in diesem Kapitel exemplarisch anhand von AWS-Diensten dargestellt wurden, sind im vergleichbaren Umfang bei Microsoft Azure und Google Cloud Platform verfügbar. Die Anbieter bieten nahezu identische Dienste an, die sich lediglich in einigen kleinen, für diese Arbeit jedoch irrelevanten Aspekten unterscheiden. In Tabelle 5.2 sind die Dienstnamen der beschriebenen Dienste für die gewählten Cloud-Anbieter aufgeführt.

Tabelle 4.2: Dienstnamen im Vergleich

Dienst \ Anbieter	AWS	MSA	GCP
Computing	EC2	Azure Virtual Machines	Google Cloud Compute Engine
Objektspeicher	S3	Azure Blob Storage	Google Cloud Storage
Blockspeicher	EBS	Azure Disk Storage	Google Persistent Disk
Datenbank	RDS	Azure SQL Databases	Google Cloud SQL

4.4 Preis-APIs

In diesem Abschnitt wird der Aufbau der ausgewählten Preis-APIs für die genannten Cloud-Anbieter beschrieben. Zudem wird eine alternative Lösung betrachtet und die abschließende Designentscheidung erläutert.

Die gewählten Preis-APIs für diesen Prototypen können als Preiskatalog-APIs kategorisiert werden. Sie bieten einen programmatischen Zugangspunkt, um an die Preisinformationen der verschiedenen Dienste zu gelangen. Die Antwort der Preis-APIs beinhaltet einen Preiskatalog mit allen benötigten Informationen, eingebettet in einer JSON-Datenstruktur. Dabei ist gewährleistet, dass die Daten zum Zeitpunkt des Abrufs immer aktuell sind. Der Aufbau dieser JSON-Datenstruktur ist Anbieterabhängig und kann nicht standardisiert werden.

4.4.1 AWS Price List Bulk API

Die Preis-API von Amazon Web Services trägt den Namen AWS Price List Service API. Um die benötigten Preisinformationen abzurufen, wird der folgende Endpunkt angesprochen:

```
https://pricing-us-east-1.amazonaws.com/offer/v1.0/aws/  
    {Service-Code}/current/{Region}/index.json
```

Der bereitstellende Server wird hierbei durch eine GET-Anfrage angesprochen. Nach der Verarbeitung dieser Anfrage erhält der Client den Preiskatalog in Form einer JSON-Datei zurück.

Die Größe des erhaltenen Preiskatalogs beträgt mehrere Gigabyte, wodurch zunächst das Herunterladen der Datensätze, je nach Verbindungsstärke, eine lange Zeit in Anspruch nehmen kann. Im Aspekt der Datenverarbeitung ist es ebenfalls kritisch. Die Dateigröße erhöht die Suchzeit für Algorithmen gewaltig. Dadurch wird das gesamte System verlangsamt und auch anfälliger für Ausfälle. Aus diesem Grund bietet die URL zwei Filtervariablen an. Die erste Variable ist ein Platzhalter für den Service-Code. Das ermöglicht die Filterung nach einem speziellen Dienst. Einige Beispiele für Service-Codes sind: AmazonEC2, AmazonS3 oder AmazonRDS. Die zweite Filtervariable erlaubt die Angabe einer AWS-Region, um die Preise für eine bestimmte Region zu erhalten. Beispiele für diese Regionen sind us-east-1, eu-central-1 und viele weitere. [25] Es existiert einschließlich für die Verwendung dieser Preis-API ein Software-Development-Kit für die Go-Programmiersprache. Dieses ist von Amazon Web Services bereitgestellt und vereinfacht die Kommunikation mit der Schnittstelle durch vorgefertigte Funktionen und Strukturen. Für die erfolgreiche Nutzung der Funktionen dieses Development-Kits ist die Erstellung eines AWS-Kontos nötig. Der Grund dafür ist das diese Funktionen die Benutzerdaten eines AWS-Mitglieds benötigen. Dies kann durch die Nutzung dieser Preis-API vermieden werden.

4.4.2 Azure Retail Prices REST API

Die Azure Retail Prices REST API ist die Preiskatalog-API für die Dienste von Microsoft Azure. Sie agiert wie die zuvor beschriebene Preis-API von Amazon Web Services, unterscheidet sich allerdings in der Struktur der URL. Die Azure Retail Prices API wird über den folgenden Endpunkt aufgerufen:

```
https://prices.azure.com/api/retail/prices?  
currencycode='USD'&$filter= serviceFamily eq '' and  
serviceName eq '' and armRegionName eq ''
```

Im Vergleich zu AWS bietet die URL-Struktur von Azure zusätzliche Filteroptionen. Innerhalb der URL kann zunächst die Währung angegeben werden, auf die sich der Preiskatalog beziehen soll. Anschließend erfolgt die Filterung nach der „serviceFamily“, also der Dienstfamilie. Beispielwerte für dieses Attribut sind Computing, Storage und Databases. Der folgende Platzhalter „serviceName“ ermöglicht eine weitergehende Spezifizierung des benötigten Dienstes innerhalb der jeweiligen Dienstfamilie. Für die genannten Dienstfamilien Computing, Storage und Databases könnten die Werte beispielsweise Virtual Machines, Blob Storage und SQL Database sein. Die letzte relevante Filtervariable ist „armRegionName“ und wird für die Filterung der Preisregion genutzt. [28]

4.4.3 Google Cloud Billing API

Die Google Cloud Billing API ist eine umfassende Schnittstelle, die mehrere Funktionalitäten beinhaltet [29]. Für diesen Prototypen wird jedoch nur die Preis-API verwendet, die über einen bestimmten Endpunkt erreichbar ist. Diese Schnittstelle ist über die folgende URL erreichbar:

```
https://cloudbilling.googleapis.com/v1/services/  
{Service-Id}/skus/?key={API-Key}
```

Mit der Nutzung dieser Preis-API erscheint ein problematischer Aspekt. Die erste Filtervariable „Service-Id“ besteht aus einer Identifikationskennung des spezifischen Dienstes. Sie besteht aus insgesamt sechzehn Zeichen und beinhaltet Buchstaben, Zahlen und Bindestriche. Die Kennung „6F81-5844-456A“ repräsentiert alle Computing-Instanzen, die vom Dienst Google Compute Engine angeboten werden. Die einzelnen Instanzen innerhalb dieser Dienstkategorie sind durch zusätzliche Kennungen spezifiziert, auf die später verwiesen wird.

Die Problematik erscheint mit der zweiten Filtervariable, dem „API-Key“. Ein API-Schlüssel wird genutzt, um den Zugang auf eine gewisse Ressource zu regulieren und in diesem Fall nur Nutzern mit einem aktiven GCP-Konto zu gewähren. Dieser Punkt sollte, ähnlich wie bei der AWS-API, vermieden werden. Da es sich jedoch um die benötigte Lösung handelt, muss ein Kompromiss gefunden werden. Der API-Schlüssel kann durch ein aktives GCP-Konto generiert werden. Es ist allein die Erstellung eines Kontos notwendig. Weiterhin müssen keine weiteren Informationen außer der E-Mail-Adresse angegeben werden. Dabei besitzt der API-Schlüssel keine Begrenzung und muss ebenfalls nicht erneut generiert werden. Dies lässt darauf schließen, dass die Verfügbarkeit und Zuverlässigkeit weiterhin gewährleistet bleibt, trotz Nutzung eines kontobasierten API-Schlüssels.

4.4.4 Preiskalkulator-API

Eine alternative Möglichkeit an die Preise der Cloud-Dienste zu gelangen, wäre die Nutzung der Preiskalkulator-API. Alle genannten Cloud-Anbieter besitzen solch eine Schnittstelle, wodurch Interessenten die Möglichkeit geboten wird, den Preis vormals abzuschätzen. Dafür gibt der Benutzer die gewünschte Konfiguration an und der Kalkulator gibt den monatlichen Kostenbetrag aus. Im Zusammenhang mit diesen Prototypen kann diese API allerdings nicht effizient genutzt werden. Der Grund hierfür ist, dass keine programmatische Schnittstelle zur Verfügung liegt. Die Kosten müssten manuell abgefragt und abgespeichert werden, um diese APIs zu nutzen. Die fehlende Automatisierungsmöglichkeit macht diese Art der APIs für dieses Projekt irrelevant.

4.5 Konzeption der Datenerlangung

Dieses Kapitel beschreibt das Konzept der Datenerlangung durch Kommunikation mit den zuvor ausgewählten Preis-APIs. Die Kommunikation kann nach dem Prinzip der Abbildung 4.2 gestaltet werden.

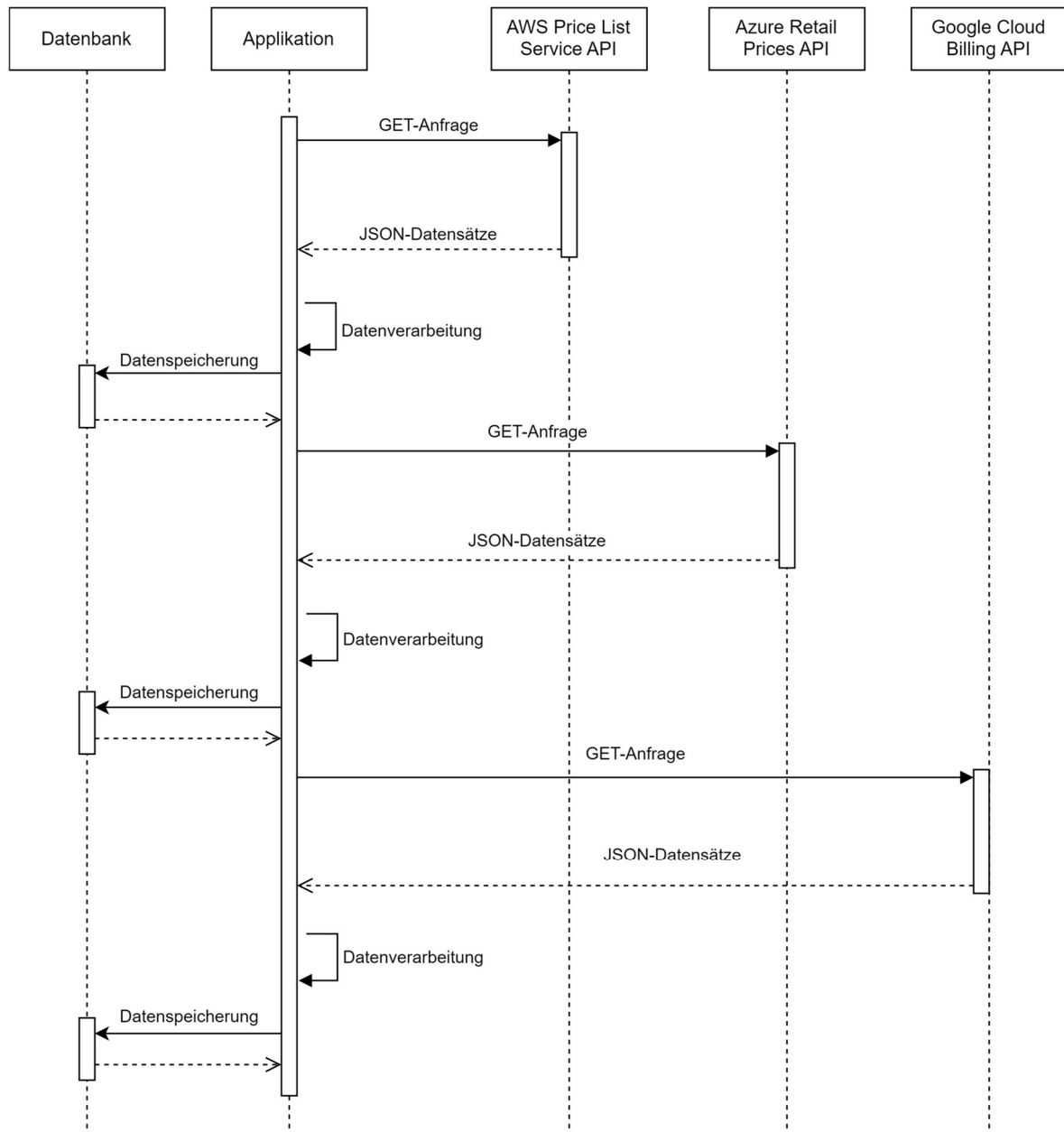


Abbildung 4.2: Ablauf der Datenerlangung

In diesem Konzept werden insgesamt neun automatisierte GET-Anfragen an alle zuvor beschriebenen Endpunkte durchgeführt. Dabei werden pro Anbieter jeweils drei Anfragen für jeden der benötigten Dienste gesendet. Die HTTP-Antworten der Preis-APIs besitzen die Preiskataloge in Form von unterschiedlichen JSON-Datenstrukturen. Diese müssen je nach Anbieter in einem separaten Verfahren in eine einheitliche Datenstruktur konvertiert werden. Daraufhin erfolgt die Kommunikation mit der Datenbank, in welcher die verarbeiteten Datensätze persistent abgespeichert werden. Diese regelmäßige Abfrage der Preis-APIs ist durch die Nutzung der Go-Routine automatisierbar und wird im Rahmen dieser Arbeit auf eine Iteration pro Stunde konfiguriert. Durch diesen Aspekt der Automatisierung ist die Bereitstellung dauerhaft aktueller Daten sichergestellt.

4.6 Konzeption der Datenverarbeitung

Der Prototyp muss in der Lage sein die erhaltenen Antworten der Preis-APIs einzulesen und zu analysieren. Diese Datensätze müssen dann in eine einheitliche Datenstruktur transformiert werden. Diese Struktur soll folgende Elemente beinhalten und unabhängig von der Dienst-Art verwendbar sein:

- Anbieter
- Dienstfamilie
- Dienstnummer / Identifikationskennung
- Name der Ressource
- Preis pro Einheit
- Einheit

Die Cloud-Ressourcen der verschiedenen Anbieter müssen in dieser einheitlichen Struktur in einer SQL-Datenbank abgespeichert werden. Um dabei die Erstellung redundanter Daten zu vermeiden ist vor der erfolgreichen Abspeicherung eine Überprüfung notwendig. Dieser Ablauf könnte wie in Abbildung 4.3 implementiert werden.

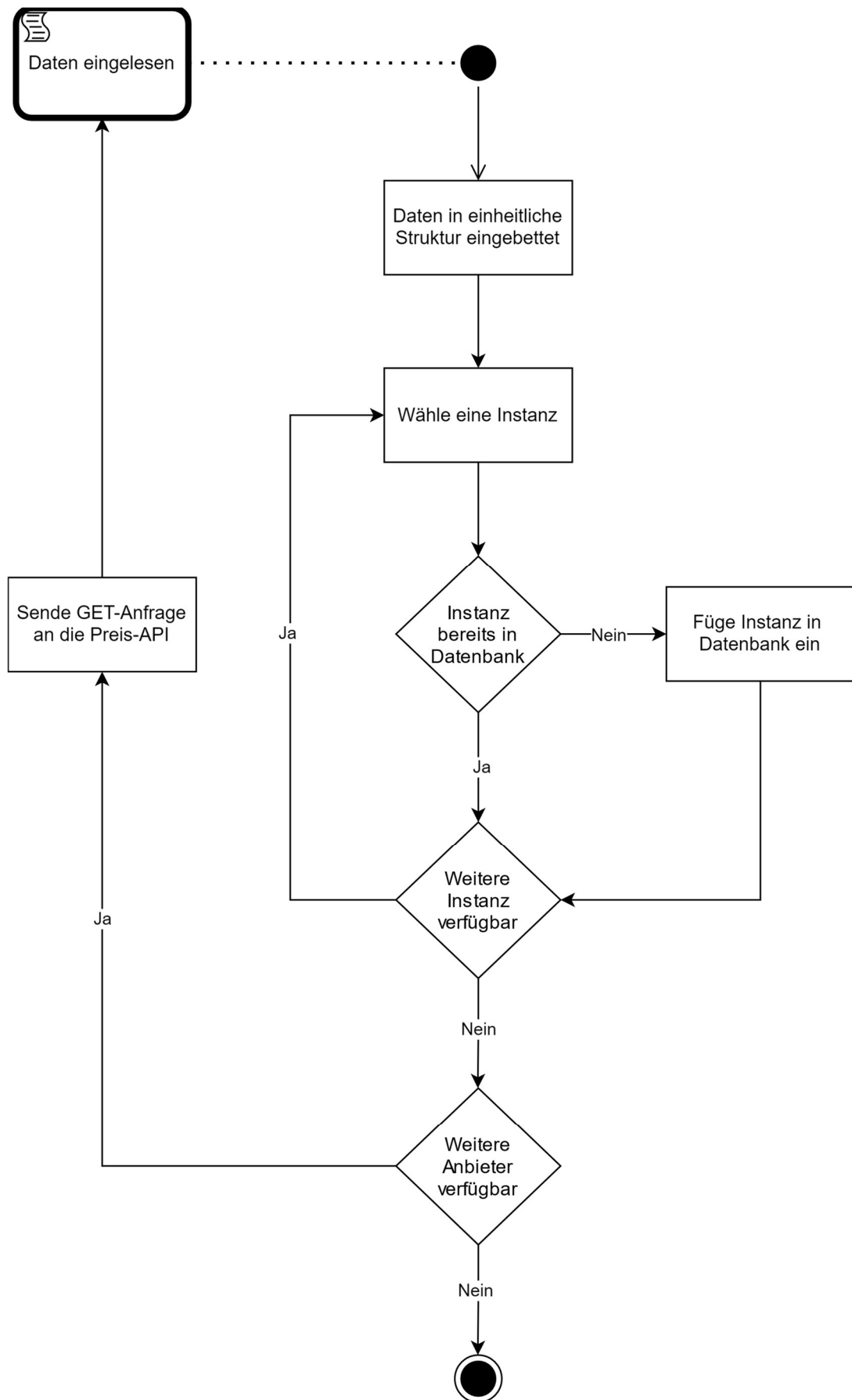


Abbildung 4.3: Ablauf der Datenverarbeitung

4.7 Konzeption der Datenaufbereitung

Die abgespeicherten Daten sollen in einer Tabelle ausgegeben werden. Das soll den Vergleich der verschiedenen Ressourcen ermöglichen. Die Tabelle soll in einer lokalen Webserver-Applikation aufrufbar sein. Dabei sollen die Attribute der einheitlichen Datenstruktur aus Abschnitt 4.6 dargestellt werden. In Abbildung 4.4 ist eine potenzielle Darstellung der Datensätze visualisiert. Die vereinheitlichten Daten sollen durch eine Filterfunktion in der Tabelle geordnet werden können. Optionen für die Filterung sind die folgenden:

- Anbieter
- Dienst-Art
- Dienstnummer
- Name der Ressource

Web-Applikation

Anbieter

Alle
AWS
MSA
GCP

Dienst

Alle
Compute
Speicher
Datenbank

Dienstnummer

Ressourcenname

Anbieter	Dienst	Dienstnummer	Ressourcenname	Preis pro Einheit	Einheit
AWS	EC2	A123456789	AWS-Ressource	\$0.01	Stunde
MSA	Virtual Machines	B123456789	MSA-Ressource	\$0.001	Tag
GCP	Compute Engine	C123456789	GCP-Ressource	\$0.0001	Minute

Abbildung 4.4: Entwurf der Benutzeroberfläche

5 Implementierung

Dieses Kapitel umfasst eine Beschreibung des Implementierungsprozesses des Prototypen. Es umfasst die wesentlichen Entwicklungsschritte, die zur Umsetzung des gewählten Designs benötigt sind. Die Entwicklungsschritte können in vier Punkte gegliedert werden: Datenerlangung, Datenverarbeitung, Datenaufbereitung und Automatisierung.

5.1 Datenerlangung

In diesem Abschnitt erfolgt eine detaillierte Prozessbeschreibung der Datenerlangung. Es wird der Kommunikationsprozess zwischen den Prototypen und den drei verschiedenen Preis-APIs beschrieben. In diesem Abschnitt liegt der Fokus auf den Erhalt der Daten.

5.1.1 AWS-Kommunikation

Um den Preiskatalog von Amazon Web Services zu erhalten, wird eine GET-Anfrage an die AWS Price List Bulk API aus Kapitel 4.4.1 gesendet. Da drei Dienste untersucht werden, sind drei GET-Anfragen erforderlich. Diese müssen zunächst passend konfiguriert werden, indem die Platzhalter innerhalb der URL mit den benötigten Informationen ersetzt werden. Die drei Ziel-URLs werden anschließend für eine vereinfachte Wiederverwendung in einer Go-Map gespeichert, wie in Listing 5.1 dargestellt.

```
urlMap["Amazon EC2"] = "https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonEC2/current/eu-central-1/index.json"

urlMap["Amazon S3"] = "https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonS3/current/eu-central-1/index.json"

urlMap["Amazon RDS"] = "https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonRDS/current/eu-central-1/index.json"
```

Listing 5.1: AWS URL-Map

Der Abruf einer der konfigurierten URLs resultiert in den Erhalt einer JSON-Datei. Die Dateigröße variiert je nach Dienst. So besitzt der Preiskatalog der Computing-Instanzen eine Größe von 302 Megabyte (MB). Währenddessen besteht die Preisliste der S3-Ressourcen aus nur 420 Kilobyte (KB) und die Liste der Datenbankressourcen wiederum aus 14 MB. Die Struktur der verschiedenen JSON-Dateien ist jedoch dienstunabhängig. In Listing 5.2 ist das Grundgerüst dieser Datenstruktur anhand einer S3-Preisliste schaubildlich dargestellt.

```
{
  "formatVersion": "v1.0",
  "disclaimer": "This list is for informational purposes only",
  "offerCode": "AmazonS3",
  "version": "20240711003858",
  "publicationDate": "2024-07-11T00:38:58Z",
  "products": {[+]},
  "terms": {[+]},
}
```

Listing 5.2: Grundstruktur der AWS-Preisliste

Das Feld `publicationDate` gibt das Datum der Veröffentlichung der Preisliste an. In diesem Fall wurde die Preisliste zuletzt am 11.07.2024 um 00:38:58 Uhr aktualisiert. Das Feld mit dem Namen `products` beinhaltet als Wert eine Liste mit allen verfügbaren Artikelnummern, auch bekannt als Stock Keeping Units (SKU). Eine SKU ist ein weiteres Objekt und beinhaltet eine Liste an Attributen einer spezifischen Ressource. In Listing 5.3 ist der Aufbau eines SKU-Objekts dargestellt. Dabei besitzen die Attribute `sku` und `usageType` eine große Bedeutung für diesen Prototypen. Diese Werte sind für die weitere Datenverarbeitung notwendig.


```

"products": {
  "5MVK9WXZCUGU7T4": {
    "sku": "5MVK9WXZCUGU7T4",
    "attributes": {
      "serviceCode": "AmazonS3",
      "transferType": "InterRegion Inbound",
      "fromLocation": "Asia Pacific (Osaka)",
      "fromLocationType": "AWS Region",
      "toLocation": "EU (Frankfurt)",
      "toLocationType": "AWS Region",
      "usageType": "EUC1-APN3-MRAP-In-Bytes ",
      "operation": "MRAP-Dtransfer",
      "fromRegionCode": "ap-northeast-3",
      "servicename": "Amazon Simple Storage Service",
      "toRegionCode": "eu-central-1",
    },
    "E8UGH653MXMRQS97": { [+ ] },
    "RDKN86J3SD7QAJNU": { [+ ] },
  }
}

```

Listing 5.3: Struktur des AWS-Produkt-Objekts

Die dazugehörigen Preisinformationen befinden sich am Ende der Datei. Dies erschwert die manuelle Analyse, da die zugehörigen Werte über mehrere tausend Zeilen verteilt sind. Die Preise befinden sich in einem Überobjekt mit dem Feldnamen `terms` und daraufhin in dem Objekt `OnDemand`, wie in Listing 5.4 erkennbar. Darin ist eine Liste mit allen SKUs enthalten, die die Felder mit allen benötigten Preisinformationen besitzen. Für diesen Prototypen sind die Werte der Felder `unit` und `pricePerUnit` von Bedeutsamkeit.

```

"terms": {
  "OnDemand": {
    "PESBDUG2BX9ZGHNN": { [+ ] },
    "3GU9FU26M8K8CF4N": {
      "3GU9FU26M8K8CF4N.JRTCKXETXF": {
        "offerTermCode": "JRTCKXETXF",
        "sku": " 3GU9FU26M8K8CF4N",
        "effectiveDate": "2024-07-01T00:00:00Z",
        "priceDimensions": {
          {
            "3GU9FU26M8K8CF4N.JRTCKXETXF.6YS6EN2CT7": {
              "rateCode": "3GU9FU26M8K8CF4N.JRTCKXETXF.6YS6EN2CT7",
              "description": "$0.036 per 1,000 PUT",
              "beginRange": "0",
              "endRange": "Inf",
              "unit": "Requests",
              "pricePerUnit": {
                "USD": "0.0000360000",
              },
            }, }, },
          },
        },
      },
    },
  },
}

```

Listing 5.4: Struktur des AWS-Terms-Objekts

Um diese JSON-Datenstruktur erfolgreich in den Prototypen einlesen zu können, ist die Erstellung einer identischen Go-Struktur notwendig. Diese Struktur muss denselben Aufbau wie die JSON-Datei besitzen, um die Werte erfolgreich aus den Feldern entziehen zu können. Wie in Listing 5.5 zu erkennen ist, erlaubt die Nutzung sogenannter JSON-Tags das Entziehen der Werte, durch die Angabe des spezifischen Feldnamens. Das Attribut innerhalb der Go-Struktur erhält während der Auswertung den Wert des angegebenen Feldes zugewiesen. Dieser Wert ist dann innerhalb der Applikation verwendbar ist.

```

type Terms struct {
    OnDemand map[string]map[string]OnDemandDetail `json:"OnDemand"`
}

type OnDemandDetail struct {
    SKU string `json:"sku"`
    PriceDimensions map[string]PriceDimension `json:"priceDimensions"`
}

type PriceDimension struct {
    Unit string `json:"unit"`
    PricePerUnit map[string]string `json:"pricePerUnit"`
}

type AwsPricingData struct {
    Products map[string]AwsProduct `json:"products"`
    Terms Terms `json:"terms"`
}

type AwsProduct struct {
    SKU string `json:"sku"`
    Aws_Instances AwsInstance `json:"attributes"`
}

type AwsInstance struct {
    ID uint
    ServiceID uint
    SKU string
    UsageType string
    Unit string
    PricePerUnit float64
}

```

Listing 5.5: Go-Struktur für die AWS-Preis-API

Die Prozedur des Einlesens der JSON-Datenstruktur ist unabhängig von der spezifischen Preis-API, weshalb sie nun einmalig beschrieben wird. Diese Methode wird jedoch in allen hier implementierten API-Kommunikationen genutzt. Sie ist in Listing 5.6 detailliert dargestellt.

```

response, err := http.Get(url)
if err != nil {
    return models.AwsPricingData{},
        fmt.Errorf("Error fetching AWS data: %v", err)
}
defer response.Body.Close()

body, err := ioutil.ReadAll(response.Body)
if err != nil {
    return models.AwsPricingData{},
        fmt.Errorf("Error reading AWS response body: %v", err)
}

err = json.Unmarshal(body, &awsPricingData)
if err != nil {
    return models.AwsPricingData{},
        fmt.Errorf("Error unmarshaling AWS JSON: %v", err)
}

```

Listing 5.6: Prozedur zur Einlesung der Daten

Die HTTP-Funktion in der ersten Zeile ist in der Go-Standardbibliothek verfügbar. Sie sendet eine GET-Anfrage, an die im Parameter übergeben URL. Die Antwort auf diese GET-Anfrage wird in der Variable `response` abgespeichert.

Die Funktion `ioutil.ReadAll` im zweiten Block liest den Body der Antwort aus und konvertiert die Informationen in Byte-Arrays. Dies ist eine notwendige Voraussetzung für den nächsten Schritt.

Die Funktion `json.Unmarshal` nimmt die Byte-Arrays und die Go-Struktur `AwsPricingData` aus Listing 5.5 als Parameter entgegen. Diese Funktion dekodiert das JSON-Format der Datensätze und speichert diese Daten in die angegebene Struktur. Dies wird durch die vorher besprochenen JSON-Tags ermöglicht, die eine Zuordnung der JSON-Felder zu den Feldern der Go-Struktur gewährleisten.

5.1.2 MSA-Kommunikation

Für die erfolgreiche Nutzung der Azure Retail Prices REST API ist eine passende Konfigurierung der API-Endpunkte notwendig. Aus diesem Grund wird die URL-Map aus Listing 5.1 um die folgenden URL-Strings in Listing 5.7 ergänzt.

```
urlMap["Azure Virtual Machines"] =  
"https://prices.azure.com/api/retail/prices?currencyCode='USD'&$filter  
=serviceFamily eq 'Compute' and armRegionName eq 'westeurope' and  
serviceName eq 'Virtual Machines'"
```

```
urlMap["Azure Storage"] =  
"https://prices.azure.com/api/retail/prices?currencyCode='USD'&$filter  
=serviceFamily eq 'Storage' and armRegionName eq 'westeurope' and  
productName eq 'Blob Storage'"
```

```
urlMap["Azure SQL Database"] =  
"https://prices.azure.com/api/retail/prices?currencyCode='USD'&$filter  
=serviceFamily eq 'Databases' and armRegionName eq 'westeurope' and  
serviceName eq 'SQL Database'"
```

Listing 5.7: MSA-URL-Strings

In Listing 5.8 ist die Grundstruktur der erhaltenen JSON-Preisliste zu erkennen. Das Feld `NextPageLink` ist ein Platzhalter, der in diesem Falle leer ist. Wie der Feldname es bereits erahnen lässt, wird hier die URL abgespeichert, die auf den nächsten Satz an Daten verweist. In diesem Preiskatalog besitzt ein Endpunkt eine maximale Anzahl von 1.000 Einträgen. Sollte es für einen Dienst mehr als 1.000 Ressourcen geben, werden die nächsten Datensätze durch dieses Element verknüpft. Das Feld `Count` gibt die Anzahl der Einträge für den aktuell angesprochenen Endpunkt an.

```
{  
  "BillingCurrency": " USD",  
  "CustomerEntityId": " Default",  
  "Items": [ + ] ,  
  "NextPageLink": null,  
  "Count": 452  
}
```

Listing 5.8: Grundstruktur der MSA-Preisliste

Das Feld `Items` besteht aus einer Liste aus Objekten. Wie in Listing 5.9 zu erkennen, stellt dabei jedes Objekt eine Ressource dar. Dabei können die Ressourcen durch das Feld `skuId` identifiziert werden. Da in jedem Ressourcenblock alle spezifischen Informationen enthalten sind, vereinfacht das die Adaption in eine Go-Struktur.

```
"Items": [
  { [+]},
  {
    "currencyCode": "USD",
    "tierMinimumUnits": 0.0,
    "retailPrice": 360.0,
    "unitPrice": 360.0,
    "armRegionName": "westeurope",
    "location": "EU West",
    "effectiveStartDate": "2016-05-01T00:00:00Z",
    "meterId": "34b1ff9d-ff18-4e62-af3f-9c63b26a4978",
    "meterName": "eDTUs",
    "productId": "DZH318Z0BQH7",
    "skuId": "DZH318Z0BQH7/01HD",
    "productName": "SQL Database Elastic Pool - Premium",
    "skuName": "2000 DTU Pack",
    "serviceName": "SQL Database",
    "serviceId": "DZH3180HX10K",
    "serviceFamily": "Databases",
    "unitOfMeasure": "1/Day",
    "type": "Consumption",
    "isPrimaryMeterRegion": false,
    "armSkuName": "",
  },
]
```

Listing 5.9: Struktur des MSA-Items-Objekts

Die für den Prototyp relevanten Werte sind unter den folgenden JSON-Tags zu finden. Die `skuId` dient zur Identifizierung einer speziellen Ressource. Die Preisinformationen sind unter den Feldern `unitPrice` und `unitOfMeasure` zu finden. Abschließend wird der Wert des Feldes `meterName` genutzt, um diesen als eindeutig identifizierbaren Ressourcennamen zu verwenden. Das Listing 5.10 stellt die Go-Struktur dar, die für die JSON-Datenstruktur der MSA-Preis-API entwickelt wurde.

```
type AzurePriceList struct {
    Items          []Item `json:"Items"`
    NextPageLink string `json:"NextPageLink"`
}

type Item struct {
    SkuId          string `json:"skuId"`
    UnitPrice      float64 `json:"unitPrice"`
    UnitOfMeasure string `json:"unitOfMeasure"`
    MeterName      string `json:"meterName"`
}
```

Listing 5.10: Go-Struktur für die MSA-Preis-API

5.1.3 GCP-Kommunikation

Für die Kommunikation mit der Google Cloud Billing API werden die folgenden URL-Strings aus Listing 5.11 in die URL-Map aus Listing 5.1 hinzugefügt. Diese URL benötigt die genaue Angabe einer Dienst-ID, die durch die Nutzung einer von Google bereitgestellten Suchfunktion leicht gefunden werden kann [30]. Da dieser Prototyp auf drei vergleichbare Dienste eingegrenzt ist, wurde mithilfe SKU-Suchfunktion die folgenden Dienstkennungen eingesammelt:

- Google Cloud Compute Engine: 6F81-5844-456A
- Google Cloud Storage: 95FF-2EF5-5EA1
- Google Cloud SQL: 9662-B51E-5089

```
urlMap["Google Cloud Compute Engine"] =  
„https://cloudbilling.googleapis.com/v1/services/  
6F81-5844-456A/skus/?key=API_Key“  
  
urlMap["Google Cloud Storage"] =  
"https://cloudbilling.googleapis.com/v1/services/  
95FF-2EF5-5EA1/skus/?key=API_Key"  
  
urlMap["Google Cloud SQL"] =  
"https://cloudbilling.googleapis.com/v1/services/  
9662-B51E-5089/skus/?key=API_Key"
```

Listing 5.11: GCP-URL-Strings

Die Grundstruktur des erhaltenen Preiskatalogs ist in Listing 5.12 zu sehen. Dabei sind die Daten wie bei der Azure Retail Prices REST API in Ressourcen-Objekte eingeteilt. Die relevanten Informationen befinden sich hauptsächlich in den Objekten `category` und `pricingInfo`. Das Objekt `category` besteht aus insgesamt vier Attributen, die die Ressource näher beschreiben. Die für diesen Prototypen relevanten Informationen befinden sich in den Feldern `skuId` und `description`.


```

"skus": [
  {
    "name": "services/95FF-2EF5-5EA1/skus/002B-81E7-32DC",
    "skuId": "002B-81E7-32DC",
    "description": "Nearline Storage Finland Dual-region",
    "category": {
      "serviceDisplayName": "Cloud Storage",
      "resourceFamily": "Storage",
      "resourceGroup": "NearlineStorage",
      "serviceDisplayName": "OnDemand",
    },
    "serviceRegions": [+],
    "pricingInfo": [+],
    "serviceProviders": "Google",
    "geoTaxonomy": [+],
  },
  { [+]},
  { [+]},
]

```

Listing 5.12: Grundstruktur der GCP-Preis-API

Wie in Listing 5.13 erkennbar, besitzt die Struktur des Objekts `pricingInfo` zusätzliche Werte für einen programmatischen Währungswechsel. Diese Informationen fallen jedoch außerhalb des Umfangs dieser Arbeit, da der US-Dollar als Basiswährung für den Vergleich gewählt wurde.

Innerhalb dieses Objekts befinden sich erneut Felder, dessen Daten für den Vergleich benötigt werden. Die Felder sind unter den JSON-Tags `usageUnit`, `units` und `nanos` gekennzeichnet. Alle drei Elemente werden für die Berechnung des konkreten Preises benötigt. In Listing 5.14 ist die dazugehörige Go-Struktur dargestellt.

```

"pricingInfo": [
  {
    "summary": "",
    "pricingExpression": {
      "usageUnit": "GiBy.mo",
      "displayQuantity": "1",
      "tieredRates": [{
        "startUsageAmount": 0,
        "unitPrice": {
          "currencyCode": "USD",
          "units": "0",
          "nanos": 11000000,
        }
      }],
      "usageUnitDescription": "gibibyte month",
      "baseUnit": "By.s",
      "baseUnitDescription": "byte second",
      "usageUnitConversionFactor": 2.8759101014016e+15,
    },
    "currencyConversionRate": 1,
    "effectiveTime": "2024-07-28T12:07:16:257331Z"
  },
],

```

Listing 5.13: Struktur des GCP-PricingInfo-Objekts

```

type GcpPricingData struct {
    Name          string    `json:"skuId"`
    Description    string    `json:"description"`
    PricingInfo    []Pricing `json:"pricingInfo"`
}

type Pricing struct {
    PricingExpression PricingExpression `json:"pricingExpression"`
}

type PricingExpression struct {
    UsageUnit    string    `json:"usageUnit"`
    TieredRates  []TieredRate `json:"tieredRates"`
}

type TieredRate struct {
    UnitPrice UnitPrice `json:"unitPrice"`
}

type UnitPrice struct {
    Units string `json:"units"`
    Nanos int    `json:"nanos"`
}

type SKUs struct {
    SKUs []GcpPricingData `json:"skus"`
}

```

Listing 5.14: Go-Struktur für die GCP-Preis-API

5.2 Datenverarbeitung

Dieses Kapitel beschreibt die Verarbeitung der Daten, die im vorherigen Kapitel erlangt wurden. Dabei liegt der Fokus auf der Vereinheitlichung der anbieterabhängigen Daten und dem detaillierten Speicherungsprozess dieser in einer MySQL-Datenbank.

5.2.1 Einheitliche Datenstruktur

Da sich nun alle benötigten Informationen der Preis-APIs in der Anwendung befinden, ist die Entwicklung einer einheitlichen Datenstruktur notwendig. Diese Struktur muss, die in Kapitel 4.6 beschriebenen Attribute enthalten, um eine vereinfachte Kommunikation mit der relationalen Datenbank zu ermöglichen. Die zuvor erstellten anbieterabhängigen Go-Strukturen können nun zu dieser einheitlichen Struktur, wie in Listing 5.15 dargestellt, transformiert werden. Diese Transformation ermöglicht die Speicherung der Daten unabhängig von ihrem ursprünglichen Anbieter in einer identischen Datenstruktur.

```
type Instances struct {  
    ID          uint  
    ServiceID   uint  
    SkuID       string  
    Name        string  
    UnitPrice   string  
    Unit        string  
}
```

Listing 5.15: Vereinheitlichte Go-Struktur

Durch die Nutzung des GORM-Frameworks muss keine manuelle Namenskonvertierung implementiert werden. Die Struktur-Namen werden durch GORM automatisch in ihre Snake-Case Schreibweise konvertiert. Beispiele für diese Konvertierungen sind:

- Instances: instances
- ServiceID: service_id

5.2.2 Speicherungsprozess

Da nun eine vereinheitlichte Datenstruktur innerhalb der Anwendung besteht, kann ein Datenbankschema entwickelt werden, um die Datensätze in diesem Format abzuspeichern. Das dafür entwickelte Schema ist in Abbildung 5.1 dargestellt und besteht aus insgesamt vier Tabellen. Die Werte der ersten Tabelle, `Providers`, sind in Tabelle 5.1 veranschaulicht. Sie beinhaltet die in dieser Anwendung unterstützten Cloud-Anbieter und weist diesen eine eindeutige ID zu. Die zweite Tabelle 5.2 `Services` besitzt die Dienste der Anbieter, die in dieser Anwendung analysiert werden und weist ihnen ebenfalls eine ID zu. Die beiden Tabellen sind durch einen Primärschlüssel und einem Fremdschlüssel miteinander verknüpft. Die dritte Tabelle 5.3 `Service_Types` beinhaltet zusätzliche Informationen für die Service-Tabelle. Durch die Verknüpfung dieser Tabellen ist die Kategorisierung der Dienste in spezifische Dienst-Typen ermöglicht. Die letzte Tabelle `Instances` beinhaltet alle vorhandenen Ressourcen und deren Attribute. Jede Ressource ist durch eine Service-ID mit dem entsprechenden Dienst in der `Services`-Tabelle verknüpft.

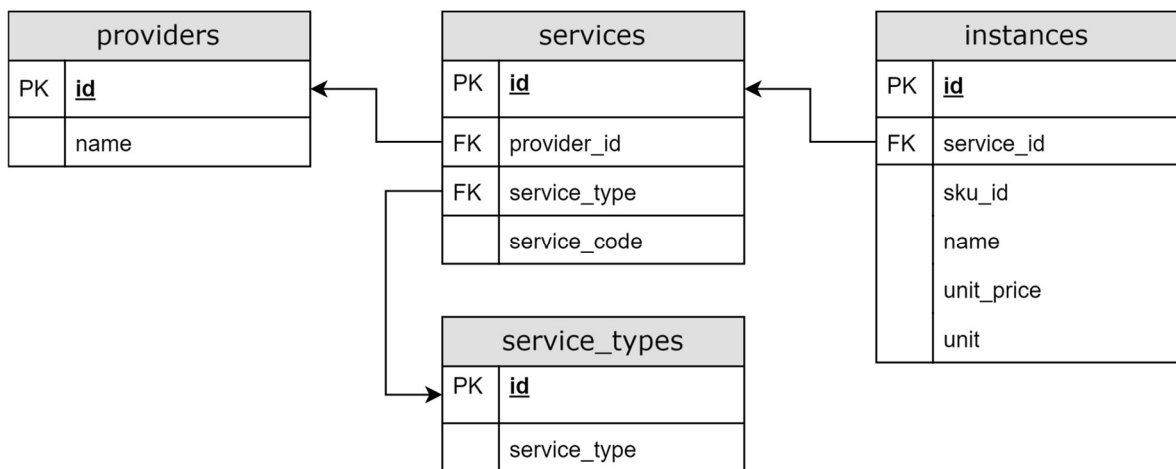


Abbildung 5.1: Datenbankschema

Tabelle 5.1: Anbieter-Tabelle

id	name
1	Amazon Web Services
2	Microsoft Azure
3	Google Cloud Platform

Tabelle 5.2: Dienst-Tabelle

id	provider_id	service_type	service_code
1	1	1	Amazon EC2
2	1	2	Amazon S3
3	2	1	Azure Virtual Machines
4	2	2	Azure Storage
5	3	1	Google Cloud Compute Engine
6	3	2	Google Cloud Storage
7	1	3	Amazon RDS
8	2	3	Azure SQL Database
9	3	3	Google Cloud SQL

Tabelle 5.3: Dienst-Typ-Tabelle

id	service_type
1	Compute
2	Storage
3	Database

In Tabelle 5.2 ist am Ende der Tabelle eine Sammlung der Datenbankdienste zu erkennen. Diese Struktur resultiert aus der nachträglichen Integration dieser Dienste. Dieses Ereignis besitzt für das Kapitel der Validierung beachtliche Bedeutung und wird dort ausführlich erörtert.

Die Abspeicherung der Datensätze gemäß der Konzeption aus Abbildung 4.3 zeigt die Ausführung der SQL-Abfragen ohne die spezifische Angabe eines Anbieters oder eines Dienstes. Diese Implementierung ermöglicht, dass derselbe Speicherungsprozess für alle Anbieter und Dienste verwendet werden kann. Aus diesem Grund wird der Speicherungsprozess exemplarisch anhand der Datensätze der Preis-API von Microsoft Azure detailliert beschrieben.

Zur erfolgreichen Nutzung der MySQL-Datenbank ist zunächst ein Verbindungsaufbau benötigt. Durch die Verwendung des GORM-Frameworks ist dafür der Aufruf einer einzelnen Methode notwendig. Der Aufruf dieser Methode ist in Abbildung 5.2 dargestellt.

```
dsn := "root:password@tcp(127.0.0.1:3306)/SkyComputing"
db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{
    Logger: logger.Default.LogMode(logger.Error),
})
if err != nil {
    log.Fatalf("failed to connect to database: %v", err)
}
```

Abbildung 5.2: Datenbankverbindung

Der Speicherungsprozess beginnt mit der Erstellung einer leeren Struktur. Dieses Objekt trägt den Namen `existingInstances` und wird mit dem Ergebnis der SELECT-Abfrage aus Listing 5.16 initialisiert. Das Objekt besitzt nach Abschluss der Abfrage, alle bereits existierenden Instanzen, die dieselbe Dienst-ID besitzen, wie der Dienst, der im Moment aktualisiert wird. Die momentan benötigte Dienst-ID wird vor Aufruf der Funktion anhand eines Parameters übergeben.

```
var existingInstances []*Instances
result := db.
    Table("instances").
    Where("service_id = ?", serviceId).
    Find(&existingInstances)
```

Listing 5.16: Sammlung der existierenden Instanzen

Die erhaltenen Instanzen werden in eine Go-Map konvertiert. Eine Go-Map besteht aus Schlüssel-Werte-Paaren, wodurch die Instanzen nach ihrer SKU geordnet und effizient abgefragt werden können. Dies erleichtert die Überprüfung, ob eine bestimmte Instanz bereits vorhanden ist.

Es folgt eine Iteration durch die erhaltenen Instanzen der Preis-API. Diese sind in Listing 5.17 unter dem Namen `items` deklariert. In jeder Iteration wird eine einzelne Instanz verarbeitet und unter dem Namen `item` referenziert. Zunächst wird geprüft, ob die aktuelle Instanz bereits in der zuvor beschriebenen Go-Map vorhanden ist. Diese Überprüfung erfolgt durch einen Abgleich der SKUs.

Sollte dieser Abgleich zutreffen und die Instanz somit bereits existieren, werden der Name, die Einheit und der Preis der Instanz aktualisiert. Im Falle, dass die Instanz nicht existiert, wird ein neues Objekt initialisiert. Dieses Objekt trägt den Namen `newInstance` und erhält die Attribute der neuen Instanz zugewiesen. Abschließend werden alle neuen und aktualisierten Instanzen in die Datenbank abgespeichert. Dies erfolgt durch den Aufruf der `save`-Methode, die von GORM bereitgestellt wird.

```
for _, item := range items {
    instance, exists := existingInstanceMap[item.SkuId]
    if exists {
        instance.Name = item.MeterName
        instance.UnitPrice = item.UnitPrice
        instance.Unit = item.UnitOfMeasure
    } else {
        newInstance := &Instances{
            ServiceID: serviceId,
            SkuID:      item.SkuId,
            Name:       item.MeterName,
            UnitPrice:  item.UnitPrice,
            Unit:       item.UnitOfMeasure,
        }
        existingInstances =
            append(existingInstances, newInstance)
        existingInstanceMap[item.SkuId] = newInstance
    }
}
```

Listing 5.17: Iteration zur Speicherung der Instanzen

5.3 Datenaufbereitung

Das Kapitel der Datenaufbereitung befasst sich mit der Visualisierung der verarbeiteten Daten in einer lokalen Web-Applikation. Dabei wird die Konzeption aus Kapitel 4.7 verfolgt und eine entsprechende Lösung erarbeitet. Das Ziel hierbei ist die Darstellung der Datensätze mit einer Gewährleistung an Benutzerfreundlichkeit durch die Bereitstellung einer Filterfunktion.

Um den Datenfluss zu beginnen, muss der folgende Endpunkt angesprochen werden:

```
http://localhost:8888/price-overview
```

Nach dem Aufruf dieser URL wird im Backend eine Datenbankabfrage durchgeführt. Diese enthält keine genauen Spezifikationen, da alle Attribute auf ihren Standardwert gesetzt sind. Das bedeutet, dass die Abfrage ohne jegliche Filteroptionen alle verfügbaren Datensätze abrufen. Daraufhin wird das HTML-Template ausgeführt und zusätzlich alle benötigten Daten in einer Struktur an das Template angefügt. Diese Go-Struktur ist in Listing 5.18 erkennbar.

```
data := struct {  
    Instances      []models.Instances  
    AllProviders   []models.Provider  
    AllServices    []models.Service  
    SearchForm     models.InstanceSearchForm  
}
```

Listing 5.18: Go-Struktur im Frontend

Die Werte dieser Struktur werden im Frontend bereitgestellt und können dort dem Nutzer präsentiert werden. Das Objekt `Instances` besteht aus einer Sammlung aus Instanzen, die durch die Datenbankabfrage erhalten wurden.

Die Objekte `AllProviders` und `AllServices` bestehen aus Sammlungen der Strukturen `Provider` und `Service`. Diese Objekte beinhalten die Auswahlmöglichkeiten der Dropdown-Listen, die für die Filterung der Datensätze benötigt werden. Das Objekt `SearchForm` besitzt die Werte, die vom Nutzer für die Filterung ausgewählt wurden. Sollte der Benutzer die Filteroptionen verwendet haben, werden diese in der nächsten Ausführung in die Datenbankabfrage eingearbeitet und dem Nutzer präsentiert.

Die Erstellung der Tabelle erfolgt über die Nutzung der Hypertext Markup Language (HTML) und den Cascading Style Sheets (CSS). Die Entwicklung der grafischen Oberfläche steht hierbei nicht im Vordergrund der Arbeit, weshalb nur ein kurzer Einblick auf die Entwicklung der Tabelle folgt. Der Aufbau der Tabelle kann in Listing 5.19 betrachtet werden. Um die Werte der Instanzen in die Tabelle zu füllen ist eine Iteration durch das Objekt `Instances` notwendig. Dabei werden die Elemente jeder Instanz in eine Tabellenreihe eingefügt bis keine Instanzen mehr übrig sind.

```
{{range .Instances}}  
  <tr data-key="{{.ID}}">  
    <td>{{.Provider}}</td>  
    <td>{{.ServiceID}}</td>  
    <td>{{.SkuID}}</td>  
    <td>{{.Name}}</td>  
    <td>{{.UnitPrice}}</td>  
    <td>{{.Unit}}</td>  
  </tr>  
{{end}}
```

Listing 5.19: Aufbau der Tabelle

Die Optionen der Dropdown-Menüs für die Wahl des Anbieters und des Dienstes sind durch Identifikationsnummern gekennzeichnet. Sie folgen derselben Nummerierung, wie sie in der Datenbank hinterlegt sind (siehe Tabellen 5.1 und 5.2). Der Standardwert für beide Dropdown-Menüs ist 0, was bedeutet, dass keine spezifische Filterung angewendet wird.

Sobald alle benötigten Komponenten im Browser geladen sind, kann der Benutzer die Tabelleneinträge einsehen. Möchte der Nutzer die Einträge filtern oder zur nächsten Seite gelangen, beginnt der Kreislauf erneut. Der Unterschied hierbei liegt in der angesprochenen URL. Da eine GET-Anfrage zur Kommunikation genutzt wird, werden alle benötigten Daten über die URL an den Server gesendet. Die URL wird modifiziert, um dem Server die gewählten Filteroptionen mitzuteilen. Die neue URL, die ab dem zweiten Aufruf verwendet wird, ist wie folgt aufgebaut:

```
http://localhost:8888/price-overview?  
provider=0&service=0&skuField=""&resourceField=""  
&page=1&pageSize=20
```

Die Variable `page` enthält einen Wert zur Lokalisierung der aktuell benötigten Datensätze. Die Variable `pageSize` legt fest, wie viele Einträge pro Seite angezeigt werden sollen. Die URL wird im Backend ausgewertet und die enthaltenen Filter- und Paginierungsparameter werden in die Datenbankabfrage integriert. Die Veranschaulichung dieses Prozesses im Sinne eines Kreislaufs ist in Abbildung 5.3 dargestellt.

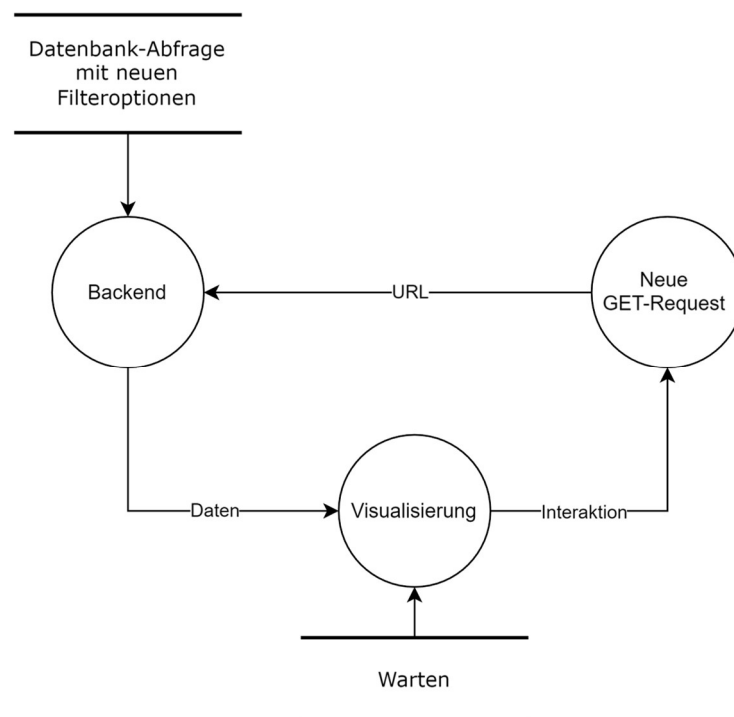


Abbildung 5.3: Datenflussdiagramm

5.4 Automatisierung

Während die Go-Applikation den Webserver hostet und über den zuvor beschriebenen Endpunkt ansprechbar ist, läuft zeitgleich im Backend eine Methode, die für die automatisierte Abfrage der Preis-APIs zuständig ist. Die Methode trägt den Namen `FetchPriceListsHourly` und wird stündlich ausgeführt. Die Konfigurierung der stündlichen Ausführung wird durch die Nutzung der Go-Routine und der `time`-Bibliothek ermöglicht. Die `time`-Bibliothek bietet die Methode `time.NewTicker`, die es ermöglicht, einen Codeabschnitt in regelmäßigen Zeitintervallen automatisch auszuführen. Die Methode `FetchPriceListsHourly` besitzt nun alle Funktionalitäten, die in den Kapiteln der Datenerlangung und der Datenverarbeitung beschrieben wurden. Das Resultat ist eine stündliche Aktualisierung der Preisliste der verschiedenen Cloud-Dienste. Um diesen Prozess nachzuvollziehen, enthält Listing 5.20 eine vereinfachte Version. Zu Beginn wird die Variable `ticker` initialisiert, die ein Zeitintervall in Sekunden festlegt. Sobald dieses Zeitintervall erreicht ist, werden alle Preislisten aktualisiert, und der Ticker setzt das Intervall zurück, um den nächsten Aktualisierungszyklus zu starten.

```
func FetchPriceListsHourly() {  
    ticker := time.NewTicker(1 * time.Hour)  
    defer ticker.Stop()  
  
    for {  
        select {  
        case <-ticker.C:  
            UpdateAmazonWebServicesPriceList()  
            UpdateMicrosoftAzurePriceList()  
            UpdateGoogleCloudPlatformPriceList()  
        }  
    }  
}
```

Listing 5.20: Automatisierungsprozess

6 Validierung

Dieses Kapitel befasst sich mit der Validierung des implementierten Prototyps. Darin wird geprüft, ob die beschriebenen Anforderungen erfüllt werden konnten. Darüber hinaus werden verschiedene Anwendungsfälle der Web-Applikation geprüft und es wird ein Testlauf durchgeführt, um den Automatisierungsprozess zu prüfen.

6.1 Web-Applikation

Dieser Abschnitt validiert die zuvor implementierte Web-Applikation. Der Prototyp ist in der Lage die erhaltenen Datensätze der verschiedenen Cloud-Anbieter einheitlich abzuspeichern und diese über einen Browser tabellarisch darzustellen. Die Datensätze können wie in der Anforderungsanalyse beschrieben gefiltert werden. Alle gewünschten Filteroptionen sind vollkommen funktionsfähig. Darüber hinaus können sämtliche Filtermöglichkeiten in Kombination miteinander verwendet werden. Abbildung 6.1 veranschaulicht den fertigen Prototypen und zeigt, wie die verschiedenen Filteroptionen effektiv genutzt werden können. Mit den ausgewählten Filteroptionen zeigt die Tabelle ausschließlich die Speicherdienste von AWS an. Zudem wurden die Einträge nach ihren Ressourcennamen gefiltert, sodass nur die Ressourcen aufgeführt sind, deren Namen den Begriff „EUC1“ enthalten.

Provider:	Service:	SKU:	Resource Name:
<input type="text" value="AWS"/>	<input type="text" value="Storage"/>	<input type="text"/>	<input type="text" value="EUC1"/>

Provider	Service	SKU	Resource Name	Price per Unit	Unit
AWS	AmazonS3	GX7WDTAN4VSYPH8C	EUC1-CloudFront-In-Bytes	\$0.0000000000	GB
AWS	AmazonS3	3F88WBE2UM7QDP6H	EUC1-USE1-S3RTC-Out-Bytes	\$0.0150000000	GB
AWS	AmazonS3	GVM9PNY2DAV9BNCE	EUC1-CAN2-S3RTC-Out-Bytes	\$0.0150000000	GB

Abbildung 6.1: Filterung eines AWS-Speicherdienstes

In Listing 6.1 ist die zugehörige URL dargestellt. Wie in der Implementierung beschrieben, enthält sie die Werte der gewählten Filteroptionen und informiert den Server mittels einer GET-Anfrage. Die Variable `provider` erhält den Wert 1 zugewiesen, da AWS als Anbieter ausgewählt wurde. Die Variable `service` erhält durch die Filterung nach den Speicherdiensten den Wert 2 zugewiesen. Da in diesem Beispiel keine Artikelnummer angegeben wurde, bleibt die Variable `sku` leer. Der letzte Platzhalter `resourceField` erhält den vom Nutzer angegebenen Begriff, in diesem Fall „EUC1“.

```
http://localhost:8888/price-overview?
provider=1&service=2&skuField=&resourceField=EUC1
```

Listing 6.1: AWS-Speicherdienst-URL

Die Filterung nach einer präzisen Artikelnummer ist ebenfalls möglich. In Abbildung 6.2 ist dieser Fall veranschaulicht. Das SKU-Textfeld ist hierbei mit einer spezifischen Artikelnummer befüllt. Die restlichen Filteroptionen sind auf ihren Standardwert belassen. Das Resultat ist eine spezielle Datenbankressource des SQL-Dienstes der Google Cloud Platform.

Provider:	Service:	SKU:	Resource Name:		
All ▾	All ▾	0192-75D3-E391			

Provider	Service	SKU	Resource Name	Price per Unit	Unit
GCP	Google Cloud SQL	0192-75D3-E391	Cloud SQL for SQL Server: Regional - Standard storage in Los Angeles	\$0.408000000	GiBy.mo

Page 1

Abbildung 6.2: Filterung einer SQL-Ressource der GCP

Alle Funktionalitäten, die in Abschnitt 4.1 an den Prototypen beschrieben wurden, konnten in der Implementierung vollständig realisiert werden. Es wurde eine prototypische Web-Applikation entwickelt, die automatisiert die Preis-API von drei public Cloud-Anbietern abfragt. Die Applikation fragt die Preislisten von Amazon Web Services, Microsoft Azure und Google Cloud Platform für drei spezifizierte Dienste ab. Dabei handelt es sich jeweils um einen Computing-Dienst, einen Speicherdienst und einen Datenbankdienst. Die erlangten Informationen werden in der Applikation verarbeitet, und in einer MySQL-Datenbank abgespeichert. Über einen Endpunkt können die gespeicherten Informationen tabellarisch dargestellt werden. Darüber hinaus wurde eine Filterfunktion implementiert, um eine präzisere Datenanalyse zu ermöglichen. Die Filterfunktion erlaubt die Sortierung nach Anbieter, Dienst, Artikelnummer und Ressourcennamen.

Die nicht-Funktionalen Metriken die an das System gestellt wurden, konnten im Rahmen dieser Arbeit ebenfalls ermöglicht werden. Durch die Vereinheitlichung der erhaltenen Datensätze der verschiedenen Cloud-Anbieter wurde eine wesentliche Grundlage für die Förderung der Interoperabilität geschaffen.

Die Metrik der Erweiterbarkeit kann durch die nachträgliche Integration der Datenbankdienste veranschaulicht werden. Das kreierte Datenbankschema aus Abbildung 5.1 erlaubt die Erweiterung neuer Dienste innerhalb von drei Schritten. Diese Schritte sind in zwei Arbeitsbereiche eingeteilt. Die Datenbank und die Applikation. Um dies zur Schau zu stellen, folgt ein detaillierter Verlauf zur Erweiterung von Amazon RDS in dieses System.

Die erste Arbeitsumgebung, die Datenbank, benötigt die Kenntnis des Dienstes in der Dienst-Tabelle. Dies wird durch die INSERT-Query erreicht, die in Listing 6.2 dargestellt ist. Die Attribute erhalten die angegebenen Werte und der Eintrag wird in der Datenbank erstellt.

```
INSERT INTO services (provider_id, service_type, service_code)
VALUES (1, 3, 'Amazon RDS');
```

Listing 6.2: INSERT-Query

Die zweite Arbeitsumgebung befindet sich im Code der Applikation. Zuerst wird die URL der neuen Preisliste in die URL-Map eingefügt aus Listing 5.1 eingefügt. Daraufhin wird die URL in den Automatisierungsprozess aus Listing 5.20 inkludiert. Durch die Integration der neuen URL in die URL-Map und in den Automatisierungsprozess wird die neue Preisliste in den bestehenden Verarbeitungsablauf eingebunden. Dies umfasst das Einlesen der Preisdaten, die Verarbeitung und die anschließende Speicherung in der Datenbank.

Der gesamte Prozess kann innerhalb von wenigen Minuten durchgeführt werden. Der Aufbau einer neuen Struktur oder sonstigen Objekten ist nicht erforderlich. Dadurch ist die Metrik der Erweiterbarkeit für neue Dienste bestätigt und erfüllt.

6.2 Testlauf

Um zu messen, wie viel Zeit eine Aktualisierung der Preislisten in Anspruch nimmt, wird die Methode `FetchPriceListsHourly` mit Kommandozeilenausgaben erweitert. Die erste Ausgabe erscheint beim Start der Methode. Daraufhin erfolgt eine Ausgabe für jeden abgeschlossenen Anbieter. Die letzte Ausgabe ist am Ende der Methode platziert. In Abbildung 6.3 ist der zeitliche Aufwand hinter einer Aktualisierung zu sehen. Insgesamt beansprucht eine Aktualisierung aller Preislisten circa 64 Sekunden. Dabei nimmt die Aktualisierung der Amazon Web Services Preislisten mit ganzen 51 Sekunden die längste Zeit in Anspruch. Die Preislisten von Microsoft Azure sind in 5 Sekunden aktualisiert und die Listen von Google Cloud Platform in 8 Sekunden.

```
Server is running on port 8888
##### Iteration: 1 #####
Price list update started at: 2024-08-03 16:12:06
Amazon Web Service price list updated at: 2024-08-03 16:12:57
Microsoft Azure price list updated at: 2024-08-03 16:13:02
Google Cloud Platform price list updated at: 2024-08-03 16:13:10
Price list update finished at: 2024-08-03 16:13:10
#####
```

Abbildung 6.3: Zeitlicher Aufwand einer Aktualisierung

Eine zentrale Frage betrifft die Verfügbarkeit des Prototyps. Für einen Automatisierungsprozess ist es von höchster Wichtigkeit kontinuierlich und ohne Abstürze aktiv zu sein. Um dies an diesen Prototypen zu testen, wurde dieser über einen ganzen Tag kontinuierlich betrieben. Um dies zu testen, wurde der Prototyp über einen Zeitraum von 24 Stunden betrieben, um sicherzustellen, dass in jeder Stunde eine Aktualisierung der Preislisten erfolgte. Dabei wurde nur die Kommandozeilenausgabe für den Start und das Ende einer Aktualisierung genutzt. Das Ergebnis dieses Testlaufs ist in Tabelle 6.1 dargestellt.

Der Prototyp wurde über 24 Stunden hinweg kontinuierlich betrieben. Die Applikation wurde um 15:49 Uhr am 05.08.2024 gestartet, wobei die erste Aktualisierung um 16:49:28 Uhr erfolgte. Die letzte Aktualisierung fand um 15:49:28 Uhr am 06.08.2024 in der 24. Iteration statt, woraufhin die Applikation manuell ausgeschaltet wurde. Dabei ist erstmalig eine signifikante Verringerung der Dauer der Aktualisierung in der 8. Iteration aufgetreten. Die Dauer der Aktualisierung in der 7. Iteration betrug 61 Sekunden. In der 8. Iteration, sank diese auf eine Gesamtdauer von 36 Sekunden. Die Gründe für diese Veränderung sowie die erneute Zeitverlängerung in der 13. Iteration sind nicht eindeutig feststellbar. Aufgrund des erheblichen zeitlichen Unterschieds ist anzunehmen, dass dies im Zusammenhang mit den Preislisten von Amazon Web Services steht. Dennoch ist in der Tabelle zu erkennen, dass der Prototyp in der Lage war, über den gesamten Testzeitraum hinweg stabil und ohne Probleme zu laufen. Das umfasst die Kommunikation mit den Preis-APIs der public Cloud-Anbieter als auch die fehlerfreie Verarbeitung und Speicherung der empfangenen Daten. Dieser Testlauf bestätigt, dass der Prototyp die Anforderung der Verfügbarkeit erfüllt.

Tabelle 6.1: Ergebnis des Testlaufs

Iteration	Tag	Start	Ende	Aktualisierungsdauer in Sekunden
1	05.08.2024	16:49:26	16:50:31	65
2	05.08.2024	17:49:26	17:50:31	65
3	05.08.2024	18:49:26	18:50:31	65
4	05.08.2024	19:49:26	19:50:26	60
5	05.08.2024	20:49:26	20:50:26	60
6	05.08.2024	21:49:28	21:50:29	61
7	05.08.2024	22:49:28	22:50:29	61
8	05.08.2024	23:49:28	23:50:04	36
9	06.08.2024	00:49:28	00:50:04	36
10	06.08.2024	01:49:28	01:50:04	36
11	06.08.2024	02:49:28	02:50:05	37
12	06.08.2024	03:49:28	03:50:04	36
13	06.08.2024	04:49:28	04:50:31	63
14	06.08.2024	05:49:28	05:50:04	36
15	06.08.2024	06:49:28	06:50:05	37
16	06.08.2024	07:49:28	07:50:04	36
17	06.08.2024	08:49:28	08:50:03	35
18	06.08.2024	09:49:28	09:50:03	35
19	06.08.2024	10:49:28	10:50:04	36
20	06.08.2024	11:49:28	11:50:04	36
21	06.08.2024	12:49:28	12:50:05	37
22	06.08.2024	13:49:28	13:50:04	36
23	06.08.2024	14:49:28	14:50:04	36
24	06.08.2024	15:49:28	15:50:03	35

7 Fazit

Diese Arbeit hat gezeigt, wie die automatisierte Abfrage der Preis-APIs von public Cloud-Anbietern aussehen kann. Dabei entstand die Kommunikation zu den Preis-APIs der größten public Cloud-Anbieter, diese sind Amazon Web Services, Microsoft Azure und Google Cloud Platform. Die Preisinformationen der Computing-Dienste, Speicherdienste und Datenbankdienste standen im Fokus, um eine Vergleichbarkeit der Preise zu ermöglichen. Dafür wurden die Preise zunächst eingelesen, in eine einheitliche Struktur verarbeitet und persistent in einer MySQL-Datenbank abgespeichert. Dieser Prozess wurde durch die Nutzung der Programmiersprache Go und des GORM-Frameworks automatisiert. Dies führte zur stündlichen Aktualisierung der Preisinformationen in der Datenbank. Um die Vergleichbarkeit der Preise zu veranschaulichen, wurde eine prototypische Web-Applikation mithilfe von Go, HTML und CSS entwickelt. Diese ist über einen lokalen Endpunkt ansprechbar und präsentierte die Datensätze in tabellarischer Form. Zusätzlich wurde eine Filterfunktion implementiert.

Die Validierung dieses Prototyps hat gezeigt, dass das beschriebene Design aus Kapitel 4 genutzt werden kann, um die benötigte Lösung zu implementieren. Dabei konnten alle Anforderungen an den Prototypen erfolgreich realisiert werden. Die Aspekte der Interoperabilität und der Erweiterbarkeit sind dabei große Faktoren, um eine Grundlage für Sky Computing zu schaffen und darauf aufzubauen.

Das Resultat dieser Arbeit besteht aus zwei nützlichen Anwendungen, die sowohl gemeinsam als auch separat genutzt werden können. Es wurde ein Prozess entwickelt, der automatisiert die aktuellen Preisinformationen verschiedener Cloud-Dienste in einer Datenbank speichert. Dieser Prozess kann unabhängig verwendet werden und flexibel an andere Systeme oder Anwendungen angebunden werden. Aufgrund dieser modularen Struktur ist die Integration in bestehende IT-Infrastrukturen möglich. Dies bietet gleichzeitig Raum für zukünftige Optimierungen und Erweiterungen.

7.1 Ausblick

Die erfolgreiche Abspeicherung der Preislisten der verschiedenen Anbieter in einem einheitlichen Format in einer Datenbank besitzt nun sehr viele Möglichkeiten zur Erweiterung. Diese konnten aufgrund des zeitlichen Rahmens dieser Arbeit nicht implementiert werden. Dennoch kann der erfolgreich implementierte Prototyp als Grundlage für diese Erweiterungen genutzt werden. Die Web-Applikation könnte durch die Implementierung der folgenden Funktionalität erweitert werden. Um die Preise in einer grafischen Form darzustellen, könnte eine Funktionalität implementiert werden, um die Datensätze zu exportieren. Hierzu zählen zahlreiche statistische Grafikdarstellungen sowie die Möglichkeit des Exports in verschiedene Dateiformate, wie beispielsweise CSV oder PDF.

In der Zukunft des Sky Computing könnte anhand der hier implementierten Datenverarbeitung und Datenspeicherung folgende Funktionalität entwickelt werden. Um diese Funktionalität zu entwickeln sind die gestellten Forderungen der Sky Computing Architektur dringend zu erfüllen. Es besteht die Möglichkeit eine automatische Überprüfung der gespeicherten Datensätze an den hier implementierten Prototypen anzufügen. Die Kommunikation zu anderweitigen Preis-APIs der Anbieter ermöglicht die Abfrage der anfallenden Kosten eines spezifischen Benutzers. Das System könnte sowohl über die aktuellen Preisinformationen und die derzeit anfallenden Kosten des Benutzers verfügen. Dies ermöglicht den automatisierten Abgleich dieser Werte. Darauf aufbauend könnte ein automatisiertes System entwickelt werden, dass in der Lage ist, die genutzten Ressourcen zwischen den Diensten und Anbietern zu verschieben. Dieses Szenario könnte beispielsweise wie folgt aussehen. Sollte das System ein besseres PreisLeistungsverhältnis, bei einem anderen Anbieter erkannt haben, so sollten alle Informationen automatisch in den neuen Dienst transportiert werden. Um dieses System zu realisieren sind alle Forderungen aus Abschnitt 3.3.1 dringend erforderlich.

Diese Idee stellt nur eine von zahlreichen zusätzlichen Funktionalitäten dar, die durch die Realisierung des Sky Computing entwickelt werden könnten. Sky Computing besitzt ein immenses Potenzial und kann in Zukunft dafür genutzt werden, um neue Anwendungsfelder zu erschließen und bereits bestehende Cloud-Computing-Paradigmen weiter zu optimieren.

Literatur

- [1] Clouddorado – Cloud Computing Comparison Engine. URL:
<https://www.clouddorado.com> (besucht am 16.06.2024)
- [2] BSI – Cloud Computing Grundlagen. URL:
https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html
(besucht am 19.06.2024)
- [3] Intel – Eine Übersicht über Cloud-Bereitstellungsmodelle. URL:
<https://www.intel.de/content/www/de/de/cloud-computing/deployment-models.html> (besucht am 19.06.2024)
- [4] Amazon – Cloud-Modelle. URL: <https://aws.amazon.com/de/types-of-cloud-computing> (besucht am 23.06.2024)
- [5] Amazon – Amazon Web Services. URL: <https://aws.amazon.com/de>
(besucht am 23.06.2024)
- [6] Microsoft – Microsoft 365. URL: <https://www.microsoft.com/de-de/microsoft-365>
(besucht am 23.06.2024)
- [7] Google – Google Workspace. URL: <https://workspace.google.com>
(besucht am 23.06.2024)
- [8] Dropbox – URL: <https://www.dropbox.com> (besucht am 23.06.2024)
- [9] Dreher Consulting – Cloud Computing Nachteile. URL: <https://www.dreher-consulting.com/de/blog/cloud-computing/#nachteile-cloud>
(besucht am 28.06.2024)
- [10] Kubernetes – URL: <https://kubernetes.io/de> (besucht am 30.06.2024)
- [11] Docker – URL: <https://www.docker.com> (besucht am 30.06.2024)
- [12] Apache Spark – URL: <https://spark.apache.org> (besucht am 30.06.2024)
- [13] Apache Flink – URL: <https://flink.apache.org> (besucht am 30.06.2024)
- [14] PyTorch – URL: <https://pytorch.org> (besucht am 30.06.2024)
- [15] Ion Stoica, Scott Shenker. From Cloud to Sky Computing. URL:
<https://dl.acm.org/doi/pdf/10.1145/3458336.3465301> (besucht am 30.06.2024)

- [16] Bernard Marr. The Future Of Computing: Supercloud And Sky Computing – URL: <https://bernardmarr.com/the-future-of-computing-supercloud-and-sky-computing> (besucht am 02.07.2024)
- [17] Hava – Cloud Market Share Analysis. URL: [https://www.hava.io/blog/2024-cloud-market-share-analysis-decoding-industry-leaders-and-trends#:~:text=Who%20has%20the%20biggest%20market,Google%20Cloud%20Platform%20\(GCP\)](https://www.hava.io/blog/2024-cloud-market-share-analysis-decoding-industry-leaders-and-trends#:~:text=Who%20has%20the%20biggest%20market,Google%20Cloud%20Platform%20(GCP)) (besucht am 04.07.2024)
- [18] CloudFlare – Was ist SSH. URL: [https://www.cloudflare.com/de-de/learning/access-management/what-is-ssh/#:~:text=Das%20SSH%20Protokoll%20\(Secure%20Shell,Verschl%C3%BCsselung%20von%20Verbindungen%20zwischen%20Ger%C3%A4ten.](https://www.cloudflare.com/de-de/learning/access-management/what-is-ssh/#:~:text=Das%20SSH%20Protokoll%20(Secure%20Shell,Verschl%C3%BCsselung%20von%20Verbindungen%20zwischen%20Ger%C3%A4ten.) (besucht am 05.07.2024)
- [19] Amazon Web Services – EC2. URL: <https://aws.amazon.com/de/ec2> (besucht am 05.07.2024)
- [20] CloudFlare – Objektspeicher vs. Blockspeicher. URL: <https://www.cloudflare.com/de-de/learning/cloud/object-storage-vs-block-storage> (besucht am 06.07.2024)
- [21] MySQL – URL: <https://www.mysql.com/de> (besucht am 30.06.2024)
- [22] Go – URL: <https://go.dev> (besucht am 20.07.2024)
- [23] Go – Goroutine. URL: <https://go.dev/tour/concurrency/1> (besucht am 20.07.2024)
- [24] GORM – URL: https://gorm.io/de_DE/docs (besucht am 20.07.2024)
- [25] Amazon Web Services – AWS Price List Bulk API. URL: <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/using-the-aws-price-list-bulk-api.html> (besucht am 21.07.2024)
- [26] Amazon Web Services – S3-Speicherklassen. URL: <https://aws.amazon.com/de/s3/storage-classes> (besucht am 22.07.2024)
- [27] Google Cloud – Was ist eine Cloud-Datenbank. URL: <https://cloud.google.com/learn/what-is-a-cloud-database?hl=de> (besucht am 22.07.2024)
- [28] Microsoft Azure – Azure Retail Prices REST API Overview. URL: <https://learn.microsoft.com/en-us/rest/api/cost-management/retail-prices/azure-retail-prices> (besucht am 22.07.2024)
- [29] Google Cloud – Cloud Billing API Documentation. URL: <https://cloud.google.com/billing/docs/reference/rest> (besucht am 22.07.2024)

- [30] Google Cloud – SKU-Suchfunktion. URL: <https://cloud.google.com/skus?hl=de>
(besucht am 23.07.2024)
- [31] Richardson, L., Amundsen, M. & Ruby, S.35 (2013). RESTful Web APIs: Services for a Changing World. „O’Reilly Media, Inc.“
- [32] Amazon Web Services – Setting up price update notifications. URL: <https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/notifications-price-list-api.html> (besucht am 02.08.2024)
- [33] Google Cloud – Bekanntgabe der Preisänderungen für Cloud Storage. URL: <https://cloud.google.com/storage/pricing-announce?hl=de> (besucht am 02.08.2024)
- [34] Baun, C., Kunze, M., Nimis, J. & Tai, (2011). Cloud Computing: Web-basierte dynamische IT-Services (2. Aufl.). Springer-Verlag.
(zitiert auf Seite 1 ff., 27 ff., 46 ff., 54 ff.)
- [35] Mulder, J., S. 4 f. (2020). Multi-Cloud Architecture and Governance: Leverage Azure, AWS, GCP, and VMware vSphere to build effective multi-cloud solutions. Packt Publishing Ltd. (zitiert auf Seite 4)
- [36] Lisdorf, A. (2024). Grundlagen des Cloud Computing: Eine nichttechnische Einführung. Springer-Verlag. (zitiert auf Seite 200)
- [37] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., Zaharia, M. & University of California at Berkeley. (10.02.2009). Above the Clouds: A Berkeley View of Cloud Computing. (zitiert auf Seite 20)
- [38] Baun, C. (2018). Computernetze kompakt (6. Aufl.). Springer-Verlag.
(zitiert auf Seite 199)
- [39] Meinel, C., Schnjakin, M., Metzke, T. & Freitag, M. (2014). Anbieter von Cloud Speicherdiensten im Überblick. Universitätsverlag Potsdam. (zitiert auf Seite 62 f.)
- [40] Seidl, R., Baumgartner, M. & Sneed, H. M. (2024). Software-Metriken: Die Vermessung von Applikationen (2. Aufl.). Carl Hanser Verlag GmbH Co KG.
(zitiert auf Seite 78)
- [41] Holori – Calculator. URL: <https://app.holori.com/compare>
(besucht am 07.08.2024)