Fakultät Druck und Medien
Studiengang Medieninformatik

Bachelor Thesis
zur Erlangung des akademischen Grades Bachelor of Science

Evaluierung von Systemen zur Speicherung und Bereitstellung von Binärdaten im Kontext von Web Services

Gamze Isik

19. Juni 2023

Matrikelnummer: 39307
Bearbeitungszeitraum: 20. März - **19. Juni 2023**

Betreuer

Erstbetreuer: Prof. Martin Goik
Hochschule der Medien

Zweitbetreuer: Thomas Maier
Leomedia GmbH

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich (mit Ausnahme dieser Erklärung) als solches kenntlich gemacht.

Ort, Datum	Unterschrift

Zusammenfassung

Das Ziel der vorliegenden Arbeit ist die Evaluierung eines Systems zur Speicherung und Bereitstellung von Binärdaten im Kontext von Web Services. Diese Binärdaten sollen Kunden des Produkts leoticket der Leomedia GmbH zur Verfügung gestellt werden. Dabei werden Anforderungen wie Performance, Verfügbarkeit, Sicherheit und eine Möglichkeit für Integrationen durch APIs in Software Produkte am Beispiel des Produkts leoticket der Leomedia GmbH dargestellt. Durch die Realisierung eines Prototyps anhand ausgewählter Speicherlösungen werden die Binärdaten durch sichere, zeitlich begrenzte URLs bereitgestellt. Folgende Fragen werden gestellt: Welches Speichersystem ist im Hinblick auf Kosten, Performance und Verfügbarkeit für die Persistenz von Binärdaten besonders geeignet? Wie können Daten durch sichere, zeitlich begrenzte URLs bereitgestellt werden? Verschiedene Speichersysteme werden verglichen und anhand von Kostenkalkulationen bewertet. Durch die Auswertung des Vergleichs werden der Prototyp implementiert und Messungen auf Testdaten durchgeführt. Die Kosten-, und Performance Analyse ergibt, dass die Speicherklassen Standard-IA von AWS sowie die Nearline von GCP die Anforderungen des Produkts leoticket der Leomedia GmbH weitestgehend erfüllen. Sie bieten eine hochverfügbare, kostengünstige und leistungsfähige Speicherung an, die durch verschiedene Einstellungen wie die SSE-KMS Sicherheit bietet und durch SDKs eine gute Integration in eigene Anwendungen verspricht. Durch die bereitgestellten Methoden für die Generierung von signierten, zeitlich begrenzten URLs können die Daten an die Ticketkäufer von leoticket ohne weitere Authentifizierung als Kundensicht zur Verfügung gestellt werden. Diese Feststellung und Empfehlung dient dem Umstieg auf neue Speicherlösungen mit höherer Performance und Sicherheit bei akzeptablen Kosten.

Abstract

The aim of this thesis is to evaluate a system for storing and providing binary data within the context of web services. These binary data are intended to be made available to customers of Leomedia GmbH's leoticket product. Requirements such as performance, availability, security, and integration capabilities through APIs in software products being defined, using the leoticket product as an example. By implementing a prototype based on the selected storage solution, binary data is provided by means of secure, time-limited URLs. The following questions are addressed: Which storage system is particularly suitable for persisting binary data in terms of cost, performance, and availability? How can data be provided through secure, time-limited URLs? Various storage systems are being compared and evaluated providing cost estimations. By evaluating the comparison, the prototype is implemented and measurements are performed on test data. The cost and performance analysis reveals that AWS's Standard-IA storage class and GC's Nearline storage class largely meet the requirements of Leomedia GmbH's leoticket product. They provide highly available, cost-effective, and efficient storage solutions, offering security features such as SSE-KMS and seamless integration into custom applications through SDKs. By utilizing the provided methods for generating signed, time-limited URLs, the data can be made available to leoticket ticket buyers without requiring further authentication. This finding and recommendation aim to facilitate the transition to new storage solutions with improved performance and security at acceptable costs.

Inhaltsverzeichnis

Akronyme

Abb.	Abbildung
ADC	Application Default Credentials
Anm.	Anmerkung
API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command Line Interface
CRR	Cross Region Replication
CRUD	Create, Read, Update, Delete
dt.	deutsch
engl.	englisch
FUSE	Filesystem in Userspace
GB	Gigabytes
GC	Google Cloud
GCP	Google Cloud Platform
GCS	Google Cloud Storage
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
JSON	Javascript Object Notation
KB	Kilobytes
REST	Representational State Transfer
S3	Simple Storage Service
SDK	Software Development Kit
SRR	Same Region Replication
SSE	Server-Side Encryption
SSE-C	Server-Side Encryption with Customer-Provided Keys
SSE-KMS	Server-Side Encryption with AWS Key Management Service
SSE-S3	Server-Side Encryption with S3 Managed Keys
TB	Terabytes
URL	Uniform Resource Locator
USA	United States of America, dt. Vereinigte Staaten von Amerika
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das folgende Kapitel dient der Einführung in die Problemstellung, Motivation sowie Ziele und Vorgehensweisen der vorliegenden Arbeit.

1.1 Problemstellung und Motivation

Die steigende Menge an Daten im Kontext von Web Services, die in verschiedenen Anwendungen generiert werden, stellt eine große Herausforderung dar. Diese Bachelorarbeit richtet sich an die Herausforderung einer Full-Service-Ticketing Software „leoticket“, die vom Unternehmen Leomedia GmbH entwickelt wird. Leomedia GmbH ist ein Unternehmen, das Software für Medienunternehmen wie Zeitungsverlage, Radiosender, Veranstalter, Künstler und Kulturvereine entwickelt.[1] „leoticket“ ist eines der vielen Produkte von Leomedia, das Services wie Online-Kartenvorverkäufe, Abendkassen, den Einlass bei der Veranstaltung, Statistiken, Abrechnungen und die Planung der Veranstaltung realisiert.[2]

Die erfassten Daten umfassen verschiedene Arten von Dokumenten, wie beispielsweise PDF-Dateien, insbesondere Tickets und Rechnungen, die aus dem Kaufprozess über die leoticket Plattform resultieren. Dabei ist es von großer Bedeutung, dass diese Daten von leoticket-Kunden und Leomedia sicher, zuverlässig und schnell gespeichert und abgerufen werden können. Vor diesem Hintergrund stellen sich Fragen nach der Auswahl eines geeigneten Speichersystems, welches die Performance, Verfügbarkeit, Sicherheitsanforderungen und die Möglichkeit der Integration in Software-Produkten wie leoticket erfüllt. Zudem müssen Mechanismen bereitgestellt werden, um den Zugriff der Ticketbesitzer auf die Daten durch sichere, zeitlich begrenzte URL's zu beschränken.

Die Herausforderung von leoticket betrifft die Speicherung und Bereitstellung von Daten wie Tickets und Rechnungen an die Ticketkäufer. Galera Cluster ist eine Multi-Master-Replikationslösung für relationale Datenbanken. Es basiert auf dem Konzept der synchronen Replikation, bei dem mehrere Knoten zu einem Cluster miteinander verbunden. Im Rahmen des Replikationsprozesses werden Daten synchronisiert, wodurch sich die Leistung bei Anfragen verringert, da das Datenbanksystem durch die Ausführung des Replikations- bzw. Synchronisierungsjobs beansprucht wird. Dabei erreicht der Arbeitsspeicher seine Kapazitätsgrenze von 200 GB. Die Hauptaufgaben der Datenbank umfassen beispielsweise die Durchführung von JOINS.

Eine weitere Herausforderung ist die Bereitstellung der Dateien über Email Anhänge. Anhänge dürfen eine bestimmte Speichergröße nicht überschreiten. Wenn Ticketkäufer mehrere Tickets in einer Bestellung tätigen, müssen diese über Email Anhänge bereitgestellt werden.

[1] leomedia-web Leomedia: Bachelor- und Master-Themen, [10].
[2] leomedia-web ebd.

Leomedia plant eine Neugestaltung der leoticket-Anwendung. In dieser Untersuchung werden keine relationalen Datenbanken berücksichtigt. Relationale Datenbanksysteme basieren auf einem festen Schema, das vorab definiert werden muss. Dies kann problematisch sein, wenn die Struktur der erfassten Daten des Produkts leoticket häufig geändert oder erweitert werden muss. NoSQL-Datenbanken bieten in dieser Hinsicht oft mehr Flexibilität, da sie schemalos oder mit flexiblen Schemata arbeiten. Weitere Gründe sind die Skalierbarkeit, Komplexität und der Speicherplatzbedarf. Es ist anzumerken, dass diese Gründe nicht bedeuten, dass relationale Datenbanksysteme grundsätzlich ausgeschlossen werden sollten. Sie sind nach wie vor eine bewährte und weit verbreitete Lösung für viele Anwendungen. Die Entscheidung, ein relationales Datenbanksystem auszuschließen, hängt von den spezifischen Anforderungen und Herausforderungen ab, die vorab für diese Arbeit definiert wurden.

1.2 Zieldefinition und Vorgehensweise

Das Ziel dieser Arbeit besteht darin, anhand der Anforderungen von leoticket eine geeignete Speicherlösung zu empfehlen und die Realisierung eines Prototypen basierend auf den ausgewählten Cloud-Providern zu erstellen. Der Prototyp soll die Bereitstellung vordefinierter Daten durch sichere und zeitlich begrenzte URLs ermöglichen.
Dabei werden folgende Fragen gestellt:

- Welches Speichersystem ist im Hinblick auf Kosten, Performance und Verfügbarkeit für die Persistenz von Daten besonders geeignet?
- Wie können diese Daten durch sichere, zeitlich begrenzte URLs bereitgestellt werden?

Zur Beantwortung werden verschiedene Arten von Speichersystemen untersucht. Dabei erfolgt eine Analyse aktuell verfügbarer Speichertechnologien hinsichtlich ihrer nicht-funktionalen Eigenschaften wie Sicherheit, Verfügbarkeit, Performance und Kosten, die im Zusammenhang mit den verwendeten Technologien entstehen können. Aus funktionaler Sicht wird auch die Möglichkeit der Integration des Speichersystems in Software Produkte und die sichere Bereitstellung der erfassten Dateien betrachtet. Cloud-Provider werden miteinander verglichen und Kosten und Performance Kalkulationen durchgeführt. Die Ergebnisse werden anschließend ausgewertet und interpretiert, um eine geeignete Speicherlösung zu empfehlen.

Der Prototyp wird auf Basis der von Cloud Providern angebotenen ausgewählten Technologien und Empfehlungen eingebunden. Nach der Durchführung von Messungen zur Performance auf Testdaten erfolgt eine Zusammenfassung der Implementierung.

Zum Abschluss werden die Ergebnisse präsentiert, interpretiert und eine Bewertung des Prototyps vorgenommen. Die am Anfang gestellten Fragen der Arbeit werden beantwortet und potenzielle Anwendungen des Prototyps aufgelistet.

2 Speichersysteme

Speichersysteme sind eine entscheidende Komponente der IT Infrastruktur eines Unternehmens. In der heutigen Zeit kann von Speichersystemen kaum abgesehen werden, da Big Data mehr an Wichtigkeit gewinnt. Big Data ist ein Begriff, der sich auf große Mengen an strukturierten, semi-strukturierten und unstrukturierten Daten bezieht, die mit hoher Geschwindigkeit und in unterschiedlichen Formaten generiert, gespeichert und analysiert werden. Diese umfasst Daten, die aus verschiedenen Quellen stammen, wie beispielsweise soziale Medien, Sensoren, Maschinenprotokolle, Transaktionsdaten und Textdokumente. Laut Oracle fallen die bekannten drei V-Begriffe Variety, Volume und Velocity darunter, die im Artikel „Was versteht man unter Big Data?“ erwähnt werden. Sie bieten eine Möglichkeit, große Mengen an Daten zu speichern und zu verwalten, um den Zugriff und die Nutzung zu erleichtern. Es gibt eine Vielzahl an Speichersystemen, die für verschiedene Zwecke konzipiert sind. Durch die große Auswahl in der IT und die stetig anwachsende Innovation stellt sich die Frage, welche Speichersysteme sich für bestimmte Zwecke eignen. Die Wahl des richtigen Speichersystems hängt von den Anforderungen der Applikation ab, beispielsweise der Art der zu speichernden Daten, dem benötigten Zugriff und der Skalierbarkeit. Eine gründliche Analyse der Anforderungen und Kosten ist entscheidend, um die beste Lösung zu finden, die den Bedürfnissen des Unternehmens entspricht.

oracle-bigdata

Im folgenden Kapitel werden die unterschiedlichen Typen von Speichersystemen vorgestellt, wobei der Fokus auf den drei Speicherarten File-, Object- und Block Storage liegt. Im Anschluss daran erfolgt ein Vergleich von zwei Cloud Providern in Bezug auf sichere Speicherung, Hochverfügbarkeit, Performance, Kosten, Integration in Software-Produkte sowie der Dateibereitstellung. Auf Basis dieser Kriterien wird eine Entscheidung darüber getroffen, welcher Provider den Bedürfnissen von leoticket am besten entspricht. Hierbei fließen die Kosten- und Performance-Analysen mit in die Entscheidung ein.

2.1 Arten von Speichersystemen

Die heutige IT-Landschaft bietet eine Vielzahl von Speichersystemen, die je nach Bedarf und Anforderungen ausgewählt werden können. Neben den traditionellen Speichermedien wie File Storage gibt es andere Arten, die in der Cloud oder als lokale Lösungen bereitgestellt werden können. Dazu gehören unter anderem Object Storage und Block Storage. Jede dieser Speicherarten hat ihre spezifischen Vor- und Nachteile und ist für bestimmte Anwendungsfälle besser geeignet.

2.1.1 File Storage

File Storage, auch dateiebenen- oder dateibasierter Storage genannt[1], ist eine Speicherlösung bei der Dateien auf einem Dateisystem gespeichert werden.

> Dieses System wird auch als hierarchischer Storage bezeichnet und gilt als das älteste und am weitesten verbreitete Datenspeichersystem für Direct und Network-Attached Storage. Dateisysteme organisieren Daten in hierarchischen Ordnern und Unterverzeichnissen, beispielsweise auf einem Computer. Dateien werden in der Regel auf einem Server oder einer Festplatte gespeichert und können von mehreren Benutzern gleichzeitig gelesen und geschrieben werden. Hierbei werden die Informationen in einzelnen Verzeichnissen abgelegt und können über den entsprechenden Pfad aufgerufen werden. Um dies zu ermöglichen, werden Metadaten genutzt, die dem System den Standort der Dateien mitteilen, vgl. RedHat. redHat-storage

In der folgenden Abbildung wird die hierarchische Struktur des Dateispeichers visualisiert.

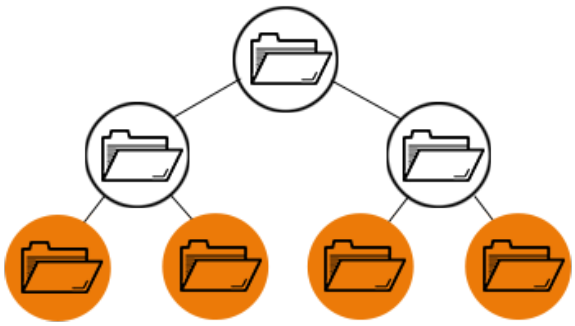

Abbildung 2.1.1: File Storage: Aufbau des Hierarchiesystems, redHat-storage https://www.redhat.com/en/topics/data-storage/file-block-object-storage

File Storage wird häufig in Unternehmen und Organisationen eingesetzt, um gemeinsame Dateiserver bereitzustellen oder Daten in Cloud-Speicherdiensten wie Dropbox oder Google Drive zu speichern. Auch wenn es von Betriebssystemen und Anwendungen gut unterstützt wird, kann die Performance und Skalierbarkeit von File Storage bei sehr großen Dateisystemen beeinträchtigt werden, was insbesondere bei stark frequentierten Anwendungen oder bei der Verarbeitung großer Datenmengen zum Problem werden kann.

[1] redHat-storage RedHat: File storage, block storage, or object storage? (2018), [13].

> Mit zunehmendem Datenvolumen erfordert das Skalieren von Dateispeichern das Hinzufügen neuer Hardwaregeräte oder den Austausch vorhandener Geräte durch solche mit höherer Kapazität. Dies kann im Laufe der Zeit teuer werden. „As data volumes expand, scaling file storage requires [...]“, (Wahlman: Data Storage: Exploring File, Block, and Object Storage (2022), [21], Übersetzung des Autors)

Laut Wahlmann (2022, Übersetzung des Autors) wird die Datenspeicherung bei zu vielen Daten nicht nur teuer, sondern auch unhandlich und zeitaufwändig. Der schnelle und einfache Zugriff auf jede Datei wird schwierig, wenn viele Dateien in sehr vielen Verzeichnissen verteilt auf viele Speichermedien gespeichert werden.

2.1.2 Block Storage

Block Storage in der lokalen Umgebung bezieht sich auf die Speicherung von Daten in Form von Speicherblöcken auf physischen Geräten. Im Gegensatz dazu erfordert File Storage ein Dateisystem, das die Organisation, Verwaltung und den Zugriff auf die gespeicherten Daten ermöglicht. Speichermedien ohne ein Dateisystem, wie eine leere Festplatte oder ein nicht formatiertes Laufwerk, können als Block Storage betrachtet werden, da sie die Daten in Blöcken speichern können, ohne eine spezifische Dateiorganisation zu haben.

Block Storage in der Cloud speichert Dateien auf Cloud-basierten Speicherumgebungen. Die Cloud-basierte Block Storage stellt dabei einzelne Speicherblöcke als Service bereit. Dies geschieht in Form von virtuellen Festplatten oder Blockvolumes, die über die Cloud-Infrastruktur angeboten werden. Bei der Verwendung von Block Storage in der Cloud können Benutzer Speicherblöcke in beliebiger Größe erstellen und diese an virtuelle Maschinen oder andere Ressourcen in der Cloud anhängen. Der Zugriff auf die Speicherblöcke erfolgt über ein Blockprotokoll wie iSCSI oder Fibre Channel.

> Wenn auf Block Storage gespeicherte Daten abgerufen werden, verwendet das Server-Betriebssystem die eindeutige Adresse, um die Blöcke wieder zusammenzufügen und so die Datei zu erstellen. Der Vorteil besteht darin, dass das System nicht durch Verzeichnisse und Dateihierarchien navigieren muss, um auf die Datenblöcke zuzugreifen. Dadurch werden Effizienzen erzielt, da der Abruf von Daten schneller erfolgen kann, vgl. IBM: What is object storage?, [8].

ibm-storage

Typische Anwendungsbereiche des Block Storage sind Datenbanken, Virtualisierungsumgebungen und Anwendungen für Big Data-Analysen. Speicherung von Daten wie Datenbanken, Virtuelle Maschinen und Betriebssysteme eignen sich besonders bei der Verwendung von Block Storage. Sie ermöglichen eine hohe Flexibilität, da Benutzer die Größe und Konfiguration der Speicherblöcke anpassen können, um den individuellen Anforderungen gerecht zu werden. Darüber hinaus bieten sie eine hohe Skalierbarkeit, da zusätzliche Speicherblöcke bei Bedarf hinzugefügt werden können, um den wachsenden Speicherbedarf zu bewältigen. Diese Art von Daten erfordert schnellen und direkten Zugriff auf bestimmte Bereiche des Speichers und muss häufig in Echtzeit ausgeführt werden. Block Storage eignet sich daher am besten für Anwendungen mit hohen Anforderungen an die Leistung und niedriger Latenzzeit.

2.1.3 Object Storage

Object Storage hat sich als Speichertechnologie in den letzten Jahren immer stärker etabliert und wird von vielen Unternehmen als Alternative zu traditionellen Speicherlösungen wie Block- oder File Storage angesehen. Die ersten Object Storage Systeme wurden bereits in den 1990er Jahren entwickelt, aber erst mit dem Aufkommen von Big Data, IoT und der Cloud-Nutzung hat es einen breiteren Einsatz gefunden. Heute bieten viele Cloud Provider wie Amazon Web Services (AWS) und Google Cloud Platform (GCP) Object Storage als einen ihrer Haupt-Cloud-Services an.

> Object Storage ist für den Umgang mit großen Datenvolumen und unstrukturierten Daten entwickelt. Sie speichert Daten als eigenständige Objekte, die aus Daten und Metadaten bestehen und einen eindeutigen Identifier (UID) haben, („Object storage (aka object-based storage) is a type of data storage used to [...]“, dataCore-OS Ivanov: File Storage vs. Object Storage: Understanding Differences, Applications and Benefits, What is Object Storage? (2020), [9], Übersetzung des Autors).

Im Gegensatz zu hierarchischen Systemen wie beim File Storage ist das Speichersystem flach strukturiert. Durch die einfache API Anbindung kann es mit vorhandenen Anwendungen integriert werden. Nutzer können detaillierte Informationen wie beispielsweise Erstellerangaben, Schlüsselwörter sowie Sicherheit-und Datenschutzrichtlinien hinterlegen. Diese Daten bezeichnet man als Metadaten. Laut nx-fileScala Wahlman, 2022 ist Skalierbarkeit der Hauptvorteil, da bei der Speicherung von Petabyte und Exabyte alle Objekte in einem Namespace abgelegt werden. Selbst wenn dieser Namespace auf Hunderten von Hardwaregeräten und Standorten verteilt ist, können alle Objekte schnell abgerufen werden. Ein weiterer Vorteil von Objekt Storage beinhaltet die Fehlercodierung bzw. Fehlerkorrekturverfahren, im Englischen bekannt als „Erasure Coding“.

Auch Object Storage hat Nachteile. Laut redHat-storage RedHat muss das Objekt nach der Speicherung bei Veränderung komplett neu überschrieben werden. Sie sind für traditionelle Datenbanken nicht geeignet, da das Schreiben von Objekten Zeit beansprucht und die Implementierung einer Applikation für die Nutzung der Object Storage API ist aufwendiger als die Nutzung von File Storage, vgl. redHat-storage RedHat.

Insgesamt bietet Object Storage eine skalierbare und flexible Methode zur Speicherung von unstrukturierten Daten. Organisationen sollten jedoch die Vor- und Nachteile von Object Storage im Kontext ihrer spezifischen Anwendungsfälle abwägen, um eine fundierte Entscheidung über die beste Speichermethode zu treffen.

Fazit

Aufgrund der Anforderungen von leoticket, Daten wie Rechnungen und Tickets zu speichern und abzurufen, erweist sich Object Storage als die geeignete Speicheroption. Da keine spezifischen Anforderungen hinsichtlich der Latenzzeit bestehen und auch keine Dateien im Petabyte- oder Exabyte-Bereich gespeichert werden müssen, scheidet die Block Storage-Variante aus. Viele Cloud-Anbieter stellen Methoden für Object Storage bereit, die Sicherheitsfunktionen, hohe Verfügbarkeit, gute Performance und eine Möglichkeit der Integration in Software-Produkten umfassen. Darüber hinaus ermöglichen Object Storage-Systeme die Bereitstellung von Dateien als Links. Die Skalierbarkeit ist ein weiterer entscheidender Faktor, der für die Wahl von Object Storage spricht.

2.2 Aktuelle Speichertechnologien im Markt

Die beiden bekanntesten Cloud-Anbieter Amazon Web Services (AWS) und Google Cloud Platform (GCP) bieten eine Vielzahl von Speicherlösungen an, die auf die Bedürfnisse von Unternehmen zugeschnitten sind.

In diesem Kapitel werden die aktuellen Speichertechnologien auf dem Markt untersucht, wobei der Fokus auf den Angeboten von AWS und GCP liegt. Um eine Vergleichsgrundlage zwischen AWS und GCP zu schaffen, werden die verschiedenen Aspekte wie sichere Speicherung, Hochverfügbarkeit, Leistung, Kosten, API Anbindung und die Bereitstellung der Dateien betrachtet. Dieses Vorgehen dient der Ermittlung der angebotenen Object Storage Speichersysteme der Cloud Provider, die am besten die genannten Anforderungen erfüllen.

AWS bietet eine Reihe von Speicheroptionen an, darunter Amazon S3 (Simple Storage Service). Amazon S3 ist ein Object Storage-Service, der für die Speicherung und den schnellen Abruf von unstrukturierten Daten wie Videos, Fotos und Dokumenten ausgelegt ist. GCP stellt mit Google Cloud Storage (GC Storage) unter anderem eine ähnliche Speicherlösung bereit. Diese ist ein Object-Storage-Service, der für die Speicherung selbiger Daten optimiert ist.

2.2.1 Eigenschaften

Im folgenden Abschnitt werden Amazon S3 und GC Storage in Bezug auf verschiedene Kriterien untersucht. Die Auswahl der Kriterien erfolgt in Anlehnung an die spezifischen Anforderungen von leoticket. Dabei werden zunächst die Eigenschaften von Amazon S3 erläutert, gefolgt von einer Betrachtung von GC Storage. Ziel ist es, Funktionen und Angebote beider Cloud Provider zu untersuchen, um eine Entscheidungshilfe für leoticket zu bieten.

2.2.1.1 Sichere Speicherung

Viele Anbieter von Object Storage-Lösungen bieten integrierte Verschlüsselungsmöglichkeiten an, um sicherzustellen, dass Daten sowohl während der Übertragung als auch in Persistenz geschützt sind. Dabei können unterschiedliche Verschlüsselungsmethoden zum Einsatz kommen. In Bezug auf die sichere Speicherung bieten sowohl AWS als auch GCP Optionen für die Verschlüsselung von Daten. Die Sicherheit der gespeicherten Daten ist von entscheidender Bedeutung, um ihre Vertraulichkeit zu gewährleisten und den Zugriff auf sie einzuschränken.

Amazon S3

IAM (Identity and Access Management) ist ein essentieller Bestandteil von AWS und ermöglicht Benutzer, Gruppen und Rollen zu erstellen, um den Zugriff auf S3 zu verwalten. Benutzern können individuelle Berechtigungen zugewiesen werden, während Gruppen und Rollen mehrere Benutzer mit denselben Berechtigungen zusammenfassen können. Auf S3-Buckets und Objekte kann eine granulare Zugriffssteuerung angewendet werden. Benutzer, Gruppen oder Rollen können so konfiguriert werden, den Zugriff auf bestimmte Buckets und Objekte zu erlauben. "Beim Erteilen von Berechtigungen in Amazon S3 entscheiden Sie, wer die Berechtigungen erhält, für welche Amazon-S3-Ressourcen die Berechtigungen gelten und welche Aktionen zu diesen Ressourcen gestattet werden sollen."[aws-iam-s3] AWS: Security: Identity and Access Management, [3]

Eine weitere Sicherheitsfunktion von Amazon S3 ist die Datenverschlüsselung. S3 bietet eine Vielzahl von Verschlüsselungsoptionen für die serverseitige und clientseitige Verschlüsselung. Da die clientseitige Verschlüsselung für leoticket keine Anwendung findet, wird diese Methode nicht weiter in Betracht gezogen. Die clientseitige Verschlüsselung erfordert die Generierung und Kommunikation eines separaten Schlüssels für jeden Kunden, um auf die Dateien zugreifen zu können. Aus Gründen des Aufwands ist diese Vorgehensweise nicht praktikabel. Daher wird der Schwerpunkt auf die serverseitige Verschlüsselung gelegt. Es existieren drei Verschlüsselungsmethoden: die serverseitige Verschlüsselung mit Amazon S3-verwalteten Schlüsseln (SSE-S3), mit dem AWS Key-Management-Service-verwalteten Schlüsseln (SSE-KMS) und die vom Kunden verwalteten Schlüsseln (SSE-C). Durch diese Optionen haben Benutzer die Möglichkeit, die Verschlüsselung gemäß ihren individuellen Anforderungen anzupassen und somit die Datensicherheit zu gewährleisten.

Laut [aws-iam-s3] AWS nutzen Buckets standardmäßig die SSE-S3 Methode. Für die Verschlüsselung wird 256-bit Advanced Encryption Standard (AES-256) verwendet. Seit dem 5. Januar 2023 sind alle neu erstellen Buckets auf SSE-S3 ausgelegt. Alle neuen Objekte sind beim Hochladen ohne weitere Zusatzkosten und Einbußen der Leistung automatisch verschlüsselt.

> Die Serverseitige Verschlüsselung schützt die Daten bei der Übertragung von und nach S3, ebenso wenn sie auf den Datenträgern in S3-Rechenzentren gespeichert sind. Amazon S3 verschlüsselt jedes Objekt mit einem eindeutigen Schlüssel. Als zusätzliche Sicherheitsmaßnahme werden diese eindeutigen Schlüssel mit einem weiteren Schlüssel verschlüsselt, welches in regelmäßigen Abständen rotiert wird, vgl. [aws-iam-s3] ebd.

Amazon S3 bietet auch die SSE-KMS (Server-Side Encryption with AWS Key Management Service) als Option an. AWS-KMS ist ein Dienst, der ein Schlüsselverwaltungssystem zur Verfügung stellt. Es verschlüsselt die Objektdaten und speichert die S3-Prüfsumme, die sich in den Objektmetadaten befindet, in verschlüsselter Form. Die SSE-KMS kann über die AWS Management-Konsole oder die AWS KMS API verwaltet werden.

Bei der Verwendung von SSE-KMS stehen zwei Methoden zur Verfügung: AWS-Managed-Key oder Customer-Managed-Key. Diese unterstützen die sogenannte „envelope encryption“. Das bedeutet, dass die Schlüssel für die Daten mithilfe eines Master Keys verschlüsselt werden. Dies erleichtert die Verwaltung der Schlüssel.

Bei der Verwendung der AWS-Managed-Key-Variante wird automatisch ein Schlüssel generiert, sobald ein Objekt in einen Bucket hochgeladen wird. Dieser generierte Schlüssel wird anschließend für die Ver- und Entschlüsselung der Daten verwendet. Wenn jedoch ein eigener Schlüssel über KMS verwendet werden soll, muss zunächst ein symmetrischer Schlüssel vor der KMS-Konfiguration erstellt werden. Bei der Erstellung des Buckets kann danach der selbst erstellte Schlüssel angegeben werden. Die Verwendung von Customer-Managed-Keys bietet bestimmte Vorteile, die den Anforderungen von leoticket entsprechen. Durch die Verwendung von selbst erstellten Schlüsseln wird mehr Flexibilität und Kontrolle gewährleistet. Diese Schlüssel können selbst erstellt, rotiert und deaktiviert werden. Zusätzlich können Zugriffskontrollen und Auditierung konfiguriert werden, um den Schutz der Daten zu gewährleisten.

Durch die Auswahl der SSE-KMS-Variante besteht die Möglichkeit, die S3 Bucket Key-Funktion zu aktivieren. Diese Funktion kann die Anfragekosten um bis zu 99 Prozent reduzieren, indem der Anfragenverkehr von Amazon S3 zu AWS KMS verringert wird. Durch die Aktivierung des S3 Bucket Keys werden eindeutige Datenschlüssel für die Objekte im Bucket generiert. Diese Bucket Keys werden für einen festgelegten Zeitraum verwendet, wodurch die Anfragen an Amazon S3 zur Durchführung von Verschlüsselungsoperationen auf AWS KMS reduziert werden.

Die letzte Option ist SSE-C (Server-Side Encryption with Customer-Provided Keys). Bei SSE-C stellt der Kunde seinen eigenen Schlüssel zur Verfügung. Im Gegensatz zu AWS KMS speichert Amazon S3 diesen Schlüssel nicht. Mit dem bereitgestellten Schlüssel übernimmt Amazon S3 die Datenverschlüsselung während des Schreibvorgangs sowie die Datenentschlüsselung beim Zugriff auf Objekte. Anschließend entfernt Amazon S3 den Schlüssel aus dem Speicher. Da Amazon S3 den Schlüssel nicht speichert, wird der zufällig generierte HMAC (Hash-based Message Authentication Code) des Verschlüsselungsschlüssels gespeichert, um zukünftige Anfragen zu validieren. „Note: Amazon S3 does not store the encryption key that you provide.“, aws-sse-c AWS: Using server-side encryption with customer-provided keys (SSE-C), [4].

Object Ownership ist eine weitere Sicherheitsfunktion von Amazon S3. Mit Object Ownership können Benutzer oder Gruppen die Eigentümerschaft von Objekten in S3-Buckets besitzen. Dies bedeutet, dass nur autorisierte Benutzer die Berechtigung haben, Objekte zu löschen oder zu ändern, was die Sicherheit der Daten erhöht. „S3 Object Ownership ist eine Einstellung auf Amazon-S3-Bucket-Ebene, mit der Sie Zugriffskontrolllisten (ACLs) deaktivieren und das Eigentum an jedem Objekt in Ihrem Bucket übernehmen können, [...].“, aws-iam-s3 AWS: Security: Identity and Access Management, [3].

AWS empfiehlt die ACL (Access Control List) auf Bucket-Ebene deaktiviert zu lassen. Alle Objekte eines Buckets gehören so dem Bucket Owner. Laut aws-iam-s3 AWS verfügt Object Ownership über drei Einstellungen, mit denen man die Eigentümerschaft von Objekten steuern kann.

Einstellung	**Gilt für**	**Auswirkung auf Object Ownership**	**Auswirkungen auf ACLs**	**Hochladen akzeptiert**
Bucket-Eigentümer erzwungen(empfohlen)	Alle neuen und bestehenden Objekte	Bucket-Eigentümer besitzt jedes Objekt.	ACLs sind deaktiviert. Bucket-Eigentümer hat das volle Eigentum und die volle Kontrolle.	Uploads mit ACL mit vollem Zugriff des Bucket-Eigentümers oder Uploads, die keine ACL angeben
Bucket-Eigentümer bevorzugt	Neue Objekte	Wenn ein Objekt-Upload die bucket-owner-full-control vordefinierte ACL beinhaltet, gehört dem Bucket Eigentümer das Objekt. Objekte, die mit anderen ACLs hochgeladen wurden, gehören dem Schreibkonto	ACLs können aktualisiert werden und können Berechtigungen erteilen. Wenn ein Objekt-Upload die bucket-owner-full-control vordefinierte ACL enthält, hat der Bucket-Eigentümer Vollzugriff und der Objekt-Writer hat keinen Vollzugriff mehr.	Alle Uploads
Object-Writer (Standard)	Neue Objekte	Der Objekt-Writer besitzt das Objekt	ACLs können aktualisiert werden und können Berechtigungen erteilen. Der Objekt-Writer hat vollen Kontrollzugriff	Alle Uploads

Tabelle 2.2.1: Einstellungen für Object Ownership, aws-iam-s3 https://docs.aws.amazon.com/de_de/AmazonS3/latest/userguide//access-control-overview.html

aws-iam-s3
Laut AWS zeigt die obige Tabelle die Auswirkungen, die jede Einstellung für Object Ownership auf ACLs, Objekte, Objekteigentümer und Objekt-Uploads hat.

Logging ist ein weiterer Aspekt der Amazon S3-Sicherheit. S3 bietet Logging-Optionen, darunter Bucket Logging und Object-Level Logging, die es Benutzern ermöglichen, Zugriffe auf S3-Objekte aufzuzeichnen und zu überwachen. Diese Funktionen sind entscheidend, um verdächtige Aktivitäten zu erkennen. AWS bietet eine Vielzahl von Tools zur Überwachung der Amazon S3 Ressourcen. Darunter die Amazon CloudWatch Alarms, AWS CloudTrail Logs, Amazon S3 Access Logs und die AWS Trusted Advisor.

Google Cloud Storage

Mit Cloud IAM können ähnlich wie in AWS IAM auch rollenbasierte Zugriffsberechtigungen auf bestimmte Ressourcen beispielsweise für Bucket-, und Objektebene umgesetzt werden. Sowohl Rollen als Berechtigungen können in ähnlicher Funktionsweise wie bei AWS erstellt und zugewiesen werden. GC IAM verwendet das Konzept von Rollen, um Berechtigungen zu definieren. Es bietet vordefinierte Rollen wie Besitzer, Bearbeiter und Viewer, die verschiedene Berechtigungen für bestimmte Ressourcen gewähren. AWS IAM verwendet ebenfalls Rollen, um Berechtigungen zu definieren. Es bietet vordefinierte Rollen wie Administratorzugriff und Lesezugriff. Es ist zu beachten, dass beide Dienste unterschiedliche Terminologien und Konzepte verwenden können, obwohl sie ähnliche Funktionen bieten.

Auch GC bietet eine Vielzahl an Sicherheitsfunktionen, um sicherzustellen, dass Daten in der Cloud sicher gespeichert und geschützt sind. Unter anderem die Datenverschlüsselung. GC bietet wie Amazon S3 die serverseitige Verschlüsselung an. Hier werden zwischen Google-managed Keys, Customer-managed encryption keys und die Customer-supplied encryption keys unterschieden. Google-managed encryption ist die Standard Verschlüsselungsoption von GC ähnlich wie die AWS SSE-S3. AWS und GC bieten ähnliche Funktionalitäten der Datenverschlüsselung an. Dies wäre die serverseitige Datenverschlüsselung, bevor die Daten auf die Festplatte geschrieben werden.

> Die Standard Variante verwaltet für den Nutzer die Encryption Keys in ihrem eigenen Key Management System. Auch GC verwendet für die Verschlüsselung die AES-256 wie AWS. Als Nutzer muss man bei dieser Variante keine Einstellungen berücksichtigen. Daten werden automatisch beim Abruf entschlüsselt, vgl. gcp-encrypt Storage: Google-managed encryption keys, [18].

Wenn ein höheres Maß an Kontrolle über die Schlüssel gewünscht wird, steht die Option der Customer-Managed Encryption zur Verfügung, die mit der AWS SSE-KMS customer-managed-Funktion gleichwertig ist. Die Schlüssel werden durch den Cloud KMS (Cloud Key Management Service) erstellt und verwaltet. Der Benutzer speichert diese Schlüssel extern oder in einem HSM-Cluster (Hardware Security Module). Customer-Keys können entweder für einzelne Objekte verwendet werden oder es kann ein Standard-Key für einen Bucket erstellt werden. Ähnlich wie bei den Bucket Keys in AWS können die erstellten Schlüssel zur Verschlüsselung der Objektdaten, zur CRC32C-Prüfsumme des Objekts und für den MD5-Hash verwendet werden. Um zusätzlichen Schutz zu gewährleisten, stehen Service Agents zur Verfügung, auch bekannt als Service Accounts. Diesen Agenten können bestimmte Berechtigungen zugewiesen werden, um Zugriff auf den gewünschten Encryption Key zu erhalten und Objekte zu verschlüsseln.

Schließlich besteht auch die Möglichkeit der Verwendung von Customer-supplied Encryption Keys, vergleichbar zu AWS SSE-C-Variante. Als zusätzliche Sicherheitsebene zu Google-managed encryption keys können Nutzer ihren eigenen AES-256 Encryption Key bereitstellen, welcher in Base64 encoded ist. Bei dieser Variante wird der Schlüssel nicht von GC gespeichert oder verwaltet. Genau wie bei AWS SSE-C müssen die Nutzer diese Schlüssel selber verwalten und speichern. Um zukünftige Requests zu validieren speichert GC einen kryptografischen Hash vom Schlüssel. Jedoch kann der Encryption Key nicht aus dem Hash regeneriert werden.

Darüber hinaus bietet GC auch Zugriffskontrolleinstellungen, mit denen Benutzer genau steuern können, wer auf Dateien in Buckets zugreifen kann. Der Zugriff kann sowohl auf Bucket- als auch auf Objektebene gesteuert werden, wobei Benutzern und Gruppen bestimmte Rollen zugewiesen werden können. Dabei wird zwischen „Uniform Access Control“ (UAC) und „Fine-grained Access Control“ (FGAC) unterschieden. UAC ermöglicht es, ähnlich wie bei der ACL auf Bucket-Ebene in AWS, den Zugriff auf einen gesamten Bucket in GC Storage auf der Ebene von Rollen zuzuweisen. Dabei können verschiedene vordefinierte Rollen wie Leser, Schreiber oder Besitzer verwendet werden, um festzulegen, welche Aktionen ein Benutzer auf dem Bucket ausführen darf. UAC stellt im Gegensatz zu FGAC eine einfachere Methode zur Zugriffskontrolle dar, da die Zugriffsrechte auf Bucket-Ebene vergeben werden. Das bedeutet, dass alle Objekte des Buckets automatisch dieselben Zugriffsrechte erhalten.

FGAC hingegen ermöglicht es, die Zugriffskontrolle auf die Ebene von Objekten oder sogar auf Teile von Objekten herunterzubrechen. Das bedeutet, dass jeder Benutzer oder jede Gruppe individuelle Zugriffsrechte auf bestimmte Objekte oder Teile von Objekten haben kann. Mit FGAC können feingranulare Zugriffssteuerungen implementiert werden. Es ist eine mächtige Methode, aber auch komplexer und zeitaufwändiger zu implementieren als UAC. Beide Cloud Provider empfehlen die ACL auf Objekt-Ebene deaktiviert zu lassen und nur bei speziellen Fällen anzuwenden. Standardmäßig ist die ACL auf Bucket-Ebene eingestellt.

Für das Object Logging in GC Storage stehen verschiedene Methoden zur Verfügung. GC bietet die Möglichkeit, spezielle Logging-Buckets zu erstellen, in denen alle Zugriffsprotokolle für Objekte gespeichert werden. Auch können Storage-Bucket-Audit-Logs aktiviert werden, um detaillierte Informationen über die Aktivitäten in Buckets zu erhalten.

2.2.1.2 Hochverfügbarkeit

Amazon S3

Amazon S3 ist ein skalierbarer und hochverfügbarer Object Storage, der eine Verfügbarkeit von 99.99% garantiert. „Designed to provide 99.999999999% durability and 99.99% availability of objects over a given year.“, AWS: Data Protection in Amazon S3, [2] aws-availability

Dies wird durch die Verwendung von Multi-Availability Zone Architekturen erreicht, die eine automatische Replikation von Daten in verschiedenen physischen Standorten ermöglichen. Die Multi-Availability Zone Architektur von Amazon S3 basiert auf der Aufteilung von Daten in mehrere geografisch getrennte Verfügbarkeitszonen (AZs). Jede AZ besteht aus mehreren physischen Rechenzentren, die sich in einem geografisch getrennten Gebiet befinden. Jede AZ ist vollständig unabhängig und bietet eine hohe Redundanz und Verfügbarkeit. Wenn ein Benutzer ein Objekt in Amazon S3 hochlädt, wird es automatisch in mehrere AZs repliziert, um sicherzustellen, dass das Objekt auch bei Ausfällen in einer AZ weiterhin verfügbar ist. Im Falle eines Ausfalls einer AZ wird Amazon S3 automatisch die Anfragen auf eine andere AZ umleiten, um eine ununterbrochene Verfügbarkeit des Objekts sicherzustellen. Durch die Cross-Region Replikation (CRR) werden Daten automatisch in andere AWS-Regionen repliziert. Dadurch kann eine hohe Verfügbarkeit der Daten im Falle eines Ausfalls einer gesamten AWS-Region gewährleistet werden. Die Same-Region-Replikation (SRR) funktioniert, ähnlich wie die CRR, nur das hier die Daten in einer einzelnen Region auf die verfügbaren AZs repliziert werden.

Zusätzlich zu Multi-Availability Zone Architekturen verwendet Amazon S3 auch Fehlerkorrektur- und Erkennungsmechanismen wie CRC-Prüfungen, um die Integrität von Daten sicherzustellen, damit die gespeicherten Daten stets korrekt sind.

Durch Aktivierung der Versionierung wird jeder Objektversion, die in einem S3-Bucket gespeichert ist, eine eindeutige Versions-ID zugewiesen. Wenn eine Objektversion versehentlich gelöscht oder überschrieben wird, kann die vorherige Version wiederhergestellt werden.

Um eine hohe Verfügbarkeit bereitzustellen, stellt S3 außerdem noch die S3-Transfer Acceleration zur Verfügung. Durch die Verwendung von Amazon S3-Transfer Acceleration können Benutzer die Übertragung großer Datenmengen beschleunigen, indem ein optimierter Netzwerkpfad genutzt wird. Dadurch kann die Verfügbarkeit der Daten verbessert werden, indem Verbindungsprobleme minimiert werden. Um die Leistung von S3 zu überwachen und auf mögliche Probleme reagieren zu können wird CloudWatch bereitgestellt. Auf diese Weise kann die Ausfallzeit minimiert und analysiert werden, zu welchen Zeitpunkten die Verfügbarkeit am geringsten ist.

Insgesamt bietet Amazon S3 eine hochverfügbare und zuverlässige Speicherlösung, die durch Multi-Availability Zone Architekturen und Fehlerkorrekturmechanismen eine Verfügbarkeit von 99,99% gewährleistet.

Google Cloud Storage

Laut Cloud: Cloud Storage Service Level Agreement (SLA), [5] wird die Verfügbarkeit als Service Level Agreement (SLA) angegeben, welches die garantierte Verfügbarkeit des Dienstes definiert. Gemäß dem aktuellen SLA von GC Storage beträgt die garantierte Verfügbarkeit für Multi-Regionale Speicher 99,95% und für Regionale Speicher 99,9%. Diese Zahlen geben an, dass GC Storage darauf ausgelegt ist, eine sehr hohe Verfügbarkeit zu gewährleisten. AWS bietet ähnliche Verfügbarkeiten an. Beide versprechen eine nahezu lückenlose Verfügbarkeit. In der folgenden Tabelle werden diese Werte nochmals gezeigt:

Cloud Service	Monthly Uptime Percentage
Standard storage class in a multi-region or dual-region location of Cloud Storage	>=99.95%
Standard storage class in a regional location of Cloud Storage; Nearline, Coldline, or Archive storage class in a multi-region or dual-region location of Cloud Storage	>=99.9%
Nearline, Coldline, or Archive storage class in a regional location of Cloud Storage; Durable Reduced Availability storage class in any location of Cloud Storage	>=99.0%

Tabelle 2.2.2: Verfügbarkeit der Speicherklassen gemäß Google Cloud Storage SLA, https://cloud.google.com/storage/sla

Es ist jedoch zu beachten, dass die tatsächliche Verfügbarkeit von GC Storage und Amazon S3 von mehreren Faktoren abhängt, einschließlich der spezifischen Konfiguration, dem Datenzugriffsmuster, der Netzwerkverfügbarkeit und anderen betrieblichen Variablen. Es ist daher möglich, dass die tatsächliche Verfügbarkeit in der Praxis leicht von der garantierten Verfügbarkeit abweicht.

Auch Google Cloud Storage bietet die Multi-Region Storage wie die CRR von AWS an. Dies ermöglicht die Speicherung von Daten in mehreren Regionen weltweit. Dadurch werden die Daten redundant repliziert und bleiben auch im Falle eines Ausfalls einer Region verfügbar. Dieses Prinzip ähnelt dem von AWS, denn auch dort werden die Daten automatisch repliziert. Außerdem bietet GC genau wie AWS SRR die Single-Region Replikation an.

Ein weiterer Punkt ist das Monitoring und Fehlererkennung wie z.B. Stackdriver Monitoring, um die Leistung und Verfügbarkeit von GC Storage zu überwachen und auf potenzielle Probleme zu reagieren. Dies gleicht auch dem CRC von AWS.

Zuletzt bietet Object Versioning die Möglichkeit, Objektversionen beizubehalten. Dadurch können vorherige Versionen von Objekten wiederhergestellt werden, falls sie versehentlich gelöscht oder überschrieben werden. Diese Funktionalität entspricht der Objektversionierung von AWS.

2.2.1.3 Kosten

In diesem Abschnitt erfolgt eine Untersuchung der Kostenstruktur von Amazon S3 und GC Storage. Dabei werden die generellen Kostenfaktoren betrachtet, für die Cloud-Anbieter Gebühren erheben. Die Angebote der Free Tier Kontingenten werden dabei nicht betrachtet.

Amazon S3

Amazon S3 erhebt Gebühren für verschiedene Leistungsbereiche, darunter die Speicherung, Anfragen, Datenabrufe, Datenübertragung, Verwaltung, Analyse und Replikation. Die Speicherungsgebühr richtet sich nach der Objektgröße, der Speicherdauer innerhalb eines Monats und der gewählten Speicherklasse. Es stehen verschiedene Speicherklassen zur Verfügung, die für unterschiedliche Anwendungsfälle geeignet sind. Dies sind S3 Standard, S3 Intelligent-Tiering (IA), S3 Standard – Infrequent Access (IA), S3 One Zone – Infrequent Access (IA), S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval (ehemals S3 Glacier) und S3 Glacier Deep Archive. In dieser Arbeit wird S3 Glacier nicht behandelt, da die Abrufzeiten der Objekte zu lange Zeit in Anspruch nehmen, was den Anforderungen von leoticket nicht entspricht. Stattdessen werden die Speicherklassen S3 Standard, S3 IA, S3 Standard - IA und S3 One Zone - IA näher untersucht. Zur Veranschaulichung werden die Preise in der folgenden Tabelle dargestellt:

	Standard	**Intelligent Tiering**	**Standard-IA**	**One Zone-IA**
Speicher/GB im Monat	0,023 EUR	0,023 EUR	0,013 EUR	0,010 EUR
PUT-,COPY-,POST-,LIST-Anforderungen (pro 1000)	0,0050 EUR	0,0050 EUR	0,0093 EUR	0,0093 EUR
GET-,SELECT und alle anderen Anforderungen (pro 1000)	0,00040 EUR	0,00040 EUR	0,00093 EUR	0,00093 EUR
Datenwiederherstellung	–	–	0,00093 EUR	0,00093 EUR
Monitoring und Automation pro Objekt	–	0,0000023 EUR	–	–
Datenübertragung aus S3 in das Internet	0,084 EUR pro GB			

Tabelle 2.2.3: Übersicht der Kosten der AWS S3 Speicherklassen

In der vorliegenden Tabelle sind die Standardpreise für verschiedene Speicherklassen von Amazon S3 aufgeführt. Die Preise für PUT-, COPY-, POST- und LIST-Anforderungen basieren auf einer Staffelung von 1000 Anfragen, ähnlich wie bei GET-, SELECT- und anderen Anforderungen.

Sowohl die Speicherklasse Standard als auch Intelligent Tiering (IA) haben eine Speicherungsgebühr von 0,023 Euro und Anforderungsgebühren von 0,0050 Euro für POST-Anforderungen und 0,00040 Euro für GET-Anforderungen. Für die Speicherklassen Standard-IA und OneZone-IA liegen die Speicherungsgebühren bei 0,013 Euro bzw. 0,010 Euro. Die Anforderungsgebühren betragen 0,0093 Euro für POST-Anforderungen und 0,00093 Euro für GET-Anforderungen. Zusätzlich erheben beide Speicherklassen eine Datenwiederherstellungsgebühr von 0,00093 Euro. Es fallen auch Datenübertragungsgebühren von Amazon S3 zum Internet in Höhe von 0,084 Euro pro GB für alle Speicherklassen an. Es sei darauf hingewiesen, dass nur die IA-Speicherklasse zusätzliche Gebühren in Höhe von 0,0000023 Euro pro Objekt für das Monitoring und Verwaltung von Objekten erhebt.

Auch für die Replikation fallen in Amazon S3 Gebühren an. Bei der Replikation von Daten fallen zusätzlich zu den Speicherungskosten Kosten für die primäre Kopie der Daten, für Replikations-PUT-Anforderungen und Speicherabruf an. Bei CRR wird auch für den regionenüber-greifenden Datentransfer OUT von S3 zu jeder Zielregion gezahlt. Die Preise für Speicher- und PUT-Anfragen für die replizierte Kopie basieren auf den ausgewählten AWS-Zielregionen, während die Preise für Datenübertragungen zwischen den Regionen auf der AWS-Quellregion basieren. Wenn man S3 Replication Time Control nutzt, wird eine Datenübertragungsgebühr für die Replikationszeitsteuerung sowie Gebühren für S3-Replikationsmetriken erhoben, die zum selben Tarif abgerechnet werden wie angepasste Amazon-CloudWatch-Metriken. Die Kosten für die S3 Replication Time Control-Datenübertragung beträgt pro GB 0.015 USD. Unter AWS Preise können diese Informationen abgerufen werden.

Im Zusammenhang mit der Objektversionierung entstehen Kosten für die Beibehaltung älterer Versionen von Objekten, da zusätzlicher Speicherplatz benötigt wird. Es fallen keine direkten Kosten für das Tagging von benutzerdefinierten Metadaten an, die den Objekten zugeordnet sind. Es ist jedoch möglich, dass Kosten für das Abrufen von Tags über die S3-API (z.B. mit den ListObjects- oder GetObject-Operationen) anfallen, da dies als Datenabruf betrachtet wird und entsprechende Gebühren gemäß den AWS-Preisen für Datenzugriff erhoben werden.

Die genauen Preise und Kostendetails für S3 können sich im Laufe der Zeit ändern. Es wird empfohlen, die aktuellsten Informationen auf der offiziellen AWS-Preisseite oder im AWS-Kostenrechner unter AWS Calculator zu überprüfen, um eine grobe Kosteneinschätzung zu erhalten.

GC Storage

GCS setzt Preise für die Komponenten Datenspeicher, Datenverarbeitung und Netzwerknutzung. Beim Datenspeicher ist, wie bei Amazon S3, die Menge der in Buckets gespeicherten Daten relevant. Preise für Speicher hängen von der Speicherklasse der Daten und dem Standort der Buckets ab. Bei der Verarbeitung werden Gebühren für die von Cloud Storage durchgeführte Verarbeitung, einschließlich Vorgangsgebühren der Klasse A und B, Abrufgebühren und Replikation zwischen Regionen erhoben. Die Vorgangsgebühren der Klasse A beinhalten die PUT-Anforderungen während die Klasse B die GET-Anforderungen beinhalten. Bei der Netzwerknutzung werden Gebühren für die Menge der aus den Buckets gelesenen oder zwischen diesen verschobenen Daten erhoben. GC Storage bietet vier Speicherklassen an, von denen drei untersucht werden. Diese sind die Standard-, Nearline-, und Coldline Storage Klassen. Diese drei Speicherklassen haben unterschiedliche Leistungseigenschaften und Kosten. Die Archive Storage Speicherklasse wird dabei nicht berücksichtigt, da sie hauptsächlich der Archivierung von Daten ausgelegt ist und dem leoticket Use Case nicht entspricht.

Die nachfolgende Tabelle zeigt den Unterschied der Speicherklassengebühren von GC Storage und dient als Vergleich zu Amazon S3:

	Standard	**Nearline**	**Coldline**
Speicher/GB im Monat	0,021 EUR	0,012 EUR	0,0054 EUR
Vorgänge Klasse A pro 1000 Vorgänge	0,0047 EUR	0,0093 EUR	0,0093 EUR
Vorgänge Klasse B pro 1000 Vorgänge	0,00037 EUR	0,00093 EUR	0,0047 EUR
Datenabrufe pro GB	–	0,0093 EUR	0,019 EUR
Ausgehender Traffic von GC ins Internet monatlich	0,11 EUR pro 0 bis 1TB		

Tabelle 2.2.4: Übersicht der Kosten der GC Storage Speicherklassen

Die Kosten der Speicherung von GC sind im Gegensatz zu AWS abweichend verteilt. Hier gibt es kaum Unterschiede zwischen der Standardklassen beider Provider und der Nearline mit der Standard-IA. Jedoch ist die One Zone-IA mit 0,010 Euro um 46% teurer als die Coldline von GC. Für die Klasse A Vorgänge verlangt GC Gebühren in Höhe von 0,0047 Euro für Standard und 0,0093 Euro für die Nearline und Coldline. Letztere zwei Klassen haben die gleichen Kosten wie die Standard-, und One Zone-IA. Die GC Standard Klasse ist um 6% billiger als die S3 Standard. Für die Klasse B fallen Gebühren in Höhe von 0,00037 Euro für die Standard Klasse, 0,00093 Euro für die Nearline und 0,0047 Euro für Coldline an. Auch hier gibt es zwischen Standard GC und S3 Preisunterschiede von 7.5% und zwischen Coldline und One Zone-IA von 80.43%.

Es fallen extra Datenabrufgebühren für die Nearline und Coldline pro GB an. Hier sind die Gebühren um bis zu 95% höher als bei S3. Zuletzt werden für ausgehende Anfragen von GC ins Internet 0,11 Euro pro 0 bis 1TB Speicher erhoben, was 23.63% teurer als AWS ist.

Ähnlich wie bei Amazon S3 sind die Speicherkosten der Nearline und Coldline geringer als in der Standard Speicherklasse. Dafür werden Gebühren für die Datenabrufe bei Nearline und Coldline erhoben. Die verschiedenen Speicherklassen sind für unterschiedliche Anwendungsfälle ähnlich wie bei Amazon S3 konzipiert.

Die Standard Storage-Klasse beider Provider bietet hohe Performance, niedrige Latenzzeiten und hohe Verfügbarkeit. Sie eignet sich für häufig abgerufene Daten, auf die schnell zugegriffen werden muss und die geringe Latenzen aufweisen. Beispiele dafür sind Datenbanken und aktive Anwendungen, die im Vordergrund auf einem System ausgeführt werden und Benutzerinteraktionen erfordern oder Daten verarbeiten.

Die Nearline Storage-Klasse ist für seltener abgerufene Daten konzipiert, auf die jedoch mit niedriger Latenzzeit zugegriffen werden muss. Sie bietet niedrigere Speicherkosten als die Standard Storage, jedoch mit einer etwas längeren Zugriffszeit. Sie eignet sich für Backup-Daten, Archivierung, lange Speicherung und bietet ähnliche Funktionen wie die Standard-IA von S3.

Die letzte Speicherklasse Coldline Storage ist für Daten ausgelegt, auf die selten zugegriffen wird und bei denen eine längere Zugriffszeit akzeptabel ist. Sie bietet die niedrigsten Speicherkosten und eignet sich für langfristige Archivierung, Compliance-Daten und Backup-Daten.

Zusätzlich werden für Replikation von Daten Gebühren erhoben, die nicht in der Tabelle aufgelistet sind. Auf Wunsch können Nutzer in GC Storage Daten innerhalb einer Region, in Dual-Regionen oder Multiregionen replizieren. Zusätzlich zu den Daten, die in den hochgeladenen Objekten enthalten sind, werden benutzerdefinierte Metadaten auf die monatliche Speichernutzung angerechnet. Für die benutzerdefinierten Metadaten `NAME:VALUE` erfasst GC Storage beispielsweise jedes Zeichen in `NAME` und `VALUE` als Byte, das mit dem Objekt gespeichert wird.

Beim vorzeitigen Löschen eines Objekts aus den Speicherklassen Coldline und Nearline werden Gebühren erhoben, da eine Mindestspeicherdauer von 30 Tagen für Nearline und 90 Tagen für Coldline angesetzt wird. Das vorzeitige Löschen beim Nearline beträgt pro GB pro Tag ca. 0.00040 EUR und beim Coldline 0.00019 EUR. Auch für Tags fallen Gebühren pro Monat von 0.00046 EUR an, die auf Buckets angewendet werden.

Auch bei GC fallen Kosten für die Objektversionierung an. Die Kosten setzen sich aus zwei Hauptkomponenten zusammen: Speicher und Anfragen. Jede Version eines Objekts belegt Speicherplatz im Bucket. Die Kosten basieren auf der Größe der Objekte und der Anzahl der gespeicherten Versionen. Das Hochladen, Herunterladen oder Löschen von Objektversionen führt zu Anfragen an den Storage-Dienst. Für diese Anfragen können Gebühren anfallen, die sich nach ihrer Anzahl der Anfragen richten. Es ist zu beachten, dass die Kosten für die Objektversionierung zusätzlich zu den regulären Kosten für die Speicherung und den Datenverkehr in GC Storage anfallen. Daher sollten die potenziellen Kosten der Objektversionierung in die Kalkulation einbezogen werden.

2.2.1.4 Performance

Amazon S3

Amazon S3 bietet Funktionen und Dienste, welche Performance und Skalierbarkeit der Datenzugriffe verbessern. In der offiziellen Dokumentation S3: Performance Guidelines for Amazon S3, [16] werden Best Practices Guidelines empfohlen, welche die Leistung erhöhen können. Es wird empfohlen, HTTP Analyse Tools zu verwenden, um die Leistung von DNS lookup times, Latenzen und die Datentransfer-Geschwindigkeiten zu messen. performance-guide

Das Amazon CloudFront ist ein Content-Delivery-Network-Service (CDN), welche die Bereitstellung von statischen und dynamischen Webinhalten wie .html, .css, .js und Bilder auf Nutzer beschleunigt. Dadurch kann die Leistung von Webanwendungen verbessert werden, vgl. AWS: CloudFront Use Cases: Accelerate static website content delivery, [1]. aws-cdn

Eine andere Methode, welche die Leistung erhöhen kann, ist die horizontale Skalierung von Connections. Da Amazon S3 als ein sehr großes verteiltes System gilt, können Requests über getrennte Verbindungen verteilt werden, um die Bandbreite zu maximieren. „Amazon S3 doesn't have any limits for the number of connections made to your bucket.“, S3: Performance Guidelines for Amazon S3, [16]. Außerdem verspricht Amazon S3, dass Requests beim wiederholten Mal schneller sind, da sie einen anderen Pfad als beim ersten Request nehmen. „[...] if the first request is slow, a retried request is likely to take a different path and quickly succeed.“, ebd. performance-guide performance-guide

Weitere Funktionen, die zur Leistungserhöhung beitragen, sind S3 Transfer Acceleration, S3 Select und die S3 Cross-Replication. Die S3 Transfer Acceleration ermöglicht schnelle, einfache und sichere Übertragungen von Dateien über große geografische Distanzen hinweg zwischen dem Client und S3 Buckets. Die Daten werden über eine optimierte Route an Amazon S3 weitergeleitet. Diese Funktion ist für Daten in Größe von Gigabytes zu Terabytes geeignet, die regelmäßig verschickt werden müssen. Um die Dauer der Anfragen zu messen, bietet Amazon S3 das S3 Transfer Acceleration Speed Comparison Tool an, um beschleunigte und nicht-beschleunigte Uploads zu messen. Unter S3 Accelerate Speedtest kann der Vergleich zwischen aktivierter und deaktivierter Transfer Acceleration beobachtet werden. S3 Select ermöglicht das effiziente Abrufen von spezifischen Daten aus Objekten in Amazon S3. Anstatt ein gesamtes Objekt herunterladen zu müssen, können mit S3 Select nur die benötigten Daten abgefragt werden. Dies reduziert den Datenverkehr und beschleunigt den Abrufvorgang erheblich, insbesondere bei großen Daten.

Die im Abschnitt Hochverfügbarkeit (2.2.1.2) erwähnten CRR und SRR Techniken können ebenfalls für eine höhere Performance beitragen, indem Daten in andere Regionen repliziert werden und für Nutzer näher erreichbar sind.

Das AWS SDK stellt eine einfache API und wird regelmäßig gewartet, um die neuesten Technologien anzubieten. Das SDK beinhaltet automatische Retry Requests bei HTTP 503 Fehlern. Es beinhaltet auch den Transfer Manager, der dafür sorgt Connections automatisch horizontal zu skalieren. Damit können tausende von Requests pro Sekunde verschickt werden.

GC Storage

Auch GC Storage ermöglicht (analog zu CRR und SRR von AWS) die Auswahl des geeigneten Speicherorts für Daten, um die Latenzzeiten zu minimieren. Es kann zwischen multi-regionalen Speicherstandorten oder regionalen Speicherstandorten gewählt werden, um Daten näher an den Benutzer zu bringen, um den Zugriff zu beschleunigen.

Das Pendant von CloudFront von GCP ist der Cloud CDN (Content Delivery Network). Diese bietet einen ähnlichen Service wie CloudFront von AWS. Durch die Integration mit Content Delivery Networks (CDNs) kann die globale Verteilung von Daten optimiert und die Bereitstellung beschleunigt werden. Die CDN stellt eine Zwischenspeicherung von Inhalten in Edge-Servern weltweit bereit, um den Zugriff auf Daten schneller zu gestalten.

Bei der Performance hat die Request Rate eine relevante Rolle. Das bietet das Auto-Scaling von GCP an. Cloud Storage ist ein multi-tenant Service, was bedeutet, dass Benutzer die zugrunde liegenden Ressourcen gemeinsam nutzen. Um gemeinsame Ressourcen optimal zu nutzen, haben Buckets eine anfängliche I/O Kapazität.
„Cloud Storage is a multi-tenant service, meaning that users share the same set of underlying resources.[...]", gcp-autoscale Cloud: Request rate and access distribution guidelines: Auto Scaling, [6] (Übersetzung aus Google Cloud)

> Diese I/O Kapazitäten betragen etwa 1000 Schreibzugriffsanfragen pro Sekunde für Objekte, einschließlich Hochladen, Aktualisieren und Löschen von Objekten. Es ist zu beachten, dass Cloud Storage auch eine kleinere Begrenzung für wiederholte Schreibvorgänge mit demselben Objektnamen hat. Auf Lesezugriffe betragen die Kapazitäten etwa 5000 Anfragen pro Sekunde, einschließlich Auflisten von Objekten, Lesen von Objektdaten und Lesen von Objektmetadaten, vgl. gcp-autoscale ebd.

Wenn die Anfragehäufigkeit für ein bestimmtes Bucket steigt, skaliert Cloud Storage automatisch und erhöht die I/O Kapazität für diesen Bucket, indem die Anfragelast auf mehrere Server verteilt wird.

Cloud Storage ermöglicht den parallelen Upload und Download von Daten, um die Übertragungsgeschwindigkeit zu maximieren. Es können mehrere Threads oder Prozesse verwenden werden, um Daten gleichzeitig hochzuladen oder herunterzuladen und so die Leistung zu verbessern. Dies ähnelt dem horizontalen Skalieren von AWS, bei dem mehrere Verbindungen auf Buckets hergestellt werden können.

Der GC Storage Transfer Service ermöglicht den schnellen und effizienten Transfer von großen Datenmengen. Daten können aus anderen Cloud-Speicherlösungen, On-Premise-Speichern oder öffentlichen Datensätzen in GC Storage übertragen werden, um Zeit und Bandbreite zu sparen. Diese Vorgehensweise entspricht auch dem Transfer Acceleration von AWS, bei denen große Datenmengen schneller an andere Ziele übertragen werden.

GC empfiehlt Einstellungen, um die Performance zu optimieren. Um die Leistung messen zu können, bietet GC das `perfdiag` Tool an. Der Autor McAnlis empfiehlt in seinem Blog Optimizing your Cloud Storage performance: Google Cloud Performance Atlas[gcp-blog] dieses Tool zu verwenden, dass eine Reihe von Tests durchläuft, welche die aktuelle Performance von einem Cloud Bucket protokolliert. Des Weiteren empfiehlt er die Nutzung des `gsutil` Tools von GC, um kleinere Dateien schneller hochzuladen. Wenn 20 000 Dateien, die jeweils 1kb groß sind hochgeladen werden, dauert der Overhead bei jedem individuellen Upload länger als die gesamte Uploadzeit. Deshalb sollte man Batch Operationen verwenden, da diese den Overhead reduzieren und die Leistung verbessern. Das `gsutil` Tool bietet die Option an, Batch Operationen durchzuführen.

Auf dem folgenden Diagramm wird ein Test mit hundert mal 200 000 Dateien mit individuellem Upload und Batch Upload angezeigt, das mit `gsutil -m cp` in ein Storage Bucket hochgeladen wird:

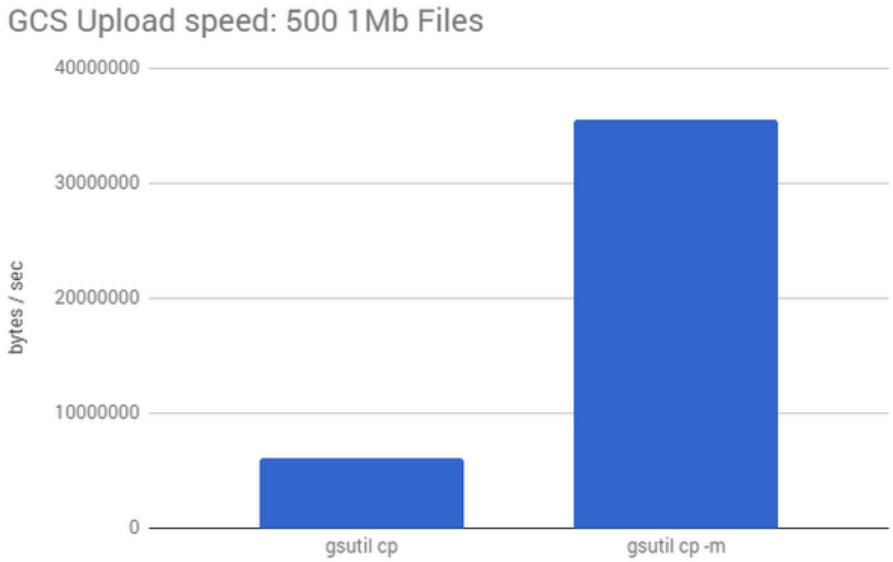

Abbildung 2.2.1: GCS Uploadgeschwindigkeit in Bytes pro Sekunde, [gcp-blog] https://cloud.google.com/blog/products/gcp/optimizing-your-cloud-storage-performance-google-cloud-performance-atlas/

Hier sieht man, wie die Leistung sich dabei um das fünffache im Gegensatz zu den indivduellen Uploads erhöht. Das Auto-balancing von GC dient der Verteilung der Upload-Connections auf "Backend Shards". Das funktioniert durch Name und Pfad der Datei und kann die Uploadgeschwindigkeit verringern, wenn sich die Dateien in unterschiedlichen Ordnern befinden oder verschiedene Namensgebungen haben. Falls sich die Dateien ähneln, kann dies die Uploadgeschwindigkeit erhöhen, da die Connections in den gleichen Shard übergehen.

Eine weitere Methode zur Performancesteigerung, sind Requests in Größen ab 1MB. Bei kleineren Requests sollten die Requests parallelisiert werden, damit die fixen Latenzkosten überlappt werden. Es ist zu beachten, dass die Performance auch von anderen Faktoren abhängt, wie der Anwendungsarchitektur und der Netzwerkverbindung. Es können spezifische Konfigurationsoptionen genutzt werden, um die Leistung weiter zu optimieren, wie Caching, asynchrone Operationen oder die Nutzung von optimierten Bibliotheken oder Frameworks.

2.2.1.5 API Anbindung

Amazon S3

Die API-Anbindung von Amazon S3 ist umfangreich und bietet Entwicklern eine Vielzahl von Möglichkeiten zur Interaktion mit dem Speicherdienst. Amazon S3 bietet eine RESTful API (Application Programming Interface) sowie verschiedene SDKs (Software Development Kits) und Tools, welche die Integration und Nutzung erleichtern.

Mit der RESTful API, die auf dem HTTP-Protokoll basiert, können Entwickler HTTP-Anfragen wie GET, PUT, POST und DELETE verschicken, um auf Buckets und Objekte zuzugreifen, zu erstellen, zu lesen, zu aktualisieren und zu löschen. Amazon empfiehlt die Verwendung des AWS SDK, da bei Verwendung der REST API zusätzlicher Code zur Berechnung der gültigen Signatur für die Authentifizierung geschrieben werden muss. „It requires you to write the necessary code to calculate a valid signature to authenticate your requests.“, aws-api S3: Amazon S3 REST API Introduction, [14].

Die Authentifizierung von Requests mit dem AWS SDK erfolgt durch die Bereitstellung von Access Keys, wodurch das Schreiben von Code dafür entfällt. Das SDK ist in verschiedenen Programmiersprachen verfügbar, darunter Java, JavaScript, Python, .NET, Ruby, PHP, iOS und Android. Die folgende Tabelle enthält eine Auflistung aller unterstützten Programmiersprachen:

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for Javascript	AWS SDK for Javascript code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS SDK for Python(Boto3)	AWS SDK for Python(Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for Swift	AWS SDK for Swift code examples

Tabelle 2.2.5: Unterstützte Programmiersprachen von AWS SDK, aws-sdk https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingAWSSDK.html

Das SDK bietet eine Schnittstelle, um die Entwicklung von Anwendungen zu erleichtern und die Interaktion mit Amazon S3 zu vereinfachen. Darüber hinaus gibt es auch Drittanbieter-Tools und Open-Source Bibliotheken, welche die Integration mit Amazon S3 unterstützen. Spring Boot stellt eine Bibliothek für die alte und neue Version von AWS SDK zur Verfügung. IntelliJ Idea bietet den Plugin `AWS Toolkit` an, mit dem Amazon Credentials zur Verfügung gestellt werden können. Dieses Plugin gibt es auch für Eclipse und Visual Studio Code.

Außerhalb des AWS SDK stellt Amazon S3 auch das AWS CLI (Command Line Interface) bereit, um API Aufrufe durchzuführen. Amazon S3 ermöglicht die Durchführung von Batch-Operationen, um mehrere Objekte in einem einzigen API-Aufruf zu verarbeiten. Dies erleichtert die effiziente Verarbeitung großer Datenmengen und reduziert die Anzahl der API-Anfragen.

Die API-Anbindung von Amazon S3 erfordert eine geeignete Authentifizierung und Autorisierung. Dies erfolgt in der Regel mithilfe von Zugriffsschlüsseln, die über AWS Identity and Access Management (IAM) verwaltet werden.

Google Cloud Storage

Auch GC Storage bietet umfangreiche API-Integrationen, um die Einbindung in eigene Anwendungen zu ermöglichen. Eine davon sind die Client Libraries für verschiedene Programmiersprachen wie Java, Python, Node.js, Go, Ruby und .NET. Diese Bibliotheken erleichtern die Integration von GC Storage in Anwendungen und bieten eine benutzerfreundliche Schnittstelle für die Interaktion mit den Speicherressourcen. Auch bei Cloud Storage wird eine RESTful API ähnlich wie in Amazon S3 angeboten. Diese ermöglicht Entwickler, über HTTP-Anfragen auf die Speicherressourcen zuzugreifen. Sie unterstützt CRUD Operationen für Buckets und Objekte sowie erweiterte Funktionen wie das Setzen von Metadaten, das Verwalten von Zugriffssteuerungen und das Durchführen von Batch-Anfragen.

Google Cloud Storage bietet sowohl JSON- als auch XML-APIs für die Interaktion mit dem Speicherdienst. Neben der direkten API-Anbindung stellt Google auch das GC SDK zur Verfügung, das verschiedene Befehlszeilentools enthält, mit denen auf Speicherressourcen zugegriffen und verwaltet werden kann. Neben den Client-Bibliotheken werden auch Tools wie `gsutil` (ein Befehlszeilen-Dienstprogramm) bereitgestellt. Dies ermöglicht, Cloud Storage von der Befehlszeile aus zu verwalten. Cloud Storage FUSE ermöglicht, Cloud Storage als Dateisystem zu mounten und direkt darauf zuzugreifen.

Die API-Anbindung von GC Storage erfordert ebenfalls eine Authentifizierung und Autorisierung. Dies wird mithilfe von GC IAM (Identity and Access Management) verwaltet, das Zugriffsrichtlinien und Rollenverwaltung bietet, ähnlich wie bei AWS.

Mit Terraform können Konfigurationen von GC Storage Buckets und Amazon S3-Buckets und die dazugehörigen Einstellungen in einer Terraform-Konfigurationsdatei spezifiziert werden. Diese Konfigurationsdatei enthält Informationen wie den Bucket-Namen, Zugriffsrechte, Verschlüsselungsoptionen und andere relevante Parameter. Terraform ist ein sogenanntes Infrastructure-as-Code-Tool, mit dem Infrastrukturressourcen in verschiedenen Cloud-Umgebungen, einschließlich GC und AWS, automatisiert erstellt und verwaltet werden können. Terraform sendet erforderliche API-Anfragen an GC oder AWS um den Bucket entsprechend der Konfiguration zu erstellen oder zu aktualisieren. Durch die Verwendung von Terraform für beide Cloud Provider kann die Infrastruktur als Code behandelt werden, wodurch die Konfiguration wiederholbar, versionierbar und leicht reproduzierbar wird. Auch können die Vorteile der Funktionen von Terraform genutzt werden, wie z.B. die Verwaltung von Abhängigkeiten zwischen Ressourcen, die Verwendung von Variablen und Modulen sowie die Integration in CI/CD-Pipelines.

2.2.2 Bereitstellung der Dateien

In diesem Abschnitt werden Methoden für die Bereitstellung der Dateien durch signierte URLs von Amazon S3 und Cloud Storage präsentiert. Unternehmen und Organisationen stehen vor der Herausforderung, Dateien sicher und effizient für ihre Benutzer bereitzustellen. Eine häufig dafür verwendete Methode, ist die Verwendung signierter URLs. Diese ermöglichen es, auf einfache Weise temporäre Zugriffsrechte für Dateien zu gewähren, ohne komplexe Zugriffskontrollmechanismen implementieren zu müssen. Sowohl AWS S3 als auch GC Storage bieten die Möglichkeit, signierte URLs für die Dateibereitstellung zu generieren. Beide bieten ähnliche Funktionen an, signierte URLs durch die SDKs zu generieren.

Amazon S3

In Amazon S3 sind alle Objekte und Buckets standardmäßig privat. Durch die Verwendung von presigned URLs können Objekte bereitgestellt werden, ohne die Voraussetzung, ein AWS Konto zu besitzen. „All objects and buckets are private by default. However, you can use a presigned URL to share objects with others.“, aws-signed-urls S3: Using presigned URLs, [17].

> Presigned URLs werden für die Generierung von Links verwendet, um auf S3 Buckets und Objekte zugreifen zu können. Bei der Erstellung solcher URLs können andere Nutzer von außerhalb AWS ohne Credentials auf die gewünschten Daten zugreifen. Jeder, der diesen Link hat, kann darauf zugreifen und Aktionen ausführen. Der Link verliert je nach Konfiguration nach einer bestimmten Zeit die Gültigkeit. Amazon S3 überprüft das Verfallsdatum des Links während des HTTP Requests, vgl. aws-signed-urls ebd.

Durch die Nutzung der URL können Nutzer Objekte lesen oder hochladen. In Fall von leoticket liegt das Lesen des Objekts im Fokus. Die URL beinhaltet spezielle Parameter, welche von einer Anwendung gesetzt werden. Es gibt drei Parameter, um den Zugriff für den Nutzer einzuschränken. Diese sind einmal Einschränkung des Buckets. Das bedeutet, das ein Nutzer nicht auf jeden Bucket zugreifen kann, sondern nur auf den Bucket, indem sich das Objekt befindet. Der zweite Parameter ist der Name des Objekts und zuletzt die Zeitspanne in der die URL gültig ist. Sobald die Zeit abgelaufen ist, kann der Nutzer nicht mehr auf den Link zugreifen.

Durch die Methode der presigned URLs können diese Links über die Email an die Clients versendet werden. Somit kann die Bereitstellung der erfassten Daten des Produkts leoticket über die presigned URLs stattfinden und Email-Anhänge vermieden werden.

Google Cloud Storage

In GC Storage funktioniert die signed URL gemäß einem ähnlichen Prinzip wie in Amazon S3. Durch zeiteingeschränkte Links können Nutzer, welche den Link besitzen, ohne GC Credentials auf Objekte und Buckets zugreifen. Dabei bietet Cloud Storage mehrere Methoden zur Generierung einer Signed URL an. Die V4 signing with service account authentication, Signing with HMAC authentication und die V2 signing with service account authentication, vgl.[2]. Letztere Methode wird ausgeschlossen, da sie von GC selbst nicht empfohlen wird. Ein Beispiel, wie eine signierte URL aussehen kann:

```
1 https://storage.googleapis.com/[BUCKET_NAME]/[OBJECT_NAME]?GoogleAccessId=[
    SERVICE_ACCOUNT_EMAIL]&Expires=[EXPIRATION_TIMESTAMP]&Signature=[URL_SIGNATURE]
```

Hierbei sind die Platzhalter wie folgt zu ersetzen:

- BUCKET_NAME - Der Name des GCS-Buckets, in dem sich das Objekt befindet.
- OBJECT_NAME - Der Name des Objekts, für das die signierte URL generiert werden soll.
- SERVICE_ACCOUNT_EMAIL - Die E-Mail-Adresse des Service Accounts, der die Zugriffsberechtigung hat.
- EXPIRATION_TIMESTAMP - Das Ablaufdatum der signierten URL in Form eines Unix-Timestamps.
- URL_SIGNATURE - Die Signatur der URL, die den Zugriff autorisiert.

Im Vergleich zu AWS unterscheiden sich die Parameter bei der Zugriffsberechtigung. In AWS wird der `ACCESS_KEY_ID` statt der Service Account Email verwendet. Die Access Key ID ist dabei der Zugriffschlüssel des AWS-Kontos, das die Zugriffsberechtigung hat.

[2] `gc-signedUrl` Storage: Signed URLs - Options for generating a signed URL, [20].

2.3 Auswahl des Speichersystems

In diesem Kapitel werden anhand der Anforderungen von leoticket die Kostenanalyse durchgeführt und Konfigurationseinstellungen empfohlen.

2.3.1 Kostenanalyse

Bei der Kostenanalyse werden die Gebühren aus dem Kostenabschnitt (2.2.1.3) übernommen und eine Kostenabschätzung für Amazon S3 und Cloud Storage durchgeführt. Es werden die Kosten für die Speicherung, das Abrufen und die Datenübertragung verglichen und dargestellt. Die Berechnungsdaten sind exemplarisch ausgewählt und dienen der groben Kosteneinschätzung. Dabei wurden die aktuellsten Gebühren aus der offiziellen Dokumentation von AWS und GC übernommen.

Für eine grobe Kosteneinschätzung wird von durchschnittlich 800 000 Bestellungen pro Jahr ausgegangen. Dies entspricht etwa 67.000 Bestellungen pro Monat , wobei jede Bestellung 4 Objekte enthält. Die durchschnittliche Größe eines Objektes beträgt dabei 100 KB. Für jede Bestellung wird von durchschnittlich drei Ticketkäufen ausgegangen, bei denen eine zusätzliche Rechnung hinzugefügt wird. Dadurch werden monatlich etwa 268.000 POST-Anfragen an Buckets verschickt. Außerdem wird erwartet, dass Objekte zweimal im Monat von leoticket Kunden abgerufen werden. Insgesamt wird es 536.000 GET-Anfragen geben. Es ist zu beachten, dass jedes Objekt einzeln hochgeladen wird.

Die Speichergröße wird durch die Multiplikation der Anzahl von 67.000 Bestellungen pro Monat mit der Größe von 400 KB pro Bestellung ermittelt. Dieses Ergebnis wird durch 1024 geteilt und das resultierende Ergebnis erneut durch 1024 geteilt, um die Größe in GB zu erhalten. Dies ergibt eine monatliche Speichergröße von 25 GB. Unter Berücksichtigung einer Aufbewahrungsdauer von 10 Jahren ergibt sich eine geschätzte Datenmenge von etwa 3 TB, wenn die Daten nach 10 Jahren gelöscht werden. Für die Preisberechnung werden diese Daten in die AWS Preisrechner und GC Preisrechner eingesetzt.

Amazon S3

Im Folgenden stellt die Tabelle die monatlichen Kosten der verschiedenen Speicherklassen dar. Die Vorabkosten, die unten berechnet werden, müssen an AWS gezahlt werden:

	Standard	**Intelligent Tiering**	**Standard-IA**	**One Zone-IA**
Speicherkosten	70,27 EUR	70,27 EUR	38,72 EUR	30,98 EUR
Vorabzahlung	162,42 EUR	162,42 EUR	300,76 EUR	300,76 EUR
PUT Requests	1,35 EUR	1,35 EUR	2,50 EUR	2,50 EUR
GET Requests	0,22 EUR	0,22 EUR	0,50 EUR	0,50 EUR
Datenabrufe	–	–	0,47 EUR	0,47 EUR
Data Transfer	4,20 EUR			
Monitoring und Automation objects	–	75,19 EUR	–	–

Tabelle 2.3.1: Übersicht der einzelnen Kosten der Datenspeicherung in Amazon S3

Zunächst werden die Einheitsumwandlungen durchgeführt. Hierbei wird die angegebene Datenmenge von 3 TB pro Monat mit 1024 GB multipliziert, um die genaue Menge in GB zu berechnen. Das ergibt einen Wert von 3072 GB. Daraufhin wird die Objektgröße von 100 KB in GB umgerechnet, was einem Wert von 0,000095367432 GB entspricht. Anschließend werden die Preise anhand dieser berechneten Werte ermittelt:

$$\frac{3072 \text{ GB per Month}}{0,000095367432 \text{ GB average item size}} = 32.212.255 \text{ number of objects} \quad (3.1)$$

$$3072 \text{ GB} \times 0,013 \text{ EUR} = 38{,}72 \text{ EUR Standard-IA costs} \quad (3.2)$$

$$268.000 \text{ PUT requests} \times 0,0093 \text{ EUR per request} = 2,50 \text{ EUR (PUT requests cost)} \quad (3.3)$$

$$536.000 \text{ GET requests} \times 0,00093 \text{ EUR per request} = 0,50 \text{ EUR (GET requests cost)} \quad (3.4)$$

$$50 \text{ GB} \times 0,00093 \text{ EUR} = 0,47 \text{ EUR (data retrievals cost)} \quad (3.5)$$

$$38,72 \text{ EUR} + 2,50 \text{ EUR} + 0,50 \text{ EUR} + 0,47 \text{ EUR} = \underline{\textbf{42,19 EUR (Total Standard-IA)}} \quad (3.6)$$

$$32.212.255 \text{ number of objects} \times 0,0000093 \text{ EUR} \quad (3.7)$$

$$= 300{,}76 \text{ EUR (Cost for PUT, COPY, POST requests for initial data)} \quad (3.8)$$

Die oben genannten Formeln ähneln denen für die Speicherklassen Standard und One Zone-IA, wobei die entsprechenden Gebühren aus dem Kostenabschnitt übernommen wurden. Bei der Intelligent Tiering gibt es jedoch einen Unterschied: Es fallen zusätzliche Kosten für Überwachung und Automatisierung an, und der Speicher wird in drei Klassen prozentual aufgeteilt. In der letzten Zeile der Tabelle werden die Gesamtkosten zusammen mit den Kosten für den Datentransfer, eventuelle Vorauszahlungen (zusätzlich in 3.8) und die reinen monatlichen Speicherkosten für die jeweiligen Speicherklassen addiert.

GC Storage

Die nachstehende Tabelle stellt eine Zusammenfassung der monatlichen Kosten für drei Speicherklassen von GC Storage dar:

	Standard	**Nearline**	**Coldline**
Speicherkosten	63,75 EUR	36,03 EUR	16,63 EUR
Datenabrufe	–	0,45 EUR	0,90 EUR
Klasse A Operationen	1,21 EUR	2,42 EUR	4,84 EUR
Klasse B Operationen	0,19 EUR	0,48 EUR	4,84 EUR
Internet Egress	3,83 EUR pro 0 bis 1TB		

Tabelle 2.3.2: Übersicht der einzelnen Kosten der Datenspeicherung in GC Storage

Die Speicherklasse mit den höchsten Kosten von 63,75 Euro ist die Standardklasse. Die Coldlineklasse ist die kostengünstigste Option mit Kosten von 16,63 Euro. Die Nearlineklasse ist teurer als die Coldlineklasse und kostet 36,03 Euro. In der Standardklasse fallen keine Gebühren für den Datenabruf an, während für Nearline 0,45 Euro und für Coldline 0,90 Euro berechnet werden. In Klasse A ist die Nearlineklasse mit 2,42 Euro doppelt so teuer wie die Standardklasse, während die Coldlineklasse mit 4,84 Euro doppelt so teuer wie die Nearlineklasse ist. In Klasse B ist die Standardklasse mit 0,19 Euro am günstigsten, während die Nearlineklasse etwa doppelt so teuer ist. Die Coldlineklasse ist hier etwa zehnmal teurer als die Nearlineklasse. Für den Internet-Ausgangsverkehr fallen Gebühren von 3,83 Euro pro 0 bis 1 TB monatlich an. Die Tabelle zeigt, dass die Standardklasse doppelt so teuer ist wie die Nearlineklasse und etwa dreimal so teuer wie die Coldlineklasse. Die Preise ergeben sich durch die unterschiedlichen Anwendungsfälle der Speicherklassen, die in der Entscheidungsfindung genauer erklärt werden.

2.3.2 Entscheidungsfindung

Um der Anforderung der sicheren Speicherung des Produkts leoticket zu entsprechen, bieten beide Cloud Provider Funktionen an, die im vorherigen Kapitel unter Eigenschaften untersucht worden sind. Beide Cloud Provider bieten ähnliche Funktionen für die Erstellung von Benutzern, Gruppen und Rollen, um den Zugriff auf die Dienste einzuschränken. Die IAM Funktionen beider Cloud Provider ermöglichen die Beschränkung des Zugriffs auf S3 und Cloud Storage. Bei der IAM-Funktion in AWS S3 werden Vorgehensweisen empfohlen, um die Sicherheit und den Zugriff auf S3-Ressourcen zu verbessern. Beispielsweise das Prinzip des geringsten Privilegs. Benutzern und Anwendungen sollten nur die Berechtigungen, die sie zum Durchführen ihrer Aufgaben benötigen, gewährt werden. Überflüssige Berechtigungen sollten vermieden werden. Das beinhaltet beispielsweise Schreib- und Leseberechtigungen für Objekte. Es können auch andere Berechtigungen, wie das Hinzufügen von Richtlinien, vergeben werden. Diese Empfehlung gilt auch für die IAM-Funktion in GC. Für die Authentifizierung von Anwendungen stehen Service Accounts zur Verfügung, die spezifische Berechtigungen haben, welche für die Anwendung relevant sind. Für jeden Dienst oder jede Anwendung kann ein eigener Service Account erstellt werden.

Auch bei der Datenverschlüsselung bieten beide Cloud Provider ähnliche Möglichkeiten, mit Encryption Keys umzugehen. Da leoticket eine sichere Speicherung erfodert, kommt die SSE-KMS customer-managed Variante in Frage. Durch die selbstständige Generierung des Schlüssels und Verwaltung hat der Nutzer im Gegensatz zu S3-managed oder Google-managed Keys mehr Kontrolle. Die generierten Schlüssel werden jedoch vom Cloud-Anbieter im KMS gespeichert. Wenn die Option SSE-KMS gewählt wird, kann die Funktion Bucket Keys in S3 aktiviert werden. Dadurch wird ein Master Key verwendet, der die Schlüssel der einzelnen Objekte verschlüsselt. Dies führt zu einer Reduzierung der Request-Kosten um bis zu 99 Prozent, da der Request-Verkehr von S3 zu AWS KMS verringert wird.

Um eine größere Unabhängigkeit vom Cloud-Anbieter zu gewährleisten, kann die Option customer-managed in S3 in Betracht gezogen werden.In diesem Fall trägt der Nutzer die volle Verantwortung für die Schlüssel. Das bedeutet, dass die Schlüssel extern vom Nutzer generiert, verwaltet und gespeichert werden. Es besteht jedoch das Risiko, dass der Schlüssel verloren geht und der Zugriff auf die Objekte in S3 oder Cloud Storage nicht mehr möglich ist. Darüber hinaus erfordert die eigenständige Verwaltung des Schlüssels einen höheren Aufwand im Vergleich zur Verwendung des KMS. Bei der Erstellung von signierten URLs müssen folgende Encryption Header der Request angehängt werden:

Name	Description
x-amz-server-side-encryption-customer-algorithm	Use this header to specify the encryption algorithm. The header value must be AES256.
x-amz-server-side-encryption-customer-key	Use this header to provide the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.
x-amz-server-side-encryption-customer-key-MD5	Use this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321. Amazon S3 uses this header for a message integrity check to ensure that the encryption key was transmitted without error.

Tabelle 2.3.3: Benötigte Encryption Header für signierte URLs mit SSE-C, aws-sse-c AWS: Using server-side encryption with customer-provided keys (SSE-C), [4]

Die Implementierung von signierten URLs in Verbindung mit customer-supplied Encryption Keys ist in der offiziellen Dokumentation von GCP nicht enthalten.

Beide Cloud-Anbieter bieten die Möglichkeit, ACLs auf Objekte und Buckets anzuwenden. Es wird jedoch empfohlen, ACLs auf Objektebene standardmäßig deaktiviert zu lassen und nur in Fällen zu verwenden, in denen der Uploader eines Objekts die vollständige Kontrolle über diese und auf andere Objekte eingeschränkten Zugriff in einem Bucket hat.

Um die Sicherheit und Verfügbarkeit von Daten zu gewährleisten, wird empfohlen, die Objektversionierung zu aktivieren, um mehrere Versionen von Objekten zur Verfügung zu stellen. Dadurch können Objekte bei versehentlichem Löschen oder Überschreiben wiederhergestellt werden. Diese Funktion wird von beiden Cloud-Providern angeboten und kann beim Erstellen eines Buckets aktiviert werden.

AWS und GC stellen eine Auswahl verschiedener Speicherklassen zur Verfügung. Beide Anbieter garantieren eine Verfügbarkeit von mindestens 99,5%. Die Speicherklassen Standard-IA und One Zone-IA bieten kostengünstigere Optionen für die Datenspeicherung. Die Standard-IA eignet sich für Daten, auf die seltener zugegriffen wird, die aber bei Bedarf schnell verfügbar sein müssen. Im Vergleich zur Standardklasse sind die Speicherkosten niedriger, jedoch fallen zusätzliche Gebühren für den Datenabruf an. Wenn Daten zwischen zwei Speicherklassen verschoben werden, entstehen Kosten für den Wechsel der Speicherklassen, die von der verschobenen Datenmenge abhängen.

Die Speicherklasse One Zone-IA eignet sich für Daten, auf die selten zugegriffen wird, weist jedoch eine Einschränkung auf. Im Gegensatz zu den anderen Speicherklassen wird One Zone-IA nur in einer einzigen Verfügbarkeitszone (AZ) in einer bestimmten Region gespeichert. Das bedeutet, dass die Daten nur in dieser einen AZ verfügbar sind und das Risiko besteht, dass die Daten nicht zugänglich sind, wenn diese spezifische AZ nicht verfügbar ist.

Für leoticket werden die Speicherklassen Standard-IA von S3 und Nearline von Cloud Storage empfohlen. Da in leoticket eine schnelle Dateiabfrage innerhalb von Millisekunden erforderlich ist, kommen die Speicherklassen von S3 Glacier nicht in Betracht, da das Abrufen von Dateien in diesen Speicherklassen Tage oder Stunden dauern kann. Die Archive Storage Klasse von GCP kommt ebenfalls nicht in Betracht, da der Prozess des Datenabrufs, um die Dateien wieder zugänglich zu machen, mehrere Sekunden dauern kann. In diesem Prozess werden die Daten vom Archive Storage zum Online Storage verschoben.

Die Daten in leoticket müssen jederzeit abrufbereit sein, jedoch wird davon ausgegangen, dass Objekte im Durchschnitt nur zweimal von leoticket Kunden abgerufen werden. Aufgrund der geringen Anzahl von Abrufen pro einzelner Objekte und der erforderlichen Speicherung von Daten über einen Zeitraum von 10 Jahren sind die Speicherklassen Standard-IA und Nearline geeignete Optionen. Diese Speicherklassen sind für Daten ausgelegt, auf die seltener zugegriffen wird, aber bei Bedarf dennoch schnell verfügbar sein müssen. Sie bieten eine kostengünstigere Möglichkeit zur Speicherung der Daten, denn die Speicherung ist günstiger als bei den Standard Klassen.

Die Speicherklasse One Zone-IA von S3, welche die Daten nur in einer einzigen AZ speichert, wird aufgrund des Risikos einer eingeschränkten Verfügbarkeit bei Ausfall dieser spezifischen AZ nicht empfohlen. Dadurch kann sich die Zeit für den Abruf von Daten verlängern. Die Abrufkosten sind ebenfalls höher als bei anderen Speicherklassen, da diese Klasse nicht für Daten ausgelegt ist, auf die mindestens zweimal zugegriffen werden muss.

Die Coldline-Speicherklasse von GC kann in Betracht gezogen werden, wenn Daten für eine längere Zeit nicht abgerufen wurden und auch in naher Zukunft nicht erwartet wird, dass sie wieder abgerufen werden. Diese Speicherklasse ist für Daten geeignet, auf die selten zugegriffen wird und bei denen eine längere Zugriffszeit akzeptabel ist. Sie eignet sich gut für die Archivierung von Daten. Hier könnte auf die Auto-Class Funktion von GC oder auf die Intelligent-Tiering von AWS zugegriffen werden, um Dateien automatisch in die passenden Speicherklassen zu verschieben. Jedoch fallen auch hier Gebühren bei der Nutzung dieser Funktionen an.

Im Hinblick auf die API-Integration gibt es einige Unterschiede zwischen den beiden Providern. Für leoticket ist es relevant, dass die API nahtlos in die leoticket-Anwendung integriert werden kann, ohne großen Aufwand betreiben zu müssen. Beide Anbieter bieten geeignete SDKs, CLIs und REST APIs an, die problemlos in die eigene Anwendung eingebunden werden können. Es gibt auch Frameworks wie Spring Boot, die Bibliotheken für sowohl die AWS SDK-Version 1 und 2 als auch für GC bereitstellen.

Ein Vorteil von AWS besteht darin, dass On-Premise-Anwendungen wie MinIO die AWS API verwenden und somit eine Kompatibilität mit S3 gewährleisten. Anwendungen oder Skripte können so konfiguriert werden, dass sie die S3 API verwenden, um mit MinIO zu interagieren. Die API-Endpunkte und Authentifizierungsmechanismen sind mit der S3 API kompatibel, was bedeutet, dass S3 Anwendungen auf MinIO umgestellt werden können.

GC kann ebenfalls mit MinIO integriert werden. Dabei dient MinIO als Gateway. Nach den nötigen Konfigurationen kann auf GC Storage über den MinIO Server Endpoint zugegriffen werden. Dabei wird das Standard AWS S3 SDK für die Integration mit Cloud Storage verwendet. Jedoch erfordert es eine Einarbeitung in das AWS SDK als GC Storage Nutzer.

Für die Authentifizierung mit der API bei GC wird empfohlen, die Verwendung von Service Accounts aus dem Google SDK zu nutzen. Es gibt jedoch auch andere Methoden zur Authentifizierung, und über Application Default Credentials (ADC) in GC kann die bevorzugte Methode festgelegt werden. AWS bietet das `AWS Toolkit` zur Authentifizierung an, das von IntelliJ Idea, Eclipse und anderen IDEs unterstützt wird. Die Verwendung des Toolkits wird zur Authentifizierung empfohlen. Insgesamt bieten sowohl AWS als auch GC gute Möglichkeiten zur API-Integration, wobei spezifische Faktoren wie Vorwissen, vorhandene Technologien und Anforderungen berücksichtigt werden sollten.

Um eine optimale Performance für leoticket zu gewährleisten, empfiehlt es sich, Performance-Analysen durchzuführen. Sowohl AWS als auch GC bieten Tools zur Unterstützung an. Bei GC kann die Performance-Diagnose mithilfe des `gsutil` CLI durchgeführt werden. Dies ist auch erforderlich, wenn der Support von GC kontaktiert wird, um bei der Fehlerbehebung zu helfen. AWS hingegen bietet im S3-Bucket Metriken an, die zur Performance-Analyse genutzt werden können. Es werden verschiedene Widgets für Request-Metriken bereitgestellt, die analysiert werden können. AWS hat jedoch keine speziellen Performance-Analyse Tools wie das `gsutil perfdiag` von GC. Im Kapitel „Prototypische Umsetzung“ werden Performance-Messungen durchgeführt und anschließend analysiert, um einen besseren Vergleich der beiden Cloud-Anbieter in Bezug auf die Performance zu ermöglichen. Es ist ratsam, diese Analysen regelmäßig durchzuführen, um mögliche Leistungsprobleme frühzeitig zu erkennen und zu optimieren.

Die Entscheidung zwischen den Cloud Providern hängt sowohl von den Anforderungen von leoticket als auch von den Präferenzen der Entwickler ab. Wenn Entwickler bereits Erfahrung und Kenntnisse mit einem bestimmten Cloud Provider haben, ist es ratsam, sich für diesen Provider zu entscheiden. Dies bietet den Vorteil, dass man sich nicht in eine neue Technologie einarbeiten muss und Zeit bei der Implementierung und Integration der API sparen kann. Es ist durchaus legitim zu sagen, dass beide Anbieter eine Vielzahl von Funktionen und ähnlichen Produkten anbieten, die an die Anforderungen von leoticket angepasst werden können. Diese Arbeit soll lediglich als Leitfaden dienen, um bei der Entscheidung zu unterstützen und letztendlich eine fundierte Wahl zu treffen.

3 Prototypische Umsetzung

Im folgenden Kapitel wird der Prototyp genauer betrachtet und die verschiedenen Aspekte seiner Entwicklung und Implementierung werden erläutert. Es werden dabei die verwendeten Technologien und die Art der Speicherung und Bereitstellung von Binärdaten beleuchtet. Um die Performance zu bewerten, werden Messungen auf generierte Testdaten durchgeführt. Insgesamt dient das Kapitel als Grundlage für weitere Untersuchungen und die Optimierung des Prototyps.

3.1 Überblick und Vorgehensweise

Zunächst wird auf die eingesetzten Technologien eingegangen, die bei der Entwicklung des Prototyps verwendet werden. Dies umfasst das Framework Spring Boot und Programmiersprachen, die zur Umsetzung des Prototyps genutzt werden. Ein besonderes Augenmerk wird auf die Speicherung der Binärdaten gelegt. Hier werden verschiedene Ansätze betrachtet, wie beispielsweise die Verwendung von dem AWS SDK und der Google Client Libraries. Des Weiteren wird die Bereitstellung der Binärdaten behandelt. Hier wird die Methode der Signed URLs betrachtet, um die Daten effizient an die Anwender zu übertragen. Um die Leistung des Prototyps zu bewerten, werden Testdaten mit zufälligem Inhalt generiert. Dies ermöglicht eine realistische Simulation der Tickets und Rechnungen in leoticket und erlaubt eine Bewertung der Performance der Cloud Provider. Die Messungen werden auf einem virtuellen Server durchgeführt, um eine isolierte Umgebung zu gewährleisten. Dies ermöglicht es, Performance Messungen unabhängig von anderen Prozessen oder Anwendungen auf dem Host-System durchzuführen, ohne Störungen zu verursachen.

3.2 Eingesetzte Technologien

Für die Umsetzung des Prototyps werden die folgenden Technologien eingesetzt:

- Spring Boot v3
- AWS SDK 2.0 Version
- GC Storage client library
- Java SDK v17
- Apache Maven v3.9.1
- Maven Schema v4.0.0
- AWS Toolkit
- aws cli
- gcloud cli

Für die Implementierung wurde die Entwicklungsumgebung IntelliJ IDEA Ultimate verwendet. IntelliJ bietet ein Plugin namens `AWS Toolkit` an. Als Framework wurde die aktuellste Version von Spring Boot (Version 3) zum Zeitpunkt der Erstellung des Prototyps verwendet. Spring Boot stellt SDKs beider Cloud-Provider als Maven-Abhängigkeiten zur Verfügung.

> Am 17. März 2021 wurde die neue Version des Spring Cloud AWS 2.3 veröffentlicht. Spring Cloud GCP und Spring Cloud AWS sind nicht mehr Teil des Spring Cloud Releases. Nicht Teil des Releases zu sein bedeutet auch, dass sie aus der Spring Cloud Organisation auf Github herausgenommen worden sind und dadurch neue Maven Package Namen haben. Das neue Package für Spring Cloud AWS heißt nun „io.awspring.cloud“, vgl. Gibb: Spring Cloud AWS 2.3 is now available, [7].

`spring-cloud-announce`

Die unten aufgeführten Maven Abhängigkeiten werden für AWS S3 und Cloud Storage verwendet:

```
<dependency>
        <groupId>com.google.cloud</groupId>
        <artifactId>spring-cloud-gcp-starter-storage</artifactId>
  </dependency>
```

```
dependency>
        <groupId>io.awspring.cloud</groupId>
        <artifactId>spring-cloud-aws-s3</artifactId>
        <version>3.0.0</version>
  </dependency>
```

Als Spring Cloud GCP Version wird die 4.2.0 verwendet. Beide Spring Cloud Abhängigkeien werden von der Community auf Github verwaltet und aktualisiert. Für die Erstellung des Spring Boot Projekts wurde der Spring Initializier von Spring selbst verwendet (https://start.spring.io/). Zudem wird das Java SDK Version 17 verwendet. Für Apache Maven wird die Version 3.9.1 verwendet. Das Listing 8.1 im Anhang beinhaltet die vollständige pom.xml Datei, in der die Abhängigkeiten des Prototyps dargestellt werden.

Für die Authentifizierung wird das `gloud` cli verwendet. Dieses wird über die offizielle Dokumentation installiert. Siehe https://cloud.google.com/sdk/docs/install?hl=de. Das AWS Toolkit wird für die Authentifizierung mit AWS angewendet. Um sich mit GC zu verbinden wird eine Methode des ADC verwendet, welches im nächsten Abschnitt genauer erläutert wird.

3.3 Speicherung von Binärdaten

Um Daten in S3 oder Cloud Storage speichern zu können, wurde der Prototyp so implementiert, dass der Nutzer sich zwischen S3 oder Cloud Storage entscheiden kann. Dies geschieht über die Klasse `CloudStorageServiceFactory`, (siehe Listing 8.3 im Anhang). Hier kann der Nutzer über die Umgebungsvariable `cloud_provider` den gewünschten Provider mit `aws` oder `google cloud` angeben. Dabei wird die Groß-, und Kleinschreibung nicht berücksichtigt. Die Umgebungsvariablen können unter Linux, OSX und Unix im System durch `export <EnvironmentVariable>=<value>` exportiert werden. Wenn eine IDE wie Intellij verwendet wird, können sie unter den Run-Einstellungen eingefügt werden. Nach der Eingabe des Cloud Providers wird das Programm die entsprechende Klasse aufrufen. Für AWS die Klasse `AWSS3StorageService` und für GC die Klasse `GCStorageService`. Beide Klassen implementieren das Interface `CloudStorageService`. Dieses definiert zwei Methoden, die zum Hoch- und Herunterladen der Daten zuständig sind:

```
void uploadObject(String bucketName, String key, String file, String encryptionKey,
     String storageClass) throws IOException;
```

```
URL getPresignedUrl(String bucketName, String key, Integer minutes, String
    encryptionKey);
```

Der `uploadObject` Methode wird der Bucket Name, der Name des Objekts, der Pfad des Objekts und der Encryption Key übergeben. Der Encryption Key kann dabei der Schlüssel sein, der in AWS KMS oder GC KMS generiert wurde. Die Implementierung dieser Methode ist für beide Cloud Provider ähnlich gestaltet.

```
@Override
public void uploadObject(String bucketName, String key, String file, String
    encryptionKey, String storageClass) {
    try {

(1)     PutObjectRequest putObjectRequest = PutObjectRequest.builder()
(2)             .bucket(bucketName)
(3)             .key(key)
(4)             .serverSideEncryption(ServerSideEncryption.AWS_KMS)
(4)             .ssekmsKeyId(encryptionKey)
(5)             .storageClass(storageClass)
                .build();

(6)     Path filePath = Paths.get(file);
(7)     byte[] fileBytes = Files.readAllBytes(filePath);
(8)     RequestBody requestBody = RequestBody.fromBytes(fileBytes);

(9)     this.s3Client.putObject(putObjectRequest, requestBody);
        logger.info(
        "File " + file + " uploaded to bucket " + bucketName + " as " + key
        );

    } catch (S3Exception | IOException e) {
        logger.error(e.getMessage());
    }
}
```

Listing 3.1: Prototyp Code Snippet - Hochladen eines Objekts nach S3

Der vorliegende Code (3.1) beschreibt den Vorgang des Speicherns eines Objekts in AWS S3. Zunächst wird ein PUT-Request-Objekt erstellt (1), wobei Parameter wie der Bucket-Name (2), der Objektname als Key (3) und die Authentifizierungsmethoden (4) angegeben werden. Dabei wird die AWS KMS-Methode verwendet und der entsprechende Schlüssel bereitgestellt (4). Anschließend wird die Speicherklasse angegeben, in der das Objekt gespeichert werden soll (5). In diesem Beispiel wird die Standard-IA-Klasse verwendet. Danach wird der Pfad der angegebenen Datei gelesen (6), in ein Byte-Array umgewandelt (7) und dem `RequestBody` übergeben (8). Der RequestBody wird mit dem `PutObjectRequest` an den `S3Client` übergeben und mit der AWS `.putObject()` Methode in S3 hochgeladen (9). AWS S3 verschlüsselt das Objekt mit dem angegebenen Schlüssel und lädt es in S3 hoch.

Das folgende Code Snippet zeigt die Methode der Klasse `GCStorageService`. Ähnlich wie bei der Methode für AWS S3 wird auch hier ein Objekt nach Cloud Storage hochgeladen:

```
@Override
public void uploadObject(String bucketName, String key, String file, String
    encryptionKey, String storageClass) throws IOException {

    Map<String, String> kmsKeyName = new HashMap<>();
    kmsKeyName.put("kmsKeyName", encryptionKey);

    // Get a reference to the bucket
(1) BlobId blobId = BlobId.of(bucketName, key);
(2) BlobInfo blobInfo = BlobInfo.newBuilder(blobId)
            .setMetadata(kmsKeyName)
            .build();

    Storage.BlobWriteOption precondition;
    if (this.storage.get(bucketName, key) == null) {
        precondition = Storage.BlobWriteOption.doesNotExist();
    } else {
        precondition =
                Storage.BlobWriteOption.generationMatch(
                        this.storage.get(bucketName, key).getGeneration());
    }
(3) this.storage.createFrom(blobInfo, Paths.get(file), precondition);

    logger.info(
    "File " + file + " uploaded to bucket " + bucketName + " as " + key
    );

}
```

Listing 3.2: Prototyp Code Snippet - Hochladen eines Objekts nach Cloud Storage

Dabei werden ähnliche Parameter der Methode wie in AWS S3 übergeben. Um ein Objekt in einen Bucket hochladen zu können, wird eine Referenz zu diesem erstellt. Dies geschieht durch die `BlobId`, welche den Bucket Namen und den Namen des Objekts beinhaltet (1). Anschließend wird diese blobId dem `BlobInfo` Objekt übergeben und der Encryption Key über die Metadaten `kmsKeyName` gesetzt (2). Nachdem überprüft worden ist, ob das Objekt bereits im Bucket existiert, wird das Objekt hochgeladen (3).

Um das Programm auszuführen, wird die Hauptklasse `CloudGcStorageAwsS3Application` gestartet. Damit das Programm erfolgreich läuft, müssen die Umgebungsvariablen im System exportiert werden. Alle Umgebungsvariablen sind in der `application.properties` hinterlegt, (siehe Listing 8.2 im Anhang). Diese werden beim Start des Programms gelesen und angewendet. Außerdem müssen die AWS und GC Credentials hinterlegt werden. Dies kann entweder über das AWS Toolkit Plugin gesteuert werden oder in der Kommandozeile mit dem Befehl `aws configure`. Für GC Credentials kann mit dem Befehl:
`export GOOGLE_APPLICATION_CREDENTIALS=<service-account-json-file>` der Service Account hinterlegt werden oder durch Ausführen des Befehls `gcloud auth application-default login` in der Kommandozeile, was die Credentials lokal speichert und für ADC verwendet wird.

3.4 Bereitstellung der Binärdaten

Bei der Bereitstellung von Binärdaten in leoticket geht es darum, den Kunden Dateien über Links zugänglich zu machen. Hierfür werden signierte URLs verwendet. Im folgenden Abschnitt werden die Implementierungen und Erläuterungen des entsprechenden Codes von beiden Cloud Providern vorgestellt.

Folgender Code Snippet (3.3) zeigt die Methode `getPresignedUrl` zur Generierung einer signierten URL der Klasse **AWSS3StorageService**:

```
1  @Override
2  public URL getPresignedUrl(String bucket, String key, Integer minutes, String
       encryptionKey) {
3      try {
4
5  (1)     GetObjectRequest getObjectRequest =
6        GetObjectRequest.builder()
7  (2)             .bucket(bucket)
8  (3)             .key(key)
9                  .build();
10
11         GetObjectPresignRequest getObjectPresignRequest =
12               GetObjectPresignRequest.builder()
13 (4)             .getObjectRequest(getObjectRequest)
14 (5)             .signatureDuration(Duration.ofMinutes(minutes))
15                 .build();
16
17         PresignedGetObjectRequest presignedGetObjectRequest =
18 (6)           presigner.presignGetObject(getObjectPresignRequest);
19
20 (7)     URL url = presignedGetObjectRequest.url();
21
22         logger.info("Presigned URL: " + url);
23
24         return url;
25
26     } catch (S3Exception e) {
27         logger.error(e.getMessage());
28     }
29
30     return null;
31 }
```

Listing 3.3: Prototyp Code Snippet - Generierung einer signierten URL durch AWS

Die Methode erhält ähnlich wie beim Hochladen des Objekts die Parameter Bucket-Name und Objektname. Zudem wird eine Zeitdauer in Minuten übergeben, die festlegt, wie lange der generierte Link gültig sein soll. Für die Entschlüsselung der Daten muss der gleiche Encryption Key wie beim Hochladen verwendet werden. Das `GetObjectRequest` Objekt wird erstellt (1) und mit dem Bucket-Namen (2) und dem Objektnamen (3) versehen. Anschließend wird dieses Objekt an das `GetObjectPresignRequest` Objekt übergeben (4) und die Gültigkeitsdauer der Signatur angegeben (5). Diese bestimmt, wie lange die signierte URL valide ist.

Um die signierte URL zu generieren, wird das `GetObjectPresignRequest` Objekt an den `S3Presigner` übergeben (6), auf welche die Methode `presignGetObject()` aufgerufen wird (6). Das generierte `PresignedGetObjectRequest` wird als URL gespeichert (7) und ausgegeben.

Folgende Abbildung zeigt die Methode zur Generierung einer signierten URL der Klasse `GCStorageService`:

```
@Override
public URL getPresignedUrl(String bucketName, String key, Integer minutes, String
    encryptionKey) {
    try {
        Map<String, String> kmsKeyName = new HashMap<>();
        kmsKeyName.put("kmsKeyName", encryptionKey);

        // Define resource
(1)     BlobInfo blobInfo = BlobInfo.newBuilder(BlobId.of(bucketName, key))
                .setMetadata(kmsKeyName)
                .build();

(2)     URL url = this.storage.signUrl(
(3)             blobInfo,
(4)             minutes,
                TimeUnit.MINUTES,
(5)             Storage.SignUrlOption.withV4Signature()
        );

        logger.info("Generated GET signed URL: " + url);

(6)     return url;

    } catch (StorageException e) {
        logger.error(e.getMessage());
    }

    return null;
}
```

Listing 3.4: Prototyp Code Snippet - Generierung einer signierten URL durch GC

Ähnlich wie bei S3 wird eine Referenz zum Objekt erstellt, indem der Bucket Name und der Name des Objekts dem `BlobInfo` Objekt mitgegeben wird (1). Anschließend kann die URL durch Aufrufen der Methode `.signUrl()` erstellt werden (2). Hier werden das `BlobInfo` Objekt (3), die Minuten (4) und die Signatur Methode (5) übergeben. Zuletzt wird die URL ausgegeben (6).

Mit signierten URLs können die Anforderungen von leoticket an die sichere Speicherung und Bereitstellung der Daten berücksichtigt werden. Dies bedeutet, Kunden können über diese Links, die durch Emails versendet werden, die Dateien für Tickets und Rechnungen herunterladen. Die Herausforderung von leoticket, große Email-Anhänge zu versenden, kann vermieden werden.

3.5 Messung der Performance

In diesem Abschnitt erfolgt die Messung der Performance für die Dienste von AWS und GC. Dabei werden bis zu 1000 generierte Dateien sowohl für den Upload als auch für den Download betrachtet. Die Performance Analyse wird auf einer virtuellen Maschine in der Hetzner Cloud ausgeführt, um eine isolierte Umgebung zu gewährleisten. Die Performance-Messung beim Hoch- und Herunterladen kann jedoch von verschiedenen Faktoren wie Netzwerklatenz, verfügbarer Bandbreite, Serverkapazität, Datenmenge und der Auslastung der Server beeinflusst werden. Außerdem kann der Hetzner Server näher an AWS oder GC liegen und dies könnte dazu führen, dass die Dauer des Hoch- und Herunterladens aus diesem Grund kürzer ist. Die Messungen dienen lediglich dem groben Vergleich beider Cloud Provider. Die Performance-Messung wird auf verschiedene Speicherklassen durchgeführt, um einen Vergleich zwischen diesen zu ermöglichen und eine Grundlage für die Bewertung ihrer Leistungsfähigkeit zu schaffen.

AWS

AWS bietet Dienste für die Performance Analyse. Unter anderem die Amazon CloudWatch, S3 Storage Lens und die S3 Transfer Acceleration. Die AWS CLI stellt einfache Methoden zum S3 Upload und Download Tests vor. Beispielsweise kann man mit dem Befehl:

```
1  aws s3 cp <lokaler_pfad> s3://<Bucket_Name>/<Ziel_Dateipfad>
```

Dateien hoch- und herunterladen und die Zeit für die benötigte Request messen. Auch mit dem AWS SDK können Performance Test Skripte geschrieben werden. Diese Methode wird für den Prototypen angewendet. Dabei werden Tests bereitgestellt, die mehrere Dateien automatisch hoch- und herunterladen und dabei die vergangene Zeit messen.

GC Storage

GC bietet ein eigenes `gsutil` Tool für die Performance Analyse. Wie im Abschnitt Performance bereits erwähnt, können durch die `perfdiag` Funktion Performance Diagnosen erstellt werden.

Mehrere Testdateien werden aus einem angegebenen Bucket hoch- und heruntergeladen. Nach der Analyse werden alle Testdateien nach erfolgreicher Diagnose gelöscht. Die `gsutil` Performance kann von Faktoren wie vom Client, Server oder Netzwerk beeinflusst werden. Möglich sind die Leistung, der verfügbare Speicher, die Netzwerk Bandbreite, Firewalls und Fehlerraten zwischen `gsutil` und den Google Servern. Die `perfiag` Funktion wurde bereitgestellt, damit Nutzer Messungen durchführen können, die beim Troubleshooting von Performance Problemen helfen, vgl. `gc-perfdiag` Storage: perfdiag - Run performance diagnostic, [19].

Um die Performance Diagnose auszuführen, kann der folgende Befehl verwendet werden:

```
gsutil perfdiag -o test.json -n 67000 -s 400kb gs://leoticket-bucket
```

Die `-o` Option schreibt den Output des Ergebnisses in eine Datei. Die Output Datei ist eine JSON Datei mit System Informationen und enthält die Performance Diagnose Ergebnisse. Die -n Option setzt die Anzahl der Objekte, die heruntergeladen und hochgeladen werden sollen während des Tests. Mit der `-s` Option kann man die Objektgröße angeben. Zum Schluss wird der Name des Buckets angegeben.

Um die Performance der SDKs zu analysieren, werden Objekte mit jeweils 100 KB Objektgröße in verschiedenen Speicherklassen hoch- und heruntergeladen. Anschließend wird die Performance über das SDK von Cloud Storage ähnlich wie bei AWS getestet. Der Test wird ebenfalls auf einer virtuellen Maschine von Hetzner ausgeführt.

Die Performance wird in Zehner-Vielfachen gemessen, beginnend mit einer Datei bis hin zu 1000 Dateien. Dies bedeutet, dass Messungen für eine Datei, zehn Dateien, 100 Dateien und 1000 Dateien durchgeführt werden. Zur Durchführung der Messung wird eine Jar Datei des Prototyps erstellt, in der die Testmethoden (siehe Listing 8.4 bis 8.8 im Anhang) ausgeführt werden. Diese generieren zunächst Testdaten, die mit zufälligen String-Werten der Größe 100 KB gefüllt werden, (siehe Listing 8.9 und 8.10 im Anhang). Anschließend werden die Testmethoden in der Kommandozeile ausgeführt.

3.5.1 Messungsergebnisse

In diesem Abschnitt werden die Messungsergebnisse der Performance Messung in Millisekunden präsentiert. Die Speicherklassen Standard, Standard-IA und One Zone-IA von AWS wurden jeweils mit Standard, Nearline und Coldline von GC verglichen. Im Folgenden werden die Ergebnisse der Upload und Download Dauer aller Speicherklassen präsentiert:

	Standard-IA	**Nearline**
1 Datei		
Upload	705	841
Download	18	26
10 Dateien		
Upload	1862	3217
Download	35	94
100 Dateien		
Upload	16681	22683
Download	129	269
1000 Dateien		
Upload	71175	185134
Download	663	1489

Tabelle 3.5.1: Ergebnisse der Upload und Download Dauer der Speicherklassen Standard-IA und Nearline

	Standard S3	**Standard GCS**
1 Datei		
Upload	585	660
Download	28	19
10 Dateien		
Upload	1369	3002
Download	70	108
100 Dateien		
Upload	8110	20750
Download	180	396
1000 Dateien		
Upload	73992	176004
Download	650	1619

Tabelle 3.5.2: Ergebnisse der Upload und Download Dauer der Speicherklassen Standard S3 und Standard GCS

	OneZone-IA	Coldline
1 Datei		
Upload	505	636
Download	28	18
10 Dateien		
Upload	1105	2855
Download	65	89
100 Dateien		
Upload	7539	20391
Download	187	250
1000 Dateien		
Upload	73086	163661
Download	701	1559

Tabelle 3.5.3: Ergebnisse der Upload und Download Dauer der Speicherklassen OneZone-IA und Coldline

Die Ergebnisse werden im Kapitel 4.2 Zusammenfassung der Ergebnisse untersucht und dargestellt.

3.6 Zusammenfassung der Implementierung

Der Prototyp soll einen Vergleich zwischen den beiden Cloud Providern bieten. Das Ziel dabei ist es, ähnliche Technologien von beiden Providern zu verwenden und die Performance mit Testdaten messen zu können.

Der Prototyp implementiert zwei wesentliche Funktionen: Das Hochladen und Herunterladen von Dateien unter Berücksichtigung der Anforderungen von leoticket. Dabei werden signierte URLs zur Bereitstellung der Dateien verwendet und die SSE KMS von beiden Providern für die Datenverschlüsselung angewendet. Der Prototyp soll auch als externe Bibliothek für eigene Anwendungen integriert werden können. Bei der Implementierung wurden die offiziellen Dokumentationen von AWS und Google Cloud verwendet, um die aktuellsten Versionen zum Zeitpunkt der Arbeit zu verwenden. Auf Wunsch von Leomedia wurde Spring Boot als Framework gewählt, um eine Enterprise-Anwendung bereitzustellen. Java wurde aus Präferenzgründen als Programmiersprache gewählt, obwohl beide Provider viele weitere Sprachen, die bereits im Kapitel zur API-Anbindung erwähnt wurden, unterstützen.

Der Prototyp umfasst eine Implementierung, die es ermöglicht, über eine Umgebungsvariable zwischen AWS und GC zu wählen. Die Klasse `CloudStorageServiceFactory` stellt diese zwischen AWS und GC zur Verfügung. Je nachdem welchen Cloud Provider der Nutzer angegeben hat, wird die entsprechende Klasse instanziiert und aufgerufen. Die Klassen `AWSS3StorageService` und `GCStorageService` implementieren die Methoden des Interfaces `CloudStorageService`.

Die bereitgestellten Tests dienen der Performance Messung. Dabei werden Dateien automatisch als 100 KB große Objekte erstellt und durch die Testmethoden verwendet, um Objekte hoch-, und herunterzuladen. Sie unterstützen dabei, die Dauer der verschiedenen Funktionen zu messen.

Da die Maven Abhängigkeiten kein Teil der Spring Cloud Release Train sind, hängt es von der Community ab, die Versionen aktuell zu halten. Für AWS bedeutet dies, dass die AWS SDK 2.0 Version nicht komplett abgedeckt ist. Jedoch ist sie so weit, dass S3 bereits unterstützt wird.

4 Ergebnisse dieser Arbeit

In diesem Kapitel werden die Funktionalitäten des Prototyps beschrieben. Zum Schluss werden die Kalkulationsergebnisse der Kostenanalyse als Tabelle und die Messungsergebnisse der Performance Messung als Liniendiagramm bereitgestellt und auf einzelne Ergebnisse eingegangen. Außerdem dient dieses Kapitel zur kurzen Übersicht über die Ergebnisse der Performance und Kosten.

4.1 Beschreibung und Funktionalität des Prototyps

Der Prototyp hat die Funktionalität des Uploads und Downloads von Objekten nach S3 oder Cloud Storage. Der Nutzer kann sich zwischen AWS und GC entscheiden, indem die cloud_provider Umgebungsvariable vor dem Ausführen des Programms gesetzt wird. Damit das Programm erfolgreich durchlaufen kann, müssen alle Umgebungsvariablen in der `application.properties` Datei gesetzt werden. Außerdem wird eine Authentifizierung für GC und AWS verlangt, sonst schlägt das Programm fehl. Die Authentifizierung für GC kann über die ADC erfolgen. In der offiziellen Dokumentation werden die angebotenen Methoden aufgelistet. Für den Prototypen wurde die lokale Entwicklungsumgebung für die Authentifizierung durch:

```
export GOOGLE\_APPLICATION\_CREDENTIALS=<jsonKeyPath>
```

oder durch den `gcloud` cli Befehl:

```
gcloud auth application-default login
```

verwendet. Dabei ist der `jsonKeyPath` der Pfad zum erstellten Service Account, welcher Rechte für das Uploaden und Downloaden besitzt. Unter https://cloud.google.com/docs/authentication/provide-credentials-adc können die ADC Credentials Methoden eingesehen werden.

Für die Authentifizierung mit AWS benötigt es lediglich das AWS Toolkit Plugin. Dieses stellt AWS für einige IDEs zur Verfügung. Unter anderem für Visual Studio, Eclipse und Intellij Idea. Falls das AWS Toolkit nicht zur Verfügung steht, kann die Authentifizierung auch über die `aws cli` erfolgen. Durch `aws configure` können die Credentials gesetzt werden. Unter https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/setup-basics.html können die AWS Credentials Methoden eingesehen werden. Nachdem die Authentifizierungseinstellungen und die Umgebungsvariablen gesetzt wurden, kann das Programm gestartet werden.

Damit die Dateien in der richtigen Speicherklasse hochgeladen werden, muss für GC der korrekte Bucket Name angegeben werden, der die Speicherklasse bereits bei der Bucket Erstellung konfiguriert hat. In AWS muss jedoch die gewünschte Speicherklasse für ein Objekt spezifisch angegeben werden. Dieser Unterschied besteht, da in GC die Speicherklasse über die GC Konsole für ein Bucket ausgewählt werden kann. In AWS ist dies jedoch nicht möglich, deshalb haben alle Buckets in AWS bei Erstellung die Standard Speicherklasse eingestellt. Die Objektklasse muss speziell im Fall von AWS beim Hochladen angegeben werden.

Das Objekt wird durch die SSE KMS verschlüsselt hochgeladen. Die Verschlüsselung wird automatisch von beiden Cloud Providern durchgeführt. Dabei muss der entsprechende KMS Schlüssel des ausgewählten Cloud Providers der Environment Variable `sse_kms_key_id_arn` übergeben werden. Beim Herunterladen des Objekts muss dieser ARN Schlüssel erneut mitgegeben werden, damit das Objekt entschlüsselt werden kann. Die Hauptklasse `HandsonAwsGcApplication` ruft die entsprechende Klasse für AWS oder GC auf und initialisiert diese. Vorher müssen die Parameter für die `.uploadObject(bucketName, key, filePath, encryptionKey, storageClass)` und `.getPresignedUrl(bucketName, key, minutes, encryptionKey)` übergeben werden. Die Minuten stellen die Zeit ein, in der die generierte signierte URL gültig ist. Der `storageClass` Parameter gilt nur für die AWS Funktion und kann die Werte `STANDARD, STANDARD_IA` und `ONEZONE_IA` annehmen.

Über Terraform werden die Buckets erstellt und die Konfigurationen eingestellt. Diese Konfigurationen sind an die Anforderungen von leoticket angepasst. Die Buckets müssen vor dem Programmstart bereits existieren, deshalb muss die Terraform Datei am Anfang durchlaufen. Die Terraform Dateien werden im entsprechenden Ordner bereitgestellt. Diese werden vor dem Programmstart als erstes ausgeführt, um die benötigten Ressourcen zu erstellen.

Der Prototyp dient außerdem dazu, die APIs miteinander zu vergleichen und jeweils ähnliche Funktionalitäten für beide Provider bereitzustellen.

4.2 Zusammenfassung der Ergebnisse

In diesem Abschnitt werden die Kosten der verschiedenen Speicherklassen nochmal aufgegriffen und die Gesamtkosten als Tabelle dargestellt. Auch werden die Messungsergebnisse der Performance Analyse als Liniendiagramm bereitgestellt und die Zahlen der Kosten und Messungen untersucht.

4.2.1 Kalkulationsergebnisse

Im Folgenden werden die Gesamtkosten der Speicherklasse von beiden Cloud Providern dargestellt und untersucht. Zuerst werden in der unteren Tabelle die Kosten von AWS untersucht:

	Standard	**Intelligent Tiering**	**Standard-IA**	**One Zone-IA**
Monatliche Speicherkosten	70,27 EUR	70,27 EUR	38,72 EUR	30,98 EUR
PUT Requests	1,35 EUR	1,35 EUR	2,50 EUR	2,50 EUR
GET Requests	0,22 EUR	0,22 EUR	0,50 EUR	0,50 EUR
Datenabrufe	–	–	0,47 EUR	0,47 EUR
Data Transfer	4,20 EUR			
Monitoring und Automation objects	–	75,19 EUR	–	–
Gesamtkosten pro Monat(ohne Vorauszahlung)	76,04 EUR	151,23 EUR	46,39 EUR	38,65 EUR
Vorabzahlung	162,42 EUR	162,42 EUR	300,76 EUR	300,76 EUR
Gesamtkosten im ersten Monat	**238,46 EUR**	**313,65 EUR**	**347,15 EUR**	**339,41 EUR**

Tabelle 4.2.1: Zusammenfassung der Gesamtkosten für AWS S3 pro Speicherklasse

In dieser Tabelle werden in der letzten Zeile die Gesamtkosten der Speicherklassen pro Monat veranschaulicht. Die Gesamtkosten im ersten Monat der Speicherklasse Standard betragen 238,46 Euro. In diesen Kosten stecken die Vorabzahlung, die PUT und GET Request Gebühren und die Datenübertragungsgebühren. Im Intelligent Tiering und Standard sind die Datenabrufgebühren nicht enthalten, da dafür keine Gebühren abgezogen werden. Die Gesamtkosten des Intelligent Tiering pro Monat betragen 313,65 Euro. Die Kosten des Intelligent Tiering hängen jedoch von der Speicherverteilung ab. Die Kosten sind grobe Werte und können von der Verteilung des Speichers in verschiedene Speicherklassen abhängen. Bei der Standard-IA Speicherklasse sind es mit 347,15 Euro die teuersten Kosten im Vergleich zu den anderen Speicherklassen. Die One Zone-IA ist mit 339,41 Euro die günstigste Speicherklasse.

In Google Cloud gibt es keine Vorabzahlungen im Gegensatz zu AWS. Hier unterscheiden sich die Gesamtkosten der verschiedenen Speicherklassen zu AWS. In der folgenden Tabelle werden die Gesamtkosten zusammengefasst:

	Standard	**Nearline**	**Coldline**
Monatliche Speicherkosten	63,75 EUR	36,03 EUR	16,63 EUR
Datenabrufe	–	0,45 EUR	0,90 EUR
Klasse A Operationen	1,21 EUR	2,42 EUR	4,84 EUR
Klasse B Operationen	0,19 EUR	0,48 EUR	4,84 EUR
Internet Egress monatlich	3,83 EUR pro 0 bis 1TB		
Gesamtkosten pro Monat	**68,98 EUR**	**43,21 EUR**	**31,04 EUR**

Tabelle 4.2.2: Zusammenfassung der Gesamtkosten für GC Storage pro Speicherklasse

Die Gesamtkosten bestehen laut Tabelle hauptsächlich aus den Speicherkosten und die Internet Egress, ausgehend von GC zum Internet, beispielsweise beim Abrufen von Dateien. Es fallen keine Gebühren für die Datenabrufe der Standard Speicherklasse an. Die Gesamtkosten des Standards betragen 68,98 Euro, der Nearline 43,21 Euro und des Coldline 31,04 Euro.

4.2.2 Messergebnisse

In diesem Abschnitt werden die Messergebnisse der Performance Analyse bereitgestellt. Die Speicherklassen Standard, Standard-IA und One Zone-IA von AWS wurden jeweils mit Standard, Nearline und Coldline von GC verglichen. Im Folgenden werden die Ergebnisse der Upload Dauer aller Speicherklassen als Liniendiagramm dargestellt:

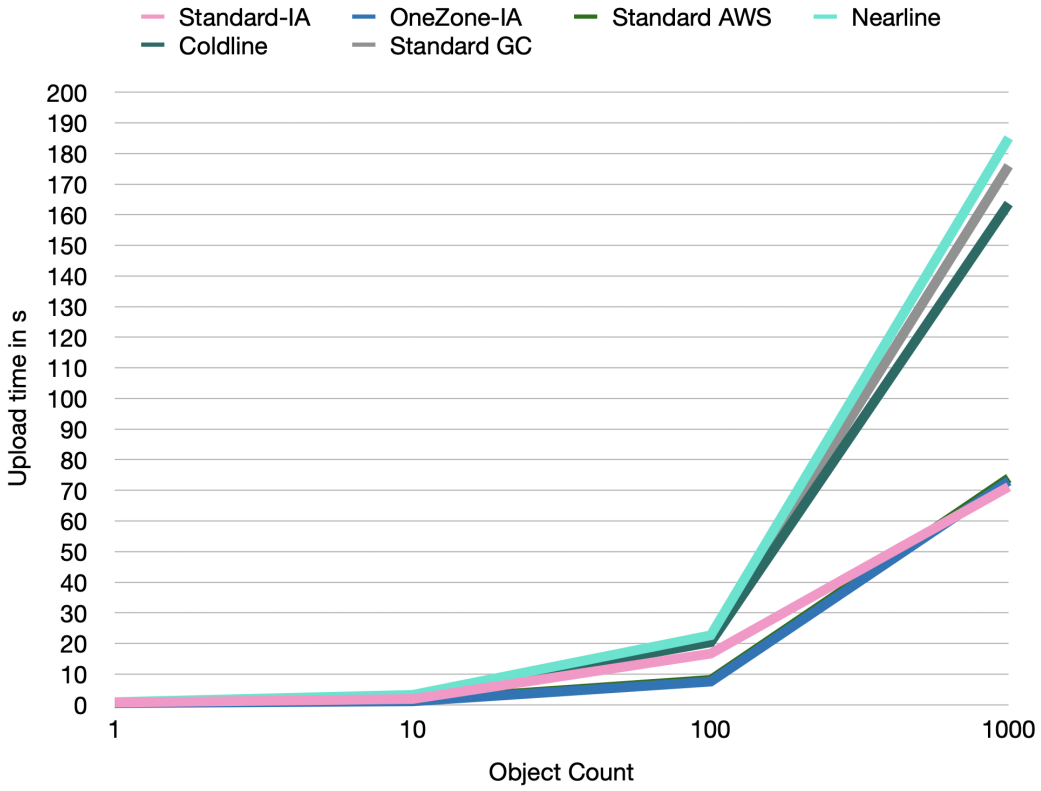

Abbildung 4.2.1: Upload Zeit der verschiedenen Speicherklassen

Die X-Achse beschreibt die Objektanzahl, die für jede Speicherklasse hochgeladen wurde. Die Farben unterscheiden die verschiedenen Speicherklassen, wobei die Speicherklassen von AWS jeweils nebeneinander platziert sind, genauso wie die Speicherklassen von GC. Die Zahlen vom Abschnitt 3.5.1 wurden in dieses Diagramm eingefügt, um die Unterschiede zwischen den Speicherklassen beider Provider zu visualisieren. So liegen die Speicherklassen von AWS im Graph nah beieinander, ähnlich wie jene von GC. Beim Hochladen von einer Datei bis hin zu zehn Dateien gibt es minimale Unterschiede. Diese Werte liegen unter 3 Sekunden. Die Linien gehen ab dem Hochladen von 11 bis 1000 Dateien stärker auseinander. Die drei Speicherklassen von GC entfernen sich von den drei Speicherklassen von AWS, was die Folge von höheren Uploadzeiten hat. Ab 100 Dateien gibt es erneut einen Knick und die Speicherklassen von GC schneiden schlechter ab. Die GC Speicherklassen wachsen steiler nach oben als bei AWS und bewegen sich in höheren Sekunden Bereichen von ungefähr 20 bis hin zu 190 Sekunden. AWS bewegt sich dabei maximal bis 140 Sekunden.

Die Unterschied sind beim Download weniger stark ausgeprägt als beim Upload45, denn die Zahlen für den Download bewegen sich in Bereichen von 16 bis hin zu 1556 Millisekunden. Hier steigen die Werte nicht höher als 2 Sekunden. In der folgenden Abbildung wird dies veranschaulicht:

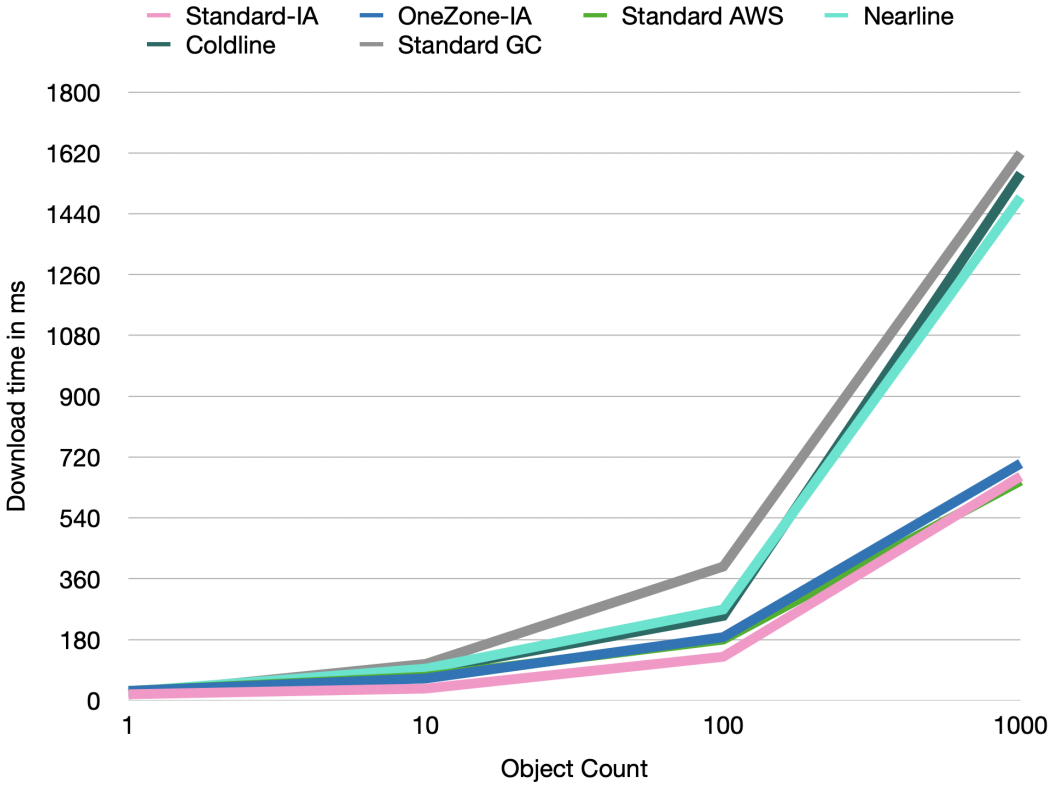

Abbildung 4.2.2: Download Zeit der verschiedenen Speicherklassen

Dieser Graph ähnelt dem Graphen des Uploads in Abbildung 4.2.3. Denn auch hier entfernen sich die Linien der GC Speicherklassen von den AWS Speicherklassen und haben eine höhere Download Zeit. Ab Downloads von zehn Dateien trennen sich die Cloud Provider voneinander. Die GC Standard Speicherklasse hat dabei mit 1619 Millisekunden am längsten für den Download von 1000 Dateien gebraucht. Die schnellste Speicherklasse von AWS ist die Standard Klasse mit 650 Millisekunden beim Download von 1000 Dateien. Bei GC dauern die Downloads von 1000 Dateien mindestens 1489 Millisekunden. Im Bereich von einer bis zehn Dateien bewegen sich alle Speicherklassen in unter 108 Millisekunden Bereichen. Jedoch gibt es bei zehn Dateien bereits erste minimale Unterschiede, die langsam steigen.

5 Diskussion

In diesem Kapitel werden die Kalkulations-, und Messergebnisse der Performance analysiert und interpretiert. Anschließend wird der Prototyp bewertet und dabei auf dessen Grenzen eingegangen.

5.1 Analyse und Interpretation der Ergebnisse

Basierend auf der Kostenanalyse und den Ergebnissen der Kalkulation ergibt sich, dass die Speicherklassen OneZone-IA von AWS und Coldline von GC die niedrigsten Kosten für die Speicherung von Daten aufweisen. Jedoch sind Datenabrufe in diesen Speicherklassen als kostenintensiv zu betrachten. In der Coldline-Klasse belaufen sich die Kosten für Klasse A-, und B-Operationen jeweils auf 4,84 Euro, während bei AWS PUT-Anfragen, die mit Klasse A gleichzusetzen sind, 2,50 Euro und GET-Anfragen, die mit Klasse B gleichzusetzen sind, 0,50 Euro betragen. Insgesamt sind die Kosten für die Coldline-Klasse im Vergleich zur OneZone-IA-Klasse jedoch günstiger.

Die Entscheidung für OneZone-IA als Speicherklasse birgt das Risiko auf die Nicht-Verfügbarkeit von Daten bei Ausfall der AZ, da die Daten nur in einer Verfügbarkeitszone gespeichert werden. Dies steht im Widerspruch zu den Anforderungen von leoticket an eine hohe Verfügbarkeit. Obwohl die Speicherkosten in dieser Klasse am niedrigsten sind, sind die Kosten für Datenabrufe im Vergleich zu anderen Speicherklassen höher. Für leoticket ist es jedoch von Bedeutung, auf Objekte mindestens zweimal zugreifen zu können und sie den Kunden zur Verfügung zu stellen. Bei älteren Daten, die mehrere Jahre zurückliegen, kann es ratsam sein, die Objekte in S3 Glacier Instant Retrieval oder in GC Archive Storage zu verschieben, um Kosten für die Archivierung einzusparen und die Daten in Bereichen von Millisekunden zur Verfügung stehen können.

Die Verfügbarkeit der Coldline-Klasse ähnelt der OneZone-IA-Klasse. Die Datenabrufe in dieser Klasse können mehrere Stunden dauern. Sie ist nicht für Szenarien geeignet, die einen sofortigen Zugriff auf Daten erfordern oder in denen häufiger auf die Daten zugegriffen werden muss. Im Gegensatz zur OneZone-IA ist die Coldline jedoch nicht auf eine einzelne Verfügbarkeitszone beschränkt. Aufgrund ihrer geringeren Kosten und der Verfügbarkeit in mehreren Availability Zones ist Coldline die bessere Option für die langfristige Archivierung von Daten. Da die Kosten für Datenabrufe in dieser Klasse höher sind als in allen anderen Speicherklassen, ist sie nicht geeignet für Daten, auf die mehr als einmal zugegriffen werden muss.

Im Rahmen der Performance-Analyse zeigte die OneZone-IA eine bessere Leistung als die Coldline. Bereits beim Upload von zehn Dateien war die OneZone-IA deutlich schneller. In diesem Szenario war sie 20.6% schneller wie die Coldline. Beim Upload von 100 Dateien wurde ein Unterschied von 63% festgestellt, wobei die Coldline etwa 20 Sekunden und die OneZone-IA nur etwa sieben Sekunden benötigte. Bei 1000 Dateien bewegten sich beide Speicherklassen im Minutenbereich, wobei auch hier die OneZone-IA 55.34% schneller wie die Coldline war. Beim Download von Dateien bewegten sich beide Klassen in Millisekunden Bereichen. Ab 1000 Dateien war die Coldline

ungefähr 55% langsamer als die OneZone-IA.

Die Speicherklassen Standard-IA und Nearline weisen bei den Speicherkosten nur geringfügige Preisunterschiede auf, die sich auf maximal zwei Euro belaufen. Dabei ist Nearline maximal zwei Euro günstiger als Standard-IA. Die Standard-IA eignet sich für seltene Datenabrufe, die jedoch eine schnellere Zugriffszeit erfordern, wenn sie benötigt werden. Die Nearline-Klasse ähnelt der Standard-IA und ist ebenfalls für Daten gedacht, die gelegentlich im Zeitraum von Wochen oder Monaten abgerufen werden. Sie ist ideal für Datensicherung, Datenmigration und Datenarchivierung. Beide Speicherklassen bieten einen Kompromiss zwischen Kosteneinsparungen und Datenzugriffszeiten.

Während der Performance Analyse wurden diese beiden Speicherklassen verglichen, da sie ähnliche Eigenschaften aufweisen. Es stellte sich heraus, dass sich die Dauer des Uploads und Downloads ab 10 Dateien zu unterscheiden began. Dabei war die Standard-IA beim Hochladen und Herunterladen der zehn Dateien fast dreimal so schnell wie die Nearline-Klasse. Die Nearline-Klasse schnitt beim Hochladen schlechter ab. Beim Herunterladen von Dateien gab es jedoch nur minimale Unterschiede, wobei die Nearline bei 1000 Dateien 61.55% langsamer war wie die Standard-IA.

Die Standardklassen beider Anbieter weisen kaum Unterschiede bei den PUT- und GET-Anfragen auf. Allerdings sind die Speicherungskosten in der Standardklasse von AWS 9.27% höher als bei GC. Beide Klassen eignen sich für Daten, auf die häufig zugegriffen wird und die eine hohe Verfügbarkeit, schnelle Zugriffszeiten und geringe Latenzzeiten erfordern. Beide Standardklassen sind für den allgemeinen Gebrauch optimiert, wobei die Speicherungskosten in diesen Klassen im Vergleich zu anderen Speicherklassen am höchsten sind.

Bei der Performance-Analyse schnitt die Standardklasse von AWS ab zehn Dateien etwa 54.4% besser ab als die entsprechende Klasse von GC. Beim Hochladen von 100 Dateien war die AWS-Standardklasse etwa 60.9% schneller und bei 1000 Dateien etwa 58% schneller wie die GC Standardklasse. Beim Herunterladen von Dateien waren die Unterschiede nicht groß genug, um eine eindeutige Aussage über die Überlegenheit einer Klasse zu treffen. Allerdings war die AWS-Standardklasse bei 1000 Dateien etwa 60% schneller.

Es ist anzumerken, dass bei den AWS Kosten auch die Vorabzahlung der Speicherung aller Objekte enthalten ist. Bei GC gibt es hingegen keine Vorabzahlung für die Speicherung von Objekten. Aus diesem Grund und auch durch die niedrigeren Gebühren der Speicherung ist GC die kostengünstigere Datenspeicherung.

Insgesamt war die Dauer des Uploads und Downloads der Speicherklassen von AWS in der Performance Analyse niedriger als bei GC. Ab einer Datei traten bereits erste Unterschiede auf, wobei GC mehr Zeit in Anspruch nahm. Die Performance-Messungen basieren jedoch lediglich auf groben Schätzungen innerhalb einer virtuellen Umgebung, die von Hetzner Cloud bereitgestellt wurde. Hier ist anzumerken, dass der Hetzner Server näher am AWS Server liegen und deshalb bessere Ergebnisse in der Performance als GCP erzielen kann. Faktoren wie die Auslastung des Netzwerks können die Ergebnisse beeinflussen und stellen keine aussagekräftigen Performance-Ergebnisse dar.

5.2 Bewertung des Prototyps

Für die Bewertung des Prototyps wurden die Anforderungen von leoticket herangezogen. Dabei war es entscheidend, den Prototypen so zu bauen, dass die sichere Speicherung gedeckt war. Für die Deckung der sicheren Speicherung wurden Methoden betrachtet, die beide Cloud Provider anboten. Dabei implementiert der Prototyp die SSE-KMS Methode beider Provider. Durch die eigene Erstellung des Schlüssels im KMS von AWS und GC hat der Nutzer mehr Kontrolle, da die Schlüssel von ihm selbst erstellt werden. Der Schlüssel wird vom KMS gespeichert und muss daher nicht vom Nutzer extern gespeichert und verwaltet werden. Der Prototyp kann jedoch so umgebaut werden, dass auch eine andere Methode wie die SSE-C verwendet werden kann. Die SSE-C bietet eine höhere Sicherheit, da der Nutzer den Schüssel selbst generieren und speichern muss. Dies führt aber auch zum Risiko, den Schlüssel zu verlieren. In diesem Fall kann auf die Objekte im Bucket nicht mehr zugegriffen werden. Noch ein Kriterium von leoticket war die hohe Verfügbarkeit der Daten. Der Prototyp wurde so gebaut, dass der Nutzer die Speicherklasse für AWS selbst wählen kann. In GC funktioniert das, indem das Bucket die richtige Speicherklasse bereits eingestellt hat. Für die Verfügbarkeit sind jedoch die Cloud Provider verantwortlich. Hier versprechen beide Provider eine Verfügbarkeit von mindestens 99.5%. Diese Verfügbarkeit hängt von den verschiedenen Speicherklassen ab. Über Terraform werden Buckets automatisch mit den Einstellungen konform zu den Anforderungen von leoticket erstellt. Diese beinhalten die Konfigurationen von:

- Object Versioning
- Lifecycle Rules
- Object Ownership
- Data Encryption
- Object Logging
- Bucket ACL
- Public Access Block

Die Objekt Versionierung dient zur Steigerung der Verfügbarkeit, falls Daten unerwünscht gelöscht oder überschrieben werden. Für die Anforderung der Integration in Software-Produkte wie leoticket sorgen die SDKs der beiden Provider. Diese unterstützen mehrere Programmiersprachen. Sie werden durch Spring Boot unterstützt und können auch mit Maven oder Gradle gebaut werden. Die Anbindung verläuft gemäß der offiziellen Dokumentation der beiden Cloud Provider. Die Dokumentationen sind verständlich und auf dem aktuellsten Stand und beschreiben deren API. AWS und GC bieten auch neue Versionen an und updaten die SDKs. Die Generierung der signierten URLs ist für die Bereitstellung der Dateien zuständig. Diese Anforderung war das Hauptmerkmal von leoticket. Dabei ist es bei beiden Providern möglich, signierte, zeitlich begrenzte URLs zu generieren und diese Nutzern bereitzustellen. Empfänger der URL können so ihre Tickets und Rechnungen herunterladen. Für die Performance Analyse wird eine Methode zur Verfügung gestellt, die Testdateien in der Größe von 100 KB erstellen und in Buckets hoch- und herunterladen kann.

Insgesamt ist der Prototyp ausbaufähig und stellt für diese Arbeit einen Vergleich zwischen beiden Cloud Providern dar. Für den besseren Vergleich beider Cloud Provider wurden die Technologien ausprobiert und genutzt. Durch Änderungen an der Terraform Konfiguration, der Speicherklassen Variable für AWS und an der Methode der Datenverschlüsselung kann der Prototyp angepasst werden. Eine Schwäche des Prototyps besteht darin, dass keine genauen Performance Messungen durchgeführt werden können aufgrund genannter Einflüsse, welche die Ergebnisse beeinflussen können. Er dient lediglich dem groben Vergleich. Eine weitere Schwäche des Prototyps besteht darin, dass Entwickler bei der Integration des Prototyps in eigenen Anwendungen Anpassungen durchführen müssen, indem die Hauptklasse des Prototyps so umgebaut werden muss, dass sie dem gewünschten Verhalten entspricht.

6 Fazit

In diesem Kapitel werden die zuvor gestellten Fragen beantwortet. Darüber hinaus wird die potenzielle Anwendung des Prototyps diskutiert.

6.1 Beantwortung der Forschungsfrage

Folgende Fragen wurden am Anfang gestellt:

- Welches Speichersystem ist im Hinblick auf Kosten, Performance und Verfügbarkeit für die Persistenz von Binärdaten besonders geeignet?
- Wie können Daten durch sichere, zeitlich begrenzte URLs bereitgestellt werden?

Um die erste Frage zu beantworten, werden die Punkte des theoretischen Teils aufgegriffen. Es wurden verschiedene Speicherarten wie Objekt-, Block-, und File Storage untersucht. Dabei stellte sich heraus, dass Objekt Storage als Speichersystem für die Anforderungen von leoticket geeignet ist. Einige Cloud Provider stellen Objekt Storage zur Speicherung von Daten zur Verfügung und sind im Markt stark vertreten. Es wurden die zwei größten Cloud Provider AWS und GCP betrachtet und eine Vergleichsbasis hergestellt im Hinblick auf Kosten, Performance, Verfügbarkeit, Sicherheit, Bereitstellung der Daten und API Anbindungen. Bei der sicheren Speicherung sollten die Daten vertraulich gespeichert werden und nur Berechtigte Zugriff auf sie haben.

Datenverschlüsselungsmethoden wurden bei beiden Cloud Providern betrachtet und dabei festgestellt, dass die SSE C zwar die stärkste unabhängige Sicherheit bietet, jedoch das Risiko besteht, selbstverwaltete und gespeicherte Schlüssel zu verlieren. Außerdem müssten Mitarbeiter dafür geschult werden, was extra Aufwand bedeutet. Aus diesem Grund wurde für die Implementierung des Prototyps die SSE-KMS customer-managed Methode verwendet, damit der Nutzer die Schlüssel im KMS von den Providern selbst erstellen und verwalten kann. So bleibt die Kontrolle erhalten und verschafft höhere Sicherheit.

Je nach den gewählten Speicherklassen versprechen beide Anbieter eine Verfügbarkeit von mindestens 99.5%. Bei der Untersuchung der Speicherklassen in Verbindung mit Kosten und Performance stellte sich heraus, dass die Standard-IA von AWS und die Nearline von GC besser zu den Anforderungen von leoticket passen. Da die Latenz für leoticket kein Kriterium darstellt und vernachlässigt werden kann, fallen die Standard Klassen beider Provider weg. Die Standardklassen bieten zwar eine leistungsfähige Speicherlösung, jedoch gehen die eingesetzten Mittel über das hinaus, was tatsächlich benötigt wird.

Da Daten über einen Zeitraum von bis zu zehn Jahren gespeichert werden müssen, sind die Speicherkosten für die Standardklassen im Vergleich zu anderen Optionen zu hoch. Die Speicherung der Daten auf längerer Sicht steht mehr im Fokus, da auf eine Datei im Durschnitt nur zweimal zugegriffen wird. Die OneZone-IA fällt ebenfalls weg, da die Daten nur in einer Availability Zone gespeichert werden. Das Risiko ergibt sich durch Nicht-Verfügbarkeit der Daten durch Speicherung

in lediglich einer Zone. Daten müssen auf Abruf schnell zugreifbar sein, deshalb sind die OneZone-IA und die Coldline nicht geeignet, da sie eher für selten abgerufen Daten angepasst sind. Die Abrufkosten dieser Speicherklassen sind am höchsten und nicht zu empfehlen.

Insgesamt wird für die Persistenz von Binärdaten ein Object Storage mit den Speicherklassen Standard-IA von AWS und Nearline von GC empfohlen. Sie bieten die nötigen Funktionen an, um die Anforderungen zu decken und kostengünstig Daten für längere Zeit zu speichern und dabei eine hohe Verfügbarkeit und Performance zu bieten. Die Entscheidung hängt auch von den persönlichen Präferenzen des Unternehmens ab. Beide Provider bieten eine gute Objektspeicherung kostengünstig und leistungsfähig an.

Bei der zweiten gestellten Frage geht es um die Bereitstellung der Daten durch signierte zeitlich begrenzte URLs. Diese Frage wurde durch Vergleichen der SDKs bei der Implementierung des Prototypen untersucht und bewertet. Es stellt sich heraus, dass beide Cloud Provider die Funktionen anbieten, signierte URLs zu erstellen und zeitlich begrenzt bereitzustellen. Mithilfe des Prototypen kann man Dateien hochladen und sie durch URLs bereitstellen. Durch Klicken auf den generierten Link werden die Daten heruntergeladen. So kann verhindert werden, Dateien direkt in Email Anhängen hinzuzufügen, sondern durch Links bereitzustellen. Diese Dateien werden von den Buckets entschlüsselt heruntergeladen und Nutzer können ohne AWS oder GC Credentials darauf zugreifen. Über den Prototypen kann man die Minuten, in denen der Link valide ist, angeben. GC und AWS stellen ausführliche Dokumentationen auf den offiziellen Seiten bereit, um diese Funktionen zu implementieren und in verschiedenen Programmiersprachen anzuwenden.

So kann leoticket vom alten System zu der neuen empfohlenen Speicherlösung wechseln, um Daten sicher und schnell bereitzustellen und für längere Zeit zu speichern. Es ist zu beachten, dass diese Arbeit lediglich dem Vergleich und der Veranschaulichung beider Cloud Provider dient und dass jedes Unternehmen unterschiedliche Anforderungen aufweist. Diese Arbeit dient als Stütze und zum Testen der Technologien auf Basis des Prototypen.

6.2 Potenzielle Anwendung des Prototyps

Der Prototyp wurde entwickelt, um sich mit den Technologien der Cloud Provider auseinanderzusetzen. Durch die Anwendung konnten Performance Analysen durchgeführt werden, die zur Auswahl des Speichersystems beitragen. Die Anwendung kann ausgebaut werden, sodass sie den Bedürfnissen der Unternehmen entspricht. Sie stellt eine Bibliothek dar, die in eigene Anwendungen integriert werden kann. Um sich mit den Technologien vertraut zu machen, können Dateien hoch- und heruntergeladen werden. Außerdem ist es möglich Terraform Buckets mit den benötigten Einstellungen und Rechten zu erstellen ohne die Cloud Konsole verwenden zu müssen.

Der Prototyp wurde dabei an die Anforderungen von leoticket angelehnt und angepasst, damit er in das Produkt integriert werden kann. Er dient zur Hilfestellung für das Wechseln von Galera Cluster in ein neues Objekt Storage System für Binärdaten.

7 Literaturverzeichnis

Internetquellen

aws-cdn [1] AWS: CloudFront Use Cases: Accelerate static website content delivery. Abgerufen am 12.06.2023. URL: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/IntroductionUseCases.html.

ws-availability [2] AWS: Data Protection in Amazon S3. Abgerufen am 14.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/DataDurability.html.

aws-iam-s3 [3] AWS: Security: Identity and Access Management. Abgerufen am 11.05.2023. URL: https://docs.aws.amazon.com/de_de/AmazonS3/latest/userguide//access-control-overview.html.

aws-sse-c [4] AWS: Using server-side encryption with customer-provided keys (SSE-C). Abgerufen am 12.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ServerSideEncryptionCustomerKey html.

gcp-sla [5] Google Cloud: Cloud Storage Service Level Agreement (SLA). Abgerufen am 17.05.2023. URL: https://cloud.google.com/storage/sla.

gcp-autoscale [6] Google Cloud: Request rate and access distribution guidelines: Auto Scaling. Abgerufen am 17.05.2023. URL: https://cloud.google.com/storage/docs/request-rate.

-cloud-announce [7] Spencer Gibb: Spring Cloud AWS 2.3 is now available. Abgerufen am 24.05.2023. URL: https://spring.io/blog/2021/03/17/spring-cloud-aws-2-3-is-now-available#why-has-the-package-name-changed.

ibm-storage [8] IBM: What is object storage? Abgerufen am 07.05.2023. URL: https://www.ibm.com/topics/object-storage.

dataCore-OS [9] Mike Ivanov: File Storage vs. Object Storage: Understanding Differences, Applications and Benefits, What is Object Storage? (2020). Abgerufen am 07.05.2023. URL: https://www.datacore.com/blog/file-object-storage-differences/.

leomedia-web [10] GmbH Leomedia: Bachelor- und Master-Themen. URL: https://www.leomedia.de/unternehmen/abschlussarbeiten/.

gcp-blog [11] Colt McAnlis: Optimizing your Cloud Storage performance: Google Cloud Performance Atlas. Abgerufen am 18.05.2023. URL: https://cloud.google.com/blog/products/gcp/optimizing-your-cloud-storage-performance-google-cloud-performance-atlas/.

oracle-bigdata [12] Oracle: Was versteht man unter Big Data? URL: https://www.oracle.com/de/big-data/what-is-big-data/.

redHat-storage [13] RedHat: File storage, block storage, or object storage? (2018). Abgerufen am 06.05.2023. URL: https://www.redhat.com/en/topics/data-storage/file-block-object-storage.

aws-api [14] Amazon S3: Amazon S3 REST API Introduction. Abgerufen am 18.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html.

aws-sdk [15] Amazon S3: Developing with Amazon S3 using the AWS SDKs, and explorers: Using this service with an AWS SDK. Abgerufen am 18.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingAWSSDK.html.

rformance-guide [16] Amazon S3: Performance Guidelines for Amazon S3. Abgerufen am 18.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/optimizing-performance-guidelines.html.

aws-signed-urls [17] Amazon S3: Using presigned URLs. Abgerufen am 19.05.2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-presigned-url.html.

gcp-encrypt [18] Google Cloud Storage: Google-managed encryption keys. Abgerufen am 13.05.2023. URL: https://cloud.google.com/storage/docs/encryption/default-keys.

gc-perfdiag [19] Google Cloud Storage: perfdiag - Run performance diagnostic. Abgerufen am 21.05.2023. URL: https://cloud.google.com/storage/docs/gsutil/commands/perfdiag.

gc-signedUrl [20] Google Cloud Storage: Signed URLs - Options for generating a signed URL. Abgerufen am 13.06.2023. URL: https://cloud.google.com/storage/docs/access-control/signed-urls.

nx-fileScala [21] Lauren Wahlman: Data Storage: Exploring File, Block, and Object Storage. (2022). Abgerufen am 06.05.2023. URL: https://www.nutanix.com/blog/block-storage-vs-object-storage-vs-file-storage.

8 Anhang

8.1 Repositories

8.1.1 Github Link

Prototyp:

HTTPS: https://github.com/gmzbae/cloud-gcs-aws3.git

`SSH: git clone git@github.com:gmzbae/cloud-gcs-aws3.git`

Thesis:

HTTPS: https://github.com/gmzbae/bachelor-thesis-latex.git

`SSH: git clone git@github.com:gmzbae/bachelor-thesis-latex.git`

8.1.2 Code Snippets

```
  <?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org
    /2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache
    .org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.7</version>
        <relativePath/>
    </parent>
    <groupId>de.leomedia</groupId>
    <artifactId>cloud_gcStorage_AwsS3</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>cloud_gcStorage_AwsS3</name>
    <description>cloud_gcStorage_AwsS3</description>
    <properties>
        <java.version>17</java.version>
        <spring-cloud-gcp.version>4.3.1</spring-cloud-gcp.version>
        <spring-cloud.version>2022.0.3</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.google.cloud</groupId>
            <artifactId>spring-cloud-gcp-starter-storage</artifactId>
        </dependency>

        <dependency>
            <groupId>io.awspring.cloud</groupId>
            <artifactId>spring-cloud-aws-s3</artifactId>
            <version>3.0.0</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>com.google.cloud</groupId>
                <artifactId>spring-cloud-gcp-dependencies</artifactId>
                <version>${spring-cloud-gcp.version}</version>
                <type>pom</type>
```

```
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Listing 8.1: pom.xml - Abhängigkeiten des Prototyps

```
CLOUD_PROVIDER=${cloud_provider}
PROJECT_ID=${GOOGLE_PROJECT_ID}
BUCKET_NAME=${bucket_name}
GC_JSON_KEY_PATH=${GOOGLE_SERVICE_ACCOUNT_FILEPATH}
ENCRYPTION_KEY=${sse_kms_key_id_arn}
STORAGE_CLASS=${storage_class}

TEST_BUCKET_NAME=${test_bucket_name}
TEST_FILES_PATH=${test_files_path}
TEST_OBJECT_KEY_PATTERN=${test_object_key_pattern}
TEST_FILE_COUNT=${test_file_count}
TEST_GC_ENCRYPTION_KEY=${test_gc_encryption_key}
TEST_AWS_ENCRYPTION_KEY=${test_aws_encryption_key}
```

Listing 8.2: application.properties Datei - Umgebungsvariablen des Prototyps

```
public static CloudStorageService getCloudStorageService(String cloudProvider,
    String projectId, String jsonKeyPath, S3Presigner presigner, S3Client s3Client)
     throws IOException {
     if ("AWS".equalsIgnoreCase(cloudProvider)) {
         return new AWSS3StorageService(s3Client, presigner);
     } else if ("Google Cloud".equalsIgnoreCase(cloudProvider)) {
         return new GCStorageService(projectId, jsonKeyPath);
     } else {
         throw new IllegalArgumentException("Invalid cloud provider: " +
    cloudProvider);
     }
 }
```

Listing 8.3: getCloudStorageService() Methode der Klasse CloudStorageServiceFactory

```java
/**
* This test method calls the given methods and starts the performance runner.
*
*/
@Test
public void performanceRunner() throws IOException {

    generateFiles();

    calculateUploadObjectsToS3_TimeMeasure();
    calculateUploadObjectsToCloudStorage_TimeMeasure();

    calculateDownloadObjectsFromS3_TimeMeasure();
    calculateDownloadObjectsFromCloudStorage_TimeMeasure();

}
```

Listing 8.4: performanceRunner() Methode der Klasse PerformanceTest

```java
/**
 * This method uploads multiple objects to S3
 * and measures the current time it took in milliseconds.
 */
public void calculateUploadObjectsToS3_TimeMeasure() {

    AWSS3StorageService awss3StorageService = new AWSS3StorageService(s3Client,
    S3Presigner.builder().build());

    long startTime = System.currentTimeMillis();

    for(int i = 1; i <= fileCount; i++){
        awss3StorageService.uploadObject(bucketName, key + i + ".txt", filePath +
    key + i + ".txt", aws_encryption_key, storageClass);
    }

    long endTime = System.currentTimeMillis();
    long elapsedTime = endTime - startTime;

    logger.info("Elapsed Time for " + fileCount + "  Object Uploads in S3: " +
    elapsedTime +  " ms.");

}
```

Listing 8.5: calculateUploadObjectsToS3_TimeMeasure() Methode der Klasse PerformanceTest

```java
/**
 * This method downloads multiple objects from S3
 * and measures the current time it took in milliseconds.
 */
public void calculateDownloadObjectsFromS3_TimeMeasure(){

    AWSS3StorageService awss3StorageService = new AWSS3StorageService(s3Client,
    S3Presigner.builder().build());

    long startTime = System.currentTimeMillis();

    for(int i = 1; i <= fileCount; i++){
        awss3StorageService.getPresignedUrl(bucketName, key + i + ".txt", 60,
    aws_encryption_key);
    }

    long endTime = System.currentTimeMillis();
    long elapsedTime = endTime - startTime;

    logger.info("Elapsed Time for " + fileCount + " Object Downloads in S3: " +
    elapsedTime +  " ms.");

}
```

Listing 8.6: calculateDownloadObjectsFromS3_TimeMeasure() Methode der Klasse PerformanceTest

```java
/**
 * This method uploads multiple objects to Cloud Storage
 * and measures the current time it took in milliseconds.
 */
public void calculateUploadObjectsToCloudStorage_TimeMeasure() throws IOException {
    GCStorageService googleCloudStorageService = new GCStorageService(projectId,
    gcJsonKeyPath);

    long startTime = System.currentTimeMillis();

    for(int i = 1; i <= fileCount; i++){
        googleCloudStorageService.uploadObject(bucketName, key + i + ".txt",
    filePath + key + i + ".txt", gc_encryption_key, storageClass);
    }

    long endTime = System.currentTimeMillis();
    long elapsedTime = endTime - startTime;

    logger.info("Elapsed Time for " + fileCount + " Object Uploads in Cloud Storage
    : " + elapsedTime +  " ms.");

}
```

Listing 8.7: calculateUploadObjectsToCloudStorage_TimeMeasure() Methode der Klasse PerformanceTest

```java
/**
 * This method downloads multiple objects from Cloud Storage
 * and measures the current time it took in milliseconds.
 */
public void calculateDownloadObjectsFromCloudStorage_TimeMeasure() throws
    IOException {

    GCStorageService googleCloudStorageService = new GCStorageService(projectId,
    gcJsonKeyPath);

    long startTime = System.currentTimeMillis();

    for(int i = 1; i <= fileCount; i++){
        googleCloudStorageService.getPresignedUrl(bucketName, key + i + ".txt", 60,
     gc_encryption_key);
    }

    long endTime = System.currentTimeMillis();
    long elapsedTime = endTime - startTime;

    logger.info("Elapsed Time for " + fileCount + " Object Downloads in Cloud
    Storage: " + elapsedTime +  " ms.");

}
```

Listing 8.8: calculateDownloadObjectsFromCloudStorage_TimeMeasure() Methode der Klasse PerformanceTest

```java
/**
 * This method writes the generated content into the generated file.
 */
public void generateFiles(){

    for(int i = 1; i <= fileCount; i++){
        String filename = filePath + key + i + ".txt";
        String content = generateContent();

        try {
            FileOutputStream fos = new FileOutputStream(filename);
            fos.write(content.getBytes(StandardCharsets.UTF_8));
            fos.close();
            logger.info("File generated: " + filename);
        } catch (IOException e) {
            logger.error("An error occurred while generating the file: " + e.
    getMessage());
        }
    }
}
```

Listing 8.9: generateFiles() Methode der Klasse PerformanceTest

```java
/**
 * This method generates random string content in size  of 100kb.
 * @return a random generated string content
 */
public String generateContent() {
    StringBuilder sb = new StringBuilder();
    int contentSize = 100 * 1024; // Convert KB to bytes

    while (sb.length() < contentSize) {
        sb.append("Lorem ipsum dolor sit amet, consectetur adipiscing elit. ");
    }

    logger.info("Generated string content");

    return sb.substring(0, contentSize);
}
```

Listing 8.10: generateContent() Methode der Klasse PerformanceTest