HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Skalierbare Plattform zur Verarbeitung von Geodaten auf Basis von Kubernetes

Richard Bischof

Master-Arbeit im Studiengang „Angewandte Informatik“

Korrigierte Fassung vom 26. Oktober 2020

Autor Richard Bischof
Matrikelnummer 1498918
richard.bischof@stud.hs-hannover.de

Erstprüferin: Prof. Dr. Arne Koschel
Abteilung Informatik, Fakultät IV
Hochschule Hannover
arne.koschel@hs-hannover.de

Zweitprüfer: Dr. Marcel Ziems
Landesamt für Geoinformation und Landesvermessung
Landesbetrieb Landesvermessung und Geobasisdaten
marcel.ziems@lgln.niedersachsen.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Master-Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 26. Oktober 2020 Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis des Quelltexts

Abkürzungsverzeichnis

CSC Cloud Service Customer . . . 13
CSP Cloud Service Provider . . . 13
NIST National Institute Of Standards And Technology . . . 13
INSPIRE Infrastructure for Spatial Information in the European Community
OZG Online-Zugangsgesetz
NDIG Gesetz über die digitale Verwaltung und Informationssicherheit
IaaS Infrastructure-As-A-Service . . . 14
PaaS Plattform-As-A-Service . . . 14
SaaS Software-As-A-Service . . . 14
BaaS Backend-As-A-Service . . . 16
BPaaS Business-Process-As-A-Service . . . 16
KaaS Kubernetes-As-A-Service . . . 16
FaaS Function-As-A-Service . . . 16
DSL Domain Specific Language . . . 18
VM Virtuelle Maschinen . . . 18
API Application Programming Interface . . . 19
OCI Open Container Image . . . 19
CLI Command Line Interface . . . 21
CNCF Cloud Native Computing Foundation . . . 22
CPU Central Processing Unit . . . 24
RAM Random Access Memory . . . 24
GPU Graphics Processing Unit . . . 24
TPU Tensor Processing Unit . . . 24
IP Internet Protocol . . . 24
DNS Domain Name System . . . 24
URL Uniform Resource Locator . . . 25
REST Representational State Transfer . . . 25
SSL Secure Sockets Layer . . . 25
TLS Transport Layer Security . . . 25
YAML Yet Another Markup Language . . . 27
EPSG European Petroleum Group Geodesy . . . 29
OSGeo Open-Source Geospatial Foundation . . . 29
GeoJSON Geography JavaScript Object Notation . . . 29
ECW Enhanced-Compress-Wavelets . . . 30
GeoTIFF Geography Tagged Image File Format . . . 30
TIFF Tagged Image File Format . . . 33

1 Einführung

1.1 Motivation

Kaum ein Begriff wurde in der kürzeren Vergangenheit innerhalb der IT-Branche so kontrovers diskutiert wie *Cloud Computing*. Schnellere Skalierbarkeit von Leistungen, verteilter Zugriff auf Ressourcen und geringere Kosten sind nur wenige der von Befürwortern oftmals ins Feld geführten Argumente [Bitkom Research, 2019]. Die heutzutage ganz selbstverständliche Verwendung von mobilen Endgeräten wie Smartphones und die damit verbundene Nutzung von Kommunikations-, Medien-, oder Backupdiensten erscheint nur durch skalierbare Plattformen möglich. Der Siegeszug von Cloud Plattformen ist kaum zu übersehen, bekannte IT-Firmen bewerben ihre Cloud Plattformen aktuell sogar im Fernsehen und stellen die Einfachheit der Nutzung von komplexen Systemen auf ihren Plattformen in den Mittelpunkt.

Behörden und staatliche Organisationen wenden sich eher zögerlich Cloud Computing zu [Handelsblatt, 2015]. Obwohl sie meist außerhalb des privat-wirtschaftlichen Marktes agieren, müssen sie sich mit veränderten Gewohnheiten und Erwartungen der Nutzer auseinandersetzen. So wird unter dem Schlagwort Digitalisierung die Verfügbarmachung jeglicher staatlichen Dienstleistung über digitale Plattformen und Kommunikationskanäle gefordert. Der Innovationsdruck auf Behörden und staatliche Organisationen wird so stark erhöht. Zahlreiche Gesetze und Normen wie z.B. das *Online-Zugangsgesetz* (OZG), *Infrastructure for Spatial Information in the European Community* (INSPIRE) oder das momentan im Beschlussverfahren befindliche *Gesetz über die digitale Verwaltung und Informationssicherheit* (NDIG) verleihen dieser Forderung Nachdruck. Bei Nichteinhaltung oder Verzögerung drohen Behörden empfindliche Strafen.

Für geodatenhaltende Stellen ist besonders die Verarbeitung und Analyse von vergleichsweise großen Datenmengen zur Bereitstellung digitaler Services herausfordernd [de Lange, 2013]. Beispielhaft sind Satellitenbilder, Laserscans oder Schrägluftbilder zu nennen. Diese bilden die Grundlage für amtliche Produkte wie digitale Orthophotos, Gelände- und Oberflächenmodelle oder Landnutzungsklassifikationen.

Diesen neuen Anforderungen stehen komplexe historisch gewachsene Altsysteme gegenüber. In diesen Altsystemen stecken bereits erhebliche Investitionsmaßnahmen der Vergangenheit. Dadurch ist die Architektur dieser Altsysteme derart komplex geworden, dass die notwendigen Anpassungen zur Digitalisierung nahezu unmöglich scheinen. Einen Ausweg aus solchen *Big Ball Of Mud* Architekturen stellt die Kapselung und

sukzessive Überführung einzelner Komponenten in eine neue und definierte Architektur dar [Foote and Yoder, 2000]. Gleichzeitig werden immer mehr Forderungen nach dem Einsatz von marktüblichen Technologien und standardisierten Architekturen laut([Bundesministerium des Inneren, 2008] [Bundesministerium des Inneren, 2017]).

Es ist daher zu klären, wie eine performante Verarbeitung von amtlichen Geodaten durch Cloud Computing möglich ist.

1.2 Ziele

Das Ziel dieser Arbeit ist die Konzeption und Implementierung einer skalierbaren Plattform für die laufzeit- und ressourcenoptimierte Verarbeitung von großen Rasterdatenbeständen.

Der in dieser Arbeit vorgeschlagene plattformbasierte Ansatz soll im Wesentlichen zwei Probleme lösen.

1. Den zumeist hohen Zeitaufwand von sequentieller Verarbeitung von Geodaten.
2. Die mangelnde Flexibilität bisheriger Architekturen, welche die Anpassung durch neue Anforderungen verhindern.

Der vorgeschlagene plattformbasierte Ansatz umfasst ein Speicherkonzept für Eingabe- und Ausgabedaten, sowie eine Umgebung zur parallelen Ausführung beliebiger Algorithmen. Dazu wird entlang eines praktischen Beispiels ein Konzept erstellt und ein lauffähiger Prototyp entwickelt. Als Beispiel dient der Managementservice für historische Kriegsluftbilder des Kampfmittelbeseitigungsdienstes Niedersachsen.

Nach Recherche der relevanten Grundlagen von Cloud Computing und Geodatenverarbeitung werden anhand des konkreten Anwendungsbeispiels zielführende Bewertungskriterien für die Auswahl von Architektur und Technologie herausgearbeitet. Darauf aufbauend wird ein alternatives Speicherkonzept für Rasterdaten vorgestellt, welches auf parallele Prozessierung durch generische Algorithmen ausgelegt ist.

Dieser Entwurf wird schließlich zur Implementierung gebracht. Im Vergleich mit anderen Plattformen erfolgt abschließend die Bewertung des vorgestellten Ansatzes.

1.3 Dokumentenstruktur

Die vorliegende Arbeit gliedert sich in acht Kapitel:

In Kapitel 1 werden nach der thematischen Motivation die Ziele und Struktur dieser Arbeit erläutert.

In Kapitel 2 erfolgt eine Beschreibung der Grundlagen, auf denen das Ergebnis dieser Arbeit beruht. Dazu zählen insbesondere Cloud Computing, Skalierbarkeit und Container-Technologie. Über die Grundlagen der Geoinformatik und Geodatenverarbeitung erfolgt eine Überleitung zum Anwendungsbeispiel.

In Kapitel 3 wird das Anwendungsbeispiel der historischen Kriegsluftbilder des Kampfmittelbeseitigungsdienstes Niedersachsen erklärt. Bereits vorhandene Transformations-Prozesse werden vorgestellt und ein Ausblick auf Analyse-Prozesse gegeben. Anschließend werden strategische Ziele, relevante Stakeholder und Nutzungsszenarien vorgestellt. Schließlich werden funktionale Anforderungen, Qualitätsanforderungen und Rahmenbedingungen erhoben.

Kapitel 4 widmet sich der Analyse von Szenarien und Anforderungen. Zunächst werden geeignete Bewertungskriterien bestimmt. Aus einer Systemanalyse ergeben sich Komponenten und wesentliche Konzepte. Aus den Bewertungskriterien werden schließlich die Architektur und Technologie der Plattform abgeleitet.

In Kapitel 5 wird die skalierbare Plattform zur Verarbeitung von Geodaten auf Basis von Kubernetes entworfen. Dazu werden die grundlegende Idee und Applikations-Architektur erläutert. Besonders wird auf die Optimierung der Datenhaltung durch Cloud-Optimized GeoTIFF eingegangen.

In Kapitel 6 wird die Implementierung des Prototypens auf der *Google Cloud Platform* erklärt. Dazu werden verwendete Konzepte von Kubernetes am Beispiel der Implementierung erläutert. Das Kapitel schließt mit der konzeptionellen Beschreibung der Integration GPU-basierter Analyseverfahren.

In Kapitel 7 wird vorgestellte Ansatz und der implementierte Prototyp anhand von Experimenten, den gestellten Anforderungen und dem Vergleich mit domänenspezifischen Plattformen evaluiert.

In Kapitel 8 werden die gestellten Forschungsfragen abschließend beantwortet. In einem weiteren Ausblick werden denkbare technische und anwendungsorientierte Handlungsfelder motiviert.

1.4 Konventionen

In dieser Arbeit werden eine Reihe von Abkürzungen verwendet. Bei der ersten Verwendung werden diese ausgeschrieben und mit ihrer Abkürzung in Klammern versehen. Mehrfach verwendete Abkürzungen sind im Abkürzungsverzeichnis aufgeführt.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte und Technologien dieser Arbeit vorgestellt. Besonders relevant sind die Grundlagen von Cloud Computing, unterschiedliche Strategien von Skalierbarkeit, das Prinzip *Infrastructure-As-Code*, aktuelle Container-Technologien und die Grundlagen der Geoinformatik.

2.1 Cloud Computing

Cloud Computing kann im Allgemeinen als Nutzung von IT-Services unterschiedlicher Fertigungstiefe durch einen Cloud Service Customer (CSC) bei einem Cloud Service Provider (CSP) verstanden werden. Es existieren eine Reihe von Definitionen, diese gehen aber meist auf eine Definition des amerikanischen National Institute Of Standards And Technology (NIST) zurück [Mell and Grance, 2011]. Oftmals werden auch konzeptionelle Erweiterungen der kürzeren Vergangenheit ergänzt. Das NIST führt in seiner Definition fünf essenzielle Charakteristika, drei Service-Modelle und vier Deployment Modelle für Cloud Computing auf.

2.1.1 Charakteristika

On-Demand Self-Service

On-Demand Self-Service bedeutet, dass Ressourcen eigenständig durch den CSC über eine zentrale und automatisierbare Schnittstelle provisioniert werden können.

Broad Network Access

Broad Network Access steht für eine hohe Verfügbarkeit der Ressourcen über Standardschnittstellen innerhalb der Cloud Infrastruktur, vor allem jedoch nach Außen um durch Smartphones, Laptops oder Computer erreichbar zu sein.

Ressource Pooling

Ressource Pooling ermöglicht dem CSP die Abstraktion der bereitgestellten Ressourcen von der tatsächlich darunter liegenden Infrastruktur. Beispielsweise erfolgt eine Spezifizierung eines zu bereitstellenden Servers durch den CSC lediglich hinsichtlich der Anzahl der Prozessoren, Größe von Arbeits- und Festplattenspeicher sowie geografische Region (meist Standort des Rechenzentrums). So werden Server unterschiedlicher CSC auf derselben Infrastruktur betrieben. Mithilfe von Sicherheitsmechanismen findet eine Separierung der individuellen Daten statt.

Rapid Elasticity

Rapid Elasticity garantiert eine dynamische Anpassung der provisionierten Ressourcen durch den CSP, um auf schwankende Lastanforderungen zu reagieren. Dem CSP wird eine unbegrenzte Elastizität suggeriert, die de facto durch die Ressourcen der Cloud Infrastruktur begrenzt ist.

Measured Service

Measured Service ermöglicht das Monitoring der bereitgestellten Ressourcen, sodass sowohl dem CSC als auch dem CSP nutzbare Metriken zur Anpassung der angebotenen und bezogenen Dienste zur Verfügung gestellt werden.

2.1.2 Service-Modelle

Service-Modelle beschreiben die Fertigungstiefe des Cloud Services. Gleichzeitig erfolgt dadurch eine Abgrenzung der Betriebsverantwortung des CSP und der Zuständigkeit des CSC. Der Definition der NIST folgend, gibt es die drei Service-Modelle Infrastructure-As-A-Service (IaaS), Plattform-As-A-Service (PaaS) und Software-As-A-Service (SaaS). Diese Modelle werden im Folgenden erklärt.

Infrastructure-As-A-Service

Bei IaaS stellt der CSP ausschließlich Infrastruktur-Komponenten wie Rechenkapazität, Speicherplatz oder Netzwerk mithilfe von Virtualisierung zur Verfügung. Die Wahl eines Betriebssystems erfolgt nach zwei unterschiedlichen Verfahren: Entweder bietet der CSP eine Auswahl von auf der Cloud verfügbaren standardisierten Betriebssystemen an oder ermöglicht dem CSC die Verwendung eigener Images. Für die weitere Anpassung, wie z. B. Installation von Bibliotheken, Laufzeitumgebungen und letztendlich auch Applikation und Daten ist der CSC selbst verantwortlich.

Platform-As-A-Service

Eine erweiterte Fertigungstiefe wird im PaaS-Modell erreicht, in dem der CSP nicht nur das Betriebssystem, sondern auch die Middleware und Laufzeitumgebung verantwortet. Der CSC ist in diesem Modell lediglich für die Applikation und die Daten zuständig, was bedeutet, dass der CSC keinen Einfluss auf eingesetzte Laufzeit- und Bibliotheksversionen sowie die zugrundeliegende Infrastruktur hat.

Software-As-A-Service

Maximale Verantwortung trägt der CSP bei SaaS, dort wird auch die Verwaltung von den Applikationen und Daten von ihm verantwortet. Meist wird zur Anzeige auf Thin-Clients ein Browser oder eine native Applikation verwendet, die Kommunikation zu dem Service erfolgt über standardisierte Netzwerkprotokolle.

Zusammenfassend lassen sich die Unterschiede der erläuterten Service-Modelle der Abbildung 2.1 entnehmen.

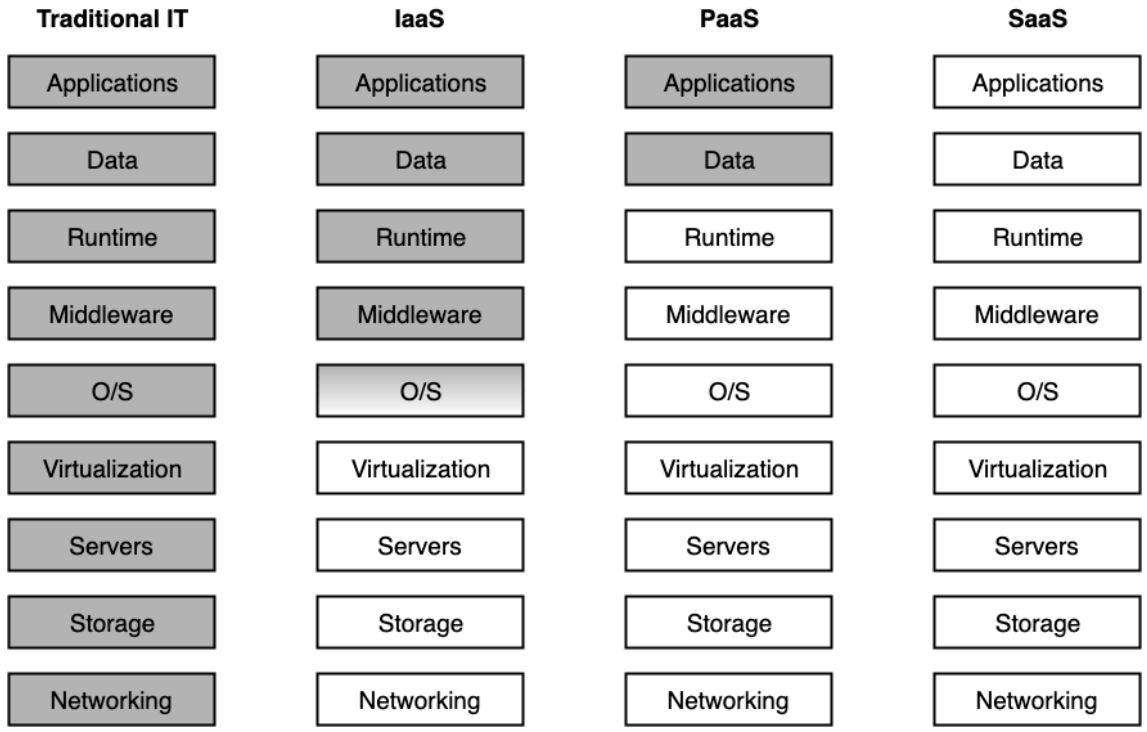

Abbildung 2.1: Vergleich unterschiedlicher Service-Modelle und traditioneller IT [Harms and Yamartino, 2010]

Diese Abbildung ist zur Verdeutlichung um das Modell der traditionellen IT ergänzt. Grau dargestellt sind die Schichten eines Systems, die in der Verantwortung des CSC stehen. Weiß dargestellt sind hingegen die Schichten, für die der CSP verantwortlich ist.

Ergänzung der Service-Modelle

Die *Cloud Computing* Definition des NIST wird von unterschiedlichen Quellen dahingehend kritisiert, dass die Definition der Service-Modelle nicht vollständig wäre. In der Literatur werden beispielsweise Backend-As-A-Service (BaaS) oder weniger technologisch getriebene Begriffe wie Business-Process-As-A-Service (BPaaS) angeführt.

Eine wichtige Ergänzung im Kontext dieser Arbeit ist Kubernetes-As-A-Service (KaaS). KaaS ist die Nutzung eines Kubernetes-Clusters, welches durch den CSP auf einer Public Cloud verwaltet und betrieben wird [Burns et al., 2018]. Das Service-Modell KaaS stellt jedoch mehr eine sprachliche, denn eine technologische Ergänzung der dargestellten Charakteristika dar, da es dieselbe Trennung der Verantwortlichkeiten wie PaaS bedeutet.

Eine auch technologisch bedeutsame Ergänzung ist mit Function-As-A-Service (FaaS) aufzuführen. In diesem Service-Modell erfolgt lediglich die Implementierung von zustandslosen Funktionen durch den CSC. In ihrer Gesamtheit bilden diese Funktionen die vollständige Geschäftslogik einer Applikation. Die Speicherung von Daten erfolgt in als Service konsumierten, außerhalb liegenden Speicherorten wie Datenbanken oder ObjectStores. Im Gegensatz zu anderen Service-Modellen gibt es keine permanent provisionierten Ressourcen, stattdessen wird erst durch den Aufruf einer einzelnen Funktion die Ausführung auf einer durch den CSP bereitgestellten Infrastruktur ausgelöst. Dadurch, dass der CSC keine klassischen Ressourcen wie Maschinen, Netzwerke oder Speicher verwaltet, wird dieses Konzept auch unter dem Schlagwort *Serverless* vermarktet. Dieses Service-Modell bietet vor allem Skalierungs- und Kostenvorteile, da die Funktionsaufrufe zustandslos sind und eine Abrechnung lediglich auf Basis der Ausführungszeit erfolgt. Zu beachten ist, dass meist durch den CSP Einschränkungen hinsichtlich verwendeter Programmiersprachen und Frameworks getroffen werden, da dieser nur für eine begrenzte Menge von Programmiersprachen einen zuverlässigen Service bieten kann.

2.1.3 Deployment Modelle

Deployment Modelle beschreiben die unterschiedlich ausgeprägten Nutzungen der bereitgestellten Ressourcen und werden vor allem durch den Nutzerkreis der jeweiligen Infrastrukturen voneinander abgegrenzt. So können bereits verfügbare Infrastrukturen integriert oder Anforderungen der Informationssicherheit, wie z. B. Vertraulichkeit, berücksichtigt werden.

Private Cloud

Eine Private Cloud steht Mitgliedern einer Organisation exklusiv zur Verfügung, beispielsweise wenn eine Cloud durch eine Firma betrieben wird und ausschließlich von ihren Mitarbeitern verwendet werden kann. Der Betrieb der Infrastruktur kann sowohl

in dem Rechenzentrum der Organisation (*On-Premise*), als auch außerhalb in einem Fremdrechenzentrum (*Off-Premise*) erfolgen.

Community Cloud

Die Community Cloud hebt die exklusive Nutzung durch eine Organisation auf und erweiterte diese um mehrere nutzungsberechtigte Organisationen, die sich zu einer Community zusammengeschlossen haben. Der Betrieb der Infrastruktur erfolgt sowohl On- als auch Off-Premise.

Public Cloud

Public Clouds können durch jedermann verwendet werden. Der Betrieb der Infrastruktur erfolgt in diesem Modell ausschließlich in Rechenzentren des CSP.

Hybrid Cloud

Hybrid Cloud vereinigen mindestens zwei von den bisherig vorgestellten Deployment-Modellen (Private, Community und Public Cloud). Der Einsatz einer Hybrid Cloud kann beispielsweise dann sinnvoll sein, wenn der Betrieb in einer Private Cloud für normale Workloads ausreichend ist, bei großen Workloads aber zu wenig Ressourcen vorhanden sind. In diesem Szenario können zusätzliche Ressourcen in der Public Cloud provisioniert werden, die einen Teil der Last auffangen. Ein anderes Szenario wäre die Verteilung von Datenspeicher und Rechenkapazität auf Private und Public Cloud, sodass besondere Anforderungen der Informationssicherheit hinsichtlich Vertraulichkeit und Verfügbarkeit von Daten nach individuellen Vorgaben (Compliances) umgesetzt werden können.

2.2 Skalierbarkeit

Als Skalierbarkeit wird die Eigenschaft eines Systems bezeichnet, veränderbar hinsichtlich seiner Leistungsfähigkeit zu sein. Gefordert wird dies bei Services meist, um einerseits bei großen Workloads weiterhin zuverlässig und performant zu sein und andererseits in Zeiten geringer Workloads wenig Ressourcen zu benutzen und damit günstig im Betrieb zu sein oder Ressourcen zur Nutzung durch andere Systeme freizugeben. Grundsätzlich werden vertikale und horizontale Skalierbarkeit unterschieden:

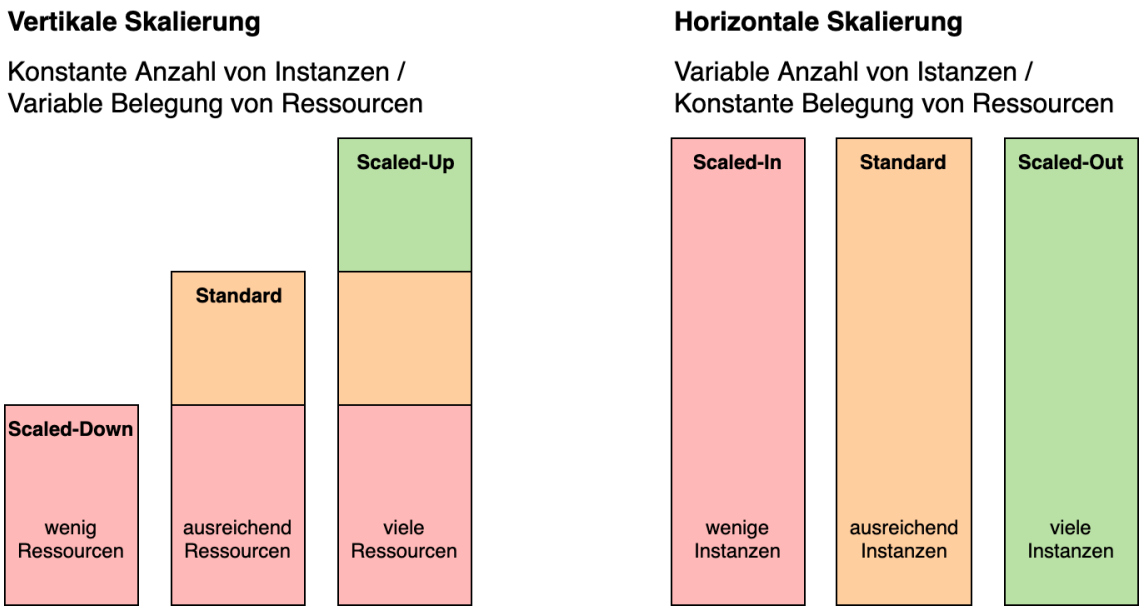

Abbildung 2.2: Vertikale (links) und horizontale (rechts) Skalierung

2.2.1 Vertikale Skalierbarkeit

Vertikale Skalierbarkeit bedeutet, dass die Anzahl der Instanzen konstant bleibt. Die Skalierbarkeit wird erreicht, indem die Ressourcen, z. B. Anzahl der Prozessorkerne oder Größe des Arbeitsspeichers, dieser Instanz verkleinert (*Scaled-Down*) oder vergrößert (*Scaled-Up*) werden. Aus Applikationssicht ist diese Art der Skalierung meist sehr einfach durchzuführen, da die veränderten Ressourcen durch das Betriebssystem verwaltet werden und somit hinter einer unveränderten Abstraktionsschicht liegen.

2.2.2 Horizontale Skalierbarkeit

Beim Verfahren der horizontalen Skalierbarkeit bleiben die Ressourcen der Instanzen hingegen unverändert. Dafür wird die Anzahl der Instanzen verringert (*Scaled-In*) oder erhöht (*Scaled-Up*). Dieses Verfahren erfordert zusätzliche Komponenten, wie z. B. Load-Balancer (Lastverteiler) oder zentralisierte Logging-Systeme. Anders als bei der vertikalen Skalierbarkeit muss dieses Verfahren durch die Applikations-Architektur explizit unterstützt werden.

2.3 Infrastructure-As-Code

Infrastructure-as-Code beschreibt das Prinzip nach dem Ressourcen (z. B. Virtuelle Maschinen (VM)) maschinenlesbar in einer Domain Specific Language (DSL) beschrieben

und in Konfigurationsdateien verwaltet werden können. Dieses Prinzip ist Grundlage für den hohen Automatisierungsgrad bei der Provisionierung von Cloud-Ressourcen, da so neben der Mensch-zu-Maschine Kommunikation über grafische Oberflächen auch eine Maschine-zu-Maschine Kommunikation über Application Programming Interface (API) ermöglicht wird. Grundsätzlich wird zwischen deklarativen und imperativen Ansätzen unterschieden [Burns et al., 2018] . Im deklarativen Ansatz wird lediglich der gewünschte Zielzustand beschrieben, wobei im imperativen Ansatz die konkrete Abfolge von Kommandos zum Erreichen des Zielzustandes definiert wird.

2.4 Container

Container sind Prozesse, die zwar innerhalb eines selben Hosts laufen, jedoch durch Isolierung voneinander und zum Host gekapselt sind. Unter Linux wird dies durch zwei Mechanismen erreicht: cgroups beschränkt den Zugriff auf Prozessor, Arbeitsspeicher, Schreib und Lesezugriffe, Netzwerk, etc. für eine Gruppe von Prozessen. Durch namespaces wird die Gruppierung von Prozessen und damit Sichtbarkeit von Prozessen untereinander erreicht. Es existieren mehrere Bibliotheken, um diese Eigenschaften zu nutzen: libvirt, LXC, systemd-spawn, runC. Auf eine tiefergehende Beschreibung der unterschiedlichen Bibliotheken wird an dieser Stelle verzichtet. Container sind im Cloud Computing eine weit verbreitete Technologie, da sie erhebliche Vorteile gegenüber der Trennung von Prozessen durch Virtualisierung (Hypervisor, VM) haben. Eine zentrale Rolle spielt hierbei die Nutzung des Host-Kernels, sodass die Portabilität von Containern (siehe Images) durch eine reduzierte Größe bedeutend höher ist. Gleichzeitig führt dies auch zu schnelleren Start- und Stoppzeiten der Container, was sich besonders durch kürzere Latenz der Skalierung bemerkbar macht.

2.4.1 Docker

Docker wurde im März 2013 durch die Firma dotCloud (seit Oktober 2013 Docker Inc.) unter der Apache 2.0 Lizenz veröffentlicht und ist ein Stack für Container. Docker besteht aus mehreren Komponenten, die zusammen ein einfach zu verwendendes Container-Ökosystem darstellen. Diese Einfachheit in der Verwendung ist der Grund für die große Verbreitung und Unterstützung in vielen Frameworks und Cloud Services. Daher wird es von vielen Quellen als Defacto-Standard betitelt [Burns et al., 2018].

Images

Images sind die Speicherabbilder, aus denen laufende Container erzeugt werden. Mit dem Open Container Image (OCI) wurde im Jahr 2017 ein Standardformat veröffentlicht. Dieser Standard hat bislang wenig Verbreitung und tatsächliche Relevanz gefunden, auch

hier zeigt Docker die größte Verbreitung. Docker Images basieren auf einem Overlay-Filesystem (z. B. AuFS oder BTRFS), in dem jeder Layer neue Dateien enthält, andere Dateien löscht oder modifiziert. Docker verwaltet für jeden Layer Metadaten wie einen eindeutigen Identifikator und optionale Tags. Die Layer werden auf dem Docker-Host in einer lokalen Registry (siehe ff.) persistiert. Dies ermöglicht eine Wiederverwendung von Layern durch Referenzierung aus verschiedenen Images heraus.

Registry

Die Registry stellt einen Speicherort von Layer-, Image- und Tag-Verwaltung dar. Sie kann nicht nur lokal auf dem Docker-Host, sondern auch als zentrale Remote-Registry betrieben werden, sodass eine breitere Wiederverwendung von Images gewährleistet werden kann. Jeder Docker-Host wird standardmäßig mit der Remote-Registry Docker Hub[1] konfiguriert. Auf Docker Hub sind ca. 2.500.000 Images zu finden, darunter eine Vielzahl von offiziellen Open-Source Projekten gepflegte Releases. Docker Hub wird von der Firma Docker Inc. als SaaS angeboten und wird durch ein Freemium Geschäftsmodell finanziert. Eigene Remote-Registries können ebenfalls bereitgestellt und in den Docker-Hosts konfiguriert werden. Viele Continuous Integration und Continuous Delivery Plattformen (z.B. GitLab, Jenkins oder Nexus) enthalten integrierte Registries. Darüber hinaus bieten CSP Registry-Services an (z. B. Microsoft Azure Container Registry oder Google Container Registry).

Dockerfiles

Die Erzeugung eines Images kann interaktiv aus dem jeweiligen Container heraus durchgeführt werden. Meist wird jedoch ein sogenanntes *Dockerfile* verwendet. Dieses enthält die notwendigen Befehle zum Erzeugen eines Images und lässt sich einfach über Source-Code-Managements Systeme versionieren und auf andere Systeme übertragen.

Quelltext 2.1: Beispiel eines Dockerfiles

```
1 # Baut auf dem Ubuntu Image auf
2 FROM ubuntu
3 # Kopiert den aktuellen Verzeichnisinhalt in den Container
4 COPY . /app
5 # Fuehrt make im Verzeichnis /app im Container aus
6 RUN make /app
7 # Startet python3 sobald Container gestartet wird
8 CMD python3 /app/app.py
```

[1] https://hub.docker.com/

Diese Datei wird meist "Dockerfile"benannt. Dadurch kann im selben Verzeichnis mit dem Command Line Interface (CLI) **docker** folgendes Kommando ausgeführt werden, welches die einzelnen Schritte ausführt.

```
1 docker run . -t beispiel-container
```

Sofern die Datei einen anderen Namen als Dockerfile trägt, benötigt das Kommando den zusätzlichen Parameter des Dateinamens.

```
1 docker run -f dockerfile-name -t beispiel-container
```

Das Ergebnis ist ein Images, welches den Namen beispiel-container trägt und in der lokalen Registry gespeichert wird.

Runtime

Docker verwendet die Runtime containerd, die Funktionalitäten zur Verwaltung von Images, Volumes und Netzwerken bietet. Zur Kommunikation mit dem Host-Kernel wird wiederum die OCI-kompatible Runtime runC verwendet. Diese Architektur wurde seit der Veröffentlichung von Docker inkrementell entwickelt, da zunächst eine eigene Runtime (libcontainer) verwendet wurde. Diese wurden zugunsten von größerer Kompatibilität und Schnittstellen für andere Container-Ökosysteme von einer monolithischen in diese komponentenbasierte Architektur überführt.

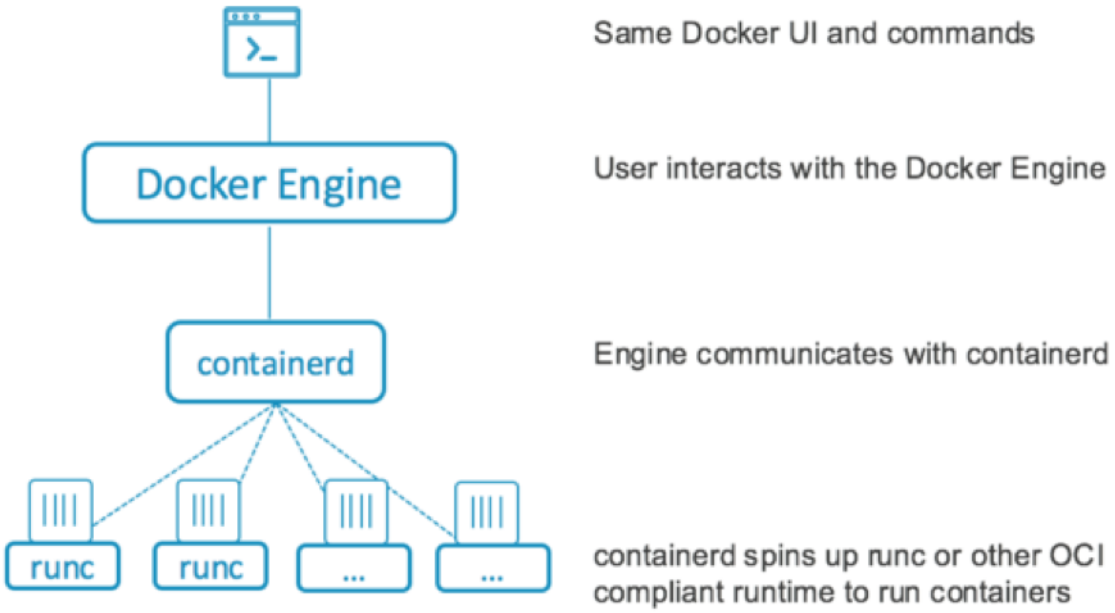

Abbildung 2.3: Beziehungen von Container Runtimes [Samipillai, 2017]

CLI

Eine Interaktion mit Docker erfolgt über das CLI **docker**. Dieses stellt Methoden zum Erstellen von Images (build), Verwalten von Images (pull, push, tag) und Verwalten von laufenden Containern (run, stop, ps, logs, exec) bereit. Der gesamte Funktionsumfang kann der offiziellen Dokumentation entnommen werden [Docker Inc., 2019].

2.4.2 Kubernetes

Kubernetes (griechisch für „Steuermann“) ist ein Framework zur Orchestrierung (Verwaltung) von Containern, um zuverlässig skalierbare und verteilte Systeme bereitstellen zu können. Ausgehend von den Erfahrungen die Google bei der Entwicklung des proprietären Frameworks Borg gesammelt hatte, wurde Kubernetes als Open-Source im Jahr 2014 unter der Lizenz *Apache-2.0* veröffentlicht. Mit Stand September 2019 hat das Kubernetes Projekt auf GitHub 2265 Beitragende mit 82922 Commits [Github, 2019]. Damit gehört es zu den aktivsten Open-Source Projekten auf GitHub. Seit 2015 engagieren sich neben Google weitere CSP, wie Amazon und Microsoft unter dem Dach der Cloud Native Computing Foundation (CNCF) in der Entwicklung. Die meisten Kubernetes Service Angebote in Public Clouds verwenden den nativen Upstream Open-Source Code. Kubernetes ist eine Abstraktion von zugrundeliegender Infrastruktur, Applikationen, Zugriffsrechten etc. Diese Objekte werden durch Kubernetes als Ressourcen über eine API zur Verfügung gestellt. Kubernetes selbst enthält keine Container-Runtime, sondern verwendet mit *CRI-O* ein Interface, welches auch durch andere Container-Runtimes implementiert werden kann, um durch Kubernetes verwendet werden zu können. Standardmäßig wird Kubernetes mit Docker verwendet, aber auch andere alternative Runtimes (z.B. *Podman*) werden unterstützt. Daher ist auch eine zentrale Container-Image-Registry nicht enthalten, stattdessen werden die Images auch Remote-Registries bezogen (siehe Docker Remote-Registries).

Architektur

Die Komplexität von Kubernetes wird teilweise als sehr groß betrachtet [Small et. al., 2019]. Um die Funktionsweise zu erklären, ist zunächst die Betrachtung einer generalisierten Architektur und Komponenten sinnvoll, die in der folgnenden Abbildung generalisiert dargestellt und anschließend beschrieben wird.

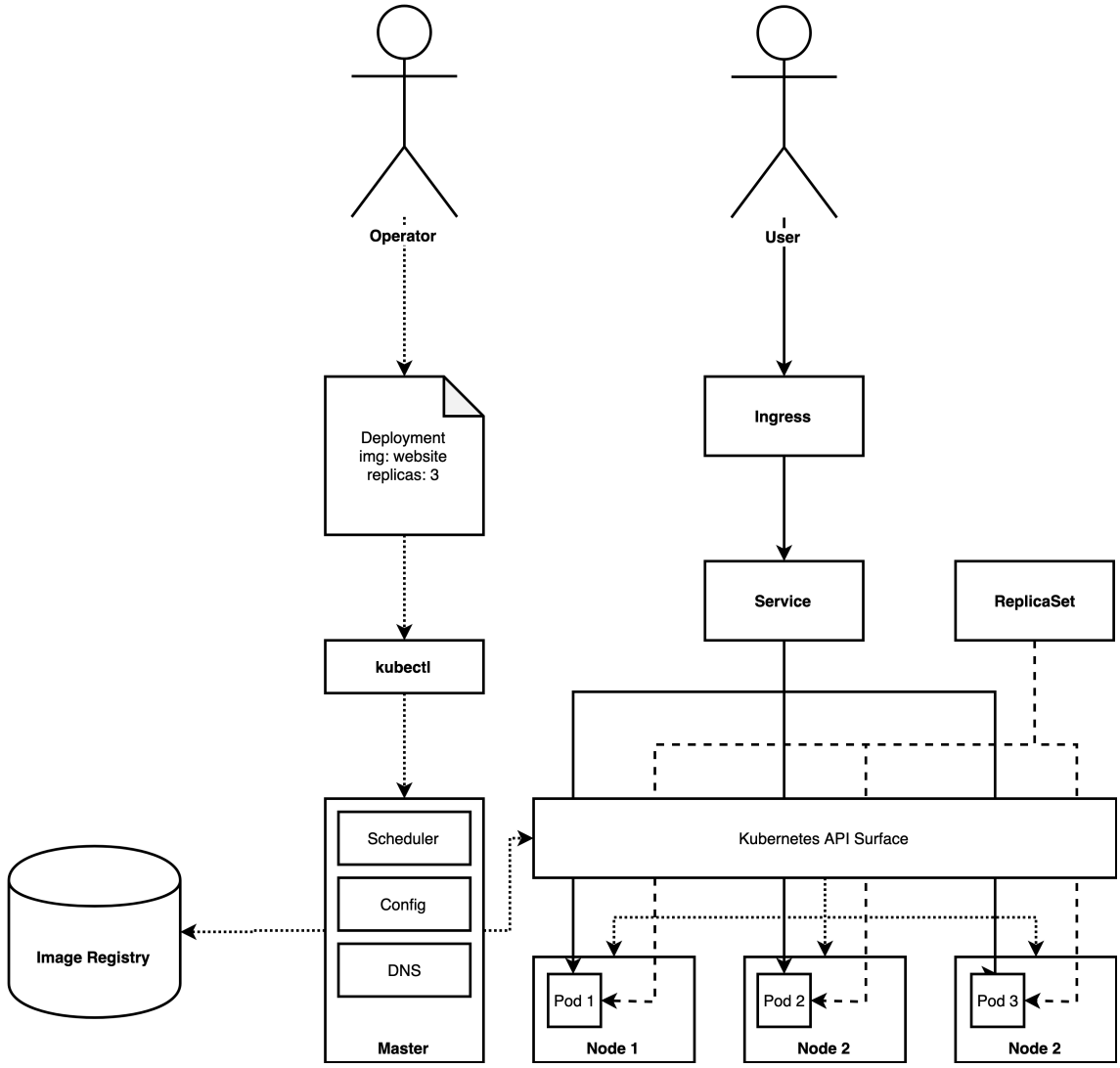

Abbildung 2.4: Generalisierte Darstellung der Kubernetes Architektur

Master

Ein Cluster besteht aus mehreren logischen Maschinen. Möglich sind sowohl VM als auch physische Maschinen (Bare Metal). Der Master übernimmt eine besondere Rolle und wird daher meist nicht direkt als vollwertiges Mitglied des Clusters betrachtet. Auf ihm werden der Scheduler und die Hauptinstanz eines verteilten Datenspeichers zur Konfigurationspersistenz (etcd) betrieben. Kubernetes arbeitet nach dem Prinzip *Infrastructure-as-Code* und verwendet den deklarativen Ansatz. Daher ist die Aufgabe des Masters, die Übereinstimmung von aktuellem Clusterzustand mit dem spezifizierten Zielzustand sicherzustellen. Mithilfe von dauerhaft laufenden Reconciliation-Schleifen wird eine mögliche Diskrepanz zwischen aktuellem Zustand und dem Zielzustand permanent beobachtet. Wenn notwendig, verwendet der Master die Container-Runtime-API

der Nodes um Anpassungen durchzuführen. In vielen KaaS auf Public Clouds werden Master außerhalb des Clusters in ausfallsicheren Infrastrukturen betrieben und auch nicht in den Ressourcenverbrauch des Clusters mit einberechnet [Microsoft, 2019].

Node

Ein Cluster besteht aus einem bis n Nodes. Nodes sind Maschinen, auf denen die Workloads in Form von Pods verteilt werden. Dazu werden die Ressourcen (Central Processing Unit (CPU), Random Access Memory (RAM), Graphics Processing Unit (GPU), Tensor Processing Unit (TPU)) des Nodes genutzt. Auf ihnen ist meist ein stark reduziertes Betriebssystem installiert, das meist lediglich eine Container-Runtime enthält. Alle Nodes müssen miteinander und mit dem Master über ein Netzwerk verbunden sein.

Pod

Ein Pod (Bedeutung aus der Biologie für Gruppe von Walen) ist die kleinste deploybare Einheit in Kubernetes. Er enthält einen oder mehrere Container, die die eigentlichen Workloads darstellen. Ein Pod wird immer vollständig auf demselben Node deployt, was ein Kriterium für die Bereitstellung der Container in einem statt zwei Pods ist. Sie teilen sich dieselbe Internet Protocol (IP)-Adresse, den Port-Space, den Hostnamen und können untereinander mittels Prozesskommunikation kommunizieren.

ReplicaSet

Um Redundanz, Skalierbarkeit und Sharding von Pods zu erreichen, bietet Kubernetes die ReplicaSet Ressource an. Diese verwalten eine bestimmte Gruppe von Pods, sind aber lediglich lose an diese gekoppelt. Die Definition, welche Pods sie verwalten geschieht über Metadaten (Labels, Annotations). So wird die Übernahme von bereits bestehenden Containern sowie Isolierung eines fehlerhaften Containers für Analysezwecke ermöglicht. Wichtigste Definition eines ReplicaSets ist die Anzahl der laufenden Pods.

Service

Eine Herausforderung bei der dynamischen Verteilung von Pods auf Nodes ist das Routing auf von den Pods bereitgestellten Services. Kubernetes bietet eine Vielzahl von Konzepten, um das sogenannte Service-Discovery ebenfalls zu abstrahieren und einfach verwendbar zu machen. Kernbestandteil ist dazu ein im Cluster integrierter Domain Name Service(Domain Name System (DNS)). Dieser löst DNS-Abfragen der Pods auf und kann so zum cluster-internen Routing genutzt werden. Gleichzeitig bietet Kubernetes die Service-Ressource an. Ein erzeugter Service vom Typ ClusterIP beinhaltet eine Zieldefinition durch die Angabe von Labels und Annotation und erhält zudem eine im Cluster

gültige IP Adresse. Zu dieser wird ein Eintrag im DNS eingefügt und ist somit für andere Pods auflösbar. Der DNS-Name folgt einem definierten Schema. Er setzt sich aus dem Service-Namen, dem Namespace sowie dem Domain-Namen des Clusters zusammen. Sofern der Zugriff innerhalb eines Namespaces erfolgt, ist auch nur die Angabe des Service-Namens ausreichend. Beispielsweise sieht der DNS-Name für einen Rest-API Service im Default-Namespace wie folgt aus: rest-api.default.svc.cluster.local . Um den Representational State Transfer (REST)-API Service von einem Frontend-Server beispielsweise zu erreichen, ist damit die Angabe des DNS-Namens ausreichend. Um einen Service auch außerhalb des Clusters erreichbar zu machen, kann ein Service vom Typ LoadBalancer angelegt werden. Dieser setzt eine weitere Integration in die Cloud-Infrastruktur voraus, sodass er mit, statt einer nur im Cluster gültigen IP, globalen IP Adresse versehen wird. Eine weitere Möglichkeit ist der Typ NodePort, der auf jedem Node ein Portforwarding zu diesem Service einrichtet und sich somit für Entwicklungstätigkeiten eignet.

Ingress

Ein Service vom Typ LoadBalancer wird meist durch einen Ingress oder auch Ingress-Controller erweitert. Dieser übernimmt die Bindung an erreichbare Uniform Resource Locator (URL) und die Terminierung von Secure Sockets Layer (SSL)/Transport Layer Security (TLS) Datenverkehr. Dazu verwendet er Zertifikate, die ihm als Secret zur Verfügung gestellt werden und leitet die Anfrage entsprechend an einen nachgelagerten Service weiter.

Weitere Konzepte

Es existieren weitere Komponenten in Kubernetes, die nicht in der generalisierten Architektur aufgeführt wurden. Diese sind für das Verständnis der generellen Funktionsweise von Kubernetes nicht erforderlich. Aufgrund ihrer Bedeutung für diese Arbeit werden sie dennoch im Folgenden erläutert.

Namespaces

Über Namespaces können Ressourcen in Kubernetes gruppiert werden. Als Standard enthält jedes Kubernetes Cluster die Namensräume default, kube-public und kube-system. Während kube-public und kube-system bereits Ressourcen für die Funktionsweise des Clusters enthalten, so ist der default Namespace zur Verwendung bestimmt. Standardmäßig interagiert kubectl mit dem default Namespace. Namespaces können mit Berechtigungen versehen werden, sodass der Zugriff auf Ressourcen eingeschränkt wird. Dies ist sinnvoll, um beispielsweise Entwicklungsumgebungen zu kapseln. Darüber hinaus können auch Cluster-Policies den Netzwerkverkehr zwischen Namensräumen einschränken.

ConfigMaps und Secrets

Besonders wichtig ist die Bereitstellung von Konfigurationen und Secrets. Dies können z. B. Zugangsdaten zu Datenbanken oder ähnlich sensible Daten sein. Nach allgemeinen Prinzipien der Wiederverwendbarkeit dürfen solche Informationen nicht Software-Artefakte und damit auch nicht in Container-Images abgelegt werden. Besser ist, diese erst zur Laufzeit des Prozesses zu injiziert, z. B. beim Programmstart durch das Auslesen einer Konfigurationsdatei oder Umgebungsvariable. Kubernetes unterstützt ConfigMaps und Secrets die Verwaltung dieser Informationen und bietet Mechanismen, um diese beim Container-Start zu injizieren.

Jobs

Durch Jobs können einmalig oder regelmäßig laufende Workloads ausgeführt werden, die keine permanent laufenden Prozesse, wie z.B. einen Webserver, benötigen. Diese werden durch Kubernetes im Vergleich zu Pods nach einem erfolgreichen Beenden (Exitcode 0) nicht erneut gestartet. Es gibt drei Typen: einmalig, parallel mit definiertem Ende und Workqueue. Jobs eignen sich z. B. für das Migrieren von Datenbanken.

DaemonSets

DaemonSets sind vergleichbar mit ReplicaSets. Auch sie verwalten eine Gruppe von Pods. Anders als ReplicaSets stellen sie jedoch sicher, dass auf jedem Node (oder einer Untermenge, siehe Labels und Annotations) genau eine Instanz eines Pods ausgeführt wird. Dies ist z. B. dann erforderlich, wenn Metriken über den Node gesammelt werden sollen oder zentrales Logging auf dem Node aggregiert werden soll.

Labels und Annotations

Jede Ressource in Kubernetes kann über Tags (Schlüssel-Wert-Paare) mit Metadaten versehen werden. Dies ist sinnvoll, weil so dynamische Selektionen von Ressourcen möglich sind, um z. B. Canary-Updates von Applikationen oder Deployment von Pods auf bestimmte Nodes durchzuführen.

Skalierbarkeit von Applikationen

Die vertikale Skalierung eines Pods kann durch die flexible Anpassung der von einem Pod nutzbaren Ressourcen (CPU, RAM) erzielt werden. Limitierend sind dennoch die maximal zur Verfügung stehenden Ressourcen der Nodes. Daher wird in erster Linie eine horizontale Skalierung von Applikationen verfolgt. Diese ist mithilfe von Kubernetes sehr einfach umzusetzen. Dazu sind zunächst ein ReplicaSet anzulegen und die entsprechenden Pods zu referenzieren. Anschließend muss lediglich das *replicas*-Attribut des ReplicaSets angepasst und diese Änderung dem Master mitgeteilt werden. Dieses Verfahren ist bei *Scaling-In* und *Scaling-Out* identisch. Beschränkt wird die maximal mögliche Skalierung von Applikationen durch die verfügbaren Ressourcen der Cluster-Nodes.

CLI

Um mit Kubernetes zu interagieren, verwenden Cluster-Operatoren das CLI **kubectl**. Dieses bildet die Schnittstelle zwischen Operator und dem Master. Über diese können alle Kubernetes Ressourcen verwaltet werden. Kubernetes arbeitet nach dem Prinzip *Infrastructure-as-Code* und verfolgt einen deklarativen Ansatz, nachdem durch den Operator lediglich der Zielzustand des Clusters definiert wird. Diese Definition erfolgt in einer DSL (meist als Yet Another Markup Language (YAML)-Dateien) die über den Programmaufruf an Kubernetes übermittelt wird.

```
1 kubectl apply -f beispiel-deployment.yaml
```

Skalierbarkeit des Clusters

Wie bereits erwähnt, kann die Skalierung des Clusters erforderlich sein. Auch hier wird horizontale Skalierung durch Veränderung der Anzahl der Cluster-Nodes eingesetzt. Bei *Scaling-Out* muss ein weiterer Node im Cluster-Netzwerk provisioniert und Kubernetes auf ihm installiert werden. Anschließend kann er dem Cluster hinzugefügt werden und der Master erkennt eine unterschiedliche Ressourcen-Auslastung der Nodes und sorgt für eine gleiche Verteilung der Workloads auf die Nodes. Dieses Verfahren bietet unter anderem auch den Vorteil, dass Betriebssystem- oder Kubernetes-Updates einfach durch sukzessives Austauschen der Nodes ermöglicht werden.

2.5 Geoinformatik

Mit der Geoinformatik hat sich in der Schnittmenge zwischen der Informatik, raumbezogenen Fachdisziplinen (Geographie, Geologie, etc.) und Geo-Informations-Technologien (Geodäsie, Photogrammetrie, etc.) die Geoinformatik als interdisziplinäre Fachdisziplin entwickelt. Eine geeignete Definition des Betrachtungsrahmens der Geoinformatik stammt von Norbert de Lange [de Lange, 2013]:

> Die Geoinformatik widmet sich der Entwicklung und Anwendung von Methoden und Konzepten der Informatik zur Lösung raumbezogener Fragestellungen unter besonderer Berücksichtigung des räumlichen Bezugs von Informationen. Die Geoinformatik beschäftigt sich mit der Erhebung oder Beschaffung, mit der Modellierung, mit der Aufbereitung und vor allem mit der Analyse sowie mit der Präsentation und Verbreitung von Geodaten.

2.5.1 Geoobjekte und Geoinformationen

Als Geoobjekt wird die digitale Repräsentation eines Objektes der realen Welt bezeichnet, welches typischerweise durch seine Geometrie, Topologie, Thematik und Dynamik geprägt ist. Die Bezeichnung *Geoobjekt* soll den Bezug zur Informatik und dem Konzept der Objektorientierung verdeutlichen [de Lange, 2013], wonach ein Objekt seine Attribute und Methoden hierarchisch weitergibt. Die grundlegenden Geometrien, wie Punkte, Linien, Flächen und Körper bilden Basisklassen, von denen jedes Geoobjekt - oder spezifischer: jede Objektklasse - erbt. Das Geoobjekt ist insofern die Instanz einer Geoobjektklasse. Von den Basisklassen erbt jedes Geoobjekt bereits Methoden, die für die Berechnung einer Reihe geometrischer Eigenschaften genutzt werden können (z. B. Berechnung des Flächeninhaltes eines flächenhaften Geoobjektes). Ebenso lassen sich Geoobjekte über ihre Topologie in Beziehung zueinander setzen, beispielsweise indem berechnet wird, ob zwei flächenhafte Geoobjekte eine Schnittfläche aufweisen und wie groß diese ist. Über Attribute wird die Thematik von Geoobjekten bestimmt. Diese weisen verschiedene Skalenniveaus auf (Nominal-, Ordinal-, Ratio- oder Intervallskala). Der Dynamik unterliegen Geoobjekte insofern, als das Veränderungen hinsichtlich Geometrie, Topologie und thematischer Attributwerte möglich ist.

2.5.2 Koordinatensysteme

Um Geoobjekte mithilfe ihrer Geometrie in einen räumlichen Kontext setzen zu können, ist die Angabe eines Koordinatensystems notwendig. Es existieren mehrere Arten von Koordinatensystemen (Kartesisches Koordinatensystem, Homogenes Koordinatensystem und Geographisches Koordinatensystem, siehe nachfolgende Abbildung).

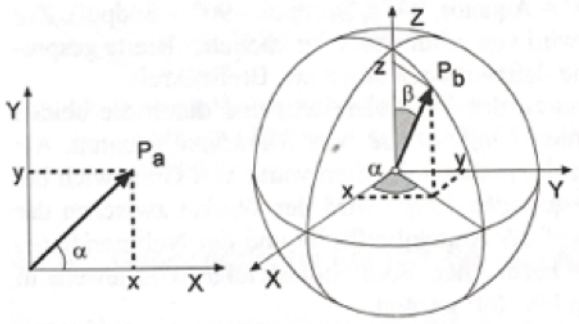

Abbildung 2.5: Kartesisches und Geographisches Koordinatensystem im Vergleich [de Lange, 2013]

Die größte Bedeutung für die Darstellung und Verarbeitung von Geoobjekten hat das kartesische Koordinatensystem, da die meisten Verfahren zur geometrischen Berechnung oder graphischen Datenverarbeitung kartesische Koordinatensysteme voraussetzen. Jedes Koordinatensystem enthält eine Metrik, sozusagen eine Abstandsfunktion,

die zwischen zwei Koordinaten gilt. In der Geoinformatik ist dies zumeist die Euklidische Metrik (Luftlinienentfernung). Da die Abbildung einer großen Fläche der Erde auf ein kartesisches Koordinatensystem mit euklidischer Metrik nicht möglich ist (Strecken-, Winkel oder Flächenverzerrung) gibt es eine Vielzahl von lokalen Koordinatensystemen, die auch lokale Anpassungen durch unterschiedliche Erdellipsoide berücksichtigen. Da Geoobjekte meist in unterschiedlichen Koordinatensystemen erhoben und verarbeitet werden, kann eine Koordinatentransformation von einem in das andere Koordinatensystem erforderlich sein. Dies erfordert, dass entsprechende Transformationsparameter für die Koordinatensysteme bekannt sind.

EPSG Codes

Um die Koordinaten von Geoobjekten nun mit einem Koordinatensystem zu kennzeichnen, werden meist die sogenannten European Petroleum Group Geodesy (EPSG)-Codes verwendet. Diese wurden seit 1986 durch die EPSG standardisiert und identifizieren jedes Koordinatensystem über einen 4- bis 5-stelligen Code. Es existieren eine Reihe von Programmier-Bibliotheken, in denen über den EPSG-Code die Eigenschaften und Parameter zur Transformation der Koordinatensysteme gefunden und verwendet werden können. Zu nennen ist insbesondere das Open-Source Projekt *PROJ*, das unter dem Dach der Open-Source Geospatial Foundation (OSGeo) geführt wird. Auf eine tiefere Darstellung des Themas der Koordinatensysteme wird an dieser Stelle verzichtet, da es für diese Arbeit nicht von substantieller Bedeutung ist.

2.5.3 Vektormodell

Eine Art der Speicherung von Geoobjekten ist das Vektormodell. In diesem Modell werden die Koordinaten der Geometrien dediziert angegeben und beziehen sich auf den Ursprung des referenzierten Koordinatensystems. Es werden Anfangs-, End- und Stützpunkte der Geometrien in Form von Punktlisten oder Referenzen auf Punkte gespeichert, sodass letztlich auch die Basisklassen Linie, Fläche und Körper durch eine Folge von Punkten repräsentiert werden. Die Speicherung im Vektormodell hat den Vorteil, dass die geometrische Genauigkeit der Geoobjekte lediglich durch den Datentyp zur Speicherung der Koordinaten beschränkt ist und daher sehr genau ist. Bekannte Vektordatenformate sind beispielsweise Geography JavaScript Object Notation (GeoJSON), ESRI Shape oder PostgreSQL/PostGIS Geometrien.

2.5.4 Rastermodell

Eine andere Art der Speicherung von Geoobjekten ist das Rastermodell. Dieses basiert auf einem regelmäßigen Raster, dessen Zellen Informationen über die Geoobjekte

enthalten und durch die Angabe von Zeile und Spalte identifizierbar sind. Auch für Rasterdaten ist die Angabe eines Koordinatensystems erforderlich. Zusätzlich enthalten die meisten Rasterdatenformate zusätzliche Informationen über Rotation, Translation und Skalierung des Rasters in Form von sogenannten „World-Files“ oder Metadaten-Tags innerhalb der Rasterdatei, über die die Lage des Rasters im Raum abgebildet wird. Viele Datenquellen, wie z. B. Scans oder Fernerkundungssatelliten, liefern ihre Daten per se im Rasterdatenmodell. Verbreitete Rasterdatenformate sind Geography Tagged Image File Format (GeoTIFF), Enhanced-Compress-Wavelets (ECW) oder PostgreSQL/PostGIS Raster.

2.5.5 Geoinformationssysteme

Die Verarbeitung von Geodaten erfolgt klassisch in desktopbasierten Geoinformationssysteme (GIS). GIS setzen wie andere Informationssysteme auf strukturelle Komponenten wie Hardware, Software, Daten und Anwender (sog. HSDA-Modell) und bieten funktionale Komponenten zur Erfassung, Verarbeitung, Analyse und Präsentation (sog. EVAP-Modell) von Geoobjekten (de Lange, 2013). Weit verbreitet sind Geoinformationssysteme als monolithische Desktop-Applikationen, in denen alle funktionalen Komponenten lokal auf dem Rechner des Anwenders vorhanden sind. Weit verbreitete GIS sind beispielsweise das proprietäre ArcGIS der Firma ESRI oder das quelloffene QGIS , dass als Open-Source Projekt von der OSGeo geführt wird. Analog zu anderen Informationssystemen ist jedoch auch hier der Trend zu verteilten Systemen zu erkennen, bei denen lediglich die Erfassungs- und Präsentationskomponenten lokal ausgeführt und Verarbeitungs- und Analysekomponenten zentral betrieben werden. Bei Geoinformationssystemen ist die sogenannte „Web-GIS“-Architektur stark vertreten.

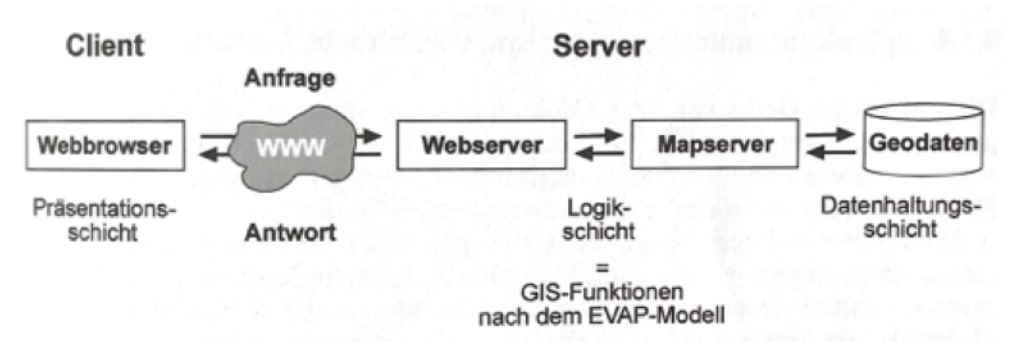

Abbildung 2.6: Schemahafte Darstellung der WebGIS Architektur [de Lange, 2013]

Gleichzeitig gibt es mit Web Map Service (WMS), Web Feature Service (WFS) oder Web Processing Service (WPS) standardisierte Protokolle für die Kommunikation zwischen Erfassungs-/Präsentationschicht und Verarbeitungs-/Analyseschicht. Diese werden durch das Open Geospatial Consortium (OGC) erarbeitet, in dem viele Hersteller

und Entwickler vertreten sind. So wird eine Interoperabilität einzelner Komponenten ermöglicht.

2.5.6 Programmierung

Die in der Geoinformatik genutzten Programmiersprachen sind maßgeblich durch die verwendeten Plattformen der Geoinformationssysteme beeinflusst. Im Desktop-GIS-Umfeld werden hier hauptsächlich Sprachen eingesetzt, mit denen GIS-Erweiterungen oder -automatisierungen entwickelt werden können. Im Web-GIS-Umfeld werden je nach Komponente meist Standard-Webtechnologien durch Erweiterungen für räumliche Daten ergänzt.

Python

Die wahrscheinlich meist eingesetzte Sprache ist Python. Sowohl QGIS als auch ArcGIS bieten API für Erweiterungen oder automatisierte Workflows an. Gleichzeitig zeigt Python im Web- und Data Science Umfeld eine hohe Verbreitung.

Java

Java ist im Web-Umfeld ebenfalls stark vertreten. Bekannte Frameworks in Java sind beispielsweise GeoTools und die Java Topology Suite (JTS) für räumliche Operationen. Mit der Software Geoserver steht eine Java Servlet-basierte Referenzimplementierung von OWS-Services (WMS, WFS, WPS) zur Verfügung. Das stark verbreitete ER-Mapping-Framework Hibernate bietet Interfaces für die Verwendung der räumlichen Datentypen der JTS und ist daher insbesondere für den Einsatz in Standard Java-Web Stacks mit Spring / Spring Boot geeignet.

JavaScript

Besonders im Web und Web-GIS ist JavaScript für die Präsentation von interaktiven Karten verbreitet. Weit verbreitete Kartenclients wie OpenLayers, Leaflet oder auch Google Maps sind in JavaScript implementiert. Für die clientseitige Verarbeitung von Geodaten im Browser steht mit turf.js auch ein JavaScript-Framework zur Verfügung.

C, C++

Die besondere Rolle spielt mit der Geodata Abstraction Library (GDAL) eine in C implementierte Bibliothek. GDAL stellt Treiber für eine Großzahl von Datenformaten, und Transformationsoperationen von Vektor- und Rasterdaten bereit. Aufgrund der Open-Source Lizensierung von GDAL wird GDAL von vielen proprietären und anderen quelloffenen Bibliotheken und Programmen verwendet. GDAL kann über Bindings von anderen Programmiersprachen verwendet werden (Python, Java, JavaScript).

SQL

Im Datenbankbereich werden Geodaten von einer Vielzahl von Systemen unterstützt. Weite Verbreitung zeigen hier Oracle (-Spatial) und PostgreSQL (-PostGIS), die jeweils über Erweiterungen spezielle räumliche Datentypen und Algorithmen integrieren. MongoDB als Vertreter von NoSQL-Datenbanken beinhaltet räumliche Funktionen sogar bereits in der Standardinstallation.

3 Anforderungen

3.1 Hintergrund

Eine Aufgabe des Kampfmittelbeseitigungsdienst Niedersachsen (KBD) ist die Auswertung historischer Kriegsluftbilder, die während und nach dem Zweiten Weltkrieg zwischen den Jahren 1939 und 1945 aufgenommen wurden. Diese wurden mit dem Ziel der Vor- und Nachbereitung von Angriffsflügen durch amerikanische und britische Aufklärungseinheiten erstellt. Nach der manuellen Auswertung wurden sie in Archiven der Alliierten verwaltet und werden nun durch den KBD zur Auswertung im Rahmen der Gefahrenabwehr bei Baumaßnahmen genutzt. Der Auswertevorgang wurde in einem Projekt Kampfmittelinformationssystem Niedersachsen (KISNi) des Landesamt für Geoinformation und Landesvermessung Niedersachsen (LGLN) im Jahr 2017 vollständig digitalisiert. Der KBD ist dem LGLN als organisatorisch untergeordnet. Der vorliegende Datenbestand von 127.167 Kriegsluftbildern wurde im Jahr 2015 mit einer geometrischen Auflösung von 1200dpi und einer radiometrischen Auflösung von 8-Bit gescannt und als Tagged Image File Format (TIFF)-Dateien mit etwa 12000 x 12000 Pixel gespeichert.

Abbildung 3.1: Scan eines Kriesluftbildes

Eine automatisierte Auswertung ist erst nach der Verarbeitung durch zwei geometrische Transformationen (Croppen und Georeferenzierung) möglich, wobei abgeleitete Datensätze entstehen. Hierbei ist weiterhin anzumerken, dass die Menge der dem KBD vorliegenden Kriegsluftbilder nicht vollständig ist. Durch kontinuierliche Archiv-Recherchen und Beschaffungen erweitert sich der Bestand um bis zu 2000 Datensätze jährlich.

3.1.1 Beschreibung des IST-Zustandes

Das derzeitige Speicherkonzept des KBD sieht eine standortübergreifende Datenhaltung vor: Im zentralen Rechenzentrum des Landesbetriebes Landesvermessung und Geobasisdaten erfolgt die Speicherung der Dateien auf einem gesicherten Fileserver. In einer dort betriebenen und gesicherten Datenbank werden Metadaten, Transformationsparameter sowie die Referenzen auf die Fileserver verwaltet. Am Standort des KBD wird ein weiterer Fileserver betrieben, welcher ausschließlich die abgeleiteten Datensätze speichert und diese im lokalen Netzwerk für die Kriegsluftbild-Auswerter netzwerk-topologisch optimal bereitstellt. Dieses Vorgehen wurde gewählt, um kurze Zugriffszeiten für Auswertungen bei gleichzeitig geringen Kosten für Datensicherungen zu erzielen. Mithilfe der gesicherten Originaldaten und Transformationsparametern kann die dezentrale Datenhaltung des KBD jederzeit neu erzeugt werden.

3.1.2 Verarbeitungsprozesse

Sowohl eine Ergänzung des Datenbestandes als auch ein Datenverlust des dezentralen Fileservers erfordern die Transformation einer 4 bis 6-stelligen Anzahl von Kriegsluftbildern. Damit stellen dies einen konkreten Anwendungsfall für den Einsatz einer skalierbaren Plattform für die Verarbeitung von Geodaten dar. Perspektivisch kommen weitere Einsatzmöglichkeiten durch die automatische Auswertung durch photogrammetrische Verfahren hinzu. Diese Einsatzmöglichkeiten werden im Folgenden dargelegt.

Transformation

Croppen

Nahezu alle gescannten Kriegsluftbilder enthalten neben dem eigentlichen Bildinhalt einen Randbereich. Dieser beinhaltet meist handschriftlich oder mit Schreibmaschinen erstellte Metadaten, wie Flugnummer oder Datum. Diese Informationen sind nicht automatisierbar auswertbar und aufgrund von vorliegenden Metadaten auch nicht von Bedeutung. Dazu kommt ein durch unterschiedlich große analoge Bildgrößen schwarzer Bereich, der beim Scanvorgang unbedeckt war. Beide Arten von Randbereichen müssen entfernt werden, um folgende Verarbeitungen und Analysen auf den photogrammetrisch auswertbaren Bereich zu begrenzen, die sonst zu Fehlern führen würden.

Abbildung 3.2: Vergleich des Kriegsluftbildes vor und nach dem Croppen

Dazu ist die Definition eines auszuschneidenden Bildbereichs über die Angabe von Pixelkoordinaten erforderlich. Durch das Croppen entsteht eine neue Rasterdatei. Dieser Datensatz wird für die 3-dimensionale manuelle Auswertung und als Eingabe für die Georeferenzierung verwendet.

Georeferenzierung

Erst die Georeferenzierung (geographische Verortung) der Kriegsluftbilder ermöglicht die Identifizierung relevanter Datensätze und die Verschneidung und Überlagerung mit weiteren geographischen Fachdaten im Geoinformationssystem.

Abbildung 3.3: Georeferenziertes Kriegsluftbild im Geoinformationssystem

Erforderlich ist dazu eine Menge von Passpunkten (Zuordnungen von Bild-Pixeln zu geographischen Koordinaten), die Wahl einer für die Passpunktverteilung und -menge geeigneten Transformationsmethode und die Wahl einer geeigneten Resamplingmethode. Durch die Georeferenzierung entstehen eine neue Rasterdatei, ein World-File und ein Footprint (geographische Fläche, die den Abbildungsbereich des entzerrten Kriegsluftbildes kennzeichnet). Zusätzlich wird ein World-File für die während des Croppens erzeugte Rasterdatei erzeugt, welches für die Orientierung bei 3D-Auswertung benutzt wird. Dieser Datensatz wird für die 2-dimensionale manuelle Auswertung im GIS verwendet. Später sollen diese Daten als Grundlage für vollautomatische photogrammetrische Verfahren verwendet werden.

Analyse

Gegenstand von zwei derzeit laufenden Forschungsprojekten des KBD mit dem Institut für Photogrammetrie und GeoInformation Universität Hannover (IPI) an der Leibniz Universität Hannover ist die Erkennung von Belastungssituationen auf Kriegsluftbildern mithilfe photogrammetrischer Verfahren. Diese Projekte sollen Ansätze und Algorithmen liefern, die eine manuelle Auswertung von Kriegsluftbildern durch menschliche Auswerter nicht mehr erforderlich machen. Motiviert wird dies durch lange Bearbeitungszeiten von Auswerte-Anträgen, die durch den Einsatz automatisierter oder sogar automatischer Verfahren stark gesenkt werden könnten und damit einen Beitrag zur Erfüllung von Anforderungen der Digitalisierung beim LGLN leisten würden.

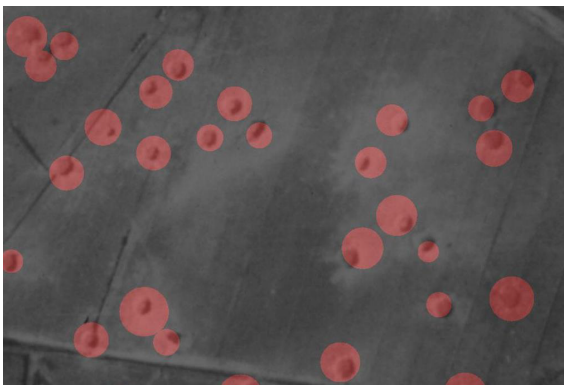

Abbildung 3.4: Referenzkrater für Analyseverfahren [Clermont, 2019]

Analyse mittels Markierter Punktprozesse

In dem ersten Forschungsprojekt wird die Erkennung von Belastungssituationen mithilfe des Verfahrens der Markierten Punktprozesse untersucht [Kruse, 2017]. Es wird die Annahme aufgestellt, dass sich Bombenkrater auf den Kriegsluftbildern lediglich in ihrer geometrischen Ausprägung unterscheiden und einen ähnlichen Gradientenverlauf aufweisen. Beim Start der Analyse werden als zufällige Ellipsen auf dem Kriegsluftbild

verteilt. Für jede Ellipse wird auf Basis ihrer Gradienten eine Energie berechnen. In vielen Interationen wird mithilfe eines stochastischen Ansatzes versucht, durch festgelegte Modifikationsoperationen auf den Ellipsen, eine globale Energiefunktion zu minimieren (*Reversible Jump Markov Chain Monte Carlo Sampling in Kombination mit Simulated Annealing*). Das Verfahren benötigt als Eingabeparameter ein Kriegsluftbild mit individuell empirisch bestimmten Parametern. Als Ergebnis wird eine Liste von Polygonen mit Metadaten bezüglich der Größe und Wahrscheinlichkeit zurückgegeben. Die Implementierung verwendet C++ und die Bibliothek OpenCV. Die Analyse eines einzelnen Kriegsluftbild benötigt zwischen 8 und 24 Stunden auf einer Workstation mit 1 CPU und 20 GigaByte Arbeitsspeicher.

Analyse mittels Convolutional Neuronal Networks

In dem zweiten Forschungsprojekt wird die Erkennung von Belastungssituationen mithilfe von Machine Learning untersucht [Clermont, 2019]. Anders als im ersten Projekt wird bei diesem Verfahren kein explizites Modellwissen über die Signatur von Bombenkratern auf Kriegsluftbildern verwendet, sondern ein abstraktes Modell durch manuell erfasste Trainingsdaten automatisch erlernt. Dazu wurden durch den KBD etwa 5000 Bombentrichter auf den Kriegsluftbildern markiert und diese zum Training eines Convolutional Neuronal Networks (CNN) eingesetzt. Die Detektion von weiteren Bombentrichtern umfasst folgende Schritte:

1. Kontrastanpassung des gesamten Kriegsluftbildes
2. Extraktion von Bombentrichter-Kandidaten mithilfe eines Blob-Detektors
3. Homogenisierung der geometrischen Auflösung des Kandidaten
4. Bewertung aller Kandidaten durch das CNN als Wahrscheinlichkeitsvektor
5. Rückgabe des Ergebnisses in Form einer Koordinatenliste

Das Verfahren benötigt als Eingabeparameter ein Kriegsluftbild und gibt als Ergebnis eine Liste von Koordinaten-Wahrscheinlichkeits-Tupeln der Bombentrichter-Kandidaten zurück. Die Implementierung wurde Python den Bibliotheken OpenCV und Tensorflow umgesetzt. Die Analyse eines einzelnen Kriegsluftbildes benötigt in etwa 30 Minuten auf einer Workstation mit NVIDIA GPU mit 8 GigaByte Grafikspeicher und CUDA Unterstützung.

Beide Verfahren bewerten ihre Qualität anhand von Korrektheit und Vollständigkeit der Detektion von Belastungssituationen und befassen sich daher maßgeblich mit der Optimierung hinsichtlich der Qualität. Außerhalb des vereinbarten Betrachtungshorizontes liegt eine tatsächliche Implementierung für den produktiven Betrieb. Bereits im derzeitigen Projektstand zeichnet sich jedoch ein erheblicher Ressourcenbedarf bei einem potentiellen Einsatz der Verfahren ab. Weil für eine Area of Interest (AOI) bis 1000 Kriegsluftbilder vorliegen, ist eine parallele Analyse der Rasterdaten erstrebenswert. Bereits im Prototypen-Status zeigen beide Projekte, dass unabhängig der letztendlichen Implementierung eine erhebliche Rechenkapazität notwendig sein wird. Daher soll der

plattformbasierte Ansatz die Integration dieser beiden Verfahren in seiner Umsetzung berücksichtigen und die dazu notwendige Rahmenbedingungen erörtert werden.

3.2 Anforderungen

Im Folgenden werden die Anforderungen beschrieben, welche durch den konkreten Anwendungsfall des KBD an die Plattform gestellt werden. Dazu werden zunächst unterschiedliche Personengruppen (Stakeholder) identifiziert, die Einfluss auf die Anforderung ausüben. Anschließend erfolgt eine Grobdarstellung der Ziele und eine Beschreibung möglicher Anwendungsszenarien durch die Nutzer. Aus dieser eher abstrakten Beschreibung ergeben sich schließlich konkrete funktionale Anforderungen, Qualitätsanforderungen und Rahmenbedingungen.

3.2.1 Stakeholder

Stakeholder sind Personen oder Gruppen, die Einfluss auf die Anforderungen an ein System haben. Diese lassen sich im konkreten Anwendungsfall in Anwender, Fachanwendungs-Entwickler, Prozess-Entwickler, Betrieb und Management gliedern.

Anwender

Die Anwender interagieren mit der Plattform indirekt über Fachanwendungen. Sie bekommen unmittelbar Ergebnisse der Verarbeitungsprozesse zu sehen. Daher sind sie im besonderen Maße durch die Verarbeitungsgeschwindigkeit der Plattform betroffen.

Fachanwendungs-Entwickler

Die Fachanwendungsentwickler entwickeln Fachanwendungen und stellen diese für Anwender bereit. Dazu verwenden sie die Schnittstellen der Plattform, um Datenhaltung und -prozessierung auf die Plattform auszulagern.

Prozessentwickler

Die Prozess-Entwickler entwickeln Algorithmen zur Verarbeitung von Daten auf der Plattform. In ihrem Fokus liegt der funktionale Aspekt ihrer Algorithmen. Die Algorithmen beziehen sich auf die Verarbeitung eines einzelnen Datensatzes.

Betrieb

Der Betrieb ist für die Verfügbarkeit der Plattform innerhalb der Organisationsinfrastruktur zuständig.

IT-Management Das IT-Management sichert durch organisatorische Maßnahmen, dass personelle und monetäre Ressourcen für den operativen Betrieb und Weiterentwicklung der Plattform zur Verfügung stehen. Besonders ist das Management an der Erfüllung

von Compliance-Anforderungen interessiert. Dazu werden in erster Linie Rahmenbedingungen durch Rahmenwerke wie *Leitsätze für Fachanwendungen* und *Leitsätze für Fachdatenhaltungen* des LGLN eingebracht.

3.2.2 Ziele

Neben den fachlich technischen Aspekten sollen mit der Plattform vor allem architektonisch-strukturelle Ziele erreicht werden. Diese sollen die Anpassbarkeit an weitere Verfahren sicherstellen und dazu führen, dass Informationen aus den historischen Kriegsluftbildern in kurzer Zeit gewonnen werden können. Erklärtes Hauptziel ist die Verkürzung der Verarbeitungsprozesse, da dies zu einer schnellen Gefährdungsbeurteilung führt. Durch die Umsetzung der Plattform sollen Folgendes erreicht werden.

1 Die Verarbeitung (Transformation und Analyse) von Rasterdaten soll mit einem hohen Durchsatz erfolgen, um die Ergebnisse innerhalb kürzester Zeit nutzbar zu machen. Die verfügbaren Rechnerkapazitäten sollen dynamisch anpassbar sein, um einerseits Kosten zu sparen und andererseits kurze Verarbeitungszeiten zu ermöglichen.

2 Die zur Verarbeitung verwendeten Algorithmen sollen in verbreiteten Programmiersprachen und Frameworks aus der Geoinformatik und dem Data Science Umfeld implementiert werden können. Dabei soll sich die Programmlogik lediglich auf die Transformation beziehungsweise Analyse eines einzelnen Datensatzes beziehen.

3 Eingabe- und Ausgabedaten sollen in einer geeigneten performanten Datenhaltung für Verwendung in internen Verarbeitungsprozessen der Plattform und die Einbindung in weitere Systeme (z.B. GIS) gespeichert werden.

3.2.3 Szenarien

Wie bereits ersichtlich ist, steht die Minderheit der Stakeholder mit der Plattform in direktem Kontakt. Daher werden zunächst Szenarien beschrieben, in denen eine direkte Nutzung der Plattform vorkommt.

1. Nach der Beschaffung einer großen Anzahl weiterer Kriegsluftbilder sollen diese für die Auswertung aufbereitet werden. Dazu soll ein Anwender über eine externe Fachanwendung die neuen Datensätze auf der Plattform bereitstellen und die entsprechenden Transformations-Parameter erfassen. Anschließend sollen für jeden Datensatz die Transformations-Prozesse ausgeführt werden. Jeder Ergebnisdatensatz soll auf der Plattform gespeichert werden. Die gespeicherten Ergebnisdaten sollen in einem Geoinformationssystem dargestellt werden können.

2. Mithilfe photogrammetrischer Analyseverfahren sollen auf allen Kriegsluftbildern Belastungen detektiert werden. Dazu soll ein Prozess-Entwickler die Plattform um einen Analyse-Prozess erweitern können. Die Implementierung des Analyse-Prozesses soll in einer beliebigen Programmiersprache möglich sein. Eine Anpassung oder Aktualisierung der Prozesse muss in kurzer Zeit möglich sein, um kurzfristig neue Methoden und Verfahren bereitstellen zu können.
3. Durch den Betrieb soll ein Monitoring der Plattform erfolgen, um die Auslastung der verwendeten Rechenkapazität feststellen zu können. Bei hohen Workloads soll die verwendete Rechenkapazität durch den Betrieb erhöht werden können. Bei geringen Workloads soll die verwendete Rechenkapazität durch den Betrieb verkleinert werden können, um Betriebskosten zu sparen.

Deutlich wird, dass diese Szenarien in erster Linie funktionale Aspekte der Plattform beschreiben. Beispielsweise gibt es kein Szenario, in dem das Management als Akteur auftritt.

3.2.4 Anforderungsverzeichnis

Aus dem aufgeführten Anwendungsbeispiel lassen sich nun Anforderungen unter der Berücksichtigung der Interessen von Stakeholdern und Szenarien definieren. Diese sind nach funktionalen Anforderungen, Qualitätsanforderungen und Rahmenbedingungen gegliedert und zur Verfolgbarkeit mit einem Schlüssel versehen. Wichtig ist bei der Formulierung von funktionalen Anforderungen und Qualitätsanforderungen, dass diese abstrakt gehalten werden, um nicht bereits vor der Anforderungsanalyse und dem Entwurf den Lösungsraum einzugrenzen. Sofern Begrenzungen notwendig sind (z.B. Compliance) werden diese als Rahmenbedingungen aufgeführt.

1. **Funktionale Anforderungen**

   a) **Management von Verarbeitungsprozessen**
   Auf der Plattform muss eine beliebige Anzahl von Verarbeitungsprozessen bereitgestellt werden können. Bereitgestellte Prozesse müssen aktualisiert oder gelöscht werden können. Die Implementierung von Verarbeitungsprozessen soll in beliebigen Programmiersprachen möglich sein. Mindestens sind Python, Java, JavaScript und C/C++ zu unterstützen. Als Eingabewert muss ein einzelner Datensatz entgegengenommen und als Ausgabewert ein neuer Rasterdatensatz oder Vektordatensatz zurückgegeben werden können.

   b) **Ausführung von Verarbeitungsprozessen**
   Die Ausführung von Verarbeitungsprozessen muss über den Aufruf einer Schnittstelle möglich sein. Verarbeitungsprozesse mit geringer Ausführungszeit sollen das Ergebnis direkt zurückgeben können, welche mit langer Ausführungszeit sollen das Ergebnis asynchron zurückgeben können.

c) **Beispielhafte Implementierung von Verarbeitungsprozessen**
Um die Umsetzbarkeit zu demonstrieren, müssen die zwei vorhandenen Transformationsalgorithmen (Croppen und Rektifizierung) implementiert werden. Die Integration weiterer, aus den KBD-IPI-Forschungsprojekten hervorgehenden, Verarbeitungsprozesse (Analyse) muss berücksichtigt und konzeptionell beschrieben werden.

d) **Datenhaltung von Eingabe- und Ausgabedaten**
Eingabe- und Ausgangsdaten müssen über eine Schnittstelle auf der Plattform gespeichert, abgerufen und gelöscht werden können. Aus der dieser Datenhaltung heraus müssen sie in Geoinformationssystemen als Datenquelle eingebunden und performant abgerufen werden können.

e) **Monitoring von Verarbeitungsprozessen und Datenhaltung**
Die Verarbeitungsprozesse müssen durch den Betrieb überwacht werden können, damit eine Skalierung der Rechenkapazität durchgeführt werden kann.

2. **Qualitätsanforderungen**

a) **Laufzeit von Verarbeitungsprozessen**
Die Laufzeit der Prozesse muss so gering wie möglich sein. Für die vorhandenen Transformationsprozesse (Croppen und Rektifizierung) darf diese maximal 10 s dauern. Auch bei großen Lastsituationen soll die Plattform einen hohen Durchsatz aufweisen. Dazu sind die vorhandenen Infrastruktur-Ressourcen vollständig auszulasten. Während der Skalierung von Verarbeitungsprozessen darf es zu keinem Ausfall der Plattform kommen.

b) **Skalierung der Rechenkapazität**
Die vorhandene Rechenkapazität der Plattform muss dynamisch und innerhalb kurzer Zeit veränderbar sein. Während der Veränderung der Rechenkapazität darf es zu keinem Ausfall der Plattform kommen.

3. **Rahmenbedingungen**

a) **Konformität mit Leitsätzen für Fachanwendungen des LGLN**
Das Dokument „Leitsätze für Fachanwendungen“ des LGLN enthält generelle Grundsätze aus Sicht einer internen Expertengruppe für Softwareentwicklung. Da diese Leitsätze sowohl einerseits sehr umfangreich als auch nicht umfassend für diese Arbeit sinnvoll sind, werden nur folgende Unteranforderungen exemplarisch betrachtet:

i. **Sourcing-Strategie**
Die Plattform soll möglichst durch quelloffene Open Source Software umgesetzt werden. Dabei sollen als Kriterien bei der Auswahl von Technologien insbesondere das Vorhandensein einer großen und offenen Entwicklergemeinschaft, sowie gegebenenfalls vorhandene kommerzielle Support-Angebote berücksichtigt werden.

ii. **Cloudfähigkeit**
Die Plattform soll auch in Public Clouds und damit außerhalb der Infrastruktur des LGLN betrieben werden können. Eine Nutzung der Plattform durch weitere Cloud-Services (lesend/schreibend) soll möglich sein.

b) **Konformität mit Leitsätzen für Datenhaltungen des LGLN**
Das Dokument „Leitsätze für Fachdatenhaltungen“ des LGLN definiert generelle Anforderungen an Datenhaltung aus Sicht einer internen Expertengruppe für Datenhaltungen. Auch hier gilt, dass diese Leitsätze umfangreich und für diese Arbeit teilweise von geringer Relevanz sind. Daher wird auch hier nur eine Unteranforderungen exemplarisch betrachtet:

i. **Wohldokumentierte Datenstrukturen**
Die Eingabe- und Ausgabedaten müssen in einem gut dokumentierten Format in der Datenhaltung gespeichert werden.

c) **Schutzbedarf der Kriegsluftbilder**
Es muss sichergestellt sein, dass die drei Schutzziele Integrität, Verfügbarkeit und Vertraulichkeit für die Datenhaltung und insbesondere für den Datenbestand der Kriegsluftbilder jederzeit gewährleistet ist.

4 Analyse

Dieses Kapitel widmet sich der Analyse von Anforderungen und ihrer Umsetzbarkeit. Dazu werden zunächst geeignete Bewertungskriterien mithilfe allgemein anerkannter und verbreiteter Qualitätsmerkmale festgelegt. Aus einer Systemanalyse ergeben sich sinnvolle Konzepte, die konkreten Herausforderungen begegnen. Aus ihnen wird unter Zuhilfenahme der Bewertungskriterien eine Plattform-Architektur abgeleitet. Für diese werden abschließend mit Docker und Kubernetes geeignete Technologien identifiziert.

4.1 Bewertungskriterien

Allgemein anerkannte Bewertungskriterien sind in den Normen für Software-Qualität ISO 9126 [ISO/IEC, 2001] und ISO-25010 [ISO/IEC, 2010] festgelegt. Die zum Zeitpunkt der Fertigstellung der Arbeit gültigen Fassung beinhaltet insgesamt acht Kategorien. Aufgrund ihres allgemeinen Charakters sind sie unzureichend konkret für die Auswahl von Architektur und Technologie. Daher werden im Folgenden die Kategorien jeweils benannt, kurz erläutert und die für die vorliegende Arbeit abgeleiteten Kriterien definiert. Abgeleitete Kriterien werden in *kursiver* Schreibweise dargestellt.

1. **Funktionalität**
   Diese Kategorie enthält Kriterien, die die grundsätzliche Eignung eines Systems zur Erreichung einer spezifizierten Funktionalität beschreiben. Dazu zählen einerseits die grundsätzliche Korrektheit, andererseits auch die Vollständigkeit der Funktionalität. Ein weiteres Kriterium ist der erforderliche Aufwand, der dazu betrieben werden muss.

   Darunter fällt, ob z. B. die verwendeten Algorithmen zur Verarbeitung der Geodaten nachweislich (d.h. formal) korrekt sind und wie schnell diese implementiert werden können. Dies kann durch die Verwendung von anerkannten überprüften Frameworks erreicht werden. Für unterschiedliche Domänen haben sich spezialisierte Programmiersprachen und Frameworks etabliert. Z. B. zeigt Python im Umfeld von Machine Learning eine hohe Verbreitung, für Statistik wird hingegen oft R eingesetzt. Damit ist die Möglichkeit zur polygloten Implementierung schon alleine aus dem Grund erforderlich, um die verbreiteten Frameworks einsetzen zu können. Daher stellt die *polyglote Implementierung* das maßgebende Kriterium der Funktionalität dar.

2. **Effizienz**
   In dieser Kategorie werden maßgebliche Kriterien hinsichtlich des Verbrauchs von Infrastruktur-Ressourcen (CPU, RAM, GPU, etc.) eines Systems während der Ausführung untersucht. Der Ressourcenverbrauch unterliegt einer zeitlichen Varianz z. B. durch auftretende Lastspitzen. Daher ist sowohl der *Ressourcenverbrauch in Ruhe* und der *Ressourcenverbrauch unter Last* zu betrachten. Durch Skalierbarkeit einer Applikation kann ihr Durchsatz erhöht werden. Unter Umständen ist die horizontale Skalierbarkeit einer Applikation grundsetzlich aufgrund ihrer inneren Architektur nicht möglich. Daher sind vor allem der *maximale Durchsatz* und *Schnelligkeit der Skalierung* maßgebende Kriterien.

3. **Kompatibilität**
   In dieser Kategorie werden Kriterien der Kompatibilität eines Systems betrachtet. Diese lassen sich grob in die Kompatibilität zu anderen Systemen in derselben Umgebung (Koexistenz) und Kompatibilität zu anderen Programmen bei Datenaustausch (Interoperabilität) trennen.

   Koexistenz bedeutet in diesem Kontext, inwieweit ein paralleler Betrieb von mehreren Systemen in einer Umgebung ohne unerwünschte Nebeneffekte zuverlässig erfolgen kann. Ein Negativ-Beispiel dafür sind installierte Programme, die verschiedene nicht-kompatible Versionen von Shared Libraries erfordern. Aus Perspektive des Software-Engineering ist es daher generell sinnvoll, Systeme durch geeignete Mechanismen zu kapseln, z. B. mithilfe von Virtualisierung oder Containern. Durch die geforderte polyglote Implementierung ist dieses Kriterium von wesentlicher Bedeutung, zumal der gewählte Isolations-Mechanismus auch erhebliche Auswirkungen auf die Effizienz eines Systems hat.

   Interoperabilität beschreibt die Kompatibilität zu anderen Systemen auf Ebene von Kommunikation und Datenaustausch. Dies kann anhand der Unterstützung für offene Standards und freie Formate beurteilt werden. Proprietäre Interfaces können die Kommunikation zu anderen Systemen erschweren oder sogar vollständig verhindern. Als Beispiel dient das Open Systems Interconnection (OSI)-Schichtenmodell [Day, 1995]. Dort werden konzeptionelle Schichten vorgestellt, für die verschiedene Protokolle existieren. Je höher die Schicht eines Protokolls, desto größer ist die Interoperabilität einer unterstützenden Software. Insofern ist Interoperabilität mit GIS über entsprechend unterstützte Protokolle und Formate (siehe Kapitel 2.5.5) ein wesentliches Kriterium.

4. **Gebrauchstauglichkeit**
   Kriterien in der Kategorie der Gebrauchstauglichkeit weisen eine hohe Schnittmenge zu Aspekten der Software-Ergonomie auf. Im Fokus der Betrachtung steht dabei die Interaktion eines Menschen mit dem System. Dabei wird die Interaktion über den gesamten Lebenszyklus betrachtet.

   Dies beginnt mit der Erlernbarkeit eines Systems. Durch geeigenete Lernangebote wie z. B. interaktive Tutorials, Blogbeiträge oder Demos werden Einstiegshürden

gesenkt. Ob ein System einen überzeugenden Eindruck beim Nutzer hinterlässt, wird maßgeblich durch seine Bedienbarkeit und Ästhetik des User-Interfaces bestimmt und liegt möglicherweise in der subjektiven Wahrnehmung eines individuellen Nutzers. Aus betrieblicher Perspektive ist zudem der Betrieb eines Systems unter diesen Aspekten zu betrachten.

Da die Gebrauchstauglichkeit sehr subjektiv und damit schwer definierbar ist, scheinen quantitative Vergleiche von Leistungskennzahlen sinnvoll. Als Indikatoren werden im Kontext freier Software oft Leistungskennzahlen wie *Commits*, *Contributoren* oder *Stars* der Projektseiten auf Kollaborations-Plattformen verwendet. Eine verbreitete Kollaborations-Plattform ist Github. Die folgende Abbildung Abbildung 4.1 zeigt, dass diese Leistungskennzahlen prominent und einfach zugänglich bereits auf dern Hauptseiten der Projekte dargestellt werden.

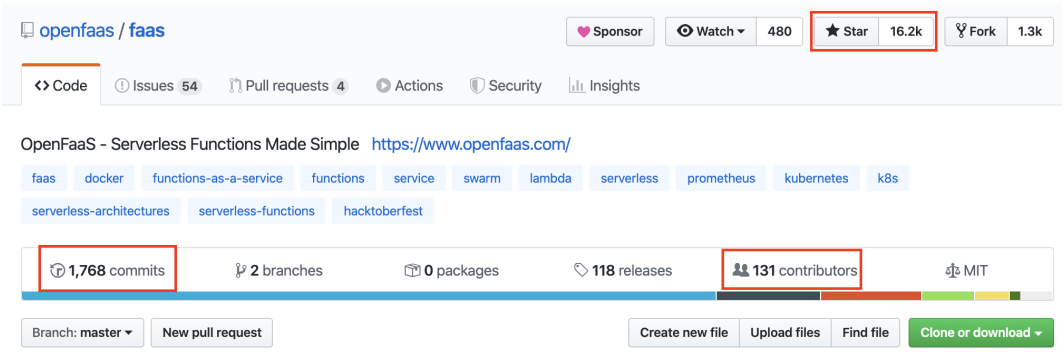

Abbildung 4.1: Leistungskennzahlen von GitHub-Projekten am Beispiel OpenFaaS

5. **Zuverlässigkeit**
   In dieser Kategorie werden Kriterien aufgeführt, mithilfe derer die Zuverlässigkeit und Robustheit eines Systems bewertet werden kann. Dazu gehört das Verhalten in Fehlersitutationen. Wenn etwa beabsichtigte oder unbeabsichtigte fehlerhafte Eingaben verarbeitet werden, kann das System in einen fehlerhaften Zustand überführt werden. Aber auch wenn das System durch viele Anfragen stark belastet wird, können Fehler auftreten. Unter Umständen werden zusätzliche Maßnahmen notwendig, wie z. B. manuelle Eingriffe oder spezielle Monitoring-Maßnahmen. Horizontale Skalierbarkeit ermöglicht demnach nicht nur einen höheren Durchsatz sondern erhöht gleichmaßen die Ausfallsicherheit. Bei schwerwiegenden Fehlern kann auch ein Neustart der Anwendung erforderlich sein. Eine Vielzahl von Systemen setzt eine korrekte Startreihenfolge von Komponenten voraus. Insofern sind maßgebende Kriterien *Self-Healing*, *Horizontale Skalierbarkeit* und *Wiederherstellbarkeit*.

6. **IT-Sicherheit**
   In dieser Kategorie sind Kriterien der IT-Sicherheit aufgeführt. Im Fokus der Betrachtung stehen die fünf Schutzziele Vertraulichkeit, Integrität, Authentizität

Verfügbarkeit und Verantwortbarkeit, auch bekannt unter dem englischsprachigen Akronym CIA[3]. Generell kann bereits während des Entwurfs durch einen Security-First-Ansatz ein erheblicher Zuwachs von IT-Sicherheit erzielt werden. Eine vollständige Betrachtung dieses Aspektes ist im Rahmen der vorliegenden Arbeit nicht möglich. Stattdessen kann die Grundlage für eine weitere Betrachtung gelegt werden. Die weitere Betrachtung kann die Implementierung mithilfe die STRIDE-Methode [Microsoft, 2009] analysieren. Aus dieser ließen sich geeignete Maßnahmen zur Erreichung eines hohen Sicherheitsniveaus ableiten. Die Voraussetzung für diese Analyse bildet das Prinzip der geringsmöglichen Rechte. Das Prinzip besagt, dass erst durch Teilung eines Systems die Ausführung einzelner Komponenten mit geringstmöglichen Rechten erlaubt wird. Demnach ist ein Kriterium die *Dekomposition* eines Systems in untergeordnete Subsysteme. Für grundlegende Sicherheit ist weiterhin erforderlich, dass Systeme nur über *sicherbare Interfaces* Zugriff auf ihre Assets[1] gewährleisten.

7. **Wartbarkeit**
   In dieser Kategorie sind Kriterien der Wartbarkeit aufgeführt. Die Wartbarkeit wird maßgeblich durch Modularisierung bestimmt, da notwendige Wartungsarbeiten auf einen reduzierten Teilaspekt des Systems erfolgen können. Durch die *Wiederverwendbarkeit* eines Modules in anderen Systemen ergeben sich Synergieeffekte. Ein Modul kann gemeinsam entwickelt und verbessert werden. Damit ermöglicht Modularität auch die *Analysierbarkeit* und *Testbarkeit*.

   Bei Wartungsarbeiten an einem Modul kann es aber gleichzeitig zu einer Beeinträchtigung des Systems kommen. Moderne Systeme verwenden daher Rolling-Update Strategien, bei denen erst das neue Modul produktiv geschaltet, dann der Datenfluss umgelenkt und zuletzt das alte Modul deaktiviert wird. Dabei darf es zu keinen Fehlersituationen kommen. Daher ist *Veränderbarkeit zur Laufzeit* ein gutes Kriterium für Wartbarkeit.

8. **Portabilität**
   Eine im Kontext dieser Arbeit sehr wichtige Kategorie ist die Portabilität. Sie bewertet, ob ein System in unterschiedlichen Betriebsumgebungen betrieben werden kann. Dies kommt dann zum Tragen, wenn eine Migration zwischen Public Clouds durchgeführt werden soll. Hier kann es zu einem Vendor-Lockin kommen, der durch die Verwendung von proprietäten Services spezifischer Public Clouds eine Migration verhindert. Dies lässt sich mit konsequenter Nutzung von Open Source Software (OSS) vermeiden. Damit ist *OSS* ein maßgebendes Kriterium.

   OSS erlaubt außerdem, dass ein System auch sehr einfach ohne komplexes Lizenz-Management in lokalen Entwicklungsumgebungen aufgesetzt werden kann. Dies setzt natürlich voraus, dass die üblicherweise sehr begrenzten Infrastruktur-Ressourcen

[1] schützenswerte und schützbare Ressourcen, z. B. Geodaten wie Kriegsluftbilder

lokaler Entwicklungsumgebungen für das System ausreichen. Das Kriterium der Installation für die *lokale Entwicklung* weist demnach eine Schnittmenge zur Effizienz auf.

Aus ähnlichen Motivationen heraus basieren zahlreiche Cloud Services ebenfalls auf OSS. Pay-Per-Instance, Pay-Per-Core, Pay-Per-User oder andere Lizenzmodelle erschweren den Betrieb von Public Clouds für CSP. Ein Beispiel für einen auf OSS basierenden Cloud Service ist Kubernetes. Das ermöglicht die Migration aus einem selbst verwalteten Kubernetes-Cluster in ein KaaS. Aber auch einzelne Komponenten, wie z.B. eine selbst verwaltete Datenbank, kann durch einen verwalteten Datenbank-Service ersetzt werden. Werden für eine Komponente verwaltete Cloud Services angeboten, so spricht dies für einen hohen Professionalisierungsgrad der Software und hohe Marktdurchdringung. Damit ist die *Ersetzbarkeit durch Cloud Services* einer Komponente ein weiteres Kriterium der Portabilität.

4.1.1 Liste der Bewertungskriterien

Die Ergebnisse dieser Aus- und Bewertung werden in einer Liste mit jeweiligen Kategorie zusammengefasst. Die derart erstellte Liste bildet die Grundlage für die weitere Auswahl der Architektur und Technologie. Ebenfalls wird diese Liste für den späteren Vergleich mit bestehenden Systemen in der Domäne der Geodatenverarbeitung verwendet.

Tabelle 4.1: Zusammengefasste Liste der Bewertungskriterien

Kategorie	**Kriterium**
Funktionalität	Polyglote Implementierung
Effizienz	Ressourcenverbrauch in Ruhe Ressourcenverbrauch unter Last Geschwindigkeit der Skalierung Maximaler Durchsatz
Kompatibilität	Koexistenz Interoperabilität
Gebrauchstauglichkeit	Sourcecode Commits Sourcecode Contributors Stars
Zuverlässigkeit	Self-Healing Horizontale Skalierbarkeit Wiederherstellbarkeit
IT-Sicherheit	Dekomposition Sicherbare Interfaces
Wartbarkeit	Wiederverwendbarkeit Veränderbarkeit zur Laufzeit Analysierbarkeit Testbarkeit
Portabilität	Open-Source Ersetzbarkeit durch Cloud Services Lokale Entwicklung

4.2 Systemanalyse

Um den Betrachtungshorizont der vorliegenden Arbeit gegenüber weiteren Systemen im Kontext abzugrenzen, wird eine Systemanalyse vorgenommen. Diese gliedert ein System in drei Mengen:

1. Das **System** beschreibt die Menge der zu erstellenden Komponenten, gegen die sich die formulierten Anforderungen richten. Diese befinden sich innerhalb der Systemgrenze. Das System wird im folgenden Abschitt 4.2.2 weiter spezifiziert.
2. Der **Systemkontext** umfasst meist andere Systeme, die in einer direkten Beziehung zu dem zu erstellenden System. Diese müssen daher bei der Umsetzung berücksichtigt werden und werden im folgenden Abschitt 4.2.1 näher erläutert.
3. Die **Systemumgebung** umfasst theoretisch sämtliche weitere Systeme, die in keiner Relation zum zu erstellenden System stehen. Aufgrund dieser Tatsache sind sie für das System von keiner Relevanz und brauchen daher nicht näher erläutert zu werden.

Sowohl Anforderungen (Kapitel 3.2.3) als auch Szenarien (Kapitel 3.2.4) nennen vielfältige Komponenten. Dazu zählen Datenhaltung, Verarbeitung, Monitoring, Geoinformationssysteme und Fachanwendungen. Die Anforderungen beschreiben zudem unterschiedliche Datenflüsse und Kommunikationsbeziehungen, die diese Komponenten miteinander verbinden. Dabei ist ersichtlich, dass sich die Anforderungen nur gegen Datenhaltung, Verarbeitung und Monitoring richten und nicht gegen die Fachanwendungen und Geoinformationssysteme. Dementsprechend umfasst die Plattform ausschließlich Datenhaltung, Verarbeitung und Monitoring. Damit sind Fachanwendungen und Geoinformationssysteme außerhalb des Systems, aber innerhalb des Systemkontextes zu verorten. Die folgende Abbildung 4.2 stellt die erläuterte Trennung dar.

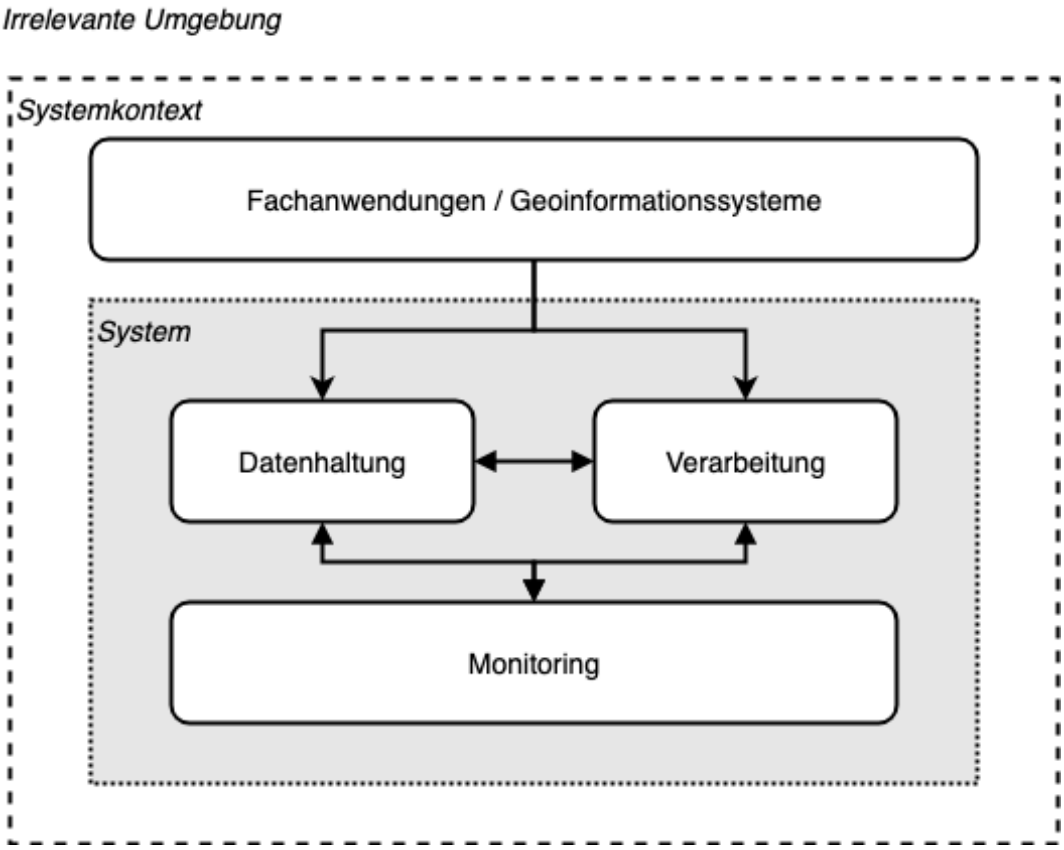

Abbildung 4.2: Kontextdiagramm der Plattform

4.2.1 Systemkontext

Der Systemkontext muss beim Entwurf des Systems berücksichtigt werden. Der Systemkontext wird gewissermaßen als starr angenommen, wobei das zu entwerfende System hingegen noch flexibel und gestaltbar ist. Daher muss der Systemkontext zunächst analysiert werden.

Fachanwendungen und Geoinformationssysteme

Fachanwendungen und Geoinformationssysteme stellen bereits existierende externe Systeme dar. Sie implementieren für eine bestimmte Domäne notwendige geographische Funktionen und bilden diese in vorgebenen Workflows der Geschäftsprozesse integriert ab. Sie bedienen sich zentralen Datenhaltungen um Arbeitsstände und -ergebnisse für weitere Prozesse nutzbar zu machen.

Ein Beispiel ist das Verwaltungssystem für die Kriegsluftbilder des KBD, das im Szenario 1 aufgeführt ist. Dieses stellt einen abgestimmten Funktionsumfang für die Vorver-

arbeitung der Kriegsluftbilder zur Verfügung und bindet die zentralen Datenhaltungen für Kriegsluftbilder und Verarbeitungsparameter ein.

Bei genauerer Analyse scheint die sprachliche Trennung von Fachanwendungen und Geoinformationssysteme ausschließlich dem allgemeinen Verständis geschuldet zu sein. Aus technischer Sicht weisen beide nahezu identische Merkmale auf. Zahlreiche Fachanwendungen enthalten Elemente von Geoinformationssystemen, wie integrierte Viewer oder Editoren für Geodaten. Umgekehrt werden viele Fachanwendungen durch anwendungsbezogene Anpassung von Geoinformationssystemen, über Programmierung von Plugins oder Konfigurationen, umgesetzt. Zur Vereinfachung können Fachanwendungen und Geoinformationssysteme aus Sicht der Plattform als eine gemeinsame Einheit gesehen werden. Aufgrund dieser Tatsache wird im Folgenden stellvertretend für beide Systeme ausschließlich das Akronym GIS verwendet.

Damit muss die Plattform geeignete Interfaces bieten, um direkt aus GIS heraus verwendet werden zu können. Bestehende Systeme verwenden oft systemnahe Integrationen, beispielsweise zum Zugriff auf Dateien über Network File Share (NFS)-Fileshares. Diese Integrationen weisen jedoch eine höhere Komplexität beim verteilten Betrieb über Wide Area Network (WAN) und unterschiedliche Sicherheitsdomänen auf. Im Bereich der heutzutage vorherrschenden webbasierten Geodaten-Infrastrukturen sind Interfaces weit verbreitet, die auf Hypertext Transfer Protocol (HTTP) basieren. Diese werden z. B. für die Übertragung von Karten mittels standardisierter Services, wie z. B. WMS oder WPS, eingesetzt. Nahezu alle aktuellen desktopbasierten GIS enthalten entsprechende Interfaces, um diese Services einzubinden.

4.2.2 System

Datenhaltung

Bei der Auswahl einer geeigneten Komponente zur Datenhaltung sind vielfältige Aspekte zu beachten. Grundsätzlich ist eine Vielzahl von Datenhaltungen für die Speicherung möglich, die aber nicht alle gleichermaßen sinnvoll sind. Zwischen einfachen und komplexen Datenhaltungen haben sich eine Vielzahl von Systemen etabliert, die je nach Anwendungsfall unterschiedliche Vor- und Nachteile bieten.

Die gewissermaßen einfachste Möglichkeit besteht in hierarchischen Filesystemen. Besonders bei großen Mengen von Daten steigt jedoch z. B. die Komplexität der Verwaltung von Zugriffsrechten, Methoden zur Durchsuchbarkeit oder Anbindung an verteilte Systeme. Eine wesentlich komplexere und stärker abstrahierte Datenspeicherung stellen Relational Database Management System (RDBMS) dar. Diese sind für die Speicherung von strukturierten Daten, die Beziehungen zwischen Objekten enthalten, entwickelt worden. Die Sicherstellung von *ACID-Compliance* (Atomacity, Consistency, Isolation, Durability) erfordert einen erheblichen Aufwand und führt zu Herausforderungen

bei verteilten Systemen, siehe *CAP-Theorem* [Gilbert and Lynch, 2002]. Daneben bieten sie zusätzliche Funktionen wie etwa die Steuerung von Zugriffsrechten. In vielen Anwendungsfällen wird keine *ACID-Compliance* benötigt, der Einsatz von RDBMS da der Nutzen in keinem Verhältnis zum Aufwand stünde.

Daher muss die Auswahl der Datenhaltung den Anwendungsfall berücksichtigen. Maßgeblich sind die Art der zu speichernden Daten und die erwarteten Zugriffsmuster zu analysieren.

Art der Daten

Bei den Geodaten handelt es sich um unstruktierte Daten, die aus Raster- und Vektordaten als Ein- und Ausgabe der Verarbeitungsprozesse bestehen. In erster Linie sind dies die Kriegsluftbilder im Rasterdaten-Modell. Ihre Größe variiert sehr stark und ist maßgeblich von der geometrischen und radiometrischen Auflösung abhängig. Die geometrische Auflösung der Kriegsluftbilder beträgt ca. 12000 x 10000 Pixel, die radiometrische Auflösung beträgt 8 Bit. Ein Datensatz ist folgenderweise ca. 96 Megabyte groß. Für die genannte Anzahl von ca. 130000 Datensätzen summiert sich der gesamte Speicherbedarf auf ca. 12,5 Terrabyte. Nach der Transformation ist anzunehmen, dass sich der Speicherbedarf auf insgesamt 37,5 Terrabyte verdreifacht. Das Ergebnis der Analyse-Prozesse können Vektor- oder Rasterdaten sein. Daher kann eine nachträgliche Vergrößerung des Speicherplatzes erforderlich sein.

Zugriffsmuster

GIS und die Verarbeitungskomponente sollen über ein Interface auf die Daten lesend und schreibend zugreifen (siehe Anforderung 1d). Der schreibende Zugriff erfolgt beispielsweise beim Import neuer Daten oder wenn Daten als Ausgabe einer Verarbeitung entstehen (siehe Anforderung 1d und Szenario 1). Dabei werden Datensätze immer vollständig innerhalb eines Vorganges gespeichert. Die partielle Aktualisierung eines Datensatzes ist nicht vorgesehen. Damit sind besondere Maßnahmen zur Transaktionssicherheit nicht erforderlich. Es ist zu erwarten, dass ein Großteil von lesenden Zugriffen bei der Visualisierung von Daten durch GIS erfolgen.

Zusammenfassung

Die Datenhaltung muss demnach für vergleichsweise große und statische Daten mit Write Once Read Many (WORM) Zugriffsmuster geeignet sein. Für genau diesen Anwendungsfall haben sich ObjectStores etabliert. Diese werden häufig in Medienportalen wie Netflix, YouTube oder Spotify zur Datenhaltung von Filmen, Musik, Fotos oder Download-Archiven eingesetzt. Diese Beispiele verdeutlichen auch das Zugriffsmuster: Filme oder Musik unterliegen keiner nachträglichen Änderung, werden aber von einer Vielzahl von Nutzern gleichzeitig konsumiert. ObjectStores interpretieren Dateien als Objekte, die über eine URL eindeutig identifiziert werden. Diese Objekte können mit Metadaten versehen werden, sodass sie innerhalb der ObjectStores indizierbar und damit performant zugreifbar sind. Der Zugriff erfolgt meist über Interfaces, die auf HTTP

basieren. Dadurch können ObjectStores sehr einfach in andere Applikationen integriert werden.

Verarbeitung

Die umfangreichsten und komplexesten Anforderungen richten sich gegen die Verarbeitungskomponente. Sie stellt damit einen Hauptbestandteil der Plattform dar und ist maßgeblich bei der Aufwahl von Architektur und Technologie zu berücksichtigen.

Eine Herausforderung liegt in der polygloten Implementierung von Verarbeitungsprozessen, die aus der Anforderung 1a hervorgeht. Ein zuverlässiger paralleler Betrieb von mehreren Programmiersprachen innerhalb eines Systems ist nicht möglich. Besser wäre die Kapselung in spezifische Ausführungsumgebungen für die einzelnen Programmiersprachen.

Eine weitere Herausforderung besteht in der flexiblen Reaktion auf temporäre Lastsituationen, die in Szenarien 1 und 2 beschrieben wird. In diesen Situationen muss die Plattform einen hohen Durchsatz aufweisen. Die beschriebenden Vearbeitungsprozesse sind *stateless*, d. h. sie können unabhängig von Zuständen der Applikationen oder Sessions ausgeführt werden und sind damit hervorragend horizontal skalierbar. Dies ermöglicht gleichzeitig auch die horizontale Skalierung der Infrastruktur-Ressourcen. Insofern scheint die Verwendung von ereignisgesteuerten Architekturmustern sinnvoll, die zusätzliche Applikations-Instanzen bei Lastsituationen flexibel auf weiteren Infrastruktur-Ressourcen bereitstellen und Anfragen über Load-Balancing verteilen.

Die polyglote Implementierung und horizontale Skalierbarkeit führen zu der Kernfrage, was unter Berücksichtigung der Bewertungskriterien überhaupt eine geeignete Applikations-Instanz zur Verarbeitung von Geodaten ist. Auf Public Clouds hat sich für diesen Anwendungsfall FaaS etabliert. FaaS ist ein Architekturmuster, bei dem eine Applikations-Instanz lediglich aus einer Funktion besteht. Als Synonym für FaaS wird ebenfalls oft das Schlagwort *Serverless* verwendet. Der Aufruf von Funktionen erfolgt zumeist über HTTP, wodurch auch FaaS sehr einfach in andere Applikationen integriert werden kann.

Um FaaS zu verstehen, ist ein Blick auf die technologischen Grundlagen des Cloud Computing hilfreich: Mithilfe von Virtualisierung wurde die Abbildung von logischen Systemen auf die physikalische Infrastruktur vollständig entkoppelt. Die Architektur von Service Oriented Architecture (SOA) verfolgt die lose Kopplung von Services, die zu Applikationen orchestriert werden können. Damit stehen nicht mehr Maschinen, sondern funktionale Einheiten im Fokus der Applikations-Architektur. FaaS bzw. Serverless stellt die vollständige Abstraktion der Infrastruktur dar. Ein CSC fokussiert sich ausschließlich auf Applikations-Architektur und überlässt die Organisation der Infrastruktor vollständig dem CSP.

4.3 Plattform-Architektur

Der Begriff Architektur ist im Software-Umfeld nicht eindeutig definiert. Es existieren eine Reihe von Definitionen, eine davon stammt von [Taylor et al., 2009]:

> A Software systems architecture is the set of principal design decisions made about the system.

Diese grundlegenden Entwurfsentscheidungen sind nachträglich nur mit erheblichem Aufwand zu verändern. Geprägt sind sie maßgeblich von nicht-funktionalen Anforderungen, da die funktionalen Anforderungen an ein System stetiger Änderung unterliegen. Die Architektur muss den Austausch und Anpassung von Komponenten an veränderte Anforderungen ermöglichen. Damit bestimmt die Architektur maßgeblich die Wartbarkeit eines Systems.

Mit einem Architekturentwurf soll eine vereinfachte Darstellung eines Systems erzeugt werden [Booch, 1994]. Architekturen können in Form von Mustern ausgetauscht und gemeinsam fortentwickelt werden. Ein weit verbreitetes Muster einer Applikations-Architektur ist beispielsweise die *N-Tier-Architektur*, die gekapselte Schichten für Datenhaltung, Business-Logik und Präsentation verwendet. Die *N-Tier-Architektur* zeigt eine sehr hohe Verbreitung, woraus auf einen hohen Reifegrad geschlossen werden kann.

Mithilfe der Bewertungskriterien kann nun zunächst eine geeignete Plattform-Architektur ausgewählt werden. Aus der Systemanalyse sind bereits ObjectStore und FaaS als geeignete Konzepte hervorgegangen. Die auszuwählende Architektur und Technologie muss diese Konzepte zu einer Plattform vereinen. Dafür geeignet sind Microservice-Architekturen, mit denen eine hohe Flexibilität durch lose Kopplung von einzeln wart-, test-, und betreibbaren Applikationen erreicht werden kann.

4.3.1 Schichten-Modell

Die erforderliche Flexibilität kann durch die Bildung von geeigneten Schichten in der Plattform-Architektur erreicht werden (siehe Abbildung 4.3). Durch die Schichten wird die bereits aufgegriffene Trennung der logischen Applikations-Architektur von der physikalischen Infrastruktur ermöglicht. Damit kann die Applikations-Architektur nachträglich verändert werden, indem Applikationen einzeln aktualisiert, verändert und sogar ersetzt werden. Die innere Architekur einer Applikationen muss explizit darauf ausgelegt sein. Beispielsweise können eine hohe Abstraktion von der Infrastruktur und parallelisierte Kontroll- und Datenflüsse nicht nachträglich hinzugefügt werden. Geeignete Applikationen werden daher auch *cloud-native* genannt.

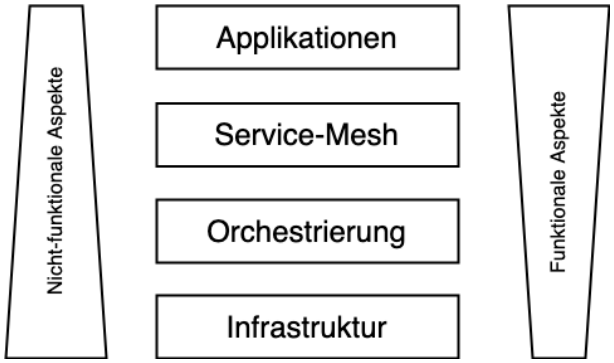

Abbildung 4.3: Schichten der Plattform

Die Architektur der Plattform enthält vier wesentliche Schichten, die Applikationen, Service-Mesh, Orchestrierung und Infrastruktur voneinander trennen. Untere Schichten setzen überwiegend nicht-funktionale Anforderungen, höhere Schichten setzen überwiegend funktionale Anforderungen um. Jede Schicht erfüllt dabei einen bestimmten Zweck:

1. **Applikationen**
   Implementierung der logischen Applikations-Architektur, bestehend aus Applikationen für Datenhaltung (ObjectStore) und Verarbeitung (FaaS)
2. **Service-Mesh**
   Implementierung nicht funktionaler Anforderungen, z.B. Monitoring, Resilienz
3. **Orchestrierung**
   Orchestrierung von Applikationen und Service-Mesh, Verteilung auf Infrastruktur
4. **Infrastruktur**
   Bereitstellung der Infrastruktur-Ressourcen wie CPU, RAM, GPU, TPU

4.3.2 Technologie

Auf Basis des Schichten-Modells erfolgt nun die Auswahl der geeigneten Technologie. Dabei widmet sich dieses Kapitel hauptsächlich der Auswahl für die Orchestrierungsschicht.

Von der untersten Schicht ausgehend, ist zunächst die Auswahl einer geeigneten Technologie für die Infrastruktur zu treffen. Bezüglich des zu verwendeten CPU-Befehlssatzes existieren Zwangspunkte. Diese resultieren aus der weiten Verbreitung von x64 - basierten Befehlssätzen, die den Quasi-Standard der verwendeten CPU definieren. Sowohl in Public Clouds, lokalen Entwicklungsumgebungen und im LGLN sind x-64 CPU vorherrschend. Weitere existierende Befehlssätze, wie z. B. *ARM* oder *RISC-V*, zeigen zum aktuellen Zeitpunkt keine praktische Relevanz.

Aus dem Modell der Schichten folgt, dass die Orchestrierung folgende Funktionalität erfüllen muss: Einerseits das Hinzufügen und Entfernen von Infrastruktur-Ressourcen und andererseits die Verwaltung der logischen Applikations-Architektur. Die Auswahl der verwendeten Technologie zur Orchestrierung ist maßgeblich von der technologischen Basis der Applikations-Instanzen bestimmt. Für diese existieren unterschiedliche Möglichkeiten:

Virtuelle Maschinen

Eine Möglichkeit sind VM. Diese enthalten eigenständige Betriebssysteme und einen eigenständigen Kernel. In VM können Applikationen installiert, konfiguriert und Services bereitgestellt werden. Als Schnittstelle des Kernels zur Infrastruktur werden Hypervisors eingesetzt, die die Zuweisung der Ressourcen vornehmen und die VMs voneinander kapseln. Verwaltung, Pflege und Instanziierung von VM kann über Configuration-Management-Tools erfolgen. Dazu existiert eine Vielzahl von Tools, die im Cloud Computing Umfeld eine hohe Verbreitung aufweisen, wie z. B. Puppet, Chef, Saltstack oder Ansible.

Prinzipiell könnte die Funktionalität vollständig mit VMs erfüllt werden. Betrachtet man zusätzlich die Kriterien der Kategorien Kompatibilität und IT-Sicherheit, weisen VM eine hervorragende Eignung auf. Auch die Untersützung durch Cloud Service ist als gut zu beurteilen, gängige IaaS-Angebote unterstützen selbst erstellte VM-Images und sind mit Open-Source Configuration-Management-Tools kompatibel. Nicht erfüllen lassen sich hingegen die Kriterien der Kategorien Effizienz, Zuverlässigkeit und Wartbarkeit. Dies ist auf den Aufbau von VM zurückzuführen: Der integrierte Kernel erfordert eine feste Zuweisung von Infrastruktur-Ressourcen. Die Veränderung dieser Zuweisung wäre nur bei ausgeschalteter VM möglich. Durch lange Starts und Stops der VM ist zudem die Skalierbarkeit nur langsam möglich. Daher werden einer VM meist mehr Ressourcen zugewiesen, als für die durchschnittliche Arbeitslast benötigt wird. Die vollständige Installation der gesamten Plattform wäre auf der üblichen Hardware lokaler Entwicklungssysteme kritisch. Dadurch wären auch Analysierbarkeit und Testbarkeit stark eingeschränkt.

Container

Eine andere Möglichkeit sind Container. Diese sind vergleichbar zu VM, weisen jedoch eine wesentlich höhere Effizienz und Portabilität auf. Sie enthalten keinen eigenen Kernel, sondern verwenden den Kernel des Hosts. Dieser benutzt interne Mechanismen für die Isolation von Container-Prozessen. Dadurch entfällt die starre Zuweisung von Hardware-Ressourcen und damit verbundene Start- und Stopzeiten. Aufgrund dieser Vorteile haben sich Container als eine zentrale Technolgie im Cloud Computing etabliert. Container können beliebige Applikationen enthalten. Sie sind sehr gut wartbar, weil sie lokal entwickelt, getestet und als *imutable*[2] Image portiert werden können. In der zentralen Docker-Registry *Docker-Hub* sind 2.895.312 Images[3] verfügbar. Von allen existierenden Container-Frameworks zeigt Docker damit die größte Verbreitung (siehe Kapitel 2.4.1),

[2] unveränderbares
[3] Stand 05.12.2019

was auch aus der GitHub-Projektseite hervorgeht. Dort sind 46.732 Commits von 2.006 Contributors und 3700 Stars angegeben[4].

Orchestrierung

Zur Orchestrierung von Containern hat sich Kubernetes etabliert(siehe 2.4.2). Kubernetes ermöglicht die Verteilung von Containern über viele Hosts. Kubernetes selbst ist sehr ressourcenschonend und kann auch in lokalen Entwicklungsumgebungen verwendet werden. Für macOS, Windows und Linux existieren angepasste Distributionen. Unter Last können Hosts vollständig ausgelastet werden. Die horizontale Skalierung von Containern erfolgt durch das Starten eines neuen Container-Prozesses auf dem Host, was sehr schnell erfolgen kann. Der maximale Durchsatz ist je nach Applikation unterschiedlich, wird jedoch maßgeblich durch die Ressourcen des Hosts begrenzt. Kubernetes erhöht die Zuverlässigkeit von Applikationen durch Mechnismen wie *Self-Healing* und *Load-Balancing*. Durch das Konzept des Horizontal Pod Autoscaler (HPA) können Container automatisch horizontal skaliert werden. Zum Update von Containern verwendet Kubernetes *Rolling-Update-Strategien*. Zur Kommunikation mit Kubernetes werden meist authentifizierende Secure Shell (SSH)- oder Hypertext Transfer Protocol Secure (HTTPS)-Verbindungen verwendet. Kubernetes selbst weist eine hohe Dekomposition auf (siehe Kapitel 2.4.2). Diese wurde im Mai 2019 einem Sicherheits-Audit unterzogen, dessen Ergebnis auch auf GitHub veröffentlicht wurde [Small et. al., 2019]. Das GitHub-Projekt von Kubernetes hat aktuell 86.231 Commits von 2375 Contributors und 60.800 Stars [Github, 2019], was eine vergleichsweise sehr große Community darstellt. Zahlreiche CSP bieten verwaltete Kubernetes-Services, darunter z. B. *Google Kubernetes Engine* von Google, *Azure Kubernetes Service* von Microsoft oder *Elastic Kubernetes Service* von Amazon. Viele CSP beteiligen sich an der Entwicklung, was den Eindruck eines hohen Professionalisierungsgrad untermauert.

Unter Berücksichtigung der Bewertungskriterien weist eine Container-Orchestrierung durch Kubernetes im Anwendungsfall erhebliche Vorteile gegenüber VM auf. Damit soll Kubernetes die Basis der skalierbaren Plattform zur Verarbeitung von Geodaten bilden.

[4]Stand 05.12.2019

5 Entwurf

In diesem Kapitel wird eine skalierbare Plattform zur Verarbeitung von Geodaten auf Basis von Kubernetes vorgestellt. Dazu wird zunächst die grundlegende Idee erläutert und anschließend die Applikations-Architektur dargestellt. Darauffolgend wird die Funktionsweise anhand der Verarbeitung eines Datensatzes dargestellt. Anschließend werden die eingesetzten Applikationen und ihre internen Architekturen vorgestellt. Zum Schluss wird eine Optimierung der Datenhaltung für Lesezugriffe in einem eigenen Unterkapitel vorgestellt.

5.1 Grundlegende Idee

Die Plattform wird als ein verteiltes System aus spezialisierten Applikationen für Datenhaltung und Verarbeitung aufgebaut. Die Applikationen werden als Container betrieben. Zur Orchestrierung der Container wird Kubernetes eingesetzt. Kubernetes ermöglicht die dynamische Anpassung der zugrundeliegenden Infrastruktur-Ressourcen und setzt gleichzeitig die horizontale Skalierung von Applikationen um.

Geodaten werden in einem ObjectStore gespeichert. Rasterdaten werden für WORM-Zugriffe optimiert und im Format Cloud-Optimized GeoTIFF (COGTIFF) abgelegt. Die Verarbeitung der Geodaten erfolgt in einer FaaS-Umgebung. Um die Komplexität der Implementierung von FaaS-Funktionen zu minimieren, werden diese durch ein Service-Mesh ergänzt, welches Resilienz-Muster verteilter Systeme integriert.

Eingesetzt werden *Minio* als ObjectStore, *OpenFaaS* als FaaS-Umgebung und *Linkerd* als Service-Mesh, sodass der gesamte Software-Stack aus OSS besteht. Als Protokoll wird HTTP verwendet, sodass eine lose Kopplung und gleichzeitig große Interoperabilität mit GIS gewährleistet wird.

Durch die Technologie und Architektur soll eine erhebliche Steigerung der Software-Qualität gegenüber bestehenden Systemen erreicht werden. Vorrangig sollen vor allem

1. ein skalierbarer Durchsatz durch parallele Verarbeitung von Geodaten und
2. Flexibilität für Anpassungen an technologische Entwicklung erzielt werden.

5.2 Applikations-Architektur

Die Plattform-Architektur (siehe Abbildung 4.3) zeigt Applikations- und Service-Mesh-Schicht voneinander getrennt. Die Plattform-Architektur muss diese differenziert betrachten, da das Service-Mesh eine Art von Infrastruktur-Schicht für die Applikationen darstellt. Tatsächlich handelt es sich bei dem Service-Mesh jedoch um eine weitere bereitgestellte Applikation. Insofern gibt es aus Sicht der Orchestrierung *de facto* keinen Unterschied zwischen Applikations- und Service-Mesh-Schicht.

Die Abbildung 5.1 zeigt eine vereinfachte schematische Darstellung der Applikations-Architektur. Abgebildet sind GIS, die mit dem ObjectStore und der FaaS-Umgebung in der Applikationen-Schicht über HTTP kommunizieren. Die Kommunikation zwischen ObjectStore und FaaS wird durch die Data-Plane in der Service-Mesh-Schicht geroutet. Dort befindet sich auch die *Control-Plane*, die Konfigurationen und Metriken mit der *Data-Plane* austauscht. Die drei Applikationen ObjectStore, FaaS und Service-Mesh besitzen eigene Architekturen und beinhalten weitere Komponenten.

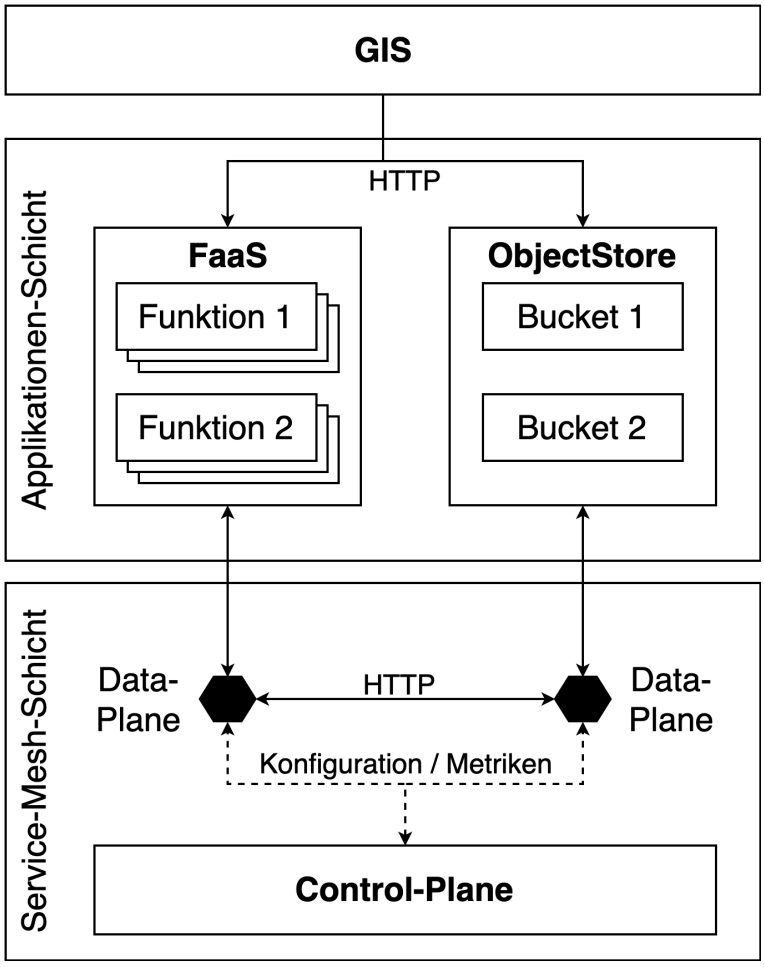

Abbildung 5.1: Logischer Aufbau der Plattform

5.3 Funktionsweise

Abbildung 5.2 zeigt eine vereinfachte Darstellung der Funktionsweise. Dort ist die synchrone Verarbeitung von Geodaten abgebildet. Nicht abgebildet sind eventuell ausgeführte Requests zum Laden von Metadaten (siehe Kapitel 5.5).

GIS speichern sämtliche Geodaten in dem ObjectStore, sodass jeder Datensatz über eine eindeutige URL identifizierbar ist. Zur Verarbeitung übermitteln GIS die URL eines Datensatzes mit optionalen Verarbeitungsparametern an eine FaaS-Funktion. Die Funktion lädt mithilfe der URL die Geodaten aus dem ObjectStore und führt anschließend die Verarbeitung durch. Das Ergebnis wird durch die Funktion wieder im ObjectStore gespeichert und die erhaltene URL als Rückgabewert an GIS zurückgegeben. GIS laden das Ergebnis der Verarbeitung mithilfe dieser URL aus dem ObjectStore.

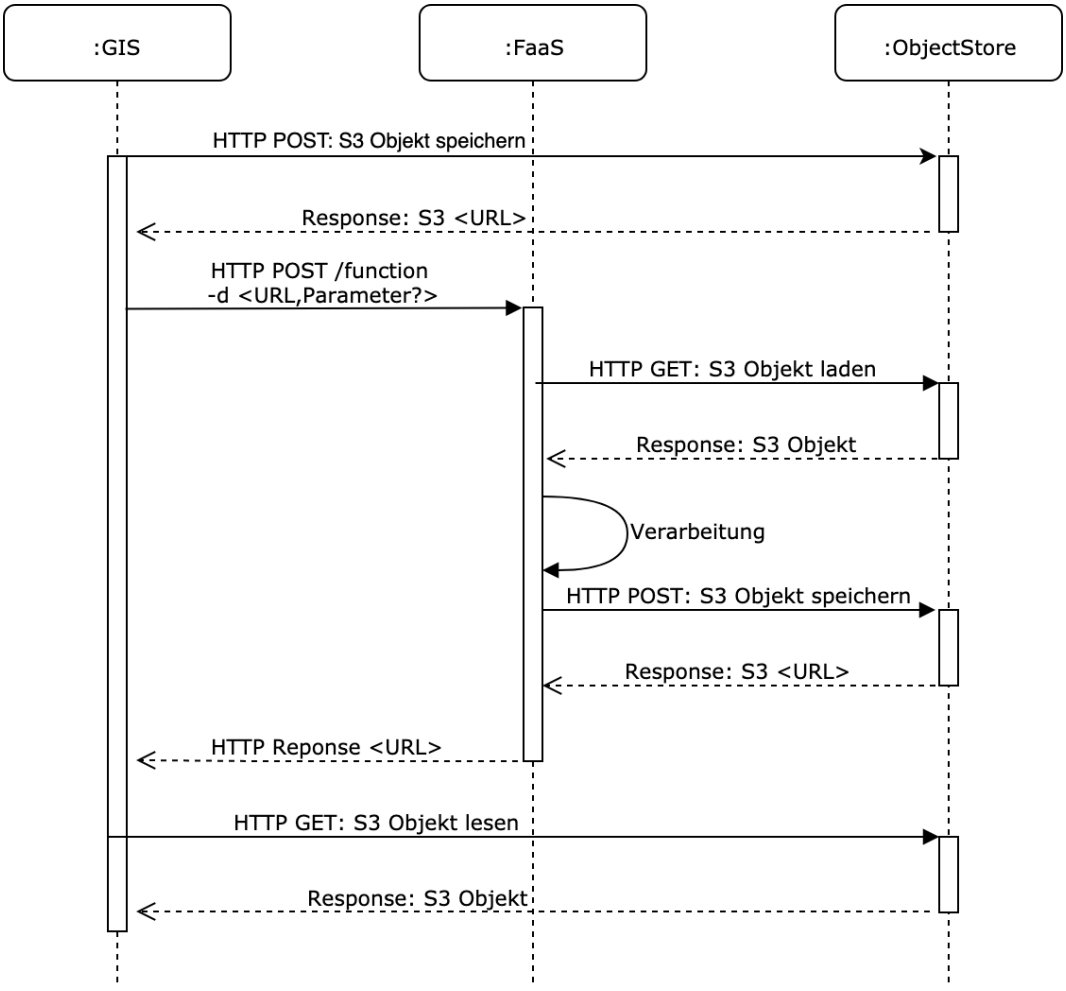

Abbildung 5.2: Sequenzdiagramm der synchronen Verarbeitung

5.4 Applikationen

Erst durch die Orchestrierung der Applikationen zu einer gemeinsamen Plattform werden die funktionalen Anforderungen vollständig erfüllt. Eingesetzt werden *Minio* als ObjectStore, *OpenFaaS* als FaaS-Umgebung und *Linkerd* als Service-Mesh. Diese sind durch die Verwendung von verbreiteten Interfaces lose gekoppelt, sodass sie jederzeit gegen eine andere interoperable Applikation ausgetauscht werden können.

5.4.1 Minio

Als ObjectStore wird Minio verwendet, der unter der Open-Source Lizenz Apache-2.0 steht. Minio stellt ein Simple Storage Service (S3)-Interface zum Zugriff auf Daten bereit. S3 ist ein von Amazon veröffentlichter freier Standard für den Zugriff auf ObjectStores. Dieser basiert auf REST über HTTP und hat sich als *de facto* Standard etabliert. Viele auf Public Clouds angebotenen proprietären ObjectStores verwenden ebenfalls S3. Für diese bietet Minio auch einen Gateway-Modus. So kann Minio innerhalb der Applikations-Architektur durch einen anderen S3-kompatiblen ObjectStore ersetzt werden.

Minio bietet einen expliziten WORM-Modus, der jegliche API für die Veränderung von Metadaten oder Objekten deaktiviert. Unterstützt werden auch HTTP-Range-Header, sodass eine optimierte Bereitstellung für Lesezugriffe durch Verwendung von COGTIFF (siehe Kapitel 5.5) möglich ist. Weiter sind komplexe Konfigurationen von On-The-Fly-Verschlüsselung, Sharding, Replikation uvm. möglich. Diese werden im Rahmen dieser Arbeit nicht weiter betrachtet.

Neben der Server-Komponente existieren zusätzlich ein Kommandozeileninterface Minio-Client und zahlreiche Software Development Kit (SDK) für verbreitete Programmiersprachen, wie z. B. Python, JavaScript, Java, .NET oder Golang. Für macOS, Linux, BSD und Windows werden Installations-Binaries zum Download angeboten. Auf Docker-Hub sind offizielle Images für Server und Minio-Client verfügbar. Die Github-Projektseite zeigt aktuell 6.195 Commits von 188 Contributoren und 19.100 Stars[1].

5.4.2 OpenFaaS

Zur Verarbeitung der Geodaten wird OpenFaaS eingesetzt, das unter der Open-Source Lizenz *MIT* veröffentlicht ist. OpenFaaS ist ein Framework für das Erzeugen, Bereitstellen und Ausführen von ereignisgesteuerten kurzläufigen Funktionen. Anders als proprietäre FaaS auf Public Clouds kann es auf eigenen Systemen betrieben werden. In

[1] Stand 05.12.2019

diesem Fall profitiert lediglich ein Funktions-Entwickler vom FaaS-Prinzip der Abstraktion von Infrastruktur. Der Betreiber einer OpenFaaS-Installation übernimmt hingegen die Aufgaben des CSP. OpenFaaS-Funktionen werden in Docker-Containern ausgeführt, wodurch die Implementierung in jeglicher Programmiersprache erfolgen kann. Für gängige Programmiersprachen existieren durch die Community gepflegte Templates. Eigene Templates können durch Dockerfiles ergänzt werden.

Die Community von OpenFaaS ist groß und sehr aktiv. Auf der Github-Projektseite sind aktuell 1.769 Commits von 131 Contributoren und 16.300 Stars verzeichnet[2]. Diese Einschätzung wird auch durch zahlreiche Konferenz-Beiträge, wie zur *KubeCon* oder *Dockercon* und den sehr aktiven Slack-Channel gestützt. Mit DigitalOcean bietet zudem ein Cloud Service Provider bereits eine native Unterstützung [DigitalOcean, 2019].

Architektur

Zur Ausführung der Funktionen sind weitere Komponenten zur Verwaltung und Skalierung von Funktionen erforderlich. Ein Teil dieser wird in separaten Docker-Containern betrieben, ein anderer Teil wird von der zugrundeliegenden Orchestrationsschicht wiederverwendet. Die folgende Abbildung 5.3 zeigt eine schematische Darstellung der Architektur.

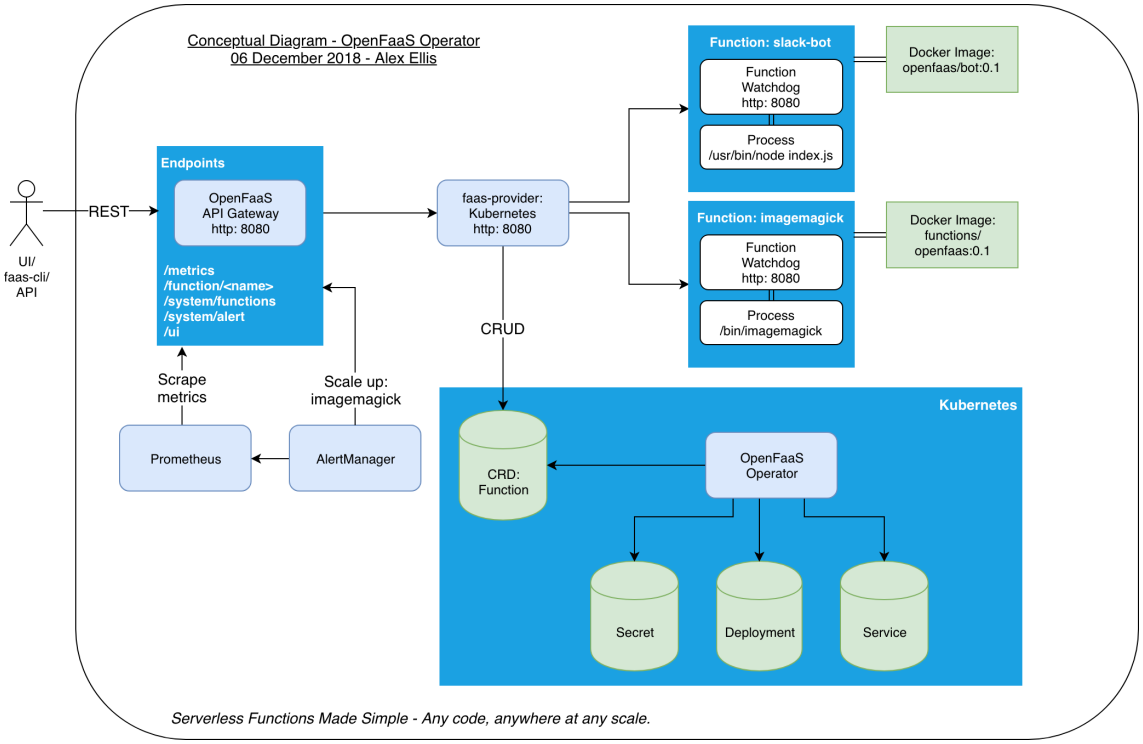

Abbildung 5.3: Architektur von OpenFaaS [OpenFaaS Project, 2019]

[2]Stand 04.12.2019

Die zentrale Komponente ist das API-Gateway. Dieses nimmt Requests über eine authentisierende REST-API über HTTP entgegen. Dies können Requests zum Bereitstellen, Löschen und Monitoring von Funktionen sein. Diese werden beispielsweise über das OpenFaaS-User-Interface oder über das Kommandozeileninterface **faas-cli** gesendet. Die Metriken des API-Gateways werden durch Prometheus[3] abgegriffen, der wiederum von dem AlertManager auf langdauernde Funktionsaufrufe überwacht wird. Sofern notwendig, veranlasst der AlertManager über einen Aufruf am API-Gateway die horizontale Skalierung von Funktionen.

Zum Bereitstellen und Löschen von Funktions-Containern verwendet das API-Gateway die zugrundeliegende Orchestrierungsschicht. Dazu wird das Interface *faas-provider* verwendet, für welches auch Implementierungen für Amazon Lamda, Docker Swarm, Hashicorp Nomad, AWS Fargate/ECS und Kubernetes verfügbar sind. Im Fall von Kubernetes wird die Implementierung *faas-netes* verwendet, die über Custom-Ressource-Definitions die nativen Kubernetes Objekte Secret, Deployment und Service verwendet.

Funktionen

Durch Kubernetes erfolgt das Bereitstellen von Funktionen als Container bzw. Pod (siehe Kapitel 2.4.2). Insofern muss für jeden Verarbeitungsprozess ein Docker-Image mit der OpenFaas-Funktion auf einer für Kubernetes zugänglichen Container-Registry bereitgestellt werden. Die Erstellung von Funktionen wird durch **faas-cli** unterstützt. Die Implementierung muss sich an der in Kapitel 5.3 vorgestellten Funktionsweise orientieren.

5.4.3 Linkerd

Als Service-Mesh wird Linkerd eingesetzt, das unter der Open-Source Lizenz *Apache-2.0* veröffentlicht ist. Die Kommunikation von Minio und OpenFaaS wird mithilfe von Sidecar-Containern durch Linkerd geleitet. Durch Dashboards werden Logs, Metriken und Datenflüsse visualisiert. Weiter können folgende Funktionalitäten transparent für die Applikationen ergänzt werden:

1. Verschlüsselung der Kommunikation
2. Retries, Timeout, Circuit Breaking
3. Telemetrie und Monitoring (z. B. Traffic, Latenz, Success, Errors)
4. A/B-Testing

[3] Collector für Metriken und Zeitreihen (https://prometheus.io/)

Prinzipiell ähneln diese Funktionalitäten dem *Netflix OSS* Stack. Gegenüber diesem bieten Service-Meshes den Vorteil, dass sie nicht in die Applikationen integriert werden müssen. Stattdessen wird das *Sidecar-Muster* verwendet (siehe Kapitel 5.4.3). Dadurch bleiben Applikationen einfach in der Implementierung und gleichzeitig portabel auf andere Umgebungen.

Linkerd zeigt eine hohe Verbreitung, wie aus aus der GitHub-Projektseite hervorgeht. Dort sind aktuell 1.402 Commits von 99 Contributoren und 5.100 Stars verzeichnet[4]. Praxisorientierte Veröffentlichungen bescheinigen Linkerd zudem eine geringe Komplexität und empfehlen es als Einstieg in die Service-Meshes [Bornkessel, 2019].

In Abbildung 5.4 ist die Architektur von Linkerd dargestellt. Linkerd teilt sich in eine Control-Plane und eine Data-Plane auf. Die Control-Plane ist für die Verwaltung des Services-Meshes, Bereitstellung von Metriken und Visualisierung zuständig. Der Controller stellt eine API bereit, die von dem dazugehörigen Kommandozeileninterface **linkerd-cli** und dem Web-User Interface (UI) angesprochen wird. Aus dem Controller heraus werden Metriken an Prometheus[5] gesendet, zur Visualisierung der Daten wird Grafana[6] verwendet. Die Data-Plane besteht aus den Sidecar-Containern, die den Applikationen im Service-Mesh hinzugefügt werden.

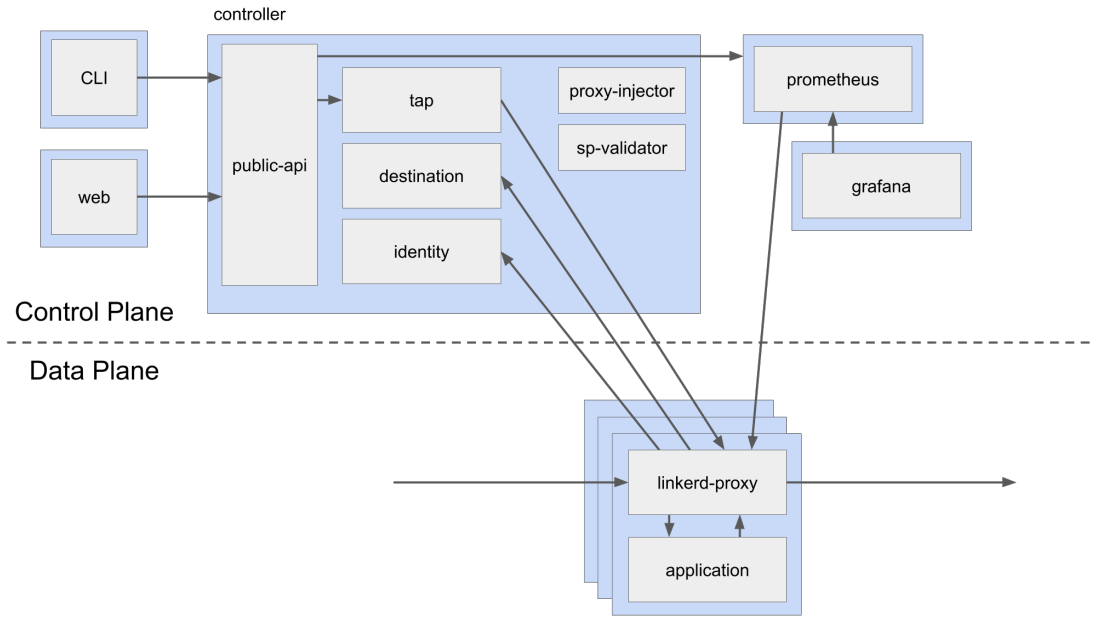

Abbildung 5.4: Architektur von Linkerd [Linkerd, 2019]

[4]Stand 06.12.2019
[5]Collector für Metriken und Zeitreihen
[6]Web-Oberfläche für Visualisierung

Sidecar-Muster

Durch das Sidecar-Muster kann eine Applikation um Funktionalität ergänzt werden, ohne dass die Applikation selbst verändert werden muss. Dazu wird eine atomare Container-Gruppe erstellt, die aus dem Applikations-Container und dem Sidecar-Container besteht. Kubernetes bietet für eine solche Container-Gruppe das native Pod-Objekt (siehe Kapitel 2.4.2).

Die Sidecar-Container können unterschiedliche Aufgaben übernehmen. Bei Service-Meshes verändern die Sidecar-Container die Netzwerkkonfiguration der Container-Gruppe derart, dass sämtlicher eingehender und ausgehender Netzwerkverkehr der Gruppe über sie geroutet wird. Damit übernehmen die Sidecar-Container die Rolle von Proxies. Die nachfolgende Abbildung 5.5 zeigt, wie zwei Applikationen miteinander durch die linkerd-proxy Sidecar-Container kommunizieren.

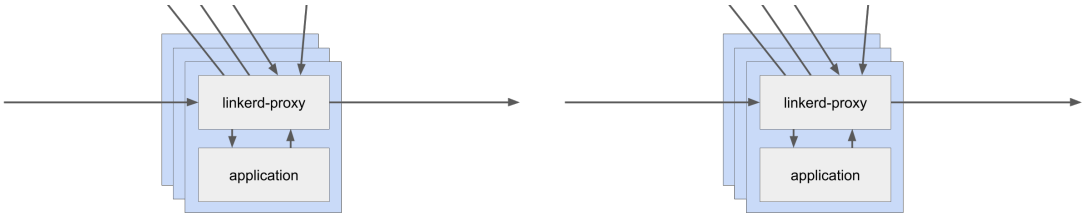

Abbildung 5.5: Data-Plane Routing in Linkerd [Linkerd, 2019]

Die Konfiguration der Proxies wird durch die Service-Mesh Control-Plane vorgenommen. Eine Änderung der Konfiguration sollte ohne Neustart der Container-Gruppe möglich sein, was verbreite Web-Proxies nicht unterstützen. Daher verwendet Linkerd einen eigenen Proxy-Server der zur Laufzeit über API konfigurierbar ist.

5.5 Cloud-Optimized GeoTIFF

Ein Großteil der Zugriffe auf Rasterdaten sind Lesezugriffe. Die Übertragung der vergleichweise großen Rasterdaten(ca. 96 MB) ist über WAN und Local Area Network (LAN) sehr zeitintesiv. Dadurch wird die Geschwindigkeit von konsumierenden Systemen stark verlangsamt. Sinkt in dieser Folge die Responsivität einer Applikation, führt dies zu einer schlechten User-Experience und wenig Akzeptanz bei Nutzern.

Gleichwohl erfordern konsumierende Systeme in vielen Fällen keinen vollständigen Raster-Datensatz. Ein Beispiel ist die Analyse einer AOI auf Kriegsbelastungen. In diesem Fall ist lediglich der zu analysierende Bereich von Interesse, außerhalb liegende Flächen brauchen nicht übertragen zu werden. Ein weiteres Beispiel ist die Visualisierung in GIS, wo die darstellbare geometrische Auflösung durch den verwendeten Monitor beschränkt wird. Bei derzeit verwendeten Desktop-Systemen beträgt diese meist 1920 x 1080 Pixel.

Bei der Visualisierung eines Kriegsluftbildes können ca. 69% der Pixel nicht dargestellt werden.

Daraus folgt, dass eine mögliche Übertragung von geometrisch reduzierten Unterbereichen eines Raster-Datensatzes zu einer wesentlich besseren User-Experience führt. Die Kombination eines ObjectStores mit dem Format COGTIFF bietet dazu geeignete Mechanismen, die in diesem Abschnitt erläutert werden.

5.5.1 TIFF

Die Grundlage dafür bildet das TIFF. TIFF ist ein freies Format [Aldus Corporation, 1992] und wird von vieler Software unterstützt. Weil es die Speicherung in einer unkomprimierten großen Farbtiefe ermöglicht, wird es häufig als Druckformat eingesetzt. Aufgrund seiner Interoperabilität und des freien Formates ist es besonders für Langzeitspeicherung geeignet.

Die folgende Abbildung 5.6 zeigt den schematischen Aufbau einer TIFF-Datei. Eine TIFF-Datei enthält einen Header, der auf ein Image File Directory (IFD) verweist. Dieses IFD beinhaltet eine Liste von Referenzen auf Directory-Entries, welche letztendlich Metadaten in Form von Key-Value-Paaren (Tags) und das tatsächliche Raster enthalten. Enthält ein TIFF mehrere Directory-Entries, wird es auch *Multipage-TIFF* genannt.

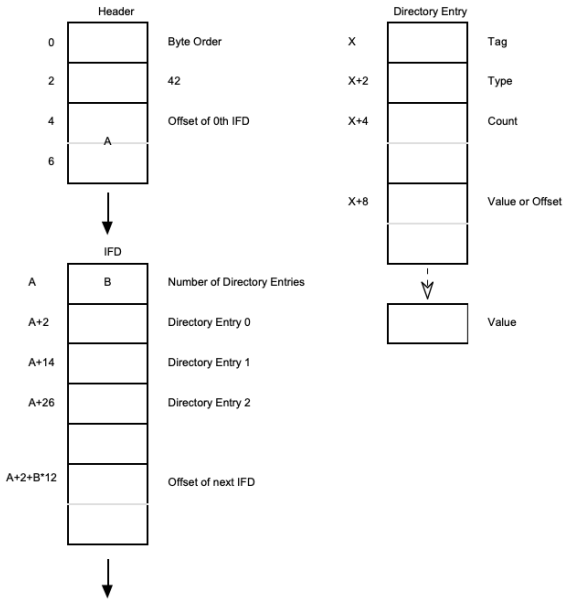

Abbildung 5.6: Struktureller Aufbau eines TIFF [Aldus Corporation, 1992]

5.5.2 GeoTIFF

Bereits kurz nach der Veröffentlichung des TIFF-Standards wurden zusätzliche Tags als Erweiterung des ursprünglichen Standards für geographische Metadaten erarbeitet [Ritter, Ruth, 1995]. Das sogenannte *Geography Tagged Image File Format* (GeoTIFF) konnte sich als Standard für Rasterdaten in der Geoinformatik etablieren. Zahlreiche GIS unterstützen die Verarbeitung von GeoTIFF.

Beispiele von Tags der GeoTIFF-Spezifikation sind *XResolution*, *YResolution*, *XPosition*, und *YPosition*, mithilfe dieser eine affine Tranformation[7] des Rasters berechnet werden kann. Durch die zusätzliche Angabe eines Koordinatenreferenzsystems, z. B. als EPSG-Code oder im Well Known Text (WKT)-Format, kann das Raster lagerichtig in GIS dargestellt werden.

GeoTIFF wird häufig für Fernerkundungsdaten eingesetzt, die üblicherweise eine hohe spektrale Auflösung aufweisen. Neben bekannten RGB-Kanälen[8] enthalten sie zusätzliche Kanäle für nicht sichtbare Wellenlängen, wie Infrarot oder Ultraviolet. Weil Fernerkundungsdaten zudem oft eine erhebliche radiometrische Auflösung aufweisen, entstehen insbesondere bei noch weit verbreiteten Systemen mit rotierenden Festplatten erhebliche Ladezeiten. Aus diesen Gründen werden Pyramiden-, und Kachel- und Kompressionsverfahren für GeoTIFF eingesetzt, die je nach Anwendungsfall unterschiedliche Szenarien abdecken.

Pyramidenbildung

Mithilfe der Bildung von Pyramiden wird das vollständige Laden eines Datensatzes verhindert, wenn die geometrische Auflösung eines Rasters die geometrische Auflösung des Monitors übersteigt. In diesem Fall wird das Raster nach dem vollständigen Laden durch die Grafikkarte dynamisch mithilfe eines Resamplingverfahrens skaliert. Um dies zu umgehen, werden verkleinerte Abbildungen des Rasters, die sogenannten Pyramiden, bereits im ursprünglichen Datensatz gespeichert und können von dort durch das Geoinformationssystem in Abhängigkeit des aktuell betrachteten Ausschnittes geladen werden. Damit wird die notwendige Ladezeit reduziert. Die Anzahl der erzeugten Pyramiden richtet sich nach der Rastergröße. Das Verhältnis der Pyramiden zueinander beträgt 2:1 Pixel.

Kachelbildung

Durch die Bildung von Kacheln wird das vollständige Laden eines Datensatzes dann verhindert, wenn lediglich ein Teilbereich eines Rasters dargestellt werden soll. Dies kann dann der Fall sein, wenn sich z. B. durch eine starke Vergrößerung oder Betrachtung eines Randbereichs weite Teile des Bildes außerhalb des Monitores befinden. Daher wird die Speichereihenfolge der Rasterzellen von einer normalerweise in Zeilen und Spalten angeordneten Struktur hin zu einer kachelbasierten Reihenfolge verändert. Das

[7] Translation, Skalierung und Rotation
[8] Rot, Grün, Blau

ursprüngliche Raster wird dementsprechend zu einem Raster von untergeordneten Rastern, den sogenannten Kacheln, verändert. Durch GIS können einzelne Kacheln eines Datensatzes geladen werden, wodurch die Ladezeit stark minimiert wird.

Kompression

Eine weitere Möglichkeit stellt die Kompression dar. Für das TIFF-Format bestehen unterschiedliche Möglichkeiten, wie z.B. Lempel Ziv Welch (LZW) oder Zstandard (ZSTD). Generelle Eigenschaften der Datenkompression lassen sich auch auf Rasterdaten anwenden. Grundsätzlich ist ein Verfahren danach auszuwählen, ob der komprimierte Datensatz verlustfrei oder verluftbehaftet sein darf. Ebenso ist grundsätzlich die Geschwindigkeit der Kompression und Dekompression gegen die resultierende Größe abzuwägen (Time-Memory-Tradeoff). Die Kompression kann individuell für jedes IFD vorgenommen werden.

5.5.3 Cloud-Optimized GeoTIFF

Das *Cloud-Optimized GeoTIFF* (COGTIFF) stellt ein für ObjectStores optimiertes TIFF dar. Es enthält eine strukturierte Folge von Pyramiden und Kacheln, die einzeln über HTTP-Range-Requests ausgeliefert werden können [Rouault et. al., 2019]. Den ersten Directory-Entry bildet ein verlustfrei komprimiertes gekacheltes Raster. Die folgenden Entries enthalten komprimierte gekachelte Pyramiden, die zugunsten einer kleinen Größe verlustbehaftet sein können.

HTTP-Range-Requests ermöglichen die begrenzte Übertragung des TIFF-Headers. Dieser enthält sämtliche Metadaten über die Byte-Position der TIFF-Directory-Entries, sowie GeoTIFF-Tags der räumlichen Orientierung. Mit diesen Informationen kann ein Client den für die Visualisierung/Verarbeitung erforderlichen Byte-Bereich im TIFF bestimmen. Über weitere HTTP-Range-Requests wird dann der relevante Bereich des TIFF übertragen. Range-Header sind nach dem HTTP-Standard optionale Elemente eines Requests, sodass der Zugriff auf die vollständige Datei weiterhin möglich ist.

Das Verfahren bietet damit erhebliche Vorteile. Der wesentliche Vorteil ist die minimierte Übertragungzeit für Lesezugriffe. Die Übertragungszeit eines TIFF-Ausschnittes konnte in einem Benchmark von ca. 3 min auf ca. als 1.5 s reduziert werden. Dieser Benchmark kann dem Anhang 9.3.1 entnommen werden. Hinzu kommt, dass Raster weiterhin verlustfrei gespeichert werden können, was die Langzeitarchivierung ermöglicht. Das Datenmanagement ist sehr einfach, weil eine einzige Datei das originale Raster und seine optimierten Pyramiden enthält.

Diesen Vorteilen stehen aber auch Nachteile entgegen. Zum Einen steigert die Erzeugung eines COGTIFF die Komplexität beim Speichern. Zum Anderen ist im Worst-Case von unkomprimierten Pyramiden ein um etwa $\frac{1}{3}$ erhöhter Speicherbedarf zu erwarten. Zudem müssen zusätzliche HTTP-Requests ausgeführt werden.

Im Kontext der vorliegenden Arbeit überwiegen die Vorteile, da Kosten für Speicherplatz gering und die zusätzlichen HTTP-Requests weitaus performanter als die Übertragung eines vollständigen Datensatzes sind. Ebenso steht mit GDAL eine weit verbreitete Bibliothek zur Verfügung, die COGTIFF nativ lesen kann. Für ein kommendes GDAL-Release ist zudem die direkte Integration von COGTIFF als Ausgabeformat geplant [Warmerdam et. al., 2019], sodass die Komplexität der Speicherung entfällt. Aus diesen Gründen erfolgt die Speicherung der Rasterdaten im ObjectStore im Format COGTIFF.

6 Umsetzung

In diesem Kapitel wird die Implementierung des vorgestellten Entwurfs dargestellt. Das Kapitel umfasst die Beschreibung von erforderlichen clientseitigen Programmen, dem eigentlichen Aufbau der Plattform auf einer Public Cloud, sowie die Implementierung der geforderten Verarbeitungsprozesse.

6.1 Voraussetzungen

Der Aufbau und die Administration der Plattform wird über eine Client-Maschine vorgenommen. Im konkreten Fall wird hierzu ein Apple MacBookPro 2017 mit macOS Version 10.15.1 Beta verwendet. Auf diesem ist die Installation einer Reihe von Programmen, beziehungsweise Kommandozeileninterfaces notwendig. Über diese kann mit den serverseitigen Pendants kommuniziert werden. Dazu gehören im Einzelnen:

1. **gcloud**: zur Verwaltung von Google Cloud Platform
2. **kubectl:** zur Verwaltung eines Kubernetes-Cluster
3. **Linkerd:** zur Verwaltung des Service-Meshes
4. **mc:** zur Verwaltung des Minio ObjectStore
5. **faas-cli:** zur Verwaltung von OpenFaaS

Eine detailierte Installationsanleitungen findet sich in Anhang 9.1. Dort wird ebenfalls beschrieben, wie die Konfiguration zur Kommunikation mit den jeweiligen Serverkomponenten eingerichtet werden muss.

6.2 Implementierung der Plattform

Die serverseitige Umsetzung erfolgt auf der *Google Cloud Plattform*. Dadurch werden die in Kapitel 7.1 durchgeführten Experimente hinsichtlich der Skalierung über die in einer lokalen Entwicklungsumgebung vorhandenen Hardware-Ressourcen hinaus ermöglicht.

Bevor Services genutzt werden können, ist das Anlegen eines Projektes erforderlich. Für den Kontext dieser Arbeit wird ein Projekt mit dem Namen **geofaas** angelegt. Projekte

bilden organisatorische Einheiten von genutzten Services und Ressourcen und werden über ein zentrales Konto abgerechnet.

6.2.1 Kubernetes-Cluster

Über **gcloud** wird ein Kubernetes-Cluster mit der Google Kubernetes Engine (GKE) angelegt. Da die Größe der Nodes unveränderbar ist, muss sie zu Beginn festgelegt werden. Dazu ist die grundlegende Funktionsfähigkeit im *Scaled-In* Zustand gegenüber den entstehenden Kosten abzuwägen. Daher wird als Größe der Nodes der Google-Standard **n1-standard-2** festgelegt. Diese Maschinen verfügen über 2 CPU und 7.5 GB RAM. Der monatliche Preis pro Maschine diesen Typs beträgt aktuell 56,31 Euro. Um die Laufzeit der Netzwerkkommunikation so minimal wie möglich zu halten, wird als Standort die Region **europe-west-3a** (Frankfurt) ausgewählt. Zur Fehlertoleranz einzelner Nodes wird das Cluster mit 3 Nodes initialisiert.

Quelltext 6.1: Initialisierung des Kubernetes-Cluster auf der Google Cloud Platform

```
gcloud beta container --project "geofaas" clusters create "
    standard-cluster-1" --zone "europe-west3-a" --no-enable-basic
    -auth --cluster-version "1.14.7-gke.10" --machine-type "n1-
    standard-2" --image-type "COS" --disk-type "pd-standard" --
    disk-size "100" --metadata disable-legacy-endpoints=true --
    scopes "https://www.googleapis.com/auth/devstorage.read_only"
    ,"https://www.googleapis.com/auth/logging.write","https://www
    .googleapis.com/auth/monitoring","https://www.googleapis.com/
    auth/servicecontrol","https://www.googleapis.com/auth/service
    .management.readonly","https://www.googleapis.com/auth/trace.
    append" --num-nodes "3" --enable-stackdriver-kubernetes --
    enable-ip-alias --network "projects/geofaas/global/networks/
    default" --subnetwork "projects/geofaas/regions/europe-west3/
    subnetworks/default" --default-max-pods-per-node "110" --
    addons HorizontalPodAutoscaling,HttpLoadBalancing --enable-
    autoupgrade --enable-autorepair --no-shielded-integrity-
    monitoring
```

Über **kubectl** wird der Status der Cluster-Nodes überprüft. Wenn die lokale Konfiguration erfolgreich war, wird eine Übersicht über die Cluster-Nodes ausgegeben.

Quelltext 6.2: Auflistung der Cluster-Nodes mit kubectl

```
kubectl get nodes
NAME                                                  STATUS
gke-standard-cluster-1-default-pool-7554fdf0-1snn     Ready
gke-standard-cluster-1-default-pool-7554fdf0-mgp8     Ready
gke-standard-cluster-1-default-pool-7554fdf0-vtzc     Ready
```

Damit ist das Aufsetzen des Kubernetes-Clusters abgeschlossen. Im Weiteren wird die Installation der Komponenten im Cluster beschrieben.

6.2.2 Linkerd

Um Linkerd im Kubernetes-Cluster zu installieren, werden Kubernetes-Manifeste durch folgenden **linkerd** Befehl ausgegeben und über eine Pipe direkt an **kubectl** weitergeleitet.

Quelltext 6.3: Installation von Linkerd im Kubernetes-Cluster

```
linkerd install | kubectl apply -f -
```

Die gesamte Installation kann nun darauf überprüft werden, ob alle Komponenten vorhanden und ausführbar sind.

Quelltext 6.4: Überprüfung der Installation von Linkerd

```
linkerd check
```

Zur grafischen Verwaltung enthält die Serverkomponente von Linkerd mehrere Dashboards. Dazu zählt eine Oberfläche für Linkerd an sich, als auch Grafana, das zusätzliche Dashboards für die Visualisierung des Kubernetes-Cluster enthält. Diese können über folgenden Befehl lokal bereitgestellt werden.

Quelltext 6.5: Lokale Bereitstellung der Linkerd-Dashboards

```
linkerd dashboard
Linkerd dashboard available at: http://localhost:50750
Grafana dashboard available at: http://localhost:50750/grafana
```

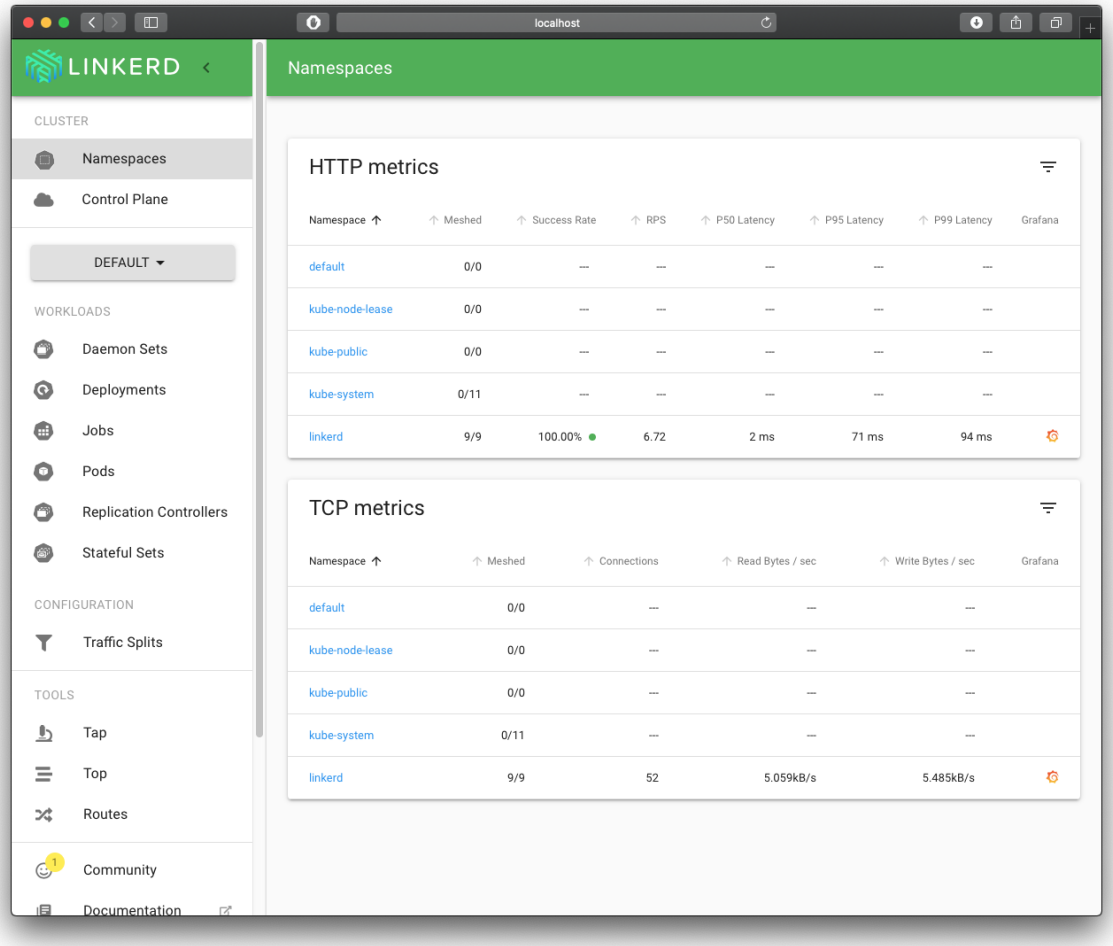

Abbildung 6.1: Linkerd Dashboard

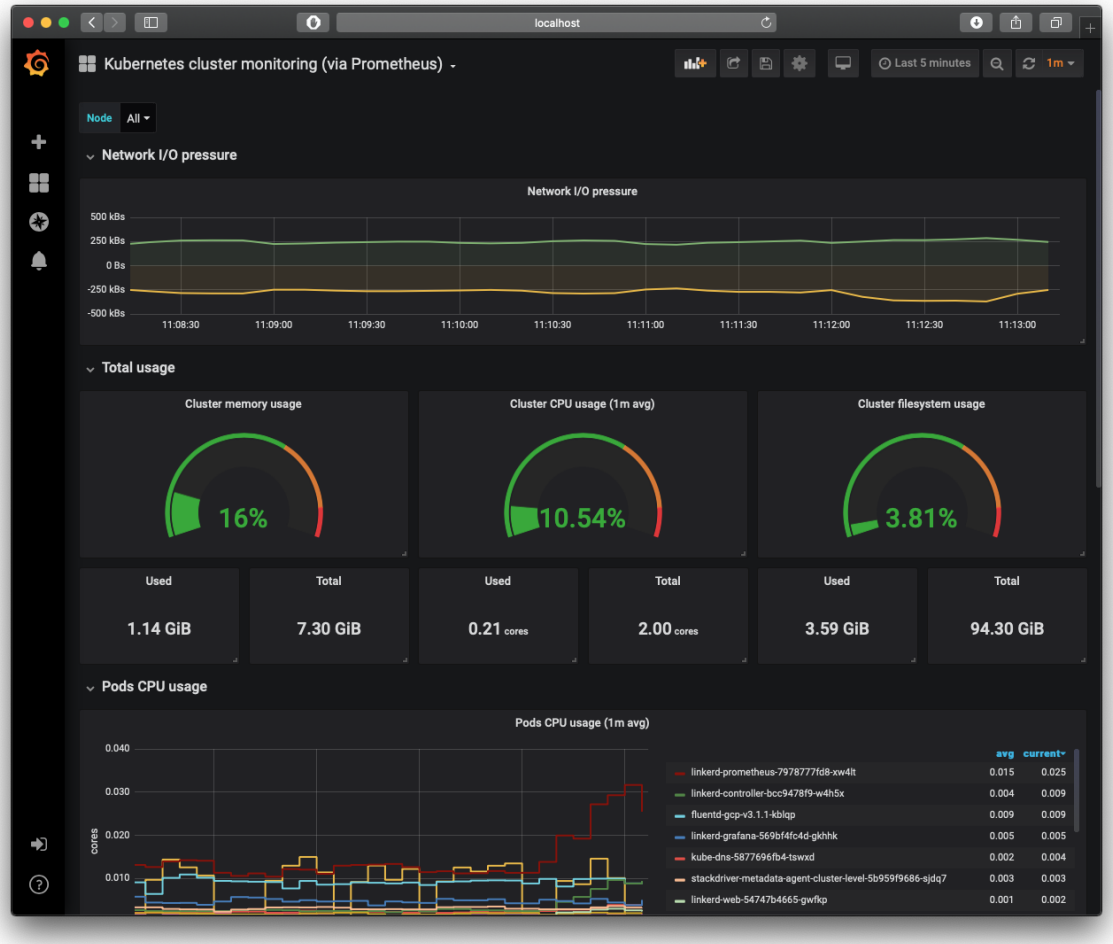

Abbildung 6.2: Grafana Dashboard

Damit ist die Installation von Linkerd abgeschlossen.

6.2.3 Minio

Die Installation von Minio im Kubernetes-Cluster umfasst mehrere Objekte, deren Konfiguration in einer gemeinsamen Manifest-Datei definiert ist. Diese kann dem Anhang 9.2.1 entnommen werden. Über **kubectl** wird die Manifest-Datei an die Kubernets-API des Clusters übergeben.

Quelltext 6.6: Installation von Minio im Kubernetes-Cluster

```
kubectl apply -f minio.yaml
```

Im Folgenden werden die einzelnen Objekte und ihre Konfiguration anhand von Auszügen der Manifest-Datei erläutert.

Zur vereinfachten Verwaltung werden alle Objekte in einen separaten Namespace gefasst. Der gesamte Netzwerkverkehr von Minio soll durch das Service Mesh Linkerd geroutet werden. Daher müssen die Minio-Pods mit Linkerd-Sidecars versehen werden. Durch die Annotation eines Namespaces wird die automatische Injizierung der Sidecars aktiviert.

Quelltext 6.7: Kubernetes-Manifest des Minio-Namespaces

```
2  kind: Namespace
3  metadata:
4    name: minio
5    annotations:
6      linkerd.io/inject: "enabled"
```

Zur Speicherung der Dateien außerhalb der Container wird ein Persistent Volume Claim (PVC) angelegt. Dieses Objekt stellt ein persistentes Dateisystem zur Verfügung. Im Fall der GKE wird im Hintergrund über die API der Google Compute Engine (GCE) ein entsprechendes Volume erzeugt. Für die beispielhafte Implementierung ist eine Speichergröße von 50 GB ausreichend. Aufgrund der wesentlich höheren Lese- und Schreibgeschwindigkeit wird ein SSD-Volume verwendet.

Quelltext 6.8: Kubernetes-Manifest des Minio-PersistentVolumeClaims

```
11 kind: PersistentVolumeClaim
12 metadata:
13   name: minio-pv-claim
14   namespace: minio
15 spec:
     ...
18   resources:
19     requests:
20       storage: 50Gi
```

Als nächstes folgt das eigentliche Deployment. Dieses verwendet das offizielle Docker-Image von Minio [Docker Hub - minio/minio, 2019]. In diesen Container wird das vorher erzeugte PVC an den Speicherort der Dateien gemountet. Aus Gründen der Vereinfachung werden die Zugangsdaten für Minio über Umgebungsvariablen des Containers festgelegt. Für einen produktiven Einsatz muss die Provisionierung von Secrets durch geeignete Mechanismen ersetzt werden.

Quelltext 6.9: Kubernetes-Manifest des Minio-Deployments

```
22 apiVersion: apps/v1
23 kind: Deployment
24 metadata:
25   name: minio
26   namespace: minio
27 spec:
28   selector:
       ...
37     spec:
38       volumes:
39       - name: data
40         persistentVolumeClaim:
41           claimName: minio-pv-claim
42       containers:
43       - name: minio
44         image: minio/minio
45         volumeMounts:
46         - name: data
47           mountPath: "/data"
48         env:
49         - name: MINIO_ACCESS_KEY
50           value: "minio"
51         - name: MINIO_SECRET_KEY
52           value: "minio123"
```

Um das Deployment innerhalb des Clusters unter einem festen DNS-Namen für andere Anwendungen nutzbar zu machen wird ein Service-Objekt vom Typ ClusterIP erzeugt. Dieses stellt sicher, dass Minio unter einem konstanten DNS-Namen ungeachtet der IP-Adresse des Pods erreichbar ist.

Quelltext 6.10: Kubernetes-Manifest des Minio-Service

```
78 apiVersion: v1
79 kind: Service
80 metadata:
81   name: minio-service
82   namespace: minio
83 spec:
84   type: ClusterIP
85   ports:
86     - port: 80
87       targetPort: 9000
88       protocol: TCP
89   selector:
90     app: minio
```

Damit ist die Installation von Minio abgeschlossen.

6.2.4 OpenFaaS

Die Installation von OpenFaaS umfasst ebenfalls mehrere Kubernetes-Objekte, weist aber eine wesentlich höhere Komplexität als Minio auf. Es werden zwei Namespaces angelegt, die alle Objekte von OpenFaas beinhalten. Der Namespace *openfaas* beinhaltet die verwaltenden Komponenten wie beispielsweise das Gateway oder die Workqueue. Der Namespace *openfaas-fn* beinhaltet hingegen die Deployments der Funktionen und wird daher analog zu Minio mit einer Annotation zur Injizierung der Linkerd-Sidecars versehen.

Quelltext 6.11: Kubernetes-Manifeste der OpenFaaS-Namespaces

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: openfaas
  labels:
    role: openfaas-system
    access: openfaas-system
    istio-injection: enabled

apiVersion: v1
kind: Namespace
metadata:
  name: openfaas-fn
  annotations:
    linkerd.io/inject: "enabled"
  labels:
    istio-injection: enabled
    role: openfaas-fn
```

Im nächsten Schritt wird ein Passwort erzeugt, das für die Authentifizierung und Authorisierung des Admin-Accounts im OpenFaaS-Gateway notwendig ist. Dieses wird zusammen mit dem Admin-Account-Benutzer als Kubernetes-Secret im Kubernetes-Cluster bereitgestellt. Von dort aus wird es vom Gateway dazu verwendet, Zugriffe über die Verwaltungsoberfläche oder **faas-cli** zu authentisieren und autorisieren.

Quelltext 6.12: Bereitstellung des OpenFaaS-Passwortes im Kubernetes-Cluster

```
PASSWORD=$(head -c 12 /dev/urandom | shasum| cut -d' ' -f1)
kubectl -n openfaas create secret generic basic-auth \
--from-literal=basic-auth-user=admin \
--from-literal=basic-auth-password="$PASSWORD"
```

Nun wird das OpenFaas-Github-Repository geclont. Dieses enthält im Verzeichnis *faas-netes* die notwendigen Kubernetes-Manifeste. Diese werden anschließend über **kubectl** der Kubernetes-Cluster-API übergeben, damit die entsprechenden Objekte erzeugt werden.

Quelltext 6.13: Installation von OpenFaaS im Kubernetes-Cluster

```
git clone https://github.com/openfaas/faas-netes
cd faas-netes & kubectl apply -f /yaml
```

Damit ist die Installation von OpenFaaS abgeschlossen.

6.3 Implementierung der Funktionen

In diesem Kapitel wird die Implementierung der beiden Funktionen Crop und Georeferenzierung beschrieben.

Zunächst wird die in Abbildung 6.3 gezeigte Verzeichnisstruktur erstellt. Diese umfasst den Quelltext der Funktionen und die Konfiguration des Deployments.

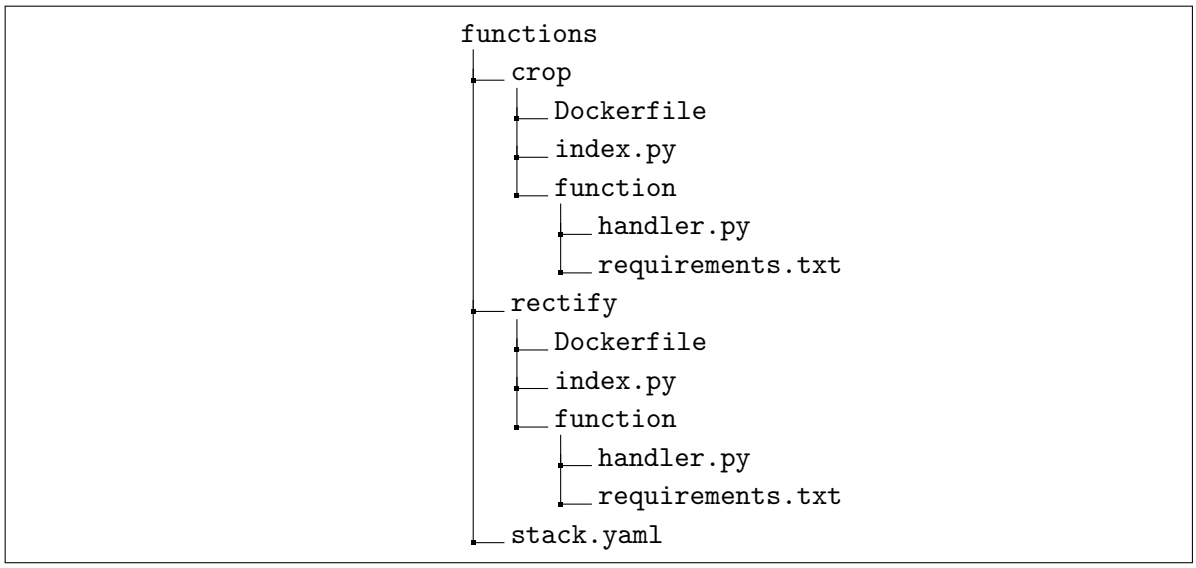

Abbildung 6.3: Verzeichnisstruktur des Sourcecodes der OpenFaaS-Funktionen

6.3.1 Konfiguration der Funktionen

Die Konfiguration der Funktionen wird in der Datei stack.yaml vorgenommen. Diese Datei wird von **faas-cli** benutzt, um das Deployment der Funktionen vorzunehmen. Dazu werden z. B. die Namen der zu erzeugenden Docker-Images definiert. Diese verwenden als Prefix den Projektnamen auf der *Google Cloud Plattform*, sodass die Images in der integrierten Container-Registry gespeichert und von dort durch das Kubernetes-Cluster lesbar sind. Zusätzlich werden die Zugangsdaten für den Minio ObjectStore über Umgebungsvariablen bereitgestellt.

Quelltext 6.14: Auszug der Deployment-Konfiguration stack.yaml

```
version: 1.0
provider:
  name: openfaas
  gateway: http://127.0.0.1:31112
functions:
  crop:
    lang: dockerfile
    handler: ./crop
    image: eu.gcr.io/geofaas/crop:latest
    environment:
      MINIO_URL: minio-service.minio.svc.cluster.local
      MINIO_BUCKET: crop
      MINIO_ACCESS_KEY: minio
      MINIO_SECRET_KEY: minio123
  rectify:
    lang: dockerfile
    handler: ./rectify
    image: eu.gcr.io/geofaas/rectify:latest
    environment:
      MINIO_URL: minio-service.minio.svc.cluster.local
      MINIO_BUCKET: rectify
      MINIO_ACCESS_KEY: minio
      MINIO_SECRET_KEY: minio123
```

Durch **faas-cli** werden die Funktionen bereitgestellt. Dies umfasst die Schritte des Build der Docker-Images, Push in die Container-Registry und des Deployments über das OpenFaaS-Gateway. Der Quelltext 6.15 führt die notwendigen Programmbefehle.

Quelltext 6.15: Deployment von OpenFaaS-Funktionen mit faas-cli

```
kubectl port-forward services/gateway -n openfaas 31112:8080
OPENFAAS_URL=http://127.0.0.1:31112
echo $PASSWORD | faas-cli login --username admin --password-stdin -
credentials saved for admin http://127.0.0.1:31112
faas-cli up
```

6.3.2 Ausführungsumgebung

OpenFaaS verwendet Docker-Container als Ausführungseinheiten der Funktionen. Damit bilden Docker-Images die Vorlagen für OpenFaaS-Funktionen. Wie in Kapitel 5.4.2

erläutert, erfolgt die Verarbeitung der Geodaten in einem mehrstufigen Verfahren. Aus diesem Grund wird die gesamte Verarbeitung in der Skriptsprache Python implementiert. Für Python existieren bereits Bibliotheken für den Minio-ObjectStore und die GDAL. GDAL erfordert zusätzlich die Installation der GDAL-Binaries. Damit ist für die Funktionen Crop und Georeferenzierung ein Docker-Image erforderlich, dass sowohl Python als auch die notwendigen GDAL-Binaries enthält. Der Quelltext 6.16 zeigt Auszüge des verwendeten Dockerfiles. Bei diesem Dockerfile handelt es sich um einen Multistage-Build, welcher die benötigten Binaries aus existierenden Images verwendet und diese mit dem Quelltext der Funktion in einem neuen Image bereitstellt.

Quelltext 6.16: Auszug Dockerfile

```
1  FROM openfaas/classic-watchdog:0.13.4 as watchdog
2
3  FROM andrejreznik/python-gdal:stable
4
5  RUN mkdir -p /home/app
6
7  COPY --from=watchdog /fwatchdog /usr/bin/fwatchdog
8  RUN chmod +x /usr/bin/fwatchdog
...
40 COPY function            function
...
47 ENV fprocess="python3 index.py"
48 ENV write_debug="true"
49 EXPOSE 8080
50
51 HEALTHCHECK --interval=3s CMD [ -e /tmp/.lock ] || exit 1
```

6.3.3 Funktion Crop

Die Funktion Crop soll aus einem vorhandenen Rasterdatensatz einen achsenparallelen Unterbereich des Bildes ausschneiden. Dazu nimmt die Funktion über einen HTTP-Post-Request ein JavaScript Object Notation (JSON)-Dokument mit den Verarbeitungsparametern entgegen. In diesem befindet sich die URL der Eingabedatei, der Name der Ausgabedatei und schließlich die Crop-Pixel-Koordinaten.

Quelltext 6.17: JSON-Dokument mit Verarbeitungsparametern für Crop

```
{
    "url":"http://minio-service.minio.svc.cluster.local/raster/scan.tif",
    "output":"1.tif",
    "parameter": {
        "x": 472,
        "y": 780,
        "w": 8648,
        "h": 10332
    }
}
```

Die Funktion verwendet diese Daten, um das Eingaberaster über GDAL aus dem Minio ObjectStore zu lesen. Mittels GDAL erfolgt anschließend auch das Croppen des Bildbereichs, sowie die Erzeugung der Ausgabedatei im Format COGTIFF. Über die Minio Bibliothek wird diese im Minio Objekt-Store gespeichert. Abschließend wird die URL der Ausgabedatei im HTTP-Response zurückgegeben. Die vollständige Implementierung kann der Datei handler.py im Anhang 9.2.2 entnommen werden.

Quelltext 6.18: Auszug relevanter Abschnitte der Datei handler.py

```
14 def handle(req):
    ...
41     crop_dataset = gdal.Translate("/vsimem/tmp.tif",
           original_dataset, format="VRT", creationOptions=["
           COMPRESS=LZW"], srcWin=[crop_x, crop_y, crop_w, crop_h])
42     gdal.SetConfigOption('COMPRESS_OVERVIEW','JPEG')
43     gdal.SetConfigOption('JPEG_QUALITY_OVERVIEW','60')
44     crop_dataset.BuildOverviews('NEAREST',[2,4,8,16,32])
       ...
61     try:
           ...
65         minio_client.put_object(BUCKET, file_output_name , io.
               BytesIO(cogtif), size)
66     except ResponseError as err:
67         print(err)
               ...
72     response_json = {}
73     response_json['result'] = 'http://minio-service.minio.svc.
           cluster.local:9000/' + bucket + '/' + file_output_name
74
75     return json.dumps(response_json)
```

6.3.4 Funktion Georeferenzierung

Die Funktion Georeferenzierung soll aus einem vorhandenen Rasterdatensatz mittels Passpunkten einen neuen rektifizierten georeferenzierten Rasterdatensatz erzeugen. Dazu nimmt die Funktion ebenfalls über einen HTTP-Post-Request ein JSON-Dokument mit den Verarbeitungsparametern entgegen. In diesem befinden sich die URL der Eingabedatei, der Name der Ausgabedatei und die Passpunkte.

Quelltext 6.19: JSON-Dokument mit Verarbeitungsparametern für Georeferenzierung

```
1  {
2      "url":"http://minio-service.minio.svc.cluster.local:9000/
           crop/1.tif",
3      "output":"1.tif",
4      "parameter": {
5          "gcps": [
6              {
7                  "srcx": 4553.203198986128,
8                  "srcy": 6241.11423702538,
9                  "dstx": 434313.886902264,
10                 "dsty": 5793159.43739078
11             },
12             {
13                 "srcx": 5257.90906525217,
                       ...
89             }
90         ]
91     }
92 }
```

Die Verarbeitung der Georeferenzierung ist nahezu identisch zur bereits vorgestellten Implementierung der Funktion Crop in Kapitel 6.3.3. Der Unterschied besteht lediglich in der Verarbeitung der gelesenen Eingabedatei durch die GDAL-Warp-Methode und Parametrisierung mit den übermittelten Passpunkten. Die Ausgabedatei wird ebenfalls im Minio ObjectStore gespeichert und die URL im HTTP-Response zurückgegeben. Die vollständige Implementierung kann der Datei handler.py im Anhang 9.2.3 entnommen werden.

Quelltext 6.20: Auszug relevanter Abschnitte der Datei handler.py

```
16 def handle(req):
    ...
35     original_dataset = gdal.Open(file_url, gdal.GA_Update)
36
37     gcp_list = []
38     for gcp in parameter['gcps']:
39         gcp_list.append(gdal.GCP(gcp['dstx'], gcp['dsty'], 0,
           gcp['srcx'], gcp['srcy']))
40     original_dataset.SetGCPs(gcp_list, original_dataset.
       GetProjection())
41     rect_dataset = gdal.Warp("/vsimem/tmp.tif", original_dataset
       , format="VRT", creationOptions=["COMPRESS=LZW"],
       polynomialOrder=2, dstSRS="EPSG:25832", dstNodata=0)
42     gdal.SetConfigOption('COMPRESS_OVERVIEW','JPEG')
43     gdal.SetConfigOption('JPEG_QUALITY_OVERVIEW','60')
44     rect_dataset.BuildOverviews('NEAREST',[2,4,8,16,32])
       ...
62     try:
         ...
66         minio_client.put_object(BUCKET, file_output_name , io.
           BytesIO(cogtif), size)
67     except ResponseError as err:
68         print(err)
           ...
73     response_json = {}
74     response_json['result'] = 'http://minio-service.minio.svc.
       cluster.local:9000/' + BUCKET + '/' + filename
75
76     return json.dumps(response_json)
```

6.4 Konzeptionelle Beschreibung der Analyseprozesse

In dieser Arbeit sollen auch die Verfahren zur Analyse von Rasterdaten (siehe Kapitel 3.1.2) betrachtet und ihre Integration in die Plattform konzeptionell beschrieben werden. Für sie liegen derzeit keine produktionsreifen Implementierungen vor, da die anzuwendenden Methodiken noch nicht abschließend evaluiert wurden. Die verwendete Technologie und generelle Funktionsweise sind hingegen bekannt.

Beide Verfahren liefern eine Liste von Koordinaten-Wahrscheinlichkeits-Paaren als Ergebnis, die z. B. im Format GeoJSON im ObjectStore gespeichert werden kann. Sie erfordern erhebliche Infrastruktur-Ressourcen und zeigen nach derzeitigem Stand lange Laufzeiten. Hinsichtlich ihrer Integrierbarkeit in die vorgestellte Plattform unterscheiden sie sich erheblich.

Markierte Punktprozesse

Die Implementierung des Verfahrens der markierten Punktprozesse führt Rechenoperationen lediglich auf der CPU aus. Somit kann es einfach als weitere OpenFaaS-Funktion in die Plattform integriert werden. Wegen der langen Laufzeit von 8 bis 24 Stunden pro Kriegsluftbild, müssen jedoch zusätzliche Aspekte betrachtet werden.

HTTP ist nicht für Requests mit hoher Laufzeit geeignet, standardmäßig werden Timeouts von wenigen Minuten verwendet. Daher sollte statt einer synchronen, eine asynchrone Verarbeitung durchgeführt werden. Idealerweise wird dies durch OpenFaaS unterstützt. Dazu muss beim Aufruf einer Funktion zusätzlich eine Callback-URL übergeben werden, an welche die Funktion das Ergebnis später über einen HTTP-Post-Request sendet. Der asynchrone Aufruf einer Funktion wird durch das OpenFaaS-Gateway mit dem HTTP-Statuscode 202 bestätigt.

Die Laufzeit des Verfahrens kann auch durch die parallele Verarbeitung von Unterbereichen des Kriegsluftbildes weiter reduziert werden. Dies ist deshalb möglich, weil das Verfahren die Topologie von Bombenkratern nicht berücksichtigt. Die Änderung einer Iterations-Konfiguration ist gewissermaßen *stateless* und kann daher isoliert erfolgen. Würde die Analyse eines Kriegsluftbildes durch die parallelisierte Analyse seiner TIFF-Kacheln in eigenständigen Funktions-Instanzen durchgeführt, sind erhebliche Geschwindigkeitszuwächse zu erwarten.

Convolutional Neuronal Networks

Weitaus größere Auswirkungen auf die Plattform-Architektur zeigt die Integration des Analyse-Verfahrens mithilfe Neuronaler Netzwerke. Auch dieses Verfahren kann als zusätzliche OpenFaaS-Funktion in die Plattform integriert werden. Das verwendete Framework *Tensorflow* benötigt eine GPU zur performanten Ausführung. Diese ist nach bisheriger Konfiguration nicht vorhanden. Weil Server meist *headless*[1] betrieben werden, sind GPU in Servern nicht weit verbreitet. Die hohe Verbreitung von Machine Learning hat dazu geführt, dass mittlerweile GPU auf vielen Public Clouds angeboten werden.

Die Integration GPU-basierter Analyseprozesse kann durch die Konfiguration des Kubernetes-Clusters umgesetzt werden. Dazu sind drei Schritte notwendig:

1. Installation von GPU in Nodes
2. Labeling der Nodes mit GPU
3. Node-Selektion-Attribut von Pods auf o.g. Label setzen

Damit verteilt der Kubernetes-Scheduler Pods mit GPU-Bedarf ausschließlich auf Nodes mit GPU.

[1] Ohne Graphical User Interface (GUI)

Eine nachträgliche Änderung von Pods, die durch OpenFaaS bereitgestellt wurden, ist nicht praktikabel. Daher kann das nodeSelector-Attribut bereits in der Deployment-Konfiguration als *constraint* durch OpenFaaS gesetzt werden (siehe Kapitel 6.3.1).

7 Evaluation

Dieses Kapitel widmet sich der Evaluation des vorgestellten Ansatzes. Dazu werden im ersten Schritt Experimente durchgeführt. Mit diesen Experimenten soll die Skalierbarkeit und ihre Auswirkungen auf den Durchsatz untersucht werden. Im zweiten Schritt wird die prototypische Implementierung den Anforderungen gegenübergestellt und die Umsetzung der Anforderungen überprüft. Zuletzt wird der Ansatz mit zwei weit verbreiteten Software-Produkten im Geodatenmanagement verglichen. Der Vergleich erfolgt anhand der in Kapitel 4.1 definierten Bewertungskriterien. Abschließend erfolgt eine Zusammenfassung.

7.1 Experimente

Der vorgestellte Ansatz und seine prototypische Implementierung werden durch drei unterschiedliche Experimente untersucht. Damit soll vornehmlich das Ziel der Verkürzung des hohen Zeitaufwandes durch parallelisierte anstelle sequentielle Verarbeitung von Geodaten überprüft werden. Darüberhinaus steht auch die Effizienz hinsichtlich des Ressourcenverbrauchs im Fokus. Folgende Fragestellungen soll anhand der Experimente beantwortet werden:

1. Wie schnell können Cluster und Funktionen skaliert werden?
2. Welchen Auswirkungen zeigt die Skalierung auf den Durchsatz?
3. Welchen Ressourcenbedarf weist die Plattform in Ruhe und unter Last auf?

7.1.1 Skalierung des Clusters

Ein wesentlicher Aspekt des vorgestellten Ansatzes ist die dynamische Anpassbarkeit der belegten Infrastruktur-Ressourcen an den tatsächlichen Bedarf. Infrastruktur-Ressourcen werden in Kubernetes durch das Hinzufügen oder Entfernen von Cluster-Nodes angepasst. Durch *Scaling-In* des Clusters können Kosten gespart werden, durch *Scaling-Out* kann der Durchsatz von Verarbeitungsprozessen erhöht werden. In welchen Situationen eine Anpassung an den tatsächlichen Bedarf sinnvoll ist, wird hauptsächlich von der Dauer für *Scaling-In* oder *Scaling-Out* bestimmt.

Dazu wird die Anzahl der Cluster-Nodes mit dem Programm **gcloud** angepasst. Das Programm terminiert, sobald die angebene Anzahl der Cluster-Nodes initialisiert und dem Cluster funktionsfähig hinzugefügt ist. Mit den beiden folgenden Experimenten wird die Dauer von *Scaling-In* und *Scaling-Out* gemessen.

Scaling-Out

Über den im Quelltext 7.1 gezeigten Programmaufruf wird das Cluster von drei auf sechs Nodes skaliert. Damit werden die Infrastruktur-Ressourcen im Kubernetes-Cluster verdoppelt. Die Ausführungszeit beträgt 1.12 min.

Quelltext 7.1: Scaling-Out des Kubernetes-Cluster mit gcloud

```
gcloud container clusters resize standard-cluster-1
--node-pool default-pool --num-nodes 6
```

Scaling-In

Über den im Quelltext 7.2 gezeigten Programmaufruf wird das Cluster von sechs auf einen Node skaliert. Dies stellt die minimal mögliche Belegung von Infrastruktur-Ressourcen dar. Die Ausführungszeit beträgt 2.13 min.

Quelltext 7.2: Scaling-In des Kubernetes-Cluster mit gcloud

```
gcloud container clusters resize standard-cluster-1
--node-pool default-pool --num-nodes 1
```

Im Vergleich benötigt das *Scaling-In* damit fast doppelt so viel Zeit, wie das *Scaling-Out*.

7.1.2 Skalierung von Funktionen

Die parallele Verarbeitung der Geodaten wird durch die horizontale Skalierung von Funktions-Instanzen ermöglicht. Mit dem Starten von vielen Funktions-Instanzen wird der Durchsatz der Plattform erheblich gesteigert. Um sämtliche bereitgestellten Funktionen performant ausführen zu können, muss die Anzahl der laufenden Funktions-Instanzen dynamisch veränderbar sein. Daher ist besonders wichtig, in welcher Zeit Funktions-Instanzen gestartet oder gestoppt werden können.

Die Docker-Images für die Funktionen wurden zuvor auf einer für das Kubernetes-Cluster zugänglichen Container-Registry bereitgestellt. Die Messungen der Experimente umfassen auch das automatische Starten und Stoppen von Sidecar-Containern für das Service-Mesh.

Mit den folgenden Experimenten wird die Geschwindigkeit von *Scaling-In* und *Scaling-Out* anhand der implementierten Transformationsprozesse untersucht. Um mögliche Einflüsse durch die Anzahl von Funktions-Instanzen oder Cluster-Nodes zu zeigen, werden einzelne Messungen für unterschiedliche Konfigurationen durchgeführt.

Scaling-Out

Bei dem Experiment des *Scaling-Out* wird der Start von einer, drei und sechs Funktions-Instanzen mit einem, drei oder sechs Cluster-Nodes gemessen. Vor jeder Messung werden die Funktionen vollständig gestoppt und damit *Scale-From-Zero* simuliert.

Tabelle 7.1: Zeitmessungen von Scaling-Out der Funktionen

		Zeit in s		
Nodes	**Instanzen** / **Funktion**	**1**	**3**	**6**
1	Crop	14	22	26
	Georeferenzierung	10	19	27
3	Crop	11	14	17
	Georeferenzierung	8	17	22
6	Crop	8	11	12
	Georeferenzierung	15	14	14

Scaling-In

Bei dem Experiment des *Scaling-In* wird der Stopp von einer, drei und sechs Funktions-Instanzen mit einem, drei und sechs Cluster-Nodes gemessen. Vor jeder Messung werden die angebenen Funktions-Instanzen gestartet und damit *Scale-To-Zero* simuliert.

Tabelle 7.2: Zeitmessungen von Scaling-In der Funktionen

		Zeit in s		
Nodes	**Instanzen** / **Funktion**	**1**	**3**	**6**
1	Crop	25	30	33
	Georeferenzierung	14	16	16
3	Crop	28	29	31
	Georeferenzierung	14	16	16
6	Crop	24	30	27
	Georeferenzierung	13	14	17

Es ist zu beobachten, dass das *Scaling-Out* wesentlich schneller als das *Scaling-In* erfolgt. Auch zu sehen ist, dass die Zeiten lediglich dann steigen, wenn mehr Funktions-Instanzen pro Cluster-Node gestartet werden.

7.1.3 Durchsatz

Beim Experiment des Durchsatzes wird die Anzahl von verarbeiteten Datensätzen pro Minute gemessen. Die Messung findet anhand der implementierten Tranformations-Funktionen statt. Analog zu den vorherigen Experimenten wird auch hier der Einfluss der unterschiedlichen Anzahl von Funktions-Instanzen und Cluster-Nodes berücksichtigt.

Für eine Messung werden jeweils zehn Funktionsaufrufe an die Plattform übermittelt. Mithilfe der gemessenen Zeit in Sekunden t wird anschließend der Durchsatz mit der Formel $Durchsatz = \frac{10\ Datensätze}{t/60\text{s}}$ berechnet.

Tabelle 7.3: Durchsatz der Funktionen

		Durchsatz in Datensätze pro min		
Nodes	**Instanzen** / **Funktion**	**1**	**3**	**6**
1	Crop	8	10	17
	Georeferenzierung	6	8	-
3	Crop	8	19	27
	Georeferenzierung	5	10	17
6	Crop	8	22	32
	Georeferenzierung	5	10	15

In den Messungen können mehrere Beobachtungen vorgenommen werden:

1. Grundsätzlich ist ein Unterschied des Durchsatzes zwischen Crop und Georeferenzierung zu erkennen. Der Unterschied ist auf komplexe Matrix-Operationen in den Transformations-Algorithmen bei der Georeferenzierung zurückzuführen. Damit entspricht der beobachtete Unterschied der Erwartung.
2. Ebenfalls erwartungsgemäß führt eine Vergrößerung des Clusters bei lediglich einer Funktions-Instanz zu keiner Steigerung des Durchsatzes. Die zusätzlichen Infrastruktur-Ressourcen können von einer Funktions-Instanz nicht genutzt werden.
3. Der Durchsatz kann durch eine Erhöhung von Funktions-Instanzen und Cluster-Nodes gesteigert werden. Relativ profitiert der Durchsatz mehr von der Skalierung der Funktions-Instanzen als von der Skalierung der Clusters. Diese Steigerung ist aber weder linear zu den Funktions-Instanzen noch zu den Cluster-Node.

4. Bei der Georeferenzierung mit einem Node und sechs Funktions-Instanzen konnte kein Durchsatz bestimmt werden. Aufgrund von konkurriendem Zugriff auf die Infrastruktur-Ressourcen des Nodes konnten die Funktions-Instanzen nicht innerhalb von 10 s ausgeführt werden und wurden durch OpenFaaS terminiert. Dieser Fehler kann dadurch verhindert werden, dass eine Funktions-Instanz nur dann bereitgestellt wird, wenn genug Ressourcen für sie zur Verfügung stehen. Dazu können die Funktionen mit Kubernetes-Labels versehen werden, sodass der Kubernetes-Scheduler diese Einschränkung berücksichtigt.

Ressourcenverbrauch

Über den Ressourcenverbrauch kann die Effizienz der Implementierung bestimmt werden. Da der Ressourcenverbrauch stark zwischen Ruhe- und Lastsituationen varriert, müssen beide Situationen betrachtet werden. Die nachfolgende Abbildung 7.1 zeigt das in Grafana dargestellte Monitoring des Ressourcenverbrauchs bei der Cluster-Größe von einem Node. In Ruhe belegt die gesamte Plattform 30 % RAM und 18 % CPU. Unter Last steigt dieser auf 66 % RAM und 83.91 % CPU. Diese Angaben beziehen sich auf die verwendeten Nodes der Größe von 7.5 GB RAM und 2 CPU.

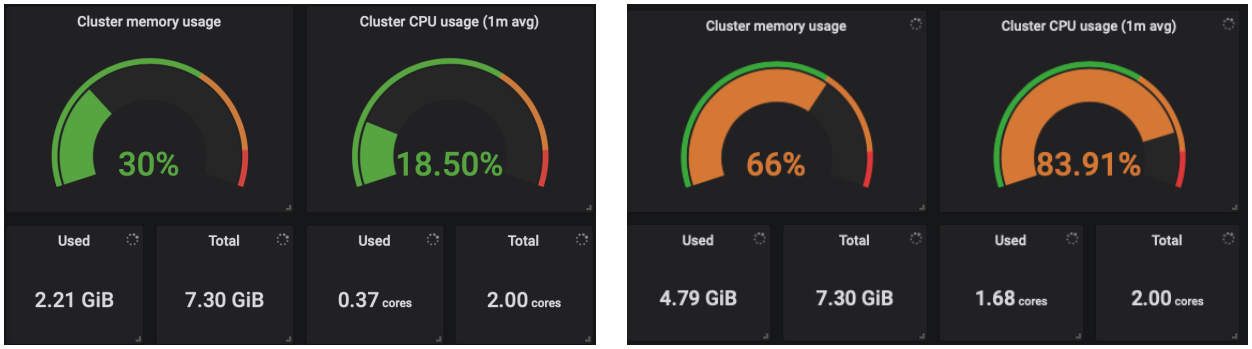

Abbildung 7.1: Ressourcenverbrauch eines Nodes in Ruhe (links) und unter Last(rechts)

Aus dem Ressourcenverbrauch unter Last lässt sich ableiten, dass die ausgeführten Funktionen im Verhältnis mehr CPU als RAM benötigen. Der vorhandene Arbeitsspeicher kann durch die Funktionen nicht vollständig ausgelastet werden. Eine effizientere Nutzung von Infrastruktur-Ressourcen ließe sich durch Cluster-Nodes mit weniger RAM oder mehr CPU realisieren.

Gleichzeitig muss auch berücksichtigt werden, dass die Verwendung von nur einem Cluster-Node die Effizienz zulasten der Ausfallsicherheit optimiert. Daher sollten mindestens drei Cluster-Nodes verwendet werden. Der Ressourcen-Verbrauch für drei Cluster-Nodes beträgt 15 % RAM und 8.75 % CPU und ist in der folgenden Abbildung 7.2 dargestellt.

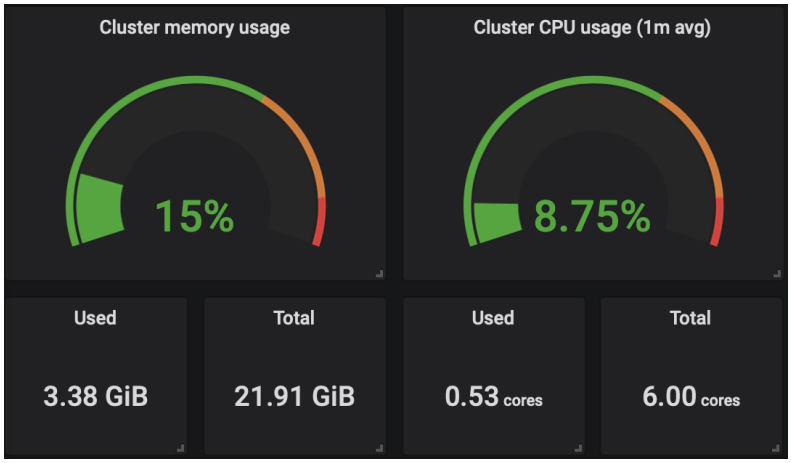

Abbildung 7.2: Ressourcenverbrauch von drei Nodes in Ruhe

7.1.4 Bewertung

Die Experimente zeigen, dass mit dem vorgestellten Ansatz die Verarbeitungszeit von Geodaten wesentlich verkürzt werden kann.

Durch die horizontale Skalierung von Funktions-Instanzen und Cluster-Nodes kann der Durchsatz um ein Vielfaches gesteigert werden. Sofern die Cluster-Nodes genügend Infrastruktur-Ressourcen zur Verfügung stellen, ist eine Skalierung der Funktionen sinnvoll. Sollten nicht ausreichend Infrastruktur-Ressourcen vorhanden sein, so muss zunächst das Cluster skaliert werden.

Die Skalierung von Funktionen ist innerhalb weniger Sekunden und die Skalierung des Clusters innerhalb weniger Minuten möglich. Dies zeigt eine hohe Elastizität und ermöglicht die kurzfristige Anpassungsfähigkeit an auftretende Lastsituationen.

Dabei haben die Experimente aber auch gezeigt, dass der Durchsatz keineswegs linear zu den ausgeführten Funktions-Instanzen oder Cluster-Nodes skaliert. Es ist zu vermuten, dass eine Optimierung der Netzwerk-Anbindung von Cluster-Nodes oder Sharding[1] der eingesetzten Datenhaltung notwendig sind.

7.2 Überprüfung der Anforderungen

Anhand der erfolgten Umsetzung und den aus den Experimenten gewonnenen Erkenntnissen wird nun die Erreichung der gestellten Anforderungen geprüft. Dazu wird entlang des Anforderungsverzeichnisses in Kapitel 3.2.4 knapp erläutert, inwiefern die jeweilige Anforderung umgesetzt werden konnte.

[1] Horizontale Partitionierung

1. **Funktionale Anforderungen**

   a) **Management von Verarbeitungsprozessen**
   Diese Anforderung ist vollständig erfüllt. Mit OpenFaas wurde neben der Funktionalität zudem ein Framework eingesetzt, welches das Management der Funktionen stark vereinfacht und für die geforderten Programmiersprachen bereits fertige Templates bietet.

   b) **Ausführung von Verarbeitungsprozessen**
   Diese Anforderung ist vollständig erfüllt. Der Aufruf von Verarbeitungsprozessen (Funktionen) erfolgt über HTTP und kann synchron oder asynchron ausgeführt werden. Die Ausführungsdauer von Verarbeitungsprozessen kann durch einen Erwartungswert begrenzt werden, nach dem ein Verarbeitungsprozess automatisch terminiert wird.

   c) **Beispielhafte Implementierung von Verarbeitungsprozessen**
   Diese Anforderung ist vollständig erfüllt. Die Transformationsprozesse wurden mithilfe von GDAL und Python auf der Plattform umgesetzt und für die Experimente verwendet. In Kapitel 6.4 wurden erforderliche Schritte zur Implementierung weiterer vornehmlich GPU-basierter Analyseprozesse beschrieben.

   d) **Datenhaltung von Eingabe- und Ausgabedaten**
   Diese Anforderung ist vollständig erfüllt. Über eine S3 basierte Schnittstelle können die Geodaten aus einem ObjectStore performant durch Verarbeitungsprozesse und GIS abgerufen werden.

   e) **Monitoring von Verarbeitungsprozessen und Datenhaltung**
   Diese Anforderung ist vollständig erfüllt. Mithilfe des Service-Meshs kann der Betrieb vollständig alle Komponenten und ihre Kommunikation überwachen. Die Injektion dieses Monitorings wird transparent und ohne zusätzlichen Entwicklungsaufwand automatisch durchgeführt.

2. **Qualitätsanforderungen**

   a) **Laufzeit von Verarbeitungsprozessen**
   Diese Anforderung ist vollständig erfüllt (siehe Kapitel 7.1).

   b) **Skalierung der Rechenkapazität**
   Diese Anforderung ist vollständig erfüllt (siehe Kapitel 7.1).

3. **Rahmenbedingungen**

   a) **Konformität mit Leitsätzen für Fachanwendungen des LGLN**

      i. **Sourcing-Strategie**
      Diese Anforderung ist vollständig erfüllt. Die eingesetzte Software besteht vollständig aus OSS.

ii. **Cloudfähigkeit**
Diese Anforderung ist vollständig erfüllt. Für die Plattform bestehen aufgrund ihrer Modularität eine Vielzahl an denkbaren Deployment-Modellen: Sie kann von lokalen Entwicklungsumgebungen, über Private- bis zu Public Cloud betrieben werden. Einzelne Module / Komponenten, wie der ObjectStore oder FaaS, können einzeln herausgelöst und durch andere Software-Produkte ersetzt werden.

b) **Konformität mit Leitsätzen für Datenhaltungen des LGLN**

i. **Wohldokumentierte Datenstrukturen**
Diese Anforderung ist vollständig erfüllt. Mit der Verwendung von COGTIFF wurde ein offenes weit verbreitetes Datenformat gefunden, dass gleichzeitig den Anforderungen an performante Datenübertragung gerecht wird.

c) **Schutzbedarf der Kriegsluftbilder**
Diese Anforderung ist teilweise erfüllt. Die notwendigen Grundlagen wurden in der prototypischen Implementierung geschaffen, um die die Schutzziele Integrität, Verfügbarkeit und Vertraulichkeit in einem Produktivbetrieb umsetzen zu können.

7.3 Vergleich domänenspezifische Plattformen

Eine weitere Eignung wird durch den Vergleich des vorgestellten Ansatzes mit existierenden Software-Produkten evaluiert. Als Vergleichsgegenstände werden *ArcGIS-Server* und *GeoServer* verwendet. Diese zeigen im Geodatenmanagement eine hohe Verbreitung und werden für vergleichbare Anwendungsfälle eingesetzt. Der Vergleich mit dem in dieser Arbeit vorgestellten Ansatz erfolgt dazu am Beispiel des Anwendungsfalles der vorliegenden Arbeit. Als Bewertungskriterien werden die für den Anwendungsfall identifizierten Bewertungskriterien aus Kapitel 4.1 herangezogen.

7.3.1 Vergleichsgegenstände

Zunächst werden die zusätzlichen Vergleichsgegenstände *ArcGIS-Server* und *GeoServer* vorgestellt.

ArcGIS-Server

ArcGIS-Server ist ein proprietäres Produkt der Firma ESRI Inc. . Er ist ein elementarer Bestandteil von ArcGIS-Enterprise basierten Geodaten-Infrastrukturen. Zu ArcGIS-Enterprise gehört auch das namensstiftende und weit verbreitete Desktop-GIS *ArcGIS*. Die nachfolgende Abbildung 7.3 zeigt den Aufbau von ArcGIS-Enterprise.

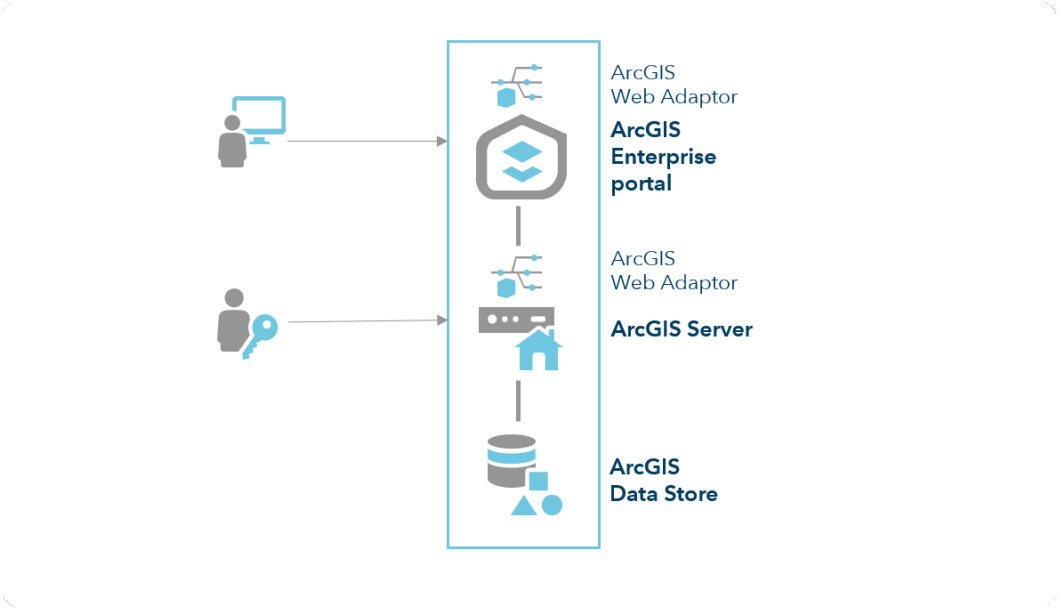

Abbildung 7.3: ArcGIS-Enterprise Stack [ESRI, 2019]

ArcGIS-Server kann auf Microsoft Windows 64-bit Betriebssystemen installiert werden. Dort müssen zudem Microsoft .NET und Core XML Services vorhanden sein. Darüberhinaus enthält die Installation einen Python-Interpreter mit den Bibliotheken NumPy und Matplotlib. Die Installation erfordert mindestens 8 GB RAM und 10 GB Speicherplatz. Auch die Installation in virtualisierten Umgebungen (VMware vSphere und Microsoft Hyper-V) wird unterstützt.

Geodaten werden über offene OGC- und proprietäre ArcGIS-Services bereitgestellt. Zur Verarbeitung von Geodaten können neben den in ArcGIS-Server enthaltenen Algorithmen auch eigene Programmen mittels Python implementiert werden.

Als Datenhaltung für die Geodaten kann eine Vielzahl von unterschiedlichen Systemen eingebunden werden. Dazu zählen weit verbreitete Datenbanken, wie z. B. Microsoft SQL Server, PostgreSQL oder IBM Db2. *ArcGIS-Server* verwendet zur Speicherung von Geodaten in RDBMS das sogenannte *ArcSDE*, ein komplexes relationales Schema über das Vektor- und Rasterdaten abgebildet werden.

GeoServer

GeoServer ist ein Open-Source Projekt der OSGeo [OSGeo, 2019] unter *GPL*-Lizenz. Mit ihm können Geodaten über Open Web Services (OWS)-Services bereitgestellt werden. Er bietet die native Unterstützung für viele offene Formate. Es existieren viele Plugins, durch welche die Funktionalitäten von *GeoServer* stark erweitert werden können.

Der *GeoServer* ist in Java implementiert und ist entweder als Java Archive (JAR) oder Java Web Archive (WAR) als Download verfügbar. Die Implementierung verwendet weit verbreitete offene Bibliotheken, wie z.B. GeoTools[2] oder JTS.

Auch *GeoServer* bietet eine große Anzahl von unterstützten Datenhaltungen. Neben klassischen RDBMS, wie z. B. PostgreSQL, Microsoft SQL Server und Oracle, können auch NoSQL-Datenbanken wie MongoDB eingebunden werden.

7.3.2 Vergleiche

Im folgenden Abschnitt werden die Vergleichsgegenstände nach den in Kapitel 4.1 vorgestellten Kriterien bewertet. Für jede Kategorie der Bewertungskriterien werden bis zu drei Punkte vergeben. Der am Besten geeignete Kandidat erhält die maximale Anzahl von Punkten, der am Schlechtesten geeignete Kandidaten erhält die minimale Anzahl. Wenn keine Ordnung getroffen werden kann, werden gleich viele Punkte vergeben. Wenn ein Kriterium nicht erfüllt wird, werden null Punkte vergeben.

Funktionalität

Sowohl ArcGIS-Server als auch GeoServer können um eigene Implementierungen von Verarbeitungsprozessen erweitert werden. Die Implementierung in ArcGIS kann jedoch ausschließlich über das sogenannte ArcPy erfolgen, was ein auf Python 2 basierendes Framework ist. GeoServer ist in Java programmiert und kann auch in Java geschriebene Erweiterungen ausführen. Zusätzlich existiert jedoch ein durch die Community entwickeltes Scripting-Modul, mit dem die Ausführung von Python, JavaScript, Groovy, Beanshell und Ruby ermöglicht wird. Der vorgestellte Ansatz dieser Arbeit ermöglicht die Implementierung in beliebigen Programmiersprachen und bietet demnach die größe Freiheit.

ArcGIS-Server: 1 Punkt | GeoServer: 2 Punkt | Vorgestellter Ansatz: 3 Punkte

Effizienz

ArcGIS-Server muss auf einem dedizierten Host installiert werden. Damit gehen die in Kapitel 4.3.2 dargestellten Nachteile von fest zugewiesenen Infrastruktur-Ressourcen einher. Der GeoServer weißt eine etwas größere Flexibilität auf, da für ihn mehrere Möglichkeiten des Deployments bestehen: Eine Möglichkeit ist die Installation innerhalb eines Servlet-Containers in einer VM. In diesem Fall wäre auch der GeoServer von den zuvor erwähnten Nachteilen von VM betroffen. Es ist aber auch möglich, dass der Servlet-Container innerhalb eines Docker-Containers deployt wird und das gleiche

[2]Open-Source Java GIS Toolkit

flexible Verhalten wie die Container des vorgestellten Ansatzes zeigen. Hinsichtlich des Ressourcenverbrauchs weisen die Vergleichsgegenstände keine Unterschiede auf. Die Skalierung von VM / Bare-Metal ist wesentlich schwerfälliger als die von Containern. Insofern kann ArcGIS nur auf den letzten Platz verortet werden. GeoServer und der vorgestellte Ansatz differenzieren sich insbesondere durch den Umfang der deploybaren Einheiten: Durch die monolithische Architektur kann der GeoServer nur vollständig skaliert werden. Der vorgestellte Ansatz dieser Arbeit ermöglicht hingegen die Skalierung von einzelnen Modulen. Aufgrund der wesentlich kleineren Größe der skalierbaren Module ist demnach die Skalierung bei dem vorgestellten Ansatz dieser Arbeit auch wesentlich schneller. Der maximale Durchsatz kann aufgrund mangelnder empirischer Untersuchungen der Vergleichsgegenstände nicht in die Bewertung aufgenommen werden.

Für die Kategorie Effizienz ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 1 Punkt | GeoServer: 2 Punkte | Vorgestellter Ansatz: 3 Punkte

Kompatibilität

Sowohl VM als auch Container sind geeignete Mechanismen, um Koexistenz von Komponenten zu ermöglichen. Demnach bestehen diesbezüglich keine Unterschiede zwischen den Vergleichsgegenständen. ArcGIS-Server und GeoServer unterstützen wesentlich mehr OWS-Services als der vorgestellte Ansatz dieser Arbeit. Damit weisen ArcGIS-Server und GeoServer eine höhere Interoperabilität auf. ArcGIS-Server bietet darüber hinaus eine breitere Unterstützung von proprietären Datenformaten, wie z. B. ECW.

Für die Kategorie Kompatibilität ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 3 Punkte | GeoServer: 2 Punkte | Vorgestellter Ansatz: 1 Punkt

Gebrauchstauglichkeit

Die Bewertung dieser Kategorien erfolgt anhand der Leistungskennzahlen der GitHub-Projektseiten. Da ArcGIS-Server keine OSS ist und keine freien Projektseiten existieren, erhält ArcGIS in dieser Kategorie null Punkte. Da GeoServer im Docker-Container auch von Kubernetes orchestriert werden kann und auch mit Minio kompatibel ist, scheint der Vergleich des OpenFaaS-Projektes mit GeoServer sinnvoll.

Tabelle 7.4: Vergleich der Leistungskennzahlen

Projekt	**Commits**	**Contributors**	**Stars**
GeoServer	20533	218	1800
OpenFaaS (vg. Ansatz)	1775	131	16300

Für die Kategorie Gebrauchstauglichkeit ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 0 Punkte | GeoServer: 3 Punkte | Vorgestellter Ansatz: 2 Punkte

Zuverlässigkeit

Self-Healing und Wiederherstellbarkeit sind zwei der maßgeblichen Kriterien für die Kategorie Zuverlässigkeit. Diese sind anhand der anhand der Vergleichsgegenstände nur schwer zu beurteilen, da Self-Healing und Wiederherstellbarkeit vor allem durch die Orchestrierung bestimmt werden. In der Dokumentation zu ArcGIS-Server und GeoServer lassen sich keine integrierten Komponenten dazu finden. Aufschlussreicher ist hingegen die Dokumentation zur horizontalen Skalierbarkeit. Sowohl ArcGIS-Server als auch GeoServer sind als vollständig duplizierte Instanzen horizontal skalierbar. Sie benötigen jedoch eine zusätzliche Synchronisation von Daten und Konfigurationen aller beteiligten Instanzen und einen externen Load-Balancer voraus. Bei der Skalierung von ArcGIS-Server ist die Lizenz zu beachten. Damit weisen beide Vergleichsgegenstände wesentliche Nachteile gegenüber dem vorgestellten Ansatz dieser Arbeit auf. Dieser ermöglicht die horizontale Skalierbarkeit von einzelnen Modulen und zeigt damit eine erheblich größere Flexibilität. Gleichzeitig geht damit einher, dass durch ein fehlerhaftes Modul nicht die vollständige Plattform wiederhergestellt werden muss.

Für die Kategorie Zuverlässigkeit ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 1 Punkt | GeoServer: 3 Punkte | Vorgestellter Ansatz: 2 Punkte

IT-Sicherheit

Zur Beurteilung der IT-Sicherheit wird überprüft, ob die Vergleichsgegenstände die Prinzipien der Dekomposition und sicherbaren Interfaces verfolgen. Dabei zeigt sich, dass der vorgestellte Ansatz diese Prinzipien verfolgt, aber aufgrund seiner Microservice-Architektur eine Vielzahl von Modulen und Kommunikationsbeziehungen aufweist. Demnach scheint eine Absicherung des Java-Servlet basierten Geoservers weitaus einfacher. Die Absicherung von Servlet-Containern, wie z. B. Tomcat oder Glassfish, ist gut untersucht. ArcGIS-Server weist eine ähnliche monolithische und damit einfach sicherbare Architektur wie GeoServer auf. Er verwendet jedoch einen eigenen proprietären Webserver, sodass auch hier lediglich eine Sicherung durch eigenes Personal über Intrusion Prevention System (IPS) oder **ips!** (**ips!**) vorgenommen werden kann.

Für die Kategorie IT-Sicherheit ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 1 Punkt | GeoServer: 3 Punkt | Vorgestellter Ansatz: 2 Punkte

Wartbarkeit

Die Wiederverwendbarkeit des vorgestellten Ansatzes ist sehr hoch, da auf weit verbreiteter Software basiert und lediglich funktionsspezifische Implementierungen enthält. GeoServer weist auch eine hohe Wiederverwenbarkeit auf und ist unter der sehr offenen *GPL Version 2.0* Lizenz veröffentlicht. Diese ermöglicht auch die Nutzung in proprietären Produkten. Negativ zur beurteilen ist hingegen die monolithische Architektur des GeoServers, sodass Erweiterungen lediglich in andere GeoServer-Instanzen portiert werden können. Bei der Veränderbarkeit zur Laufzeit weist der vorgestellte Ansatz auch erhebliche Vorteile gegenüber den Vergleichsgegenständen auf. Statt einem vollständigen Neustart verwendet er Rolling Update. GeoServer und ArcGIS-Server müssen bei Anpassungen, z.B. bei der Vergrößerung von Infrastruktur-Ressourcen, neugestartet werden. Besonders negativ ist ArcGIS-Server hinsichtlich Analysierbarkeit und Testbarkeit zu beurteilen. Im Gegensatz zu den beiden anderern Vergleichsgegenständen, deren Quelltext offen zur Verfügung steht, sind jegliche Fehler ausschließlich über kostenpflichtige Support-Verträge mit der Entwicklungsfirma ESRI Inc. zu lösen.

Für die Kategorie Wartbarkeit ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 1 Punkt | GeoServer: 2 Punkte | Vorgestellter Ansatz: 3 Punkte

Portabilität

ArcGIS-Server ist ein proprietäres Produkt und wird ausschließlich mit kostenpflichtigen Lizenzen vertrieben. GeoServer hingegen steht, wie bereits beschrieben, unter der sehr offenen *GPL Version 2.0* Lizenz. Daraus folgen Implikationen, die die Portabilität maßgeblich beeinflussen. ESRI bietet mit ArcGIS-Enterprise auch ein SaaS an, dass ArcGIS-Server enthält. Außerhalb von ESRI gibt es jedoch so gut wie keine Services für ArcGIS-Server auf Public Clouds. Ähnlich ist die Situation für GeoServer. Dieser wird aber zumindest im Microsoft Azure Marketplatz angeboten[Microsoft, 2019]. Wie bereits erwähnt, benötigt ArcGIS-Server ein Microsoft Windows Betriebssystem und 8 GB RAM. Damit ist es nicht möglich, ihn in weit verbreiteten auf macOS oder Linux basierten Entwicklungsumgebungen zu verwenden. Der GeoServer kann hingegen plattformunabhängig mit weit weniger Infrastruktur-Ressourcen betrieben werden. Aufgrund seines monolithischen Architektur ist er nicht so fein granular aufsetzbar, wie der in dieser Arbeit vorgestellte Ansatz.

Für die Kategorie Portabilität ergibt sich insgesamt folgende Bewertung:

ArcGIS-Server: 1 Punkt | GeoServer: 2 Punkte | Vorgestellter Ansatz: 3 Punkte

7.3.3 Bewertung

Aus dem Vergleich geht hervor, dass sich die Vergleichsgegenstände hinsichtlich der Bewertungskriterien sehr stark voneinander differenzieren.

Die ermittelten Ergebnisse der Vergleiche sind in der folgenden Abbildung 7.4 zusammenfassend dargestellt. In einem Radar-Diagramm ist jede Kategorie der Bewertungskriterien als eine Dimension abgebildet. Auf den Dimensionen wurden von innen nach außen die vergebenen Punkte abgetragen und die Produktzugehörigkeit durch farbliche Trennung der Graphen dargestellt. Der ArcGIS-Server ist rot, der GeoServer ist grün und der vorgestellte Ansatz dieser Arbeit ist blau dargestellt.

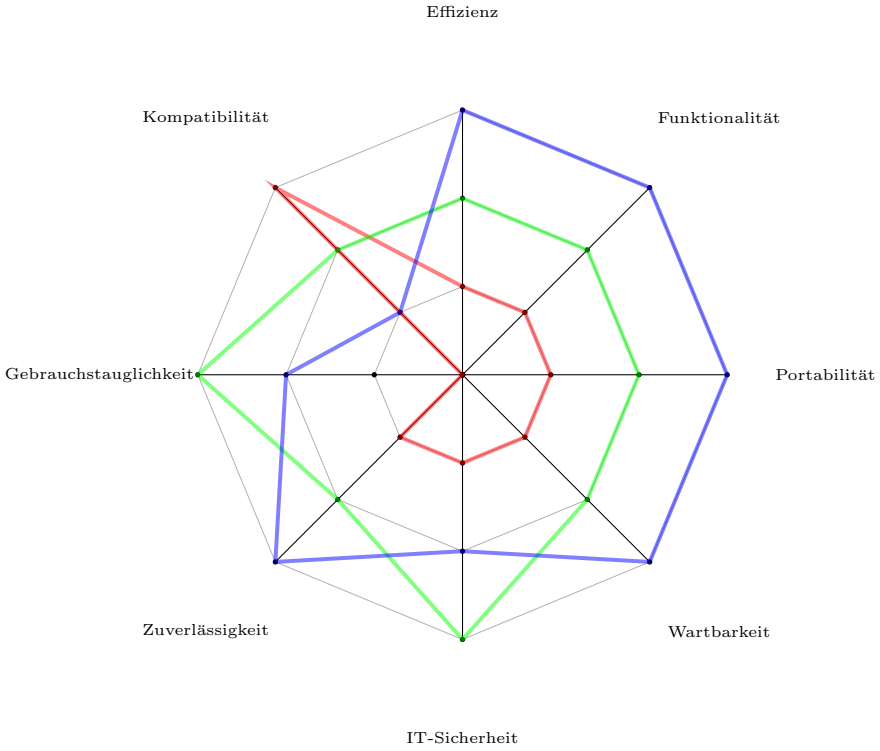

Abbildung 7.4: Bewertungen von ArcGIS-Server, GeoServer und dem vorgestellten Ansatz

ArcGIS-Server weist in den Kategorien Effizienz, Funktionalität, Portabilität, Wartbarkeit, IT-Sicherheit, Zuverlässigkeit jeweils die geringste Eignung auf. In der Kategorie Gebrauchstauglichkeit konnte zudem keine Wertung vorgenommen werden. Lediglich in der Kategorie Kompatibilität ist ArcGIS-Server das bestgeeigneteste Produkt. Diese Bewertung ist vornehmlich auf den Closed-Source Charakter von ArcGIS-Server zurückzuführen. Dieses verhinderte unter den gegebenen Bewertungskriterien eine höhere Bewertung.

GeoServer ist in den Kategorien Kompatibilität, Effizienz, Funktionalität, Portabilität, Wartbarkeit und Zuverlässigkeit jeweils das zweit geeigneteste Produkt. In den Kategorien Gebrauchstauglichkeit und IT-Sicherheit stellt er das bestgeeignetste Produkt dar. Die durchweg gute und teilweise sehr gute Bewertung ist auf den Einsatz von weit verbreiteter OSS zurückzuführen.

Der in dieser Arbeit vorgestellte Ansatz weist in den Kategorien Effizienz, Funktionalität, Portabilität, Wartbarkeit und Zuverlässigkeit die beste Eignung auf. Dies ist vor allem auf die vorgenommene Trennung von Architektur- und Applikations-Architektur zurückzuführen. In den Kategorien IT-Sicherheit und Gebrauchstauglichkeit liegt er hinter dem GeoServer. Die Kompatibilität wurde im Vergleich am geringsten eingestuft und ist durch die ausschließliche Implementierung der Transformationsprozesse zu erklären.

7.4 Zusammenfassung

Die Evaluation des in dieser Arbeiten vorgestellten Ansatzes wurde durch drei Methoden vorgenommen. Einerseits wurden Experimente durchgeführt, mit denen Skalierung, Durchsatz und Ressourcenverbrauch in unterschiedlichen Konfigurationen überprüft wurden. Mit den erlangten Erkenntnissen wurde dann geprüft, ob die Anforderungen des Anwendungsfalles erfüllt werden konnten. Abschließend erfolgte ein Vergleich mit zwei weit verbreiteten Software-Produkten, die für vergleichbare Anwendungsfälle eingesetzt werden.

Durch die Experimente konnte belegt werden, dass mit dem vorgestellten Ansatz die Verarbeitungszeit von Geodaten wesentlich verkürzt werden kann. Dabei wurde erkannt, dass der Durchsatz nicht linear zu eingesetzen Infrastruktur-Ressourcen und Funktions-Instanzen skaliert.

Aus der Überprüfung der Anforderungen ergab sich, dass nahezu sämtliche Anforderungen vollständig erfüllt werden konnten.

Der Vergleich mit anderen Plattform bestätigt die hohe Effizienz, Funktionalität, Portabilität und Wartbarkeit des Ansatzes. Er zeigt aber auch Defizite hinsichtlich der Kompatibilität auf. Diese Defizite können auf den eingeschränkten Betrachtungshorizont der vorliegenden Arbeit zurückgeführt werden. Durch die Integration weiterer Komponenten, wie z. B. GeoServer als Bereitstellung von OGC-Services lassen sich diese Defizite leicht ausbessern.

8 Fazit

Mit der vorliegenden Arbeit wurde eine Konzeption und prototypische Umsetzung einer skalierbaren Plattform für die laufzeit- und ressourcenoptimierte Verarbeitung von großen Rasterdatenbeständen vorgestellt. Der gewählte Ansatz löst das Problem eines hohen Zeitaufwandes für die Verarbeitung von Geodaten durch eine flexible Architektur, die die freiee Skalierung und parallelisierte Verarbeitung unterstützt.

Basierend auf Kubernetes wurde eine Plattform entworfen, die gleichermaßen die horizontale Skalierung der zugrundeliegenden Infrastruktur als auch von bereitgestellten Applikationen mithilfe von Docker-Containern ermöglicht. Eine dieser bereitgestellten Applikationen ist das zur Verarbeitung der Geodaten eingesetzte OpenFaaS, das die polyglote Implementierungen von Funktionen durch Ausführung in Containern ermöglicht. Eine andere Applikation ist der ObjectStore Minio, in dem Geodaten über eine weit verbreteite S3-Schnittstelle gespeichert werden. Mit dem Cloud-Optimized GeoTIFF wurde eine Möglichkeit der Optimierung von Rasterdaten für den performanten Lesezugriff gefunden.

Die eingesetzten Technologien orientieren sich an marktüblichen Technologien und Architekturen des Cloud Computing. Mit dem vorgestellen Ansatz konnte eine erhebliche Steigerung des Durchsatzes erzielt werden. Das zeigt, dass die performante Verarbeitung von Geodaten in der Cloud möglich ist. Gleichwohl wurde ausschießlich Open Source Software eingesetzt, was eine große Flexibilität hinsichtlich von Deployment- und Service-Modelle ermöglicht.

Geodatenhaltende Stellen sind jetzt gefordert, aus den Anforderungen der Digitalisierung, die *richtigen* Anforderungen abzuleiten. Wie mit der vorliegenden Arbeit gezeigt wurde, bestimmt die Auswahl von geeigneten Software-Qualitätskriterien maßgeblich die Flexibilität und Anpassbarkeit an künftige Anforderungen.

Niemals galten bessere Rahmenbedingungen, hinsichtlich verfügbarer Technologie, engagierten Open Source Communities und darauf aufbauenden Geschäftsmodellen, um unsere Gesellschaft durch Digitalisierung nachhaltig weiter zu entwickeln, als jetzt[1].

[1] Stand: 15.12.2019

8.1 Ausblick

Der in dieser Arbeit vorgestellte Ansatz wurde am Beispiel des Managementservice für historische Kriegsluftbilder des KBD Niedersachsens erarbeitet. Diese Fachanwendung würde durch eine produktive Implementierung von einer wesentlich vereinfachten Verfahrensarchitektur und konsolidierten Infrastruktur profitieren.

Bereits zum Zeitpunkt der Fertigstellung dieser Arbeit ergeben sich zudem mögliche Handlungsfelder für eine Weiterentwicklung des vorgestellten Ansatzes:

1. **Integration in GIS**
   Die Integration von FaaS und ObjectStores in GIS ist derzeit auf den lesenden Zugriff auf die Geodaten beschränkt. Durch eine tiefere Integration, z. B. als Toolbox, UI-Plugin, oder OWS-Service, könnte die performante Verarbeitung und zentrale Datenhaltung besser in Desktop- oder Web-GIS genutzt werden.
2. **Steigerung der Storage-Performance**
   Während der Experimente konnte gezeigt werden, dass die Skalierung nicht linear mit den ausgeführten Verarbeitungsprozessen skaliert. Dies ist vermutlich auf die Netzwerk- und Storage-Anbindung des ObjectStores zurückzuführen. Daher sollte das Speicherkonzept auf mögliche Verbesserungen hin untersucht werden.
3. **Komposition von Funktionen**
   Bislang ist lediglich der synchrone bzw. asynchrone Aufruf einer Funktion möglich. Eine größere Wiederverwendbarkeit einzelner Funktionen könnte durch Dekomposition der Implementierung ermöglicht werden. Benötigt werden würde ein Kompositions-Mechanismus, der Funktionen dazu miteinander verknüpft und Kontroll- und Datenflüsse abbildet. Einen Ansatz bildet die Arbeit des FaaS-Flow-Projektes [FaaS-Flow, 2019].

Darüberhinaus existieren im Geodaten-Umfeld eine Vielzahl von praktischen Anwendungsfällen, in denen der vorgestellte Ansatz eine Anwendung finden könnte. Einige Beispiele dafür sind:

1. Klassifikation von Fernerkundungsdaten für Landnutzung und -bedeckung
2. Ableitung von 3D-Modellen aus Schrägluftbildern
3. Erzeugung von Vektor-, und Raster-Tiles zur Visualierung in Karten-Clients
4. Text-, Zeichenerkennung auf historischen Vermessungsrissen

Literaturverzeichnis

[Aldus Corporation, 1992] Aldus Corporation (1992). TIFF Spezifikation. https://www.itu.int/itudoc/itu-t/com16/tiff-fx/docs/tiff6.pdf. (Accessed on 12/15/2019).

[Bitkom Research, 2019] Bitkom Research (2019). Cloud-Monitor 2019. https://www.bitkom.org/sites/default/files/2019-06/bitkom_kpmg_pk_charts_cloud_monitor_18_06_2019.pdf. (Accessed on 09/29/2019).

[Booch, 1994] Booch, G. (1994). *Object-oriented Analysis and Design with Applications*. Benjamin/Cummings series in object-oriented software engineering. Benjamin/Cummings Publishing Company.

[Bornkessel, 2019] Bornkessel, P. (2019). Alle 11 Minuten verliebt sich ein Microservice in Linkerd. https://www.heise.de/developer/artikel/Alle-11-Minuten-verliebt-sich-ein-Microservice-in-Linkerd-4511406.html. (Accessed on 11/24/2019).

[Bundesministerium des Inneren, 2008] Bundesministerium des Inneren (2008). Saga 4.0. https://www.cio.bund.de/SharedDocs/Publikationen/DE/Architekturen-und-Standards/SAGA/saga_4_0_download.pdf?__blob=publicationFile. (Accessed on 09/29/2019).

[Bundesministerium des Inneren, 2017] Bundesministerium des Inneren (2017). It-strategie für die bundesverwaltung 2017-2021. https://www.cio.bund.de/SharedDocs/Publikationen/DE/Strategische-Themen/it_strategie_der_bundesverwaltung_download.pdf?__blob=publicationFile. (Accessed on 09/29/2019).

[Burns et al., 2018] Burns, B., Hightower, K., and Beda, J. (2018). *Kubernetes: Eine kompakte Einführung*. dpunkt.verlag.

[Clermont, 2019] Clermont, D. (2019). Überwachte Detektion von Bombenkratern in historischen Luftbildern mittels Convolutional Neural Networks. https://www.ipi.uni-hannover.de/fileadmin/institut/pdf/Abschlussarbeiten/2019_MA_Dominic_Clermont.pdf.

[Day, 1995] Day, J. (1995). The (un)revised osi reference model. *SIGCOMM Comput. Commun. Rev.*, 25(5):39–55.

[de Lange, 2013] de Lange, N. (2013). *Geoinformatik: in Theorie und Praxis*. Springer Berlin Heidelberg.

[DigitalOcean, 2019] DigitalOcean (2019). Openfaas hosting — digitalocean marketplace 1-click app. `https://marketplace.digitalocean.com/apps/openfaas`. (Accessed on 12/01/2019).

[Docker Hub - minio/minio, 2019] Docker Hub - minio/minio (2019). Docker Hub - minio/minio. `https://hub.docker.com/r/minio/minio/`. (Accessed on 10/25/2019).

[Docker Inc., 2019] Docker Inc. (2019). Docker Documentation. `https://docs.docker.com/engine/reference/commandline/cli/`. (Accessed on 09/30/2019).

[ESRI, 2019] ESRI (2019). Was ist ArcGIS Enterprise? `https://enterprise.arcgis.com/de/get-started/latest/windows/what-is-arcgis-enterprise-.htm`. (Accessed on 12/14/2019).

[FaaS-Flow, 2019] FaaS-Flow (2019). Function Composition for OpenFaaS. `https://github.com/s8sg/faas-flow`. (Accessed on 12/15/2019).

[Foote and Yoder, 2000] Foote, B. and Yoder, J. W. (2000). Big Ball of Mud. In Harrison, N., Foote, B., and Rohnert, H., editors, *Pattern Languages of Program Design*, volume 4, pages 654–692.

[Gilbert and Lynch, 2002] Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59.

[Github, 2019] Github (2019). Production-Grade Container Scheduling and Management. `https://github.com/kubernetes/kubernetes`. (Accessed on 09/29/2019).

[Handelsblatt, 2015] Handelsblatt (2015). Öffentlicher Sektor: Behörden in der Cloud. `https://www.handelsblatt.com/technik/it-internet/oeffentlicher-sektor-behoerden-in-der-cloud/11249882.html?ticket=ST-8102432-a2EjSXtc4cvSdidhxemo-ap2`. (Accessed on 09/29/2019).

[Harms and Yamartino, 2010] Harms, R. and Yamartino, M. (2010). The economics of the cloud. *Microsoft whitepaper, Microsoft Corporation*, 3:157.

[ISO/IEC, 2001] ISO/IEC (2001). *ISO/IEC 9126. Software engineering and Product quality*. ISO/IEC.

[ISO/IEC, 2010] ISO/IEC (2010). ISO/IEC 25010 System and software quality models. Technical report.

[Kruse, 2017] Kruse (2017). Automatische Detektion von Bombenkratern in Kriegsluftbildern mittels markierter Punktprozesse. `https://www.dgpf.de/src/tagung/jt2017/proceedings/proceedings/papers/24_DGPF2017_Kruse_et_al.pdf`. (Accessed on 12/16/2019).

[Linkerd, 2019] Linkerd (2019). Architecture — linkerd. `https://linkerd.io/2/reference/architecture/`. (Accessed on 12/02/2019).

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD.

[Microsoft, 2009] Microsoft (2009). The STRIDE Threat Model. `https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN`. (Accessed on 12/04/2019).

[Microsoft, 2019] Microsoft (2019). Einführung in Azure Kubernetes Service. `https://docs.microsoft.com/de-de/azure/aks/intro-kubernetes`. (Accessed on 09/29/2019).

[Microsoft, 2019] Microsoft (2019). Geoserver. `https://azuremarketplace.microsoft.com/en/marketplace/apps/apps-4-rent.geoserver-windows-server-2016?tab=Overview`. (Accessed on 12/15/2019).

[OpenFaaS Project, 2019] OpenFaaS Project (2019). Gateway - openfaas. `https://docs.openfaas.com/architecture/gateway/`. (Accessed on 12/01/2019).

[OSGeo, 2019] OSGeo (2019). Geoserver. `http://geoserver.org/`. (Accessed on 12/14/2019).

[Ritter, Ruth, 1995] Ritter, Ruth (1995). GeoTIFF Format Specification. `http://download.osgeo.org/geotiff/spec/geotiff.rtf`. (Accessed on 11/25/2019).

[Rouault et. al., 2019] Rouault et. al. (2019). Cloud Optimized GeoTIFF Specification. `https://github.com/cogeotiff/cog-spec/blob/master/spec.md`. (Accessed on 11/27/2019).

[Samipillai, 2017] Samipillai, S. (2017). How containerd compares to runC. `https://stackoverflow.com/questions/41645665/how-containerd-compares-to-runc`. (Accessed on 09/29/2019).

[Small et. al., 2019] Small et. al. (2019). Kubernetes Final Report. `"https://github.com/kubernetes/community/blob/master/wg-security-audit/findings/Kubernetes%20Final%20Report.pdf`. (Accessed on 09/29/2019).

[Taylor et al., 2009] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing.

[Warmerdam et. al., 2019] Warmerdam et. al. (2019). GDAL documentation, Cloud Optimized GeoTIFF. `https://gdal.org/drivers/raster/cog.html`. (Accessed on 11/27/2019).

9 Anhang

Der Anhang kann dem beiliegenden Datenträger entnommen werden. Dieser enthält den Quelltext zur Erstellung des Prototypen und folgende Dokumente:

9.1 Installationsanleitungen für Verwaltungsprogramme

9.1.1 Installationsanleitung gcloud

Schnellstart für macOS

Auf dieser Seite erfahren Sie, wie Sie das Google Cloud SDK installieren und initialisieren sowie **`gcloud`**-Hauptbefehle über die Kommandozeile ausführen.

Hinweis: Wenn sich Ihre Umgebung hinter einem Proxy bzw. einer Firewall befindet, finden Sie auf der Seite mit den Proxyeinstellungen (https://cloud.google.com/sdk/docs/proxy-settings?hl=de) weitere Informationen zur Installation.

Vorbereitung

1. Erstellen Sie ein Google Cloud Platform-Projekt (https://console.cloud.google.com/cloud-resource-manager?hl=de), falls Sie noch keines haben.
2. Python 2.7 muss auf Ihrem System installiert sein:

   ```
   python -V
   ```

   ★ **Hinweis**: Ab Version 206.0.0 des Cloud SDK bietet die **`gcloud`**-Kommandozeile experimentelle Unterstützung für die Ausführung mit einem Interpreter für Python 3.4 und höher. Führen Sie **`gcloud topic startup`** (https://cloud.google.com/sdk/gcloud/reference/topic/startup?hl=de) aus, um Details zu Ausschlüssen und weitere Informationen zur Konfiguration des Python-Interpreters zu erhalten. Für alle anderen Cloud SDK-Tools ist weiterhin ein Interpreter für Python 2.7 erforderlich.

3. Laden Sie die geeignete Archivdatei für Ihr Betriebssystem herunter. Auf den meisten Rechnern wird das 64-Bit-Paket ausgeführt.

Plattform	Paket	Größe	SHA256-Prüfsumme
macOS 64-Bit (x86_64)	google-cloud-sdk-245.0.0-darwin-x86_64.tar.gz (https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-245.0.0-darwin-x86_64.tar.gz?hl=de)	19,6 MB	78cbe75ebf17599217d4a6cb81897897faa72d6e5e0c515a8532c8827de59af7

macOS 32-Bit (x86)	google-cloud-sdk-245.0.0-darwin-x86.tar.gz (https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-245.0.0-darwin-x86.tar.gz?hl=de)	19,6 MB	3ca3575f19a21a589a75d2ed3c2a62af7d1be12482ad4cf3efba87e071977a61

4. Entpacken Sie das Archiv in einem Verzeichnis Ihres Dateisystems, vorzugsweise im Basisverzeichnis. Unter MacOS ist dies durch Öffnen der heruntergeladenen Archivdatei **`.tar.gz`** am gewünschten Speicherort möglich.
5. Optional: Wenn der **`gcloud`**-Befehl nicht richtig ausgeführt wird, prüfen Sie, ob **`$PATH`** korrekt angegeben wurde. Fügen Sie dem Pfad mithilfe des Installationsskripts Cloud SDK-Tools hinzu. Sie können während der Installation auch die Vervollständigung von Befehlen für die **`bash`**-Shell und die Erfassung von Nutzungsstatistiken (https://cloud.google.com/sdk/usage-statistics?hl=de) aktivieren. Führen Sie das Skript mit diesem Befehl aus:

   ```
   ./google-cloud-sdk/install.sh
   ```

   Starten Sie dann das Terminal neu, damit die Änderungen wirksam werden.

   Sie können das Cloud SDK nach dem Extrahieren des heruntergeladenen Archivs auch ausführen, indem Sie die zugehörigen ausführbaren Dateien über den vollständigen Pfad aufrufen.

SDK initialisieren

Mit dem Befehl **`gcloud init`** führen Sie mehrere gängige Einrichtungsaufgaben für das SDK aus. Diese umfassen die Autorisierung der SDK-Tools für den Zugriff auf die Google Cloud Platform anhand der Anmeldedaten Ihres Nutzerkontos und die Einrichtung der SDK-Standardkonfiguration.

So initialisieren Sie das SDK:

1. Führen Sie in der Eingabeaufforderung folgenden Befehl aus:

   ```
   gcloud init
   ```

   Hinweis: Wenn der Befehl keinen Webbrowser starten soll, verwenden Sie stattdessen **`gcloud init --console-only`**. Erstellen Sie für eine nicht interaktive Autorisierung

ohne Webbrowser in der Google Cloud Platform Console (https://console.cloud.google.com?hl=de) ein Dienstkonto mit den entsprechenden Bereichen und verwenden Sie **`gcloud auth activate-service-account`** mit der entsprechenden JSON-Schlüsseldatei.

2. Stimmen Sie der Anmeldung mit Ihrem Google-Nutzerkonto zu:

```
To continue, you must log in. Would you like to log in (Y/n)? Y
```

3. Melden Sie sich bei entsprechender Aufforderung über den Browser bei Ihrem Google-Nutzerkonto an. Klicken Sie auf **Zulassen**, um die Berechtigung für den Zugriff auf Google Cloud Platform-Ressourcen zu gewähren.
4. Wählen Sie in der Eingabeaufforderung aus der Liste der Cloud Platform-Projekte, für die Sie die Berechtigungen **Inhaber**, **Bearbeiter** oder **Betrachter** haben, ein Projekt aus:

```
Pick cloud project to use:
 [1] [my-project-1]
 [2] [my-project-2]
 ...
 Please enter your numeric choice:
```

Wenn Sie nur ein Projekt haben, wählt **`gcloud init`** dieses für Sie aus.

5. Wenn Sie die Google Compute Engine API aktiviert haben, können Sie mit **`gcloud init`** jetzt eine Compute Engine-Standardzone auswählen:

```
Which compute zone would you like to use as project default?
 [1] [asia-east1-a]
 [2] [asia-east1-b]
 ...
 [14] Do not use default zone
 Please enter your numeric choice:
```

`gcloud init` bestätigt Ihnen, dass die Einrichtungsschritte erfolgreich abgeschlossen wurden:

```
gcloud has now been configured!
You can use [gcloud config] to change more gcloud settings.

Your active configuration is: [default]
```

gcloud-Hauptbefehle ausführen

Führen Sie die folgenden **gcloud**-Befehle aus, um Informationen zu Ihrer SDK-Installation anzuzeigen:

1. Auflisten der Konten, deren Anmeldedaten auf dem lokalen System gespeichert sind:

   ```
   gcloud auth list
   ```

   gcloud zeigt eine Liste von Konten mit Anmeldedaten an:

   ```
          Credentialed Accounts
   ACTIVE             ACCOUNT
   *                  example-user-1@gmail.com
                      example-user-2@gmail.com
   ```

2. Auflisten der Attribute in der aktiven SDK-Konfiguration:

   ```
   gcloud config list
   ```

   gcloud zeigt die Liste der Attribute an:

   ```
   [core]
   account = example-user-1@gmail.com
   disable_usage_reporting = False
   project = example-project
   ```

3. Anzeigen der Informationen zu Ihrer Cloud SDK-Installation und der aktiven SDK-Konfiguration:

   ```
   gcloud info
   ```

   gcloud zeigt eine Zusammenfassung der Informationen zu Ihrer Cloud SDK-Installation an. Hierzu gehören Informationen zu Ihrem System, zu den installierten SDK-Komponenten, zum aktiven Nutzerkonto und aktuellen Projekt sowie zu den Attributen in der aktiven SDK-Konfiguration.

4. Anzeigen von Informationen zu **gcloud**-Befehlen und anderen Themen über die Befehlszeile:

   ```
   gcloud help
   ```

   So zeigen Sie beispielsweise die Hilfe für **gcloud compute instances create** an:

   ```
   gcloud help compute instances create
   ```

`gcloud` zeigt ein Hilfethema mit einer Beschreibung des Befehls, einer Liste der Flags und Argumente für den Befehl sowie Verwendungsbeispiele an.

Weitere Informationen

- In der Anleitung zum `gcloud`-Tool (https://cloud.google.com/sdk/gcloud/?hl=de) finden Sie eine Übersicht über das `gcloud`-Befehlszeilentool sowie eine kurze Einführung in wichtige Konzepte, Befehlskonventionen und nützliche Tipps.
- In der Referenz zu `gcloud` (https://cloud.google.com/sdk/gcloud/reference?hl=de) erhalten Sie ausführliche Informationen zu jedem `gcloud`-Befehl, einschließlich Beschreibungen, Flags und Beispielen, mit denen Sie verschiedene Aufgaben in der Google Cloud Platform ausführen können.
- Weitere Komponenten wie die App Engine-Emulatoren oder `kubectl` können Sie mit dem Cloud SDK-Komponentenmanager (https://cloud.google.com/sdk/gcloud/guide/managing-components?hl=de) installieren.

Zuletzt aktualisiert: Juni 24, 2019

9.1.2 Installationsanleitung kubectl

Aufgaben

Installieren und konfigurieren von kubectl

Verwenden Sie das Kubernetes Befehlszeilenprogramm, kubectl, um Anwendungen auf Kubernetes bereitzustellen und zu verwalten. Mit kubectl können Sie Clusterressourcen überprüfen, Komponenten erstellen, löschen und aktualisieren; Ihren neuen Cluster betrachten; und Beispielanwendungen aufrufen.

Bevor Sie beginnen

Sie müssen eine kubectl-Version verwenden, die innerhalb eines geringfügigen Versionsunterschieds zur Version Ihres Clusters liegt. Ein v1.2-Client sollte beispielsweise mit einem v1.1, v1.2 und v1.3-Master arbeiten. Die Verwendung der neuesten Version von kubectl verhindert unvorhergesehene Probleme.

ectl installieren

Nachfolgend finden Sie einige Methoden zur Installation von kubectl.

Installieren der kubectl Anwendung mithilfe der systemeigenen Paketverwaltung

Ubuntu, Debian oder HypriotOS | CentOS, RHEL oder Fedora

```
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -
sudo apt-get update
sudo apt-get install -y kubectl
```

Installation mit snap auf Ubuntu

Wenn Sie Ubuntu oder eine der anderen Linux-Distributionen verwenden, die den snap Paketmanager unterstützen, können Sie kubectl als snap-Anwendung installieren.

1. Wechseln Sie zum Snap-Benutzer und führen Sie den Installationsbefehl aus:

   ```
   sudo snap install kubectl --classic
   ```

2. Testen Sie, ob die installierte Version ausreichend aktuell ist:

   ```
   kubectl version
   ```

Installation mit Homebrew auf macOS

Wenn Sie mit macOS arbeiten und den Homebrew Paketmanager verwenden, können Sie kubectl mit Homebrew installieren.

1. Führen Sie den Installationsbefehl aus:

```
brew install kubernetes-cli
```

2. Testen Sie, ob die installierte Version ausreichend aktuell ist:

```
kubectl version
```

Installation mit Macports auf macOS

Wenn Sie mit macOS arbeiten und den Macports Paketmanager verwenden, können Sie kubectl mit Macports installieren.

1. Führen Sie den Installationsbefehl aus:

```
sudo port selfupdate
sudo port install kubectl
```

2. Testen Sie, ob die installierte Version ausreichend aktuell ist:

```
kubectl version
```

Installation mit PowerShell von PSGallery

Wenn Sie mit Windows arbeiten und den Powershell Gallery Paketmanager verwenden, können Sie kubectl mit Powershell installieren und aktualisieren.

1. Führen Sie die Installationsbefehle aus (stellen Sie sicher, dass eine `DownloadLocation` gegeben wird):

   ```
   Install-Script -Name install-kubectl -Scope CurrentUser -Force
   install-kubectl.ps1 [-DownloadLocation <path>]
   ```

   > **Hinweis:** Wenn Sie keine `DownloadLocation` angeben, wird `kubectl` im temporären Verzeichnis des Benutzers installiert.

   Das Installationsprogramm erstellt `$HOME/.kube` und weist es an, eine Konfigurationsdatei zu erstellen

2. Testen Sie, ob die installierte Version ausreichend aktuell ist:

   ```
   kubectl version
   ```

   > **Hinweis:** Die Aktualisierung der Installation erfolgt durch erneutes Ausführen der beiden in Schritt 1 aufgelisteten Befehle.

Installation auf Windows mit Chocolatey oder scoop

Um kubectl unter Windows zu installieren, können Sie entweder den Paketmanager Chocolatey oder das Befehlszeilen-Installationsprogramm scoop verwenden.

choco | scoop

```
choco install kubernetes-cli
```

1. Testen Sie, ob die installierte Version ausreichend aktuell ist:

```
kubectl version
```

2. Navigieren Sie zu Ihrem Heimatverzeichnis:

```
cd %USERPROFILE%
```

3. Erstellen Sie das `.kube` -Verzeichnis:

```
mkdir .kube
```

4. Wechseln Sie in das soeben erstellte `.kube` -Verzeichnis:

```
cd .kube
```

5. Konfigurieren Sie kubectl für die Verwendung eines Remote-Kubernetes-Clusters:

```
New-Item config -type file
```

> **Hinweis:** Bearbeiten Sie die Konfigurationsdatei mit einem Texteditor Ihrer Wahl, z.B. Notepad.

Download als Teil des Google Cloud SDK herunter

Sie können kubectl als Teil des Google Cloud SDK installieren.

1. Installieren Sie das Google Cloud SDK.
2. Führen Sie den `kubectl` -Installationsbefehl aus:

```
gcloud components install kubectl
```

3. Testen Sie, ob die installierte Version ausreichend aktuell ist:

```
kubectl version
```

Installation der kubectl Anwendung mit curl

macOS | Linux | Windows

1. Laden Sie die neueste Version herunter:

   ```
   curl -LO https://storage.googleapis.com/kubernetes-release/release/$(cur
   ```

   Um eine bestimmte Version herunterzuladen, ersetzen Sie den Befehlsteil `$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)` mit der jeweiligen Version.

   Um beispielsweise die Version v1.16.0 auf macOS herunterzuladen, verwenden Sie den folgenden Befehl:

   ```
   curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.16
   ```

2. Machen Sie die kubectl-Binärdatei ausführbar.

   ```
   chmod +x ./kubectl
   ```

3. Verschieben Sie die Binärdatei in Ihren PATH.

   ```
   sudo mv ./kubectl /usr/local/bin/kubectl
   ```

kubectl konfigurieren

kubectl einen Kubernetes-Cluster findet und darauf zugreifen kann, benötigt es eine kubeconfig Datei. Diese wird automatisch erstellt, wenn Sie einen Cluster mit kube-up.sh erstellen oder einen Minikube-Cluster erfolgreich implementieren. Weitere Informationen zum Erstellen von Clustern finden Sie in den Anleitungen für die ersten Schritte. Wenn Sie Zugriff auf einen Cluster benötigen, den Sie nicht erstellt haben, lesen Sie die Cluster-Zugriff freigeben Dokumentation. Die kubectl-Konfiguration befindet sich standardmäßig unter `~/.kube/config` .

Überprüfen der kubectl-Konfiguration

Überprüfen Sie, ob kubectl ordnungsgemäß konfiguriert ist, indem Sie den Clusterstatus abrufen:

```
kubectl cluster-info
```

Wenn Sie eine URL-Antwort sehen, ist kubectl korrekt für den Zugriff auf Ihren Cluster konfiguriert.

Wenn eine Meldung ähnlich der folgenden angezeigt wird, ist kubectl nicht richtig konfiguriert oder kann keine Verbindung zu einem Kubernetes-Cluster herstellen.

```
The connection to the server <server-name:port> was refused - did you specify t
```

Wenn Sie beispielsweise vorhaben, einen Kubernetes-Cluster auf Ihrem Laptop (lokal) auszuführen, müssen Sie zunächst ein Tool wie minikube installieren und anschließend die oben genannten Befehle erneut ausführen.

Wenn kubectl cluster-info die URL-Antwort zurückgibt, Sie jedoch nicht auf Ihren Cluster zugreifen können, verwenden Sie Folgendes, um zu überprüfen, ob er ordnungsgemäß konfiguriert ist:

```
kubectl cluster-info dump
```

Aktivieren der automatischen Autovervollständigung der Shell

kubectl bietet Autocompletion-Unterstützung für Bash und Zsh, was Ihnen viel Tipparbeit erspart!

Im Folgenden werden die Verfahren zum Einrichten der automatischen Vervollständigung für einschließlich der Unterschiede zwische x und macOS) und Zsh beschrieben.

Bash on Linux | Bash auf macOS | Zsh

Einführung

Das kubectl-Vervollständigungsskript für Bash kann mit dem Befehl `kubectl completion bash` generiert werden. Durch das Sourcing des Vervollständigungsskripts in Ihrer Shell wird die automatische Vervollständigung von kubectl ermöglicht.

Das Fertigstellungsskript benötigt jedoch **bash-completion**. Dies bedeutet, dass Sie diese Software zuerst installieren müssen (Sie können testen, ob Sie bereits bash-completion installiert haben, indem Sie `type _init_completion` ausführen).

Installation von bash-completion

bash-completion wird von vielen Paketmanagern bereitgestellt (siehe hier). Sie können es mittels `apt-get install bash-completion` oder `yum install bash-completion` , usw.

Die obigen Befehle erstellen `/usr/share/bash-completion/bash_completion` ,Dies ist das Hauptskript für die Bash-Vollendung. Abhängig von Ihrem Paketmanager müssen Sie diese Datei manuell in Ihre `~ / .bashrc` -Datei eingeben.

Um dies herauszufinden, laden Sie Ihre Shell erneut und führen Sie `type _init_completion` aus. Wenn der Befehl erfolgreich ist, ist bereits alles vorbereitet. Andernfalls fügen Sie der `~/.bashrc` -Datei Folgendes hinzu:

```
source /usr/share/bash-completion/bash_completion
```

Laden Sie Ihre Shell erneut und vergewissern Sie sich, dass bash-completion korrekt installiert ist, indem Sie folgendes eingeben: `type _init_completion` .

Aktivieren der automatische Vervollständigung von kubectl

Sie müssen nun sicherstellen, dass das kubectl-Abschlussskript in allen Ihren Shell-
:ungen verwendet wird. Es gibt zwei M eiten, dies zu tun:

- Fügen Sie das Vervollständigungsskript Ihrer `~ /.bashrc` -Datei hinzu:

```
echo 'source <(kubectl completion bash)' >>~/.bashrc
```

- Fügen Sie das Vervollständigungsskript zum Verzeichnis `/etc/bash_completion.d` hinzu:

```
kubectl completion bash >/etc/bash_completion.d/kubectl
```

> **Hinweis:** bash-completion bezieht alle Verfollständigungsskripte aus `/etc/bash_completion.d` .

Beide Ansätze sind gleichwertig. Nach dem erneuten Laden der Shell sollte kubectl autocompletion funktionieren.

Nächste Schritte

Erfahren Sie, wie Sie Ihre Anwendung starten und verfügbar machen.

Feedback

War diese Seite hilfreich?

Ja Nein

Problem berichten Diese Seite bearbeiten

Letzte Änderung am October 12, 2019 at 9:50 AM PST durch [fix] Fix typo (#16136) (Seitenverlauf)

H e

Blog

Partner

Community

Fallstudien

Contribute

9.1.3 Installationsanleitung des mc Programms

Search documents...

MinIO Client Complete Guide slack channel 5834

MinIO Client (mc) provides a modern alternative to UNIX commands like ls, cat, cp, mirror, diff etc. It supports filesystems and Amazon S3 compatible cloud storage service (AWS Signature v2 and v4).

Copy

```
ls       list buckets and objects
tree     list buckets and objects in a tree format
mb       make a bucket
rb       remove a bucket
cat      display object contents
head     display first 'n' lines of an object
pipe     stream STDIN to an object
share    generate URL for temporary access to an object
cp       copy objects
mirror   synchronize objects to a remote site
find     search for objects
sql      run sql queries on objects
stat     stat contents of objects
diff     list differences in object name, size, and date between buckets
rm       remove objects
event    manage object notifications
watch    watch for object events
policy   manage anonymous access to objects
admin    manage MinIO servers
session  manage saved sessions for cp command
config   manage mc configuration file
update   check for a new software update
version  print version info
```

1. Download MinIO Client

Docker Stable

Copy

```
docker pull minio/mc
docker run minio/mc ls play
```

Docker Edge

Talk to the community

Copy

```
docker pull minio/mc:edge
docker run minio/mc:edge ls play
```

Note: Above examples run `mc` against MinIO *play* environment by default. To run `mc` against other S3 compatible servers, start the container this way:

Copy

```
docker run -it --entrypoint=/bin/sh minio/mc
```

then use the `mc config` command.

Homebrew (macOS)

Install mc packages using Homebrew

Copy

```
brew install minio/stable/mc
mc --help
```

Binary Download (GNU/Linux)

Platform	Architecture	URL
GNU/Linux	64-bit Intel	https://dl.min.io/client/mc/release/linux-amd64/mc
	64-bit PPC	https://dl.min.io/client/mc/release/linux-ppc64le/mc

Copy

```
chmod +x mc
./mc --help
```

Binary Download (Microsoft Windows)

Platform	Architecture	URL
Microsoft Windows	64-bit Intel	https://dl.min.io/client/mc/release/windows-amd64/mc.exe

Talk to the community

Copy

```
mc.exe --help
```

Install from Source

Source installation is intended only for developers and advanced users. `mc update` command does not support update notifications for source based installations. Please download official releases from https://min.io/download/#minio-client.

If you do not have a working Golang environment, please follow How to install Golang.

Copy

```
go get -d github.com/minio/mc
cd ${GOPATH}/src/github.com/minio/mc
make
```

2. Run MinIO Client

GNU/Linux

Copy

```
chmod +x mc
./mc --help
```

macOS

Copy

```
chmod 755 mc
./mc --help
```

Microsoft Windows

Copy

```
mc.exe --help
```

3. Add a Cloud Storage Service

Note: If you are planning to use `mc` only on POSIX compatible filesystems, you may skip this step and proceed to **Step 4**.

Talk to the community

To add one or more Amazon S3 compatible hosts, please follow the instructions below. `mc` stores all its configuration information in `~/.mc/config.json` file.

Usage

```
mc config host add <ALIAS> <YOUR-S3-ENDPOINT> <YOUR-ACCESS-KEY> <YOUR-SECRET-
```

Copy

Alias is simply a short name to your cloud storage service. S3 end-point, access and secret keys are supplied by your cloud storage provider. API signature is an optional argument. By default, it is set to "S3v4".

Example - MinIO Cloud Storage

MinIO server displays URL, access and secret keys.

```
mc config host add minio http://192.168.1.51 BKIKJAA5BMMU2RHO6IBB V7f1CwQqAcw
```

Copy

Example - Amazon S3 Cloud Storage

Get your AccessKeyID and SecretAccessKey by following AWS Credentials Guide.

```
mc config host add s3 https://s3.amazonaws.com BKIKJAA5BMMU2RHO6IBB V7f1CwQqA
```

Copy

Example - Google Cloud Storage

Get your AccessKeyID and SecretAccessKey by following Google Credentials Guide

```
mc config host add gcs  https://storage.googleapis.com BKIKJAA5BMMU2RHO6IBB V
```

Copy

Specify host configuration through environment variable

```
export MC_HOST_<alias>=https://<Access Key>:<Secret Key>@<YOUR-S3-ENDPOINT>
```

Copy

Talk to the community

Example:

```
export MC_HOST_myalias=https://Q3AM3UQ867SPQQA43P2F:zuf+tfteSlswRu7BJ86wekitnifILbZam
mc ls myalias
```

4. Test Your Setup

`mc` is pre-configured with https://play.min.io, aliased as "play". It is a hosted MinIO server for testing and development purpose. To test Amazon S3, simply replace "play" with "s3" or the alias you used at the time of setup.

Example:

List all buckets from https://play.min.io

```
mc ls play
[2016-03-22 19:47:48 PDT]     0B my-bucketname/
[2016-03-22 22:01:07 PDT]     0B mytestbucket/
[2016-03-22 20:04:39 PDT]     0B mybucketname/
[2016-01-28 17:23:11 PST]     0B newbucket/
[2016-03-20 09:08:36 PDT]     0B s3git-test/
```

5. Everyday Use

You may add shell aliases to override your common Unix tools.

```
alias ls='mc ls'
alias cp='mc cp'
alias cat='mc cat'
alias mkdir='mc mb'
alias pipe='mc pipe'
alias find='mc find'
alias tree='mc tree'
```

6. Global Options

Option [--debug]

Debug option enables debug output to console.

Example: Display verbose debug output for `ls` command.

Copy

```
mc --debug ls play
mc: <DEBUG> GET / HTTP/1.1
Host: play.min.io
User-Agent: MinIO (darwin; amd64) minio-go/1.0.1 mc/2016-04-01T00:22:11Z
Authorization: AWS4-HMAC-SHA256 Credential=**REDACTED**/20160408/us-east-1/s3/aws4_re
Expect: 100-continue
X-Amz-Content-Sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b85
X-Amz-Date: 20160408T145236Z
Accept-Encoding: gzip

mc: <DEBUG> HTTP/1.1 200 OK
Transfer-Encoding: chunked
Accept-Ranges: bytes
Content-Type: text/xml; charset=utf-8
Date: Fri, 08 Apr 2016 14:54:55 GMT
Server: MinIO/DEVELOPMENT.2016-04-07T18-53-27Z (linux; amd64)
Vary: Origin
X-Amz-Request-Id: HP30I0W2U49BDBIO

mc: <DEBUG> Response Time:  1.220112837s

[...]

[2016-04-08 03:56:14 IST]     0B albums/
[2016-04-04 16:11:45 IST]     0B backup/
[2016-04-01 20:10:53 IST]     0B deebucket/
[2016-03-28 21:53:49 IST]     0B guestbucket/
```

Option [--json]

JSON option enables parseable output in JSON lines format.

Example: List all buckets from MinIO play service.

Copy

```
mc --json ls play
{"status":"success","type":"folder","lastModified":"2016-04-08T03:56:14.577+05:30","s
{"status":"success","type":"folder","lastModified":"2016-04-04T16:11:45.349+05:30","s
{"status":"success","type":"folder","lastModified":"2016-04-01T20:10:53.941+05:30","s
{"status":"success","type":"folder","lastModified":"2016-03-28T21:53:49.217+05:30","s
```

Option [--no-color]

Talk to the community

This option disables the color theme. It is useful for dumb terminals.

Option [--quiet]

Quiet option suppress chatty console output.

Option [--config-dir]

Use this option to set a custom config path.

Option [--insecure]

Skip SSL certificate verification.

7. Commands

ls - List buckets and objects	**tree** - List buckets and objects in a tree format	**mb** - Make a bucket
cp - Copy objects	**rb** - Remove a bucket	**pipe** - Pipe to an object
share - Share access	**rm** - Remove objects	**find** - Find files and objects
diff - Diff buckets	**mirror** - Mirror buckets	**session** - Manage saved sessions
config - Manage config file	**policy** - Set public policy on bucket or prefix	**event** - Manage events on your buckets
update - Manage software updates	**watch** - Watch for events	**stat** - Stat contents of objects and folders
head - Display first 'n' lines of an object	**version** - Show version	
	sql - Run sql queries on objects	

Command `ls` - List Objects

`ls` command lists files, buckets and objects. Use `--incomplete` flag to list partially copied content.

```
USAGE:
   mc ls [FLAGS] TARGET [TARGET ...]

FLAGS:
  --recursive, -r               list recursively
  --incomplete, -I              list incomplete uploads
  --help, -h                    show help
```

Example: List all buckets on https://play.min.io.

```
mc ls play
[2016-04-08 03:56:14 IST]     0B albums/
[2016-04-04 16:11:45 IST]     0B backup/
[2016-04-01 20:10:53 IST]     0B deebucket/
[2016-03-28 21:53:49 IST]     0B guestbucket/
[2016-04-08 20:58:18 IST]     0B mybucket/
```

Command `tree` - List buckets and directories in a tree format

`tree` command lists buckets and directories in a tree format. Use `--files` flag to include files/objects in listing.

```
USAGE:
   mc tree [FLAGS] TARGET [TARGET ...]

FLAGS:
  --help, -h                    show help
  --files, -f                   include files in tree
  --depth, -d                   set the maximum depth of the tree
```

Example: List all buckets on play/test-bucket in a tree format.

```
mc tree play/test-bucket
play/test-bucket/
├─ dir_a
├─ dir_b
│  └─ dir_bb
└─ dir_x
   └─ dir_xx
```

Command `mb` - Make a Bucket

`mb` command creates a new bucket on an object storage. On a filesystem, it behaves like `mkdir -p` command. Bucket is equivalent of a drive or mount point in filesystems and should not be treated as folders. MinIO does not place any limits on the number of buckets created per user.
On Amazon S3, each account is limited to 100 buckets. Please refer to Buckets Restrictions and Limitations on S3 for more information.

Copy

```
USAGE:
   mc mb [FLAGS] TARGET [TARGET...]

FLAGS:
  --region value                specify bucket region; defaults to 'us-east-1' (defau
  --ignore-existing, -p         ignore if bucket/directory already exists
  --help, -h                    show help
```

Example: Create a new bucket named "mybucket" on https://play.min.io.

Copy

```
mc mb play/mybucket
Bucket created successfully ‘play/mybucket’.
```

Example: Create a new bucket named "mybucket" on https://s3.amazonaws.com.

Copy

```
mc mb s3/mybucket --region=us-west-1
Bucket created successfully ‘s3/mybucket’.
```

Command `rb` - Remove a Bucket

`rb` command removes a bucket and all its contents on an object storage. On a filesystem, it behaves like `rmdir` command.

Note that when a bucket is removed all policies associated with the bucket will also be removed. If you would like to just
empty the objects in a bucket use `rm` command

Talk to the community

```
USAGE:
   mc rb [FLAGS] TARGET [TARGET...]

FLAGS:
  --force                       allow a recursive remove operation
  --dangerous                   allow site-wide removal of objects
  --help, -h                    show help
```

Example: Remove a bucket named "mybucket" on https://play.min.io.

```
mc rb play/mybucket --force
Bucket removed successfully ‘play/mybucket’.
```

Command `cat` - Concatenate Objects

`cat` command concatenates contents of a file or object to another. You may also use it to simply display the contents to stdout

```
USAGE:
   mc cat [FLAGS] SOURCE [SOURCE...]

FLAGS:
  --encrypt-key value           encrypt/decrypt objects (using server-side encryption
  --help, -h                    show help


ENVIRONMENT VARIABLES:
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Display the contents of a text file `myobject.txt`

```
mc cat play/mybucket/myobject.txt
Hello MinIO!!
```

Example: Display the contents of a server encrypted object `myencryptedobject.txt`

```
mc cat --encrypt-key "play/mybucket=32byteslongsecretkeymustbegi
Hello MinIO!!
```

Example: Display the contents of a server encrypted object `myencryptedobject.txt` . Pass base64 encoded string if encryption key contains non-printable character like tab

Copy

```
mc cat --encrypt-key "play/mybucket=MzJieXRlc2xvbmdzZWNyZWFiY2RlZmcJZ2l2ZW5uMjB= pld
Hello MinIO!!
```

Command `sql` - Run sql queries on objects

`sql` run sql queries on objects.

Copy

```
USAGE:
  mc sql [FLAGS] TARGET [TARGET...]

FLAGS:
  --query value, -e value       sql query expression
  --recursive, -r               sql query recursively
  --csv-input value             csv input serialization option
  --json-input value            json input serialization option
  --compression value           input compression type
  --csv-output value            csv output serialization option
  --json-output value           json output serialization option
  --encrypt-key value           encrypt/decrypt objects (using server-side encryption
  --help, -h                    show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values

INPUT SERIALIZATION
  --csv-input or --json-input can be used to specify input data format. Format is
  specified by a string with pattern "key=value,..." for valid key(s).

  DATA FORMAT:
    csv: Use --csv-input flag
      Valid keys:

        RecordDelimiter (rd)
        FieldDelimiter (fd)
        QuoteChar (qc)
        QuoteEscChar (qec)
        FileHeader (fh)
        Comments (cc)
        QuotedRecordDelimiter (qrd)

    json: Use --json-input flag
      Valid keys:
        Type
    parquet: If object name ends in .parquet, this is automatic
```

Talk to the community

```
    parquet. If object name ends in .parquet, this is automatically interpreted.

OUTPUT SERIALIZATION
  --csv-output or --json-output can be used to specify output data format. Format is
  specified by a string with pattern "key=value,..." for valid key(s).
  DATA FORMAT:
    csv: Use --csv-output flag
      Valid keys:
        RecordDelimiter (rd)
        FieldDelimiter (fd)
        QuoteChar (qc)
        QuoteEscChar (qec)
        QuoteFields (qf)

    json: Use --json-output flag
      Valid keys:
        RecordDelimiter (rd)

COMPRESSION TYPE
    --compression specifies if the queried object is compressed.
    Valid values: NONE | GZIP | BZIP2
```

Example: Select all columns on a set of objects recursively on AWS S3

Copy

```
mc sql --recursive --query "select * from S3Object" s3/personalbucket/my-large-csvs/
```

Example: Run an aggregation query on an object on MinIO

Copy

```
mc sql --query "select count(s.power) from S3Object" myminio/iot-devices/power-ratio.
```

Example: Run an aggregation query on an encrypted object with customer provided keys

Copy

```
mc sql --encrypt-key "myminio/iot-devices=32byteslongsecretkeymustbegiven1" \
    --query "select count(s.power) from S3Object" myminio/iot-devices/power-ratio-enc
```

For more query examples refer to official AWS S3 documentation here

Command `head` - Display few lines of object

Talk to the community

`head` display first 'n' lines of an object

```
USAGE:
   mc head [FLAGS] SOURCE [SOURCE...]

FLAGS:
  -n value, --lines value       print the first 'n' lines (default: 10)
  --encrypt-key value           encrypt/decrypt objects (using server-side encryption
  --help, -h                    show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Display the first line of a text file `myobject.txt`

```
mc head -n 1 play/mybucket/myobject.txt
Hello!!
```

Example: Display the first line of a server encrypted object `myencryptedobject.txt`

```
mc head -n 1 --encrypt-key "play/mybucket=32byteslongsecretkeymustbegiven1" play/myb
Hello!!
```

Command `pipe` - Pipe to Object

`pipe` command copies contents of stdin to a target. When no target is specified, it writes to stdout.

```
USAGE:
   mc pipe [FLAGS] [TARGET]

FLAGS:
  --encrypt value               encrypt objects (using server-side encryption with se
  --encrypt-key value           encrypt/decrypt objects (using server-side encryption
  --help, -h                    show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT:      list of comma delimited prefix values
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Stream MySQL database dump to Amazon S3 directly.

```
mysqldump -u root -p ******* accountsdb | mc pipe s3/sql-backups/backups/accountsdb-
```

Command `cp` - Copy Objects

`cp` command copies data from one or more sources to a target. All copy operations to object storage are verified with MD5SUM checksums. Interrupted or failed copy operations can be resumed from the point of failure.

```
USAGE:
   mc cp [FLAGS] SOURCE [SOURCE...] TARGET

FLAGS:
  --recursive, -r                    copy recursively
  --older-than value                 copy object(s) older than N days (default: 0)
  --newer-than value                 copy object(s) newer than N days (default: 0)
  --storage-class value, --sc value  set storage class for new object(s) on target
  --attr                             add custom metadata for the object (format: KeyN
  --encrypt value                    encrypt/decrypt objects (using server-side encry
  --encrypt-key value                encrypt/decrypt objects (using server-side encry
  --help, -h                         show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT:      list of comma delimited prefixes
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Copy a text file to an object storage.

```
mc cp myobject.txt play/mybucket
myobject.txt:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00
```

Example: Copy a text file to an object storage with specified metadata.

```
mc cp --attr key1=value1;key2=value2 myobject.txt play/mybucket
myobject.txt:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00
```

Example: Copy a folder recursively from MinIO cloud storage to Amazon S3 cloud storage with specified metadata.

```
mc cp --attr Cache-Control=max-age=90000,min-fresh=9000\;key1=value1\;key2=value2 --r
https://play.minio.io:9000/mybucket/myobject.txt:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
```

Example: Copy a text file to an object storage and assign storage-class `REDUCED_REDUNDANCY` to the uploaded object.

```
mc cp --storage-class REDUCED_REDUNDANCY myobject.txt play/mybucket
myobject.txt:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00
```

Example: Copy a server-side encrypted file to an object storage.

```
mc cp --recursive --encrypt-key "s3/documents/=32byteslongsecretkeymustbegiven1" myobj
myobject.txt:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00
```

Example: Perform key-rotation on a server-side encrypted object

```
mc cp --encrypt-key 'myminio1/mybucket=32byteslongsecretkeymustgenerate , myminio2/myb
encryptedobject:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.
```

Notice that two different aliases myminio1 and myminio2 are used for the same endpoint to provide the old secretkey and the newly rotated key.

Example: Copy a javascript file to object storage and assign Cache-Control header to the uploaded object

```
mc cp --attr Cache-Control=no-cache myscript.js play/mybucket
myscript.js:    14 B / 14 B  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00 %
```

Command `rm` - Remove Objects

Use `rm` command to remove file or object

Copy

```
USAGE:
   mc rm [FLAGS] TARGET [TARGET ...]

FLAGS:
  --recursive, -r               remove recursively
  --force                       allow a recursive remove operation
  --dangerous                   allow site-wide removal of objects
  --incomplete, -I              remove incomplete uploads
  --fake                        perform a fake remove operation
  --stdin                       read object names from STDIN
  --older-than value            remove objects older than L days, M hours and N minut
  --newer-than value            remove objects newer than L days, M hours and N minut
  --encrypt-key value           encrypt/decrypt objects (using server-side encryptior
  --help, -h                    show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Remove a single object.

Copy

```
mc rm play/mybucket/myobject.txt
Removing `play/mybucket/myobject.txt`.
```

Example: Remove an encrypted object.

Copy

```
mc rm --encrypt-key "play/mybucket=32byteslongsecretkeymustbegiven1" play/mybucket/my
Removing `play/mybucket/myobject.txt`.
```

Example: Recursively remove a bucket's contents. Since this is a dangerous operation, you must explicitly pass `--force` option.

Copy

```
mc rm --recursive --force play/mybucket
Removing `play/mybucket/newfile.txt`.
Removing `play/mybucket/otherobject.txt`.
```

Example: Remove all uploaded incomplete files for an object.

Talk to the community

Copy

```
mc rm --incomplete play/mybucket/myobject.1gig
Removing `play/mybucket/myobject.1gig`.
```

Example: Remove object and output a message only if the object is created older than 1 day, 2 hours and 30 minutes. Otherwise, the command stays quiet and nothing is printed out.

Copy

```
mc rm -r --force --older-than 1d2h30m myminio/mybucket
Removing `myminio/mybucket/dayOld1.txt`.
Removing `myminio/mybucket/dayOld2.txt`.
Removing `myminio/mybucket/dayOld3.txt`.
```

Command `share` - Share Access

`share` command securely grants upload or download access to object storage. This access is only temporary and it is safe to share with remote users and applications. If you want to grant permanent access, you may look at `mc policy` command instead.

Generated URL has access credentials encoded in it. Any attempt to tamper the URL will invalidate the access. To understand how this mechanism works, please follow Pre-Signed URL technique.

Copy

```
USAGE:
   mc share [FLAGS] COMMAND

FLAGS:
  --help, -h                          show help

COMMANDS:
   download   generate URLs for download access
   upload     generate 'curl' command to upload objects without requiring access/secr
   list       list previously shared objects and folders
```

Sub-command `share download` - Share Download

`share download` command generates URLs to download objects without requiring access and secret keys. Expiry option sets the maximum validity period (no more than 7 days), beyond which the access is revoked automatically.

Talk to the community

Copy

```
USAGE:
   mc share download [FLAGS] TARGET [TARGET...]

FLAGS:
  --recursive, -r               share all objects recursively
  --expire value, -E value      set expiry in NN[h|m|s] (default: "168h")
  --help, -h                    show help
```

Example: Grant temporary access to an object with 4 hours expiry limit.

Copy

```
mc share download --expire 4h play/mybucket/myobject.txt
URL: https://play.min.io/mybucket/myobject.txt
Expire: 0 days 4 hours 0 minutes 0 seconds
Share: https://play.min.io/mybucket/myobject.txt?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-A
```

Sub-command `share upload` - Share Upload

`share upload` command generates a 'curl' command to upload objects without requiring access/secret keys. Expiry option sets the maximum validity period (no more than 7 days), beyond which the access is revoked automatically. Content-type option restricts uploads to only certain type of files.

Copy

```
USAGE:
   mc share upload [FLAGS] TARGET [TARGET...]

FLAGS:
  --recursive, -r                 recursively upload any object matching the prefix
  --expire value, -E value        set expiry in NN[h|m|s] (default: "168h")
  --content-type value, -T value  specify a content-type to allow
  --help, -h                      show help
```

Example: Generate a `curl` command to enable upload access to `play/mybucket/myotherobject.txt`. User replaces `<FILE>` with the actual filename to upload

Copy

```
mc share upload play/mybucket/myotherobject.txt
URL: https://play.min.io/mybucket/myotherobject.txt
Expire: 7 days 0 hours 0 minutes 0 seconds
Share: curl https://play.min.io/mybucket -F x-amz-date=20160408
```

Sub-command `share list` - Share List

`share list` command lists unexpired URLs that were previously shared

Copy

```
USAGE:
   mc share list COMMAND

COMMAND:
   upload:   list previously shared access to uploads.
   download: list previously shared access to downloads.
```

Command `mirror` - Mirror Buckets

`mirror` command is similar to `rsync` , except it synchronizes contents between filesystems and object storage.

Copy

```
USAGE:
   mc mirror [FLAGS] SOURCE TARGET

FLAGS:
  --overwrite                        overwrite object(s) on target
  --fake                             perform a fake mirror operation
  --watch, -w                        watch and synchronize changes
  --remove                           remove extraneous object(s) on target
  --region value                     specify region when creating new bucket(s) on ta
  -a                                 preserve bucket policy rules on target bucket(s)
  --exclude value                    exclude object(s) that match specified object na
  --older-than value                 filter object(s) older than N days (default: 0)

  --newer-than value                 filter object(s) newer than N days (default: 0)
  --storage-class value, --sc value  specify storage class for new object(s) on targe
  --encrypt value                    encrypt/decrypt objects (using server-side encry
  --encrypt-key value                encrypt/decrypt objects (using server-side encry
  --help, -h                         show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT:      list of comma delimited prefixes
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Mirror a local directory to 'mybucket' on https://play.min.io.

Copy

```
mc mirror localdir/ play/mybucket
localdir/b.txt:  40 B / 40 B
```

```
localdir/b.txt:   10 B / 10 B  ┃▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓┃  100.0
```

Example: Continuously watch for changes on a local directory and mirror the changes to 'mybucket' on https://play.min.io.

Copy

```
mc mirror -w localdir play/mybucket
localdir/new.txt:  10 MB / 10 MB  ┃▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓┃  1
```

Command `find` - Find files and objects

`find` command finds files which match the given set of parameters. It only lists the contents which match the given set of criteria.

Copy

```
USAGE:
  mc find PATH [FLAGS]

FLAGS:
  --exec value                spawn an external process for each matching object (s
  --ignore value              exclude objects matching the wildcard pattern
  --name value                find object names matching wildcard pattern
  --newer value               match all objects newer than specified time L days, M
  --older value               match all objects older than specified time L days, M
  --path value                match directory names matching wildcard pattern
  --print value               print in custom format to STDOUT (see FORMAT)
  --regex value               match directory and object name with PCRE regex patte
  --larger value              match all objects larger than specified size in units
  --smaller value             match all objects smaller than specified size in unit
  --maxdepth value            limit directory navigation to specified depth (defaul
  --watch                     monitor a specified path for newly created object(s)
  ...
  ...
  --help, -h                  show help
```

Example: Find all jpeg images from s3 bucket and copy to MinIO "play/bucket" bucket continuously.

Copy

```
mc find s3/bucket --name "*.jpg" --watch --exec "mc cp {} play/bucket"
```

Command `diff` - Show Difference

Talk to the community

`diff` command computes the differences between the two directories. It only lists the contents which are missing or which differ in size.

It *DOES NOT* compare the contents, so it is possible that the objects which are of same name and of the same size, but have difference in contents are not detected. This way, it can perform high speed comparison on large volumes or between sites

Copy

```
USAGE:
  mc diff [FLAGS] FIRST SECOND

FLAGS:
  --config-folder value, -C value  Path to configuration folder. (default: "/root/.mc
  --quiet, -q                       Disable progress bar display.
  --no-color                        Disable color theme.
  --json                            Enable JSON formatted output.
  --debug                           Enable debug output.
  --insecure                        Disable SSL certificate verification.
  --help, -h                        Show help.

LEGEND:
    < - object is only in source.
    > - object is only in destination.
    ! - newer object is in source.
```

Example: Compare a local directory and a remote object storage.

Copy

```
 mc diff localdir play/mybucket
‘localdir/notes.txt’ and ‘https://play.min.io/mybucket/notes.txt’ - only in first.
```

Option [--json]

JSON option enables parseable output in JSON lines format.

Example: diff json output.

Copy

```
mc diff minio1/diffbucket minio2/diffbucket --json
{"status":"success","first":"","second":"http://127.0.0.1:9001/diffbucket/file1.png",
{"status":"success","first":"http://127.0.0.1:9000/diffbucket/file2.png","second":"",
{"status":"success","first":"http://127.0.0.1:9000/diffbucket/file3.png","second":"ht
{"status":"success","first":"http://127.0.0.1:9000/diffbucket/file4.png","second":"ht
```

Talk to the community

Diff values in json output

Constant	Value	Meaning
differInNone	0	Does not differ
differInSize	1	Differs in size
differInTime	2	Differs in time
differInType	3	Differs in type exfile/directory
differInFirst	4	Only in source (FIRST)
differInSecond	5	Only in target (SECOND)

Command `watch` - Watch for files and object storage events.

`watch` provides a convenient way to watch on various types of event notifications on object storage and filesystem.

Copy

```
USAGE:
  mc watch [FLAGS] PATH

FLAGS:
  --events value                  filter specific types of events, defaults to all e
  --prefix value                  filter events for a prefix
  --suffix value                  filter events for a suffix
  --recursive                     recursively watch for events
  --help, -h                      show help
```

Example: Watch for all events on object storage

Copy

```
mc watch play/testbucket
[2016-08-18T00:51:29.735Z] 2.7KiB ObjectCreated https://play.min.io/testbucket/CONTRI
[2016-08-18T00:51:29.780Z]  1009B ObjectCreated https://play.min.io/testbucket/MAINTA
[2016-08-18T00:51:29.839Z] 6.9KiB ObjectCreated https://play.min.io/testbucket/README
```

Talk to the community

Example: Watch for all events on local directory

Copy

```
mc watch ~/Photos
[2016-08-17T17:54:19.565Z] 3.7MiB ObjectCreated /home/minio/Downloads/tmp/5467026530_
[2016-08-17T17:54:19.565Z] 3.7MiB ObjectCreated /home/minio/Downloads/tmp/5467026530_
...
[2016-08-17T17:54:19.565Z] 7.5MiB ObjectCreated /home/minio/Downloads/tmp/8771468997_
```

Command `event` - Manage bucket event notification.

`event` provides a convenient way to configure various types of event notifications on a bucket. MinIO event notification can be configured to use AMQP, Redis, ElasticSearch, NATS and PostgreSQL services. MinIO configuration provides more details on how these services can be configured.

Copy

```
USAGE:
  mc event COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]

COMMANDS:
  add     add a new bucket notification
  remove  remove a bucket notification. With '--force' can remove all bucket notifica
  list    list bucket notifications

FLAGS:
  --ignore-existing, -p            ignore if event already exists
  --help, -h                       show help
```

Example: List all configured bucket notifications

Copy

```
mc event list play/andoria
MyTopic        arn:minio:sns:us-east-1:1:TestTopic    s3:ObjectCreated:*,s3:ObjectRem
```

Example: Add a new 'sqs' notification resource only to notify on ObjectCreated event

Copy

```
mc event add play/andoria arn:minio:sqs:us-east-1:1:your-queue --event put
```

Example: Add a new 'sqs' notification resource with filters

Talk to the community

Add `prefix` and `suffix` filtering rules for `sqs` notification resource.

```
mc event add play/andoria arn:minio:sqs:us-east-1:1:your-queue --prefix photos/ --suf
```

Example: Remove a 'sqs' notification resource

```
mc event remove play/andoria arn:minio:sqs:us-east-1:1:your-queue
```

Command `policy` - Manage bucket policies

Manage anonymous bucket policies to a bucket and its contents

```
USAGE:
  mc policy [FLAGS] set PERMISSION TARGET
  mc policy [FLAGS] set-json FILE TARGET
  mc policy [FLAGS] get TARGET
  mc policy [FLAGS] get-json TARGET
  mc policy list [FLAGS] TARGET

PERMISSION:
  Allowed policies are: [none, download, upload, public].

FILE:
  A valid S3 policy JSON filepath.

FLAGS:
  --help, -h                      show help
```

Example: Show current anonymous bucket policy

Show current anonymous bucket policy for `mybucket/myphotos/2020/` sub-directory

```
mc policy get play/mybucket/myphotos/2020/
Access permission for ‘play/mybucket/myphotos/2020/’ is ‘none’
```

Example : Set anonymous bucket policy to download only

Set anonymous bucket policy for `mybucket/myphotos/2020/` sub-directory and its objects to `download` only. Now, objects under the sub-directory are publicly accessible. e.g `mybucket/myphotos/2020/yourobjectname` is available at https://play.min.io:9000/mybucket/myphotos/2020/yourobjectname

Copy

```
mc policy set download play/mybucket/myphotos/2020/
Access permission for ‘play/mybucket/myphotos/2020/’ is set to 'download'
```

Example : Set anonymous bucket policy from a JSON file

Configure bucket policy for `mybucket` with a policy JSON file.

Copy

```
mc policy set-json /tmp/policy.json play/mybucket
Access permission for `play/mybucket` is set from `/tmp/policy.json`
```

Example : Remove current anonymous bucket policy

Remove any bucket policy for *mybucket/myphotos/2020/* sub-directory.

Copy

```
mc policy set none play/mybucket/myphotos/2020/
Access permission for ‘play/mybucket/myphotos/2020/’ is set to 'none'
```

Command `admin` - Manage MinIO servers

Please visit here for a more comprehensive admin guide.

Command `session` - Manage Sessions

`session` command manages previously saved sessions for `cp` and `mirror` operations

Copy

```
USAGE:
  mc session COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]

COMMANDS:
  list    list all previously saved sessions
  clear   clear a previously saved session
  resume  resume a previously saved session

FLAGS:
```

Talk to the community

```
  --help, -h                    show help
```

Example: List all previously saved sessions.

```
mc session list
IXWKjpQM -> [2016-04-08 19:11:14 IST] cp assets.go play/mybucket
ApwAxSwa -> [2016-04-08 01:49:19 IST] mirror miniodoc/ play/mybucket
```

Example: Resume a previously saved session.

```
mc session resume IXWKjpQM
...assets.go: 1.68 KB / 1.68 KB  ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  100.00 % 784 B/s 2s
```

Example: Drop a previously saved session.

```
mc session clear ApwAxSwa
Session 'ApwAxSwa' cleared successfully.
```

Command `config` - Manage Config File

`config host` command provides a convenient way to manage host entries in your config file `~/.mc/config.json`. It is also OK to edit the config file manually using a text editor.

```
USAGE:
  mc config host COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]

COMMANDS:
  add, a      add a new host to configuration file
  remove, rm  remove a host from configuration file
  list, ls    lists hosts in configuration file

FLAGS:
  --help, -h                    show help
```

Example: Manage Config File

Add MinIO server access and secret keys to config file host entry. Note that, the history feature of your shell may record these keys and pose a security risk. On `bash` shell, use `set -o` and `set +o` to disable and enable history feature momentarily.

Copy

```
set +o history
mc config host add myminio http://localhost:9000 OMQAGGOL63D7UNVQFY8X GcY5RHNmnEWvD/1
set -o history
```

Remove the host from the config file.

Copy

```
mc config host remove myminio
```

List all configured host

Copy

```
mc config host list
```

Command `update` - Software Updates

Check for new software updates from https://dl.min.io. Experimental flag checks for unstable experimental releases primarily meant for testing purposes.

Copy

```
USAGE:
  mc update [FLAGS]

FLAGS:
  --quiet, -q  suppress chatty console output
  --json       enable JSON formatted output
  --help, -h   show help
```

Example: Check for an update.

Copy

```
mc update
You are already running the most recent version of ‘mc’.
```

Command `version` - Display Version

Talk to the community

Display the current version of `mc` installed

Copy

```
USAGE:
  mc version [FLAGS]

FLAGS:
  --quiet, -q  suppress chatty console output
  --json       enable JSON formatted output
  --help, -h   show help
```

Example: Print version of mc.

Copy

```
mc version
Version: 2016-04-01T00:22:11Z
Release-tag: RELEASE.2016-04-01T00-22-11Z
Commit-id: 12adf3be326f5b6610cdd1438f72dfd861597fce
```

Command `stat` - Stat contents of objects and folders

`stat` command displays information on objects (with optional prefix) contained in the specified bucket on an object storage. On a filesystem, it behaves like `stat` command.

Copy

```
USAGE:
   mc stat [FLAGS] TARGET

FLAGS:
  --recursive, -r               stat all objects recursively
  --encrypt-key value           encrypt/decrypt objects (using server-side encryption
  --help, -h                    show help

ENVIRONMENT VARIABLES:
   MC_ENCRYPT_KEY:  list of comma delimited prefix=secret values
```

Example: Display information on a bucket named "mybucket" on https://play.min.io.

Copy

```
mc stat play/mybucket
Name      : mybucket/
Date      : 2018-02-06 18:06:51 PST
Size      : 0B
Type      : folder
```

Talk to the community

Example: Display information on an encrypted object "myobject" in "mybucket" on https://play.min.io.

Copy

```
mc stat play/mybucket/myobject --encrypt-key "play/mybucket=32byteslongsecretkeymustb
Name      : myobject
Date      : 2018-03-02 11:47:13 PST
Size      : 132B
ETag      : d03ba22cd78282b7aef705bf31b8cded
Type      : file
Metadata  :
  Content-Type                                 : application/octet-stream
  X-Amz-Server-Side-Encryption-Customer-Key-Md5  : 4xSRdYsabg+s2nlsHKhgnw==
  X-Amz-Server-Side-Encryption-Customer-Algorithm: AES256
```

Example: Display information on objects contained in the bucket named "mybucket" on https://play.min.io.

Copy

```
mc stat -r play/mybucket
Name      : mybucket/META/textfile
Date      : 2018-02-06 18:17:38 PST
Size      : 1024B
ETag      : d41d8cd98f00b204e9800998ecf8427e
Type      : file
Metadata  :
  Content-Type: application/octet-stream

Name      : mybucket/emptyfile
Date      : 2018-02-06 18:16:14 PST
Size      : 100B
ETag      : d41d8cd98f00b204e9800998ecf8427e
Type      : file
Metadata  :
  Content-Type: application/octet-stream
```

Talk to the community

9.1.4 Installationsanleitung faas-cli

Installation

You can install the CLI with a `curl` utility script, `brew` or by downloading the binary from the releases page. Once installed you'll get the `faas-cli` command and `faas` alias.

Linux or macOS

Utility script with `curl`:

```
$ curl -sSL https://cli.openfaas.com | sudo -E sh
```

> The flag `-E` allows for any `http_proxy` environmental variables to be passed through to the installation bash script.

Non-root with curl downloads the binary into your current directory and will then print installation instructions:

```
$ curl -sSL https://cli.openfaas.com | sh
```

Via brew:

```
$ brew install faas-cli
```

> **Note**
>
> The `brew` release may not run the latest minor release but is updated regularly.

Windows

In PowerShell:

```
$version = (Invoke-WebRequest
"https://api.github.com/repos/openfaas/faas-cli/releases/latest" |
ConvertFrom-Json)[0].tag_name
(New-Object
System.Net.WebClient).DownloadFile("https://github.com/openfaas/faas-
cli/releases/download/$version/faas-cli.exe", "faas-cli.exe")
```

Environment variable overrides

Several overrides exist which will be used by default if set and no other command-line flag has been set.

- `OPENFAAS_TEMPLATE_URL` - to set the default URL to pull templates from
- `OPENFAAS_PREFIX` - for use with `faas-cli new` - this can act in place of `--prefix`
- `OPENFAAS_URL` - to override the default gateway URL

Running `faas-cli` with sudo

If you're running the faas-cli with `sudo` we recommend using `sudo -E` to pass through any environmental variables you may have configured such as a `http_proxy`, `https_proxy` or `no_proxy` entry.

Docker image

The `faas-cli` is also available as a Docker image making it convenient for use in CI jobs such as with a Jenkins pipeline or a task in cron.

https://hub.docker.com/r/openfaas/faas-cli/tags/
[https://hub.docker.com/r/openfaas/faas-cli/tags/]

There is no "latest" tag, so find the version of the CLI you want to use from the tags page on the Docker Hub. These correspond to the release from GitHub.

> Note: the Docker image cannot be used to perform a build directly, but you can use it to generate a build context which can be used with a container builder

> such as Docker, buildkit or Kaniko in another part of your build pipeline.

Use-cases for the Docker image:

- Generate the build context without running `docker build` - `faas-cli --shrinkwrap`
- Deploy an existing image to a remote server `faas-cli deploy`
- Manage secrets with `faas-cli secret`
- Invoke functions via cron with `faas-cli invoke`
- Check the health of your remote gateway with `faas-cli info`

Building from source

The contributing guide [/community/#contribute] has instructions for building from source and for configuring a Golang development environment.

- **Star/fork** on GitHub: faas-cli [https://github.com/openfaas/faas-cli]

Tutorial: learn how to use the CLI

Morning coffee with the OpenFaaS CLI [https://blog.alexellis.io/quickstart-openfaas-cli/]

9.2 Installation der Plattform

9.2.1 Kubernetes Manifest für Minio

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: minio
  annotations:
    linkerd.io/inject: "enabled"
spec: {}
status: {}

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-pv-claim
  namespace: minio
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi

apiVersion: apps/v1
kind: Deployment
metadata:
  name: minio
  namespace: minio
spec:
  selector:
    matchLabels:
      app: minio
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: minio
    spec:
      volumes:
      - name: data
```

```
40          persistentVolumeClaim:
41            claimName: minio-pv-claim
42        containers:
43        - name: minio
44          image: minio/minio
45          volumeMounts:
46          - name: data
47            mountPath: "/data"
48          env:
49          - name: MINIO_ACCESS_KEY
50            value: "minio"
51          - name: MINIO_SECRET_KEY
52            value: "minio123"
53          ports:
54          - containerPort: 9000
55          resources:
56            limits:
57              cpu: "1"
58              memory: "2048Mi"
59            requests:
60              cpu: "1"
61              memory: "512Mi"
62          args:
63          - server
64          - /data
65          readinessProbe:
66            httpGet:
67              path: /minio/health/ready
68              port: 9000
69            initialDelaySeconds: 120
70            periodSeconds: 20
71          livenessProbe:
72            httpGet:
73              path: /minio/health/live
74              port: 9000
75            initialDelaySeconds: 120
76            periodSeconds: 20
77
78  apiVersion: v1
79  kind: Service
80  metadata:
81    name: minio-service
82    namespace: minio
83  spec:
```

```
84    type: ClusterIP
85    ports:
86      - port: 80
87        targetPort: 9000
88        protocol: TCP
89    selector:
90      app: minio
```

9.2.2 Implementierung der Funktion Crop

```
1  from osgeo import gdal
2  from minio import Minio
3  from minio.error import (ResponseError, BucketAlreadyOwnedByYou, BucketAlre
4  import numpy as np
5  import pandas
6  import matplotlib.pyplot as plt
7  import boto3
8  import io
9  import json
10 import requests
11 import os
12 import uuid
13
14 def handle(req):
15
16     # fetch environment variables
17     BUCKET = os.environ['MINIO_BUCKET']
18     MINIO_URL = os.environ['MINIO_URL']
19     MINIO_ACCESS_KEY = os.environ['MINIO_ACCESS_KEY']
20     MINIO_SECRET_KEY = os.environ['MINIO_SECRET_KEY']
21
22     # set gdal config to log exceptions
23     gdal.UseExceptions()
24
25     # parse json from http request
26     json_req = json.loads(req)
27
28     # set input file names
29     file_url = json_req['url']
30     file_output_name = json_req['output']
31     parameter = json_req['parameter']
32
33     # read original dataset with gdal
34     original_dataset = gdal.Open(file_url)
35
36     # crop original dataset
37     crop_x = parameter["x"]
38     crop_y = parameter["y"]
39     crop_w = parameter["w"]
40     crop_h = parameter["h"]
41     crop_dataset = gdal.Translate("/vsimem/tmp.tif", original_dataset, form
42     gdal.SetConfigOption('COMPRESS_OVERVIEW','JPEG')
43     gdal.SetConfigOption('JPEG_QUALITY_OVERVIEW','60')
44     crop_dataset.BuildOverviews('NEAREST',[2,4,8,16,32])
```

```
45
46      # initialize minio client
47      minio_client = Minio(MINIO_URL, access_key=MINIO_ACCESS_KEY, secret_key=M
48
49      # upload cropped and warped datasets to minio
50      # create temporary geotiff (necessary because of compression)
51      tiff_driver = gdal.GetDriverByName('GTiff')
52      tiff_driver.CreateCopy('/vsimem/cogtif.tif', crop_dataset, options=["TILE
53      f = gdal.VSIFOpenL('/vsimem/cogtif.tif', 'rb')
54      gdal.VSIFSeekL(f, 0, 2)  # seek to end
55      size = gdal.VSIFTellL(f)
56      gdal.VSIFSeekL(f, 0, 0)  # seek to beginning
57      cogtif = gdal.VSIFReadL(1, size, f)
58      gdal.VSIFCloseL(f)
59
60      # Upload the raw data to s3
61      try:
62          # check if bucket exists
63          if not minio_client.bucket_exists(BUCKET):
64              minio_client.make_bucket(BUCKET)
65          minio_client.put_object(BUCKET, file_output_name , io.BytesIO(cogtif)
66      except ResponseError as err:
67          print(err)
68
69      gdal.Unlink("/vsimem/tmp.tif")
70      gdal.Unlink('/vsimem/cogtif.tif')
71
72      response_json = {}
73      response_json['result'] = 'http://minio-service.minio.svc.cluster.local:9
74
75      return json.dumps(response_json)
```

9.2.3 Implementierung der Funktion Georeferenzierung

```
from osgeo import gdal
from minio import Minio
from minio.error import (ResponseError, BucketAlreadyOwnedByYou, BucketAlre
import numpy as np
import pandas
import matplotlib.pyplot as plt
import boto3
import io
import json
import requests
import os
import uuid



def handle(req):

    BUCKET              = os.environ['MINIO_BUCKET']
    MINIO_URL           = os.environ['MINIO_URL']
    MINIO_ACCESS_KEY    = os.environ['MINIO_ACCESS_KEY']
    MINIO_SECRET_KEY    = os.environ['MINIO_SECRET_KEY']

    # set gdal config to log exceptions
    gdal.UseExceptions()

    # parse json from http request
    json_req = json.loads(req)

    # set input file names
    file_url = json_req['url']
    file_output_name = json_req['output']
    parameter = json_req['parameter']

    # read original dataset with gdal
    original_dataset = gdal.Open(file_url, gdal.GA_Update)

    gcp_list = []
    for gcp in parameter['gcps']:
        gcp_list.append(gdal.GCP(gcp['dstx'], gcp['dsty'], 0, gcp['srcx'],
    original_dataset.SetGCPs(gcp_list, original_dataset.GetProjection())
    rect_dataset = gdal.Warp("/vsimem/tmp.tif", original_dataset, format="V
    gdal.SetConfigOption('COMPRESS_OVERVIEW','JPEG')
    gdal.SetConfigOption('JPEG_QUALITY_OVERVIEW','60')
    rect_dataset.BuildOverviews('NEAREST',[2,4,8,16,32])
```

```
45
46      # initialize minio client
47      minio_client = Minio(MINIO_URL, MINIO_ACCESS_KEY, MINIO_SECRET_KEY, secur
48
49      # upload cropped and warped datasets to minio
50      tiff_driver = gdal.GetDriverByName('GTiff')
51      tiff_driver.CreateCopy('/vsimem/cogtif.tif', rect_dataset, options=["TILE
52
53      # Read the raw data from the virtual filesystem
54      f = gdal.VSIFOpenL('/vsimem/cogtif.tif', 'rb')
55      gdal.VSIFSeekL(f, 0, 2)  # seek to end
56      size = gdal.VSIFTellL(f)
57      gdal.VSIFSeekL(f, 0, 0)  # seek to beginning
58      cogtif = gdal.VSIFReadL(1, size, f)
59      gdal.VSIFCloseL(f)
60
61      # Upload the raw data to s3
62      try:
63          filename = str(uuid.uuid4())+'.tif'
64          if not minio_client.bucket_exists(BUCKET):
65              minio_client.make_bucket(BUCKET)
66          minio_client.put_object(BUCKET, file_output_name , io.BytesIO(cogtif)
67      except ResponseError as err:
68          print(err)
69
70      gdal.Unlink("/vsimem/tmp.tif")
71      gdal.Unlink('/vsimem/cogtif.tif')
72
73      response_json = {}
74      response_json['result'] = 'http://minio-service.minio.svc.cluster.local:9
75
76      return json.dumps(response_json)
```

9.3 Sonstige

9.3.1 Cloud-Optimized GeoTIFF Dokumentation und Benchmarks

Cloud optimized GeoTIFF

Table of Contents

NOTE

The specification now lives in

https://github.com/cogeotiff/cog-spec/blob/master/spec.md

Definition

A cloud optimized GeoTIFF is a regular GeoTIFF file, aimed at being hosted on a HTTP file server, whose internal organization is friendly for consumption by clients issuing HTTP GET range request ("bytes: start_offset-end_offset" HTTP header).

It contains at its beginning the metadata of the full resolution imagery, followed by the optional presence of overview metadata, and finally the imagery itself. To make it friendly with streaming and progressive rendering, we recommand starting with the imagery of the smallest overview and finishing with the imagery of the full resolution level.

More formally, the structure of such a file is:

- TIFF / BigTIFF signature
- IFD (Image File Directory) of full resolution image
- Values of TIFF tags that don't fit inline in the IFD directory, such as TileOffsets?, TileByteCounts? and GeoTIFF keys
- Optional: IFD (Image File Directory) of first overview (typically subsampled by a factor of 2), followed by the values of its tags that don't fit inline
- Optional: IFD (Image File Directory) of second overview (typically subsampled by a factor of 4), followed by the values of its tags that don't fit inline
- ...
- Optional: IFD (Image File Directory) of last overview (typically subsampled by a factor of 2^N), followed by the values of its tags that don't fit inline
- Optional: tile content of last overview level
- ...
- Optional: tile content of first overview level
- Tile content of full resolution image.

Unspecified points

- size of tile: 256 or 512 pixels are typical however
- compression methods allowed. typically, no compression, DEFLATE or LZW can be used for lossless, or JPEG for lossy. (note that DEFLATE while more efficient than LZW can cause compatibility issues with some software packages)

How to generate it with GDAL

Given an input dataset in.tif with already generated internal or external overviews, a cloud optimized GeoTIFF can be generated with:

gdal_translate in.tif out.tif -co TILED=YES -co COPY_SRC_OVERVIEWS=YES -co COMPRESS=LZW

This will result in a images with tiles of dimension 256x256 pixel for main resolution, and 128x128 tiles for overviews.

For an image of 4096x4096 with 4 overview levels, the 5 IFDs and their TileOffsets? and TileByteCounts? tag data fit into the first 6KB of the file.

Note: for JPEG compression, the above method produce cloud optimized files only if using GDAL 2.2 (or a dev version >= r36879). For older versions, the IFD of the overviews will be written towards the end of the file. A recent version of GDAL (2.2 or dev version >= r37257) built against internal libtiff (or libtiff >= 4.0.8) will also help reducing the amount of bytes read for JPEG compressed files with YCbCr subsampling.

How to read it with GDAL

GDAL includes special filesystems that can read a file hosted on a HTTP/FTP server by chunks.

The base filesystem is /vsicurl/ (Virtual System Interface for Curl) and the filename it accepts are of the form "/vsicurl/http://example.com/path/to/some.tif". They can be used whereever GDAL expects a dataset / filename to be passed: gdalinfo, gdal_translate, GDALOpen() API, etc...

Currently /vsicurl/ uses 16 KB as the minimum unit for downloading with HTTP range requests, and a in-memory cache of up to 1000 16KB blocks, with a least recently used eviction strategy.

To minimize the total number of HTTP requests outside of the target GeoTIFF file, setting the GDAL_DISABLE_READDIR_ON_OPEN=YES and CPL_VSIL_CURL_ALLOWED_EXTENSIONS=tif environment variables/configuration options is recommended so as to avoid any side-car files (such a .ovr, .aux.xml, .aux, etc.) to be probed.

Running gdalinfo or GDALOpen() on such a cloud optimized GeoTIFF will retrieve all the metadata with a single HTTP request of 16 KB. When reading pixels in a tile, only the blocks of 16 KB intersecting the range of the tile will be downloaded.

For files hosted on Amazon S3 storage, with non-public sharing rights, /vsis3/ can be used.

How to check if a GeoTIFF has a cloud optimization internal organization ?

The validate_cloud_optimized_geotiff.py script can be used to check that a (GeoTIFF) file follows the above described file structure

```
$ python validate_cloud_optimized_geotiff.py test.tif
```

or

```
$ python
import validate_cloud_optimized_geotiff.py
validate_cloud_optimized_geotiff.validate('test.tif')
```

Performance testing

Done with GDAL trunk r37259 with internal libtiff.

Preparation

The source image is the True Color Image of a Sentinel 2A L1C product (10980x10980 pixels, RGB bands of type Byte)

Creation of a strip oriented GeoTIFF without overviews ("traditional"):

```
gdal_translate SENTINEL2_L1C:S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_2017
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR
```

Creation of a tiled GeoTIFF without overviews:

```
gdal_translate SENTINEL2_L1C:S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_2017
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR
```

Creation of a tiled GeoTIFF with overviews:

```
gdal_translate SENTINEL2_L1C:S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_2017
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR
gdaladdo -r average S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T1114
```

Creation of a cloud optimized GeoTIFF:

```
gdal_translate SENTINEL2_L1C:S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_2017
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=DEFLATE
gdaladdo -r average  S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111
gdal_translate S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR -co COPY_
```

Creation of a cloud optimized GeoTIFF with tiles of dimension 512x512

```
gdal_translate SENTINEL2_L1C:S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_2017
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=DEFLATE
gdaladdo -r average  S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111
gdal_translate S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               S2A_MSIL1C_20170102T111442_N0204_R137_T30TXT_20170102T111441_TC
               -co TILED=YES -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR -co COPY_
               -co BLOCKXSIZE=512 -co BLOCKYSIZE=512 --config GDAL_TIFF_OVR_BI
```

Reading a single pixel

- Traditional GeoTIFF

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
    /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Report:
  Location: (5000P,5000L)
  Band 1:
GDAL: GDAL_CACHEMAX = 791 MB
VSICURL: Downloading 1523712-1540095 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
    Value: 255
  Band 2:
    Value: 255
  Band 3:
    Value: 255

real    0m0.397s
user    0m0.060s
sys     0m0.024s
```

- Tiled GeoTIFF without overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N

VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
```

```
Report:
  Location: (5000P,5000L)
  Band 1:
GDAL: GDAL_CACHEMAX = 791 MB
VSICURL: Downloading 1556480-1572863 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
    Value: 255
  Band 2:
    Value: 255
  Band 3:
    Value: 255
```

- Regular GeoTIFF with overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Report:
  Location: (5000P,5000L)
  Band 1:
GDAL: GDAL_CACHEMAX = 791 MB
VSICURL: Downloading 1556480-1572863 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
    Value: 255
  Band 2:
    Value: 255
  Band 3:
    Value: 255
GDAL: GDALClose(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_201

real    0m0.520s
user    0m0.080s
sys     0m0.012s
```

- Cloud optimized GeoTIFF:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Report:
  Location: (5000P,5000L)
  Band 1:
GDAL: GDAL_CACHEMAX = 791 MB
VSICURL: Downloading 2785280-2801663 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
    Value: 255
  Band 2:
    Value: 255
  Band 3:
    Value: 255
GDAL: GDALClose(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_201

real    0m0.527s
user    0m0.088s
sys     0m0.024s
```

- Cloud optimized GeoTIFF with 512x512 tiles

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Report:
  Location: (5000P,5000L)
  Band 1:
GDAL: GDAL_CACHEMAX = 791 MB
VSICURL: Downloading 2392064-2408447 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
    Value: 255
  Band 2:
    Value: 255
  Band 3:
    Value: 255
GDAL: GDALClose(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_201

real    0m0.407s
user    0m0.068s
sys     0m0.016s
```

The winners are the traditional GeoTIFF organization and the cloud optimized 512x512 tiled GeoTIFF since the directory fits in the first 16 KB.

In other cases, no significant time difference (individual runs may differ by a few tens of milliseconds). Same amount of I/O (64 KB read). The fact that we even read 16 + 32 KB in the case without overviews is due to the fact that the first IFD is slightly larger than 16 KB, so we need to read a bit more (and the heuristics double the chunk size as this is contiguous to the previous region read).

Note the use of CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif to avoid reading any side car files (.aux.xml, etc...) and GDAL_DISABLE_READDIR_ON_OPEN=YES to avoid any attempt of listing the files in the same directory.

Same conclusions if using a AWS S3 hosting, with both GDAL_DISABLE_READDIR_ON_OPEN=YES and CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif defined as well.

Reading a block of pixels at full resolution

- Traditional GeoTIFF

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
   -srcwin 1024 1024 256 256 out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GDAL: GDALDefaultOverviews::OverviewScan()
VSICURL: Downloading 163840-196607 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
VSICURL: Downloading 196608-262143 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
VSICURL: Downloading 262144-393215 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.923s
user    0m0.256s
sys     0m0.020s
```

The access to the pixel values requires 3 GET requests since whole lines need to be downloaded. A smarter implementation in the GeoTIFF reader with a better interaction with the GDAL HTTP code could probably collapse the 3 GET into a single larger one.

- Tiled GeoTIFF without overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N
  -srcwin 1024 1024 256 256 out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GDAL: GDALDefaultOverviews::OverviewScan()
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.527s
user    0m0.084s
sys     0m0.020s
```

- Regular GeoTIFF with overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
   -srcwin 1024 1024 256 256 out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
VSICURL: Downloading 2113536-2129919 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 2129920-2162687 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
GDAL: GDALDefaultOverviews::OverviewScan()
VSICURL: Downloading 196608-212991 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.757s
user    0m0.100s
sys     0m0.032s
```

One can see that a directory scan is done (GTiff: ScanDirectories?() trace), despite a few optimizations done in r37258 and r37259. This is due to gdal_translate trying to copy mask bands, which requires scaning directories to find a potential internal mask band. This scan can be avoided by adding the -mask none switch.

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
   /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_
   -srcwin 1024 1024 256 256 -mask none out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
```

```
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GDAL: GDALDefaultOverviews::OverviewScan()
GDAL: GDALDatasetCopyWholeRaster(): 256*256 swaths, bInterleave=1
VSICURL: Downloading 196608-212991 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.518s
user    0m0.092s
sys     0m0.012s
```

Best timing on a AWS S3 bucket (us-east-1 region, accessed from France): ~ 1.9s

This is also the performance one gets with :

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
    python -c 'from osgeo import gdal; ds = gdal.Open("/vsicurl/http://even.rc
```

- Cloud optimized GeoTIFF:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N
  -srcwin 1024 1024 256 256 out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
VSICURL: Downloading 1425408-1441791 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.519s
user    0m0.096s
sys     0m0.008s
```

Best timing on a AWS S3 bucket: ~ 1.9s

No need to specify -mask none to get the maximum performance: as the IFD are at the beginning of the files, they have been fetched with the 2 first HTTP GET requests.

- Cloud optimized GeoTIFF with tiles 512x512:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N
  -srcwin 1024 1024 256 256 out.tif
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
```

```
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
GDAL: GDALDefaultOverviews::OverviewScan()
VSICURL: Downloading 1114112-1146879 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.441s
user    0m0.096s
sys     0m0.016s
```

Compared to the default cloud optimized GeoTIFF, we save a few bytes for the reading of the IFD since all IFDs fit into the first 16 KB

Getting a subsampled version of the image

- Traditional GeoTIFF

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
     /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T11144
     out.tif -outsize 1% 1%
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GDAL: GDALDefaultOverviews::OverviewScan()
GDAL: GDALDefaultOverviews::OverviewScan()
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
VSICURL: Downloading 49152-114687 (http://even.rouault.free.fr/gtiff_test/S2A_
VSICURL: Got response_code=206
VSICURL: Downloading 114688-245759 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
VSICURL: Downloading 245760-507903 (http://even.rouault.free.fr/gtiff_test/S2A
VSICURL: Got response_code=206
VSICURL: Downloading 507904-1032191 (http://even.rouault.free.fr/gtiff_test/S2
VSICURL: Got response_code=206
VSICURL: Downloading 1064960-1081343 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1097728-1114111 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1130496-1146879 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1163264-1179647 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1196032-1212415 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1212416-1245183 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1245184-1310719 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1310720-1441791 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1441792-1703935 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 1703936-2009537 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m3.197s
user    0m0.304s
sys     0m0.056s
```

As one could anticipate, the whole file needs to be read.

- Tiled GeoTIFF without overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=
    /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T1
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C
Input file size is 10980, 10980
GDAL: GDALDefaultOverviews::OverviewScan()
GDAL: GDAL_CACHEMAX = 791 MB
GDAL: GDALDatasetCopyWholeRaster(): 109*109 swaths, bInterleave=1
GDAL: GDALDefaultOverviews::OverviewScan()
GDAL: GDALDefaultOverviews::OverviewScan()
GDAL: Potential thrashing on band 1 of /vsimem/sparse_0x7d0e10.
VSICURL: Downloading 49152-114687 (http://even.rouault.free.fr/gtiff_test
VSICURL: Got response_code=206
VSICURL: Downloading 114688-245759 (http://even.rouault.free.fr/gtiff_tes
VSICURL: Got response_code=206
VSICURL: Downloading 245760-507903 (http://even.rouault.free.fr/gtiff_tes
VSICURL: Got response_code=206
VSICURL: Downloading 507904-1032191 (http://even.rouault.free.fr/gtiff_te
VSICURL: Got response_code=206
VSICURL: Downloading 1032192-2080767 (http://even.rouault.free.fr/gtiff_t
VSICURL: Got response_code=206
VSICURL: Downloading 2080768-2123471 (http://even.rouault.free.fr/gtiff_t
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.
GDAL: GDALClose(out.tif, this=0x83bdc0)
GDAL: 59168 block reads on 32 block band 1 of /vsimem/sparse_0x7d0e10.

real    0m2.960s
user    0m0.740s
sys     0m0.096s
```

As one could anticipate, the whole file needs to be read.

- Regular GeoTIFF with overviews:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N
  out.tif -outsize 1% 1%

VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
VSICURL: Downloading 2113536-2129919 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 2129920-2162687 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
VSICURL: Downloading 3342336-3358719 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
VSICURL: Downloading 3358720-3363606 (http://even.rouault.free.fr/gtiff_test/S
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.
```

```
real    0m0.810s
user    0m0.108s
sys     0m0.020s
```

Best timing on a AWS S3 bucket: ~ 2.5s

A full scan of the IFD is necessary to find the appropriate overview level.

- Cloud optimized GeoTIFF:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
  /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442_N
  out.tif -outsize 1% 1%
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
GDAL: GDALDatasetCopyWholeRaster(): 109*109 swaths, bInterleave=1
...10...20...30...40...50...60...70...80...90...100 - done.

real    0m0.435s
user    0m0.088s
sys     0m0.028s
```

Best timing on a AWS S3 bucket: ~ 1.5s

As the IFD are at the beginning of the files, as well as the pixel data for the smallest overview, the request can be completed with the 2 first HTTP GET requests (this is a bit of an extreme case of course)

- Cloud optimized GeoTIFF with tiles of 512x512:

```
$ time GDAL_DISABLE_READDIR_ON_OPEN=YES CPL_VSIL_CURL_ALLOWED_EXTENSIONS=.tif
    /vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_20170102T111442
    out.tif -outsize 1% 1%
VSICURL: GetFileSize(http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017010
VSICURL: Downloading 0-16383 (http://even.rouault.free.fr/gtiff_test/S2A_MSIL1
VSICURL: Got response_code=206
GDAL: GDALOpen(/vsicurl/http://even.rouault.free.fr/gtiff_test/S2A_MSIL1C_2017
Input file size is 10980, 10980
GTiff: ScanDirectories()
GTiff: Opened 5490x5490 overview.
GTiff: Opened 2745x2745 overview.
GTiff: Opened 1373x1373 overview.
GTiff: Opened 687x687 overview.
GTiff: Opened 344x344 overview.
VSICURL: Downloading 16384-49151 (http://even.rouault.free.fr/gtiff_test/S2A_M
VSICURL: Got response_code=206
...10...20...30...40...50...60...70...80...90...100 - done.
real    0m0.439s
user    0m0.088s
sys     0m0.024s
```

Same performance as the default cloud optimized case (GDAL could potentially read in a less gready way for the pixel data since the tile size is only 8KB here)

Conclusions

- GDAL can make an efficient use of overviews, even on files hosted on a HTTP storage
- The cloud compatible file organization can reduce a bit the amount of GET requests required.
- Increasing the block size can reduce the size of the IFD. But larger blocks can cause more bytes to be pulled for random access if the compression rate is not high.

Last modified on 1 Aug 2018 10:18:25