Jade Hochschule Oldenburg
Fachbereich Bauwesen Geoinformation Gesundheitstechnologie

Masterarbeit

Entwicklung einer Cloud-nativen Architektur zur Verwaltung und Verarbeitung rasterbasierter Bodenbewegungsdaten

Development of a cloud-native architecture for management and processing of raster-based ground motion data

Jannes Wykhoff
Matrikelnummer: 6032253
24.Oktober 2024

Erstprüfer:	Prof. Dr. Thomas Brinkhoff Jade Hochschule Oldenburg
Zweitprüfer:	Clas Christian Borchers (M.Sc.) Landesamt für Geoinformation und Landesvermessung Niedersachsen (LGLN)
Studiengang:	Geoinformationswissenschaften - Schwerpunkt Geoinformatik

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie alle Stellen der Arbeit, die wortwörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht habe.

Ich verantworte die Übernahme jeglichen Outputs der von mir verwendeten generativen KI-Systeme vollumfänglich selbst. Ich habe alle von KI-Systemen generierten Outputs, die ich direkt oder indirekt verwendet habe, als solche ausgewiesen. Ich habe in der Anlage „Dokumentation der Verwendung generativer KI-Systeme“ zur Arbeit dargelegt, welche generativen KI-Systeme ich genutzt habe, für welchen Zweck ich diese verwendet habe und auf welche Weise die Nutzung stattfand.

Ort, Datum	Unterzeichner

Zusammenfassung

Das Ziel dieser Masterarbeit ist die Entwicklung einer Cloud-nativen Architektur, die den Vorgaben der IT-Strategie des Landesamts für Geoinformation und Landesvermessung Niedersachsen (LGLN) entspricht und eine effiziente Verwaltung, Verarbeitung sowie Bereitstellung rasterbasierter Bodenbewegungsdaten ermöglicht. Primär soll diese Architektur und deren Komponenten im Kontext des Bodenbewegungsdienstes Niedersachsens (BOB.NI) eingesetzt werden. Die Architektur zielt darauf ab, die Herausforderungen durch kontinuierliche Bodenbewegungen zu adressieren, indem das flächendeckende Bodenbewegungsmodell des LGLN automatisiert verwaltet, verarbeitet und bereitgestellt wird. Der Fokus liegt dabei auf der Umsetzung einer skalierbaren und flexiblen Cloud-Lösung, die sowohl Fachanwendern als auch der Öffentlichkeit den Zugriff auf Bodenbewegungsinformationen über eine BOB.NI-WebApp erleichtert.

Nachdem zunächst die Anforderungen an die Architektur auf Basis von Dokumenten, Stakeholdern und bestehenden Systemen identifiziert wurden, wurde eine detaillierte Analyse des bestehenden Prototyps durchgeführt. Dabei stellte sich heraus, dass der vom LGLN entwickelte Prototyp weder den Vorgaben der IT-Strategie entspricht, noch die Qualitätsanforderungen gemäß ISO/IEC 25010 erfüllt, weshalb eine Neuentwicklung erforderlich ist. Für die Neuentwicklung wurden die Subpixelinterpolation zur präzisen Abfrage von Bodenbewegungsprofilen und die Rasterstatistik zur flächenbezogenen Analyse als zentrale Funktionalitäten identifiziert. Für die Subpixelinterpolation sind sowohl die bilineare als auch die bikubische Interpolation erforderlich, um je nach Genauigkeitsanforderung optimale Ergebnisse zu erzielen. Auch die Datenverwaltung wurde umfassend analysiert. Hierbei zeigte sich, dass Cloud-optimierte GeoTIFF-Dateien in Kombination mit einem Cloud Object Storage die beste Lösung zur effizienten Speicherung des rasterbasierten Bodenbewegungsmodells darstellen. Eine Auflösung des Rasters von 100 m wurde als besonders effizient ermittelt. Für ergänzende Vektordaten wurde das FlatGeobuf-Format als optimal bewertet, da es sowohl eine schnelle Abfrage als auch eine effiziente Speicherung ermöglicht.

Auf Grundlage der Anforderungsanalyse und der gewonnenen Erkenntnisse aus der Analyse wurde eine Cloud-native Architektur in der IBM Cloud implementiert. Diese Architektur umfasst die Infrastruktur, Softwarearchitektur sowie eine automatisierte Datenverwaltung. Die Infrastruktur besteht im Wesentlichen aus einem IBM Cloud Object Storage, der in zwei Buckets unterteilt ist: einen für untransformierte Rohdaten und einen für Cloud-optimierte, produktive Daten. Zusätzlich wird die serverlose Plattform IBM Cloud Code Engine genutzt, die als Grundlage für eine Microservice-Architektur dient. Diese stellt die identifizierten Funktionen der Subpixelinterpolation und Rasterstatistik effizient und skalierbar bereit. In Kombination mit dem Cloud Object Storage ermöglicht sie außerdem die automatisierte Datenverwaltung, bei der herkömmlich genutzte Datenformate durch eventgesteuerte Jobs in Cloud-optimierte Formate überführt werden. Obwohl diese Architektur und deren Komponenten primär für den BOB.NI entwickelt wurde, ist sie generisch gestaltet, um auch für andere rasterbasierte Anwendungen eingesetzt werden zu können.

Die Evaluation der Architektur hinsichtlich der gestellten Anforderungen zeigte, dass die meisten Anforderungen erfüllt wurden, wobei dennoch Optimierungspotenziale hinsichtlich Ressourcennutzung, Sicherheitsaspekte und der Implementierung einer Continuous Integration/Continuous Deployment-Pipeline bestehen. Für die künftige Weiterentwicklung wird empfohlen, diese Optimierungen gezielt umzusetzen und die Architektur in eine produktionsreife Umgebung zu überführen.

Abstract

The aim of this master's thesis is to develop a cloud-native architecture that meets the requirements of the IT strategy of the State Office for Geoinformation and Surveying of Lower Saxony (LGLN) and enables the efficient management, processing and provision of raster-based ground movement data. This architecture and its components will primarily be used in the context of the Lower Saxony Ground Movement Service (BOB.NI). The architecture aims to address the challenges posed by continuous ground movements by automatically managing, processing and providing the LGLN's area-wide ground movement model. The focus here is on implementing a scalable and flexible cloud solution that makes it easier for both specialist users and the public to access soil movement information via a BOB.NI web app.

After first identifying the requirements for the architecture on the basis of documents, stakeholders and existing systems, a detailed analysis of the existing prototype was carried out. It emerged that the prototype developed by LGLN neither met the requirements of the IT strategy nor the quality requirements of ISO/IEC 25010, which is why a new development was necessary. Subpixel interpolation for precise querying of ground movement profiles and raster statistics for area-based analyses were identified as key functionalities for the new development. Both bilinear and bicubic interpolation are required for subpixel interpolation in order to achieve optimum results depending on the accuracy requirements. Data management was also comprehensively analysed. This showed that cloud-optimised GeoTIFF files in combination with cloud object storage are the best solution for efficient storage of the raster-based ground motion model. A grid resolution of 100 metres was found to be particularly efficient. For supplementary vector data, the FlatGeobuf format was evaluated as optimal, as it enables both fast retrieval and efficient storage.

Based on the requirements analysis and the findings from the analysis, a cloud-native architecture was implemented in the IBM Based on the requirements analysis and the findings from the analysis, a cloud-native architecture was implemented in the IBM Cloud. This architecture comprises the infrastructure, software architecture and automated data management. The infrastructure essentially consists of IBM Cloud Object Storage, which is divided into two buckets: one for untransformed raw data and one for cloud-optimised, productive data. In addition, the serverless platform IBM Cloud Code Engine is used, which serves as the basis for a microservice architecture. This provides the identified functions of subpixel interpolation and raster statistics in an efficient and scalable manner. In combination with Cloud Object Storage, it also enables automated data management, in which conventionally used data formats are converted into cloud-optimised formats using event-driven jobs. Although this architecture and its components were primarily developed for the BOB.NI, it is generically designed so that it can also be used for other grid-based applications.

The evaluation of the architecture with regard to the specified requirements showed that most of the requirements were met, although there is still potential for optimisation in terms of resource utilisation, security aspects and the implementation of a continuous integration/continuous deployment pipeline. For future development, it is recommended that these optimisations be implemented in a targeted manner and that the architecture be transferred to a production-ready environment.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis des Quelltextes

Abkürzungsverzeichnis

AFIS	Amtliches Festpunktinformationssystem
API	Programmierschnittstelle (engl. Application Programming Interface)
ASGI	Asynchronous Server Gateway Interface
BGR	Bundesanstalt für Geowissenschaften und Rohstoffe
BOB.NI	Bodenbewegungsdienst Niedersachsen
CI/CD	Continuous Integration/Continuous Deployment
CNCF	Cloud Native Computing Foundation
COG	Cloud-optimiertes GeoTIFF
COS	Cloud Object Storage
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CSC	Cloud Service Customer
CSP	Cloud Service Provider
DaaS	Database as a Service
DHHN	Deutsches Haupthöhennetz
DOM	Digitalen Oberflächenmodell
FaaS	Function as a Service
FGB	FlatGeobuf
GDAL	Geospatial Data Abstraction Library
GeoTIFF	Geography Tagged Image File Format
GIS	Geoinformationssystem
GLONASS	Global'naya Navigatsioannaya Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IaC	Infrastructure as Code
IBM CE	IBM Cloud Code Engine
IBM CR	IBM Cloud Container Registry
IEC	International Electrotechnical Commission
IaaS	Infrastructure as a Service
InSAR	Radarinterferometrie
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LGLN	Landesamt für Geoinformation und Landesvermessung Niedersachsen
LoS	Line of Sight

MVP	Minimum Viable Product
NIST	National Institute Of Standards and Technology
OGC	Open Geospatial Consortium
PaaS	Platform as a Service
PS	Persistent Scatterern
PSI	Persistent Scatterer Interferometry
REST	Representational State Transfer
S3	Simple Storage Service
SAPOS	Satellitenpositionierungsdienst der deutschen Landesvermessung
SAR	Synthetic Aperture Radar
SaaS	Software as a Service
SPA	Single-Page-Applikation
TIFF	Tagged Image File Format
TIN	Triangulated Irregular Network
URI	Uniform Resource Identifier
UUID	Universell eindeutiger Bezeichner
VKV	Vermessungs- und Katasterverwaltung
VM	Virtuelle Maschinen
WKT	Well-known Text
WORM	Write-Once-Read-Many
YAML	Yet Another Markup Language
vCPU	virtuelle CPU

1 Einleitung

Kaum ein Begriff wurde in der IT-Branche in den letzten Jahren so intensiv diskutiert wie Cloud-Computing. Vor allem der Cloud-native Ansatz, der Anwendungen gezielt für die Cloud-Umgebung optimiert, hat sich als wegweisend etabliert. Die damit verbundenen Vorteile wie schnellere Skalierbarkeit, verteilter Zugriff auf Ressourcen und geringere Kosten haben dazu beigetragen, dass Cloud-Plattformen einen unverzichtbaren Bestandteil moderner IT-Architekturen darstellen. Auch im Kontext der Digitalisierung spielt Cloud-Computing eine zentrale Rolle. Das Land Niedersachsen hat im Rahmen seiner IT-Strategie die Modernisierung der IT-Infrastruktur in allen Bereichen als Ziel gesetzt. Aufbauend darauf hat auch das Landesamt für Geoinformation und Landesvermessung Niedersachsen (LGLN) eine IT-Strategie entwickelt, welche darauf abzielt, moderne und zukunftssichere IT-Lösungen zu implementieren. Ein zentraler Aspekt dieser IT-Strategie ist die Entscheidung, alle Neuentwicklungen nach dem Prinzip einer Microservice-Architektur innerhalb einer Cloud-Umgebung zu gestalten. Insbesondere setzt das LGLN dabei auf die Einhaltung von Architekturprinzipien wie dem Cloud-nativen Ansatz, der in der Strategie verankert ist (LGLN, 2022).

Neben der Modernisierung der IT-Infrastruktur hat das LGLN auch geodätische Herausforderungen zu bewältigen. Insbesondere die fortlaufende Bewegung der Erdoberfläche, welche sich aus geologischen und anthropogenen Aktivitäten ergeben, ist eine dieser Herausforderungen. So sind nach Jahn et al., 2011 etwa 30 % der Fläche von Niedersachen Einflüssen von Bodenbewegungsphänomenen ausgesetzt. Diese horizontalen und vertikalen Bewegungen wirken sich direkt auf die Festpunktfelder des LGLN aus und können bei der Nutzung des Landesbezugssystems zu erheblichen Netzspannungen führen. Um diese Risiken zu minimieren, ist eine kontinuierliche Überwachung der Bodenbewegungen unverzichtbar. Zur Bewältigung dieser Herausforderungen wurde von Brockmeyer et al., 2020 eine Prozesskette entwickelt, die eine flächendeckende Modellierung der Bodenbewegungen in Niedersachsen ermöglicht. Diese Prozesskette basiert auf Radarinterferometrie (InSAR)-Daten, die von der Bundesanstalt für Geowissenschaften und Rohstoffe (BGR) bereitgestellt werden, und liefert unter anderem präzise, rasterbasierte Bodenbewegungsdaten für ganz Niedersachsen. Ein zentrales Ziel des Zukunftskonzepts Vermessungs- und Katasterverwaltung (VKV) 2025 ist es, diese Bodenbewegungsdaten über einen eigenen Bodenbewegungsdienst Niedersachsen (BOB.NI) zugänglich zu machen (LGLN, 2017). Dieser Dienst soll sowohl Fachanwendern[1] als auch Privatnutzern einen einfachen Zugang zu den flächenhaften Bodenbewegungsinformationen bieten. Durch die webbasierte Schnittstelle können Nutzende gezielt Informationen visualisieren und abfragen, um diese für Planungszwecke, die Überprüfung von Messergebnissen oder zur Unterstützung von Entscheidungsprozessen zu verwenden.

Zu diesem Zweck wurde innerhalb des LGLN ein Prototyp zur Bereitstellung der Bodenbewegungsdaten entwickelt. Dieser erfüllt jedoch nicht die Anforderungen der aktuellen IT-Strategie, insbesondere hinsichtlich Skalierbarkeit und Flexibilität. Ziel dieser Masterarbeit ist es daher, eine Cloud-native Architektur zu konzipieren, die den Vorgaben der IT-Strategie entspricht und den Einsatz moderner Cloud-native Technologien sicherstellt. Diese Architektur soll nicht nur eine effiziente Verwaltung und Verarbeitung rasterbasierter Bodenbewegungsdaten ermöglichen, sondern auch die Skalierbarkeit der Anwendung gewährleisten. Primär soll die Architektur auf den BOB.NI ausgerichtet sein, wobei die einzelnen Komponenten auch für andere rasterbasierte Bodenbewegungsdaten oder ähnliche Rasterdaten anwendbar sein sollen. Zudem soll der BOB.NI ganzheitlich betrachtet werden, indem

[1] Zur besseren Lesbarkeit wird in dieser Masterarbeit das generische Maskulinum verwendet, der jedoch alle Geschlechter gleichermaßen umfasst.

nicht nur die Bereitstellung, sondern auch die vorherige Prozessierung der Daten berücksichtigt wird. Auf diese Weise soll eine langfristige Erweiterbarkeit und eine nachhaltige Nutzung der Architektur und ihrer Komponenten für vergleichbare Anwendungsfälle innerhalb des LGLN ermöglicht werden.

Um ein umfassendes Verständnis der zugrunde liegenden Technologien zu schaffen, werden im ersten Teil dieser Arbeit die Grundlagen von Cloud-Computing sowie den Cloud-nativen Technologien und Konzepten dargestellt. Anschließend wird der Kontext der Arbeit durch eine Beschreibung des BOB.NI vertieft. Dieser Abschnitt beinhaltet eine Einführung in die Grundlagen von Bodenbewegungen, deren Ursachen und die verschiedenen Methoden zu deren Erfassung. Zusätzlich werden die rasterbasierten Bodenbewegungsdaten beschrieben, welche als zentrale Datengrundlage dieser Arbeit dienen. Darauf aufbauend folgt eine ausführliche Anforderungsanalyse, die zunächst den aktuellen IST-Zustand des BOB.NI im LGLN beschreibt. Im Anschluss daran werden die funktionalen und Qualitätsanforderungen an die Architektur des BOB.NI, basierend auf Stakeholdern, Dokumenten und bestehenden Systemen, abgeleitet. Im weiteren Verlauf der Arbeit wird ein ausführliches Analysekapitel präsentiert. Zunächst erfolgt eine detaillierte Untersuchung des bestehenden Prototyps zur Bereitstellung der Bodenbewegungsdaten. Im Anschluss wird eine umfassende Analyse durchgeführt, wie eine effiziente und präzise Datenabfrage der rasterbasierten Bodenbewegungsdaten ermöglicht werden kann. Zusätzlich wird die Eignung Cloud-optimierter Geodatenformate für den BOB.NI untersucht. Den Abschluss bildet eine Evaluierung der implementierten Lösungen, wobei die zuvor definierten Anforderungen als Bewertungsmaßstab dienen. Abschließend wird ein Fazit gezogen, das die wichtigsten Erkenntnisse zusammenfasst, gefolgt von einem Ausblick auf mögliche zukünftige Entwicklungen und Erweiterungen der Architektur, die auch für ähnliche Projekte des LGLN von Bedeutung sein könnten.

2 Grundlagen

Im Folgenden werden die wichtigsten theoretischen Grundlagen dargestellt, um ein umfassendes Verständnis für die in dieser Masterarbeit verwendeten Konzepte und Überlegungen zu schaffen. Zunächst wird erläutert, was unter Cloud-Computing zu verstehen ist. Anschließend werden wesentliche Konzepte und Technologien von Cloud-nativen Anwendungen beschrieben. Darauf aufbauend werden Cloud-native Architekturkonzepte beschrieben und näher erläutert. Abschließend werden Cloud-optimierte Geodatenformate vorgestellt, die innerhalb der Arbeit eine wesentliche Rolle spielen.

2.1 Cloud-Computing

Cloud-Computing hat sich in den letzten Jahren als eine der bedeutendsten technologischen Entwicklungen im Bereich der IT etabliert. Es hat die Art und Weise, wie Unternehmen und Privatpersonen IT-Ressourcen nutzen, Daten speichern und Anwendungen bereitstellen, revolutioniert. Die steigenden Nutzerzahlen sowie die fortschreitende Digitalisierung haben zu einer signifikanten Steigerung des Bedarfs an Rechen- und Speicherressourcen geführt. In diesem Zusammenhang stoßen traditionelle Datenverarbeitungsmodelle, wie das bewährte Client/Server-Modell, zunehmend an ihre Grenzen und erweisen sich als unzureichend, um den modernen Anforderungen gerecht zu werden. Im Gegensatz dazu bietet Cloud-Computing flexible und skalierbare Lösungen, die diese Anforderungen teilweise oder sogar vollständig erfüllen können (Hentschel & Leyh, 2018). Nach der Definition des amerikanischen National Institute Of Standards and Technology (NIST) beschreibt Cloud-Computing ein Modell, das allgegenwärtigen, bequemen und bedarfsgesteuerten Netzwerkzugriff auf einen gemeinsam genutzten Pool konfigurierbarer Rechenressourcen (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste) ermöglicht. Diese Ressourcen können mit minimalem Verwaltungsaufwand und minimaler Interaktion mit dem Dienstanbieter schnell bereitgestellt und freigegeben werden (Mell & Grance, 2011). Vereinfacht gesagt bedeutet dies, dass Cloud-Computing die Nutzung von IT-Diensten durch einen Cloud Service Customer (CSC) bei einem Cloud Service Provider (CSP) beschreibt. Die Dienste werden dem CSC in Echtzeit über das Internet bereitgestellt und nach der Nutzung abgerechnet (BITKOM, 2009). Dabei kann es sich um Rechenressourcen, Speicherplatz oder spezifische Software handeln, die je nach Bedarf skaliert werden kann (Münzl et al., 2015). Dadurch können Unternehmen und Organisationen ihre Investitionskosten in laufende Betriebskosten umwandeln, da teure Investitionen in Hardware entfallen.

Um ein tiefergehendes Verständnis von Cloud-Computing zu erlangen, wird in den folgenden Kapiteln Cloud-Computing näher beschrieben. Die Kapitel orientieren sich an der Definition des NIST, welche Cloud-Computing durch fünf zentrale Merkmale, drei Servicemodelle und vier Bereitstellungsmodelle charakterisiert. Die genannten Elemente stellen eine wesentliche Grundlage des Cloud-Computings dar und sind daher essenziell zum Verständnis dieser Arbeit.

2.1.1 Charakteristika

Nach der Definition von Mell und Grance, 2011 besitzt Cloud-Computing fünf charakteristische Eigenschaften, die es von traditionellen IT-Modellen unterscheiden. Diese Eigenschaften stellen eine wesentliche Grundlage für das Verständnis und die Implementierung von Cloud-Computing-Technologien dar und bilden zudem die Basis für die Entwicklung moderner, skalierbarer und flexibler IT-Infrastrukturen.

- **On-Demand self-service**: On-Demand self-service ist ein zentrales Merkmal des Cloud-Computing, welches es dem CSC ermöglicht, Rechenkapazitäten selbstständig und bedarfsgerecht bereitzustellen, ohne dass eine menschliche Interaktion mit dem CSP erforderlich ist (Mell & Grance, 2011). CSC können jederzeit über eine Programmierschnittstelle (engl. Application Programming Interface) (API) auf Rechenressourcen wie Serverzeit, Netzwerkspeicher und Anwendungen zugreifen. Diese Automatisierung verkürzt die Vorlaufzeit für die Ressourcenbereitstellung, verbessert die Agilität und Flexibilität in Organisationen und ermöglicht eine kosteneffiziente Nutzung der Ressourcen, da der CSC nur für die tatsächlich genutzten Ressourcen bezahlen.

- **Broad Network Access**: Broad Network Access ist ein weiteres wesentliches Merkmal des Cloud-Computing, das sich auf die Fähigkeit bezieht, über Standardschnittstellen auf die Cloud-Dienste zuzugreifen (Mell & Grance, 2011). Dies umfasst den Zugang über das Internet und die Unterstützung für eine Vielzahl von Endgeräten wie Laptops, Smartphones, Tablets und stationäre Computer (Diaby & Rad, 2017).

- **Ressource Pooling**: Ressource Pooling beschreibt die Möglichkeit von CSP, die physischen und virtuellen Ressourcen zu konsolidieren, um diese effizient und bedarfsgerecht einer Vielzahl von CSC bereitzustellen (Mell & Grance, 2011). Die Rechenressourcen werden gebündelt, um mehrere CSC zu bedienen. Dabei werden verschiedene physische und virtuelle Ressourcen je nach Verbraucherbedarf dynamisch zugewiesen. Um eine Trennung der Daten und Anwendungen zu gewährleisten, werden bestimmte, hier nicht näher thematisierte Sicherheitsmechanismen eingesetzt.

- **Rapid Elasticity**: Rapid Elasticity ist die Fähigkeit von Cloud-Diensten, schnell und nahtlos auf sich ändernde Nachfragen nach Ressourcen zu reagieren. Diese Eigenschaft erlaubt es den Nutzern, die Menge der genutzten Ressourcen dynamisch zu erhöhen oder zu verringern, um den aktuellen Anforderungen gerecht zu werden, ohne dabei den laufenden Betrieb zu unterbrechen (Mell & Grance, 2011). Die Elastizität des Cloud-Computing ermöglicht es Unternehmen, sowohl auf kurzfristige Lastspitzen als auch auf langfristige Veränderungen in der Ressourcennachfrage flexibel zu reagieren. Dies trägt nicht nur zur Kosteneffizienz bei, sondern stellt auch sicher, dass die Dienste kontinuierlich und zuverlässig verfügbar bleiben.

- **Measured Service**: Cloud-Systeme steuern und optimieren automatisch die Ressourcennutzung, indem sie eine Messfunktion auf einer geeigneten Abstraktionsebene für den jeweiligen Diensttyp verwenden (z.B. Speicher, Verarbeitung, Bandbreite und aktive Benutzerkonten). Die Ressourcennutzung kann überwacht, gesteuert und berichtet werden, was sowohl für den CSP als auch für den CSC des genutzten Dienstes Transparenz schafft (Mell & Grance, 2011).

2.1.2 Service-Modelle

Neben den unterschiedlichen Charakteristika unterscheiden Mell und Grance, 2011 drei wesentliche Service-Modelle: Software as a Service (SaaS), Platform as a Service (PaaS) sowie Infrastructure as a Service (IaaS). Diese Modelle bestimmen die Ebene, auf der Cloud-Services bereitgestellt werden, sowie die Verantwortlichkeiten und Kontrollen über die IT-Ressourcen. Dabei repräsentieren die Modelle den Grundsatz von Cloud-Computing, dass Software, Plattform und Infrastruktur als aufeinander aufbauende Schichten verstanden werden können (Lins & Sunyaev, 2018). Die unteren Schichten dienen als Grundlage für die höheren Ebenen, wodurch eine klare Trennung der Verantwortlichkeiten zwischen dem CSP und dem CSC erreicht wird (Sirtl, 2010).

- **Infrastructure as a Service (IaaS)**: IaaS ist die unterste Abstraktionsebene und stellt die physische IT-Basisinfrastruktur in Form von Rechenkapazität, Speicherplatz und Netzwerk bereit. Diese wird durch Virtualisierung geteilt, zugewiesen und schließlich dem CSC nach Bedarf zur Verfügung gestellt (Hentschel & Leyh, 2018). Der CSC ist in der Lage beliebige Software einzusetzen und auszuführen, darunter auch Betriebssysteme und Anwendungen (Marinescu, 2023).
- **Platform as a Service (PaaS)**: PaaS ist die mittlere Abstraktionsebene und setzt direkt auf IaaS auf. Dabei bietet PaaS eine Plattform für Entwickler, um Anwendungen zu erstellen, zu testen und bereitzustellen, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen (Hentschel & Leyh, 2018). Der CSP übernimmt zusätzlich die Verantwortung für das Betriebssystem und die Laufzeitumgebung, wodurch der CSC lediglich die Kontrolle über die bereitgestellten Anwendungen und Daten hat.
- **Software as a Service (SaaS)**: SaaS ist die oberste Abstraktionsebene im Cloud-Computing und bietet Endnutzern vollständige Anwendungen, die über das Internet zugänglich sind (Hentschel & Leyh, 2018). Diese Anwendungen werden von CSP gehostet und verwaltet, sodass der CSC weder die Infrastruktur noch die Plattform verwalten muss.

Die einzelnen Service-Modelle mit den unterschiedlichen Verantwortlichkeiten sind in der Abbildung 1 dargestellt. Dabei sind die Schichten, bei denen die Verantwortung beim CSP liegt, grau und die Schichten, bei denen die Verantwortung beim CSC liegt, weiß dargestellt. Zusätzlich werden die Service-Modelle mit einer traditionellen IT verglichen. Im traditionellen IT-Modell liegt die gesamte Verantwortung für die Infrastruktur, das Betriebssystem, die Laufzeitumgebung und die Anwendungen beim Unternehmen selbst. Im Gegensatz dazu übernehmen im Cloud-Computing-Ansatz die CSP viele dieser Aufgaben, was zu einer erheblichen Entlastung der internen IT-Abteilung führt. Durch die Verlagerung dieser Aufgaben auf spezialisierte Anbieter profitieren Unternehmen und Organisationen von höherer Flexibilität, schnelleren Anpassungsmöglichkeiten an veränderte Geschäftsanforderungen und einer verbesserten Kosteneffizienz. Die Service-Modelle verdeutlichen somit, wie sich durch Cloud-Computing die Zuständigkeiten verschieben und welche operativen Vorteile sich daraus im Vergleich zu traditionellen IT-Systemen ergeben.

Neben den in der NIST-Definition aufgeführten Servicemodellen gibt es weitere Servicemodelle, wie z.B. Database as a Service (DaaS) oder Function as a Service (FaaS). FaaS hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Dabei beschreibt FaaS eine Form des serverlosen Cloud-Computing und definiert sich durch die Ausführung ereignisgesteuerter Funktionen (Scheuner & Leitner, 2020). FaaS ermöglicht es Entwicklern, Code in Form einzelner Funktionen zu schreiben und bereitzustellen, die bei Bedarf ausgeführt werden (Tremp, 2021). Diese Funktionen laufen in sogenannten serverlosen Umgebungen (siehe Kapitel 2.3.2), sodass sich Entwickler nicht um die zugrunde liegende Infrastruktur kümmern müssen.

2.1.3 Deployment-Modelle

Nach der Betrachtung der verschiedenen Service-Modelle und ihrer Vorteile im Hinblick auf die Verteilung der Verantwortlichkeiten ist es sinnvoll, einen Blick auf die verschiedenen Cloud-Bereitstellungsmodelle zu werfen. Diese Modelle bestimmen, wie und wo die Cloud-Infrastruktur bereitgestellt wird und welche Art von Zugang und Kontrolle Organisationen darüber haben. Dabei legt die Definition von NIST vier Deployment-Modelle fest (Mell & Grance, 2011):

- **Private Cloud**: In einer Private Cloud wird die Cloud-Infrastruktur ausschließlich für eine einzelne Institution betrieben. Diese Infrastruktur kann entweder von der Institution selbst

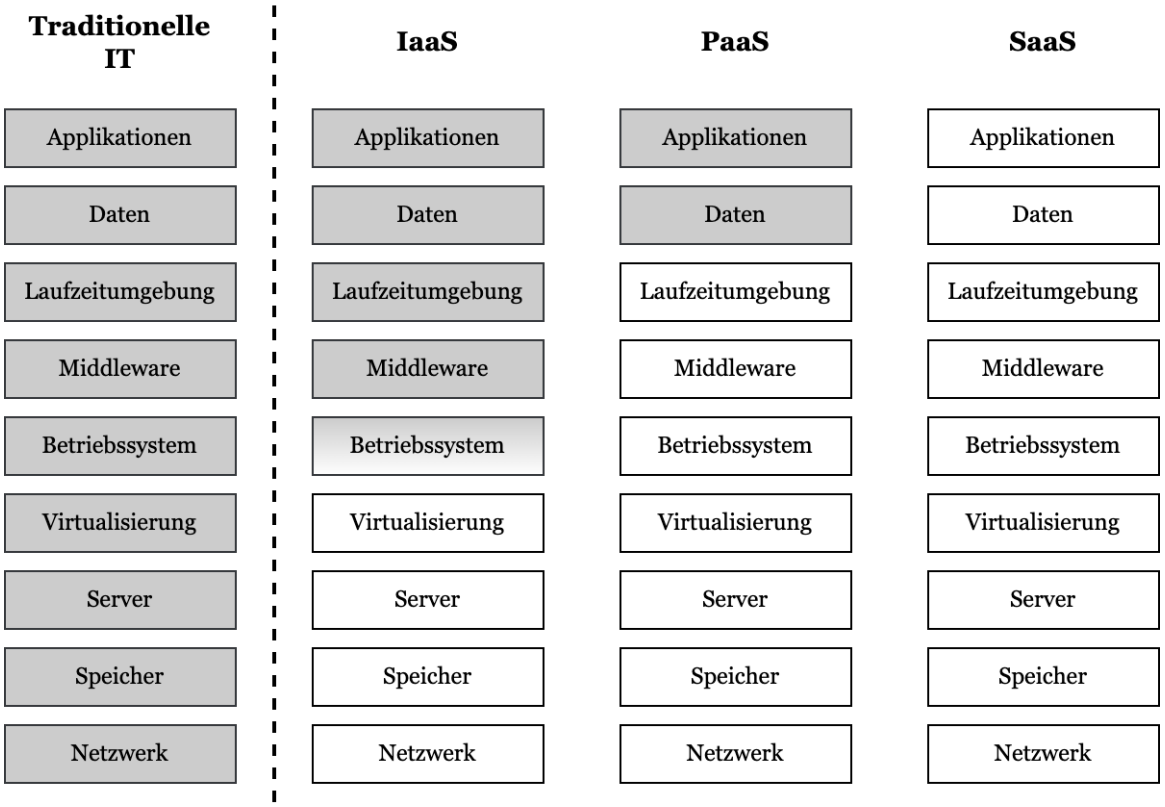


Abbildung 1: *Vergleich der unterschiedlichen Cloud-Service-Modelle (eigene Darstellung nach Harms und Yamartino, 2010)*

oder von einem externen Dienstleister organisiert und verwaltet werden. Der physische Standort der Infrastruktur kann entweder im eigenen Rechenzentrum der Institution oder in dem eines Dritten sein.

- **Public Cloud**: Eine Public Cloud hingegen stellt Cloud-Services der Allgemeinheit oder einer großen Nutzergruppe zur Verfügung, wie z.B. einer ganzen Industriebranche. Diese Services werden von einem Cloud-Anbieter bereitgestellt und verwaltet, wodurch die Nutzer keinen direkten Zugriff auf die physische Infrastruktur haben.
- **Community Cloud**: Eine Community Cloud wird von mehreren Institutionen genutzt, die gemeinsame Interessen oder Anforderungen haben. Diese Cloud-Infrastruktur kann entweder von einer der teilnehmenden Institutionen oder von einem externen Dienstleister betrieben werden. Sie ermöglicht den beteiligten Institutionen die gemeinsame Nutzung von Ressourcen und Services, um spezifische Bedürfnisse und Ziele zu erfüllen.
- **Hybrid Cloud** Eine Hybrid Cloud kombiniert mehrere eigenständige Cloud-Infrastrukturen, die über standardisierte Schnittstellen miteinander verbunden sind. Diese hybride Umgebung ermöglicht es, Workloads und Daten zwischen den verschiedenen Cloud-Typen (Private, Public und Community) nahtlos zu verschieben und zu integrieren, um Flexibilität und Skalierbarkeit zu gewährleisten.

2.2 Cloud-native Konzepte und Technologien

Mit dem Aufkommen von Cloud-Computing und der zunehmenden Bedeutung von Skalierbarkeit und Agilität hat sich der Begriff Cloud-native etabliert. Cloud-native Anwendungen sind Anwendungen, die so konzipiert, entwickelt und bereitgestellt werden, dass sie in erster Linie die Vorteile des Cloud-Computing nutzen (Surianarayanan & Chelliah, 2019). In diesem Zusammenhang definiert die Cloud Native Computing Foundation (CNCF), dass Cloud-native Technologien und Konzepte es Unternehmen ermöglichen, skalierbare Anwendungen in modernen, dynamischen Umgebungen zu implementieren und zu betreiben. Dabei kann es sich um öffentliche, private oder hybride Clouds handeln (Cloud Native Computing Foundation, 2023).

Cloud-native Anwendungen werden normalerweise mit bestimmten Prinzipien und Technologien entwickelt, um die Skalierbarkeit, Flexibilität und Robustheit in einer Cloud-Umgebung zu verbessern. In diesem Zusammenhang haben sich Konzepte wie Elastizität und Skalierbarkeit sowie Technologien wie Containerisierung, Orchestrierung und Infrastructure as Code (IaC) durchgesetzt. Diese Technologien und Praktiken bieten die nötige Flexibilität und Effizienz, um auf wechselnde Anforderungen schnell reagieren zu können. Zudem haben sich auf Basis der in diesem Kapitel beschriebenen Konzepte und Technologien Cloud-native Architekturen entwickelt, die im Kapitel 2.3 näher erläutert werden.

2.2.1 Elastizität und Skalierbarkeit

Ein wesentliches Merkmal Cloud-nativer Anwendungen ist seine Elastizität und Skalierbarkeit, die sich sowohl auf die Datenhaltung als auch auf die Datenverarbeitung beziehen kann (Hentschel & Leyh, 2018). Der Begriff der Skalierbarkeit bezeichnet die Fähigkeit eines Systems, seine Leistung dynamisch und automatisiert an unterschiedliche Anforderungen anzupassen. Dies umfasst sowohl Veränderungen der Last, Anpassung an Nutzerbedingungen als auch eine optimale Nutzung der Ressourcen. Die Fähigkeit zur Anpassung spielt vor allem bei hoher Auslastung, etwa durch eine Vielzahl von HTTP-Anfragen, eine entscheidende Rolle. Bei geringer Auslastung können Ressourcen effizienter genutzt oder freigegeben werden. Grundsätzlich wird zwischen horizontaler und vertikaler Skalierung unterschieden. Horizontale Skalierung bedeutet, dass zusätzliche Ressourcen durch Hinzufügen weiterer Instanzen oder Server erhöht (Scaled-Out) oder reduziert (Scaled-In) werden, während die Ressourcen innerhalb der einzelnen Instanzen unverändert bleiben (Quix, 2021). Vertikale Skalierung hingegen bedeutet, dass die Anzahl der Instanzen konstant bleibt, während die Ressourcen dieser Instanzen verkleinert (Scaled-Down) oder vergrößert (Scaled-Up) werden, bspw. durch Anpassung der Anzahl der Prozessorkerne oder der Größe des Arbeitsspeichers. Diese Art der Skalierung ist aus Anwendungssicht in der Regel einfacher durchzuführen, da die Software in der Regel nicht geändert werden muss, um von den zusätzlichen Ressourcen zu profitieren. Allerdings sind der vertikalen Skalierung relativ enge Grenzen gesetzt, da die Ressourcen eines Systems nicht beliebig erweitert werden können (Quix, 2021).

Die Besonderheit von Cloud-Computing ist, dass sowohl horizontale als auch vertikale Skalierung unterstützt werden. Dies ermöglicht eine hohe Flexibilität und Effizienz bei der Ressourcennutzung. CSP bieten automatisierte Skalierungsfunktionen, die auf Grundlage unterschiedlicher Parameter wie Auslastung der Central Processing Unit (CPU) oder Arbeitslast basieren (Goniwada, 2022). Dabei spielt die Containerisierung eine wichtige Rolle, da sie es ermöglicht, Anwendungen konsistent und isoliert zu betreiben.

2.2.2 Containerisierung

Ein wesentlicher Aspekt von Cloud-nativen Anwendungen ist die Containerisierung. Die Containerisierung ist eine Methode zur Virtualisierung auf Betriebssystemebene, bei der Anwendungen und ihre Abhängigkeiten in isolierte, leichtgewichtige Container verpackt werden (Goniwada, 2022). Diese Container enthalten alles was benötigt wird, um die Anwendung auszuführen, einschließlich Bibliotheken, Abhängigkeiten und Konfigurationsdateien, und sind dadurch unabhängig von der zugrunde liegenden Infrastruktur (Feil et al., 2023). Im Gegensatz zu herkömmlichen Virtuelle Maschinen (VM) teilen Container jedoch denselben Kernel des Host-Betriebssystems, anstatt ein vollständiges Betriebssystem zu duplizieren. Dadurch sind Container wesentlich leichtgewichtiger als VM. Ein grafischer Vergleich der Architektur von VM und Containern ist in Abbildung 2 zu sehen.

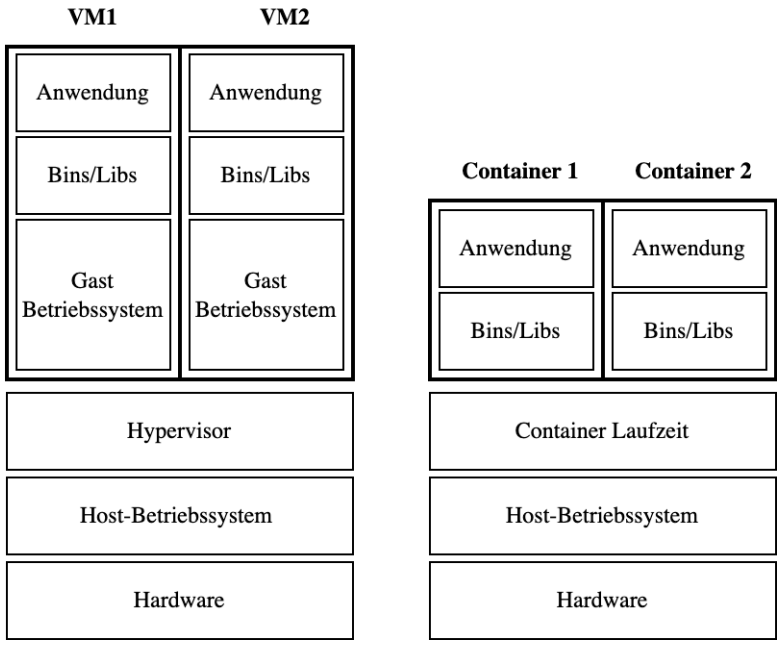


Abbildung 2: *Vergleich der Architektur von VM und Containern (eigene Darstellung nach Goniwada, 2022)*

Die Containerisierung bietet dadurch grundsätzlich mehrere Vorteile für Anwendungen (IBM, 2024c):

- **Hohe Portabilität und Konsistenz:** Container gewährleisten eine einheitliche Funktionalität über verschiedene Umgebungen hinweg, sodass eine Anwendung, die in einem Container auf einem Entwicklungssystem funktioniert, mit hoher Wahrscheinlichkeit auch in einem Produktionssystem funktioniert.
- **Skalierbarkeit:** Container lassen sich aufgrund ihrer kurzen Start- und Stoppzeiten leicht horizontal skalieren, was die Skalierbarkeit von Anwendungen erheblich verbessert.
- **Effiziente Ressourcennutzung:** Da Container weniger Ressourcen benötigen als virtuelle Maschinen, können mehr Container auf demselben Hardware-Server betrieben werden, was zu einer besseren Ausnutzung der vorhandenen Ressourcen führt.
- **Isolation:** Container laufen in einer sicheren und unabhängigen Weise, was die Stabilität und Sicherheit der Gesamtanwendung erhöht.

Die wohl bekannteste Software zur Containerisierung ist Docker (Riti, 2018). Docker automatisiert die Bereitstellung von Anwendungen in Containern und fungiert in einer Container-Umgebung als zusätzliche Schicht einer Bereitstellungs-Engine (Rad et al., 2017). Docker wurde entwickelt, um eine schnelle und leichtgewichtige Umgebung zu bieten, in der Code effizient ausgeführt werden kann. Darüber hinaus bietet Docker eine effiziente Methode, um Code vor der Produktion zu testen (Rad et al., 2017). Docker besteht im Wesentlichen aus drei Hauptkomponenten, die im Zusammenspiel die gesamte Funktionalität ermöglichen (siehe Abbildung 3): Images, Registries und Container. Ein Docker-Image dient als Container, in dem die Anwendung zusammen mit ihrer Laufzeitumgebung gebündelt ist. Ein Docker-Image enthält das Dateisystem, das der Anwendung zur Verfügung steht, sowie Metadaten wie den Pfad zur ausführbaren Datei, die beim Start des Images verwendet wird. Das Image wird mithilfe einer Dockerfile erstellt, die alle notwendigen Anweisungen und Konfigurationen zur Installation und Ausführung der Anwendung enthält. Registries dienen als zentrale Speicherorte für Docker-Images, um die gemeinsame Nutzung zwischen verschiedenen Benutzern und Systemen zu ermöglichen. Nach der Erstellung kann ein Image entweder lokal ausgeführt oder in eine Registry hochgeladen werden. Von dort aus lässt es sich auf anderen Computern herunterladen und ausführen. Es gibt sowohl öffentliche Registries, die für jedermann zugänglich sind, als auch private Registries, die nur bestimmten Benutzern oder Systemen vorbehalten sind. Ein Docker-Container ist die laufende Instanz eines Docker-Images. Wie bereits erwähnt, stellt ein Container einen isolierten Prozess dar, der auf einem Docker-fähigen System läuft. Diese Isolation schützt sowohl den Host als auch andere Prozesse. Darüber hinaus ist der Zugriff auf Ressourcen begrenzt, sodass der Container nur die ihm zugewiesenen Ressourcen nutzen kann.(Lukša, 2018)

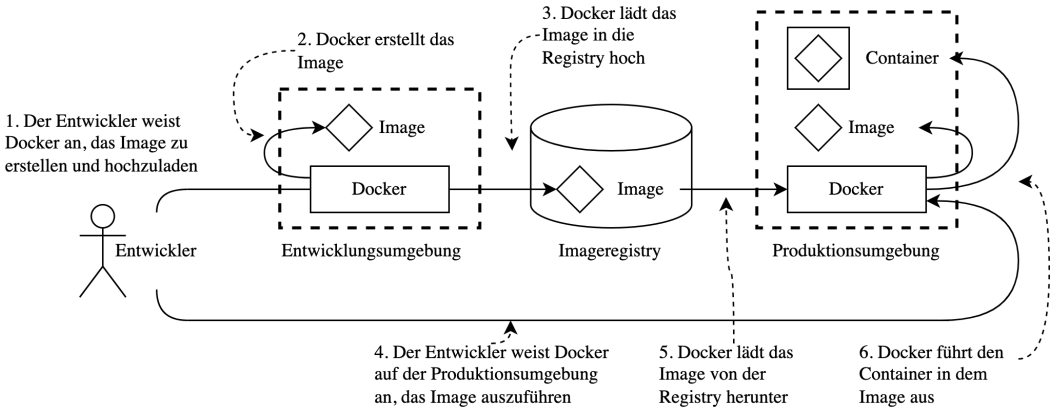


Abbildung 3: *Docker-Images erstellen, verteilen und ausführen (eigene Darstellung nach Lukša, 2018)*

Docker bietet zahlreiche Vorteile, die es zum Industriestandard gemacht haben. Einer der größten Vorteile ist die Geschwindigkeit. Aufgrund dessen, dass Container schnell erstellt und bereitgestellt werden können, kann die Entwicklung, das Testen und das Deployment beschleunigt werden. Zudem sind Docker-Container äußerst portabel, sodass Anwendungen problemlos zwischen verschiedenen Umgebungen verschoben werden können. Ein weiterer Vorteil ist die Skalierbarkeit. Durch die schnelle Erstellung von Containern ermöglicht Docker eine leichte horizontale Skalierbarkeit, sodass mehrere Instanzen derselben Anwendung problemlos auf verschiedenen Servern oder in unterschiedlichen Umgebungen parallel betrieben werden können. Schließlich ist Docker ressourceneffizient, da es ohne Hypervisor auskommt, wodurch mehr Container auf einem Host ausgeführt werden können als bei herkömmlichen virtuellen Maschinen. (Rad et al., 2017)

2.2.3 Container-Orchestrierung

Moderne, komplexe Cloud-native Anwendungen bestehen in der Regel nicht aus einem einzelnen Container, sondern aus einer Kombination von mehreren Containern (Riti, 2018). Solche zusammengesetzten Anwendungen basieren i.d.R. auf einer Microservice-Architektur (siehe Kapitel 2.3.1). Mit der steigenden Anzahl von Containern wächst jedoch auch die Notwendigkeit, diese effizient zu verwalten, zu skalieren und zu überwachen (Lukša, 2018). Die Container-Orchestrierung automatisiert und vereinfacht den Prozess der Bereitstellung, Verwaltung, Skalierung und Vernetzung von Containern. Sie kann in jeder Umgebung, sei es Public Cloud, Private Cloud oder Hybrid Cloud, eingesetzt werden. Container-Orchestrierungs-Tools verwenden in der Regel Yet Another Markup Language (YAML)- oder JavaScript Object Notation (JSON)-Dateien, welche die Konfiguration der Anwendung beschreiben. Diese Dateien werden verwendet, um Container-Images abzurufen, Netzwerke zwischen Containern zu erstellen, Protokolldaten zu speichern und Storage-Volumes zu fixieren (Goniwada, 2022). Des Weiteren verwalten diese Tools die Einsatzplanung von Containern in Clustern und identifizieren automatisch den richtigen Host (Goniwada, 2022). Tools für die Container-Orchestrierung bieten üblicherweise folgende Hauptfunktionen (Casalicchio, 2019):

- **Ressourcenkontrolle**: Reservierung von CPU und Speicher für Container, um Ressourcenbeschränkungen zu setzen und Interferenzen zu minimieren.
- **Planung**: Platzierung von Containern auf Knoten basierend auf Ressourcenkriterien oder Knotenzugehörigkeit.
- **Lastverteilung**: Verteilung der Arbeitslast auf mehrere Containerinstanzen.
- **Gesundheitsprüfung**: Überwachung, ob Container Anfragen beantworten können, einschließlich Überprüfung von Verbindungen und HTTP-Anfragen.
- **Fehlertoleranz**: Verwaltung von Container-Replikationen und Hochverfügbarkeit, um die gewünschte Anzahl von Containern zu gewährleisten und Orchestrator-Ausfälle zu bewältigen.
- **Autoskalierung**: Automatische Anpassung der Anzahl von Containern basierend auf CPU- und Speichernutzung.

Für die Orchestrierung von Containern gibt es eine Vielzahl von Lösungen, wie z.B. Docker Swarm oder Mesosphere Marathon. Ähnlich wie Docker hat sich jedoch Kubernetes in den letzten Jahren als Industriestandard etabliert. Kubernetes ist eine von Google entwickelte Open-Source-Plattform, die von der Cloud Native Computing Foundation verwaltet wird (Riti, 2018). Die Hauptaufgabe von Kubernetes ist die einfache Bereitstellung und Verwaltung von containerisierten Anwendungen. Kubernetes ermöglicht die automatische Bereitstellung, Überwachung und Verwaltung von Containern in einer Produktionsumgebung über mehrere Hosts hinweg. Darüber hinaus übernimmt Kubernetes den Umgang mit ausgefallenen Containern sowie deren Skalierung und sorgt so für eine effiziente und zuverlässige Container-Orchestrierung.(Surianarayanan & Chelliah, 2019).

Kubernetes basiert auf einer Architektur, die in Cluster und Knoten unterteilt ist. Ein Cluster besteht aus einem Master-Knoten und mehreren Worker-Knoten, die entweder virtuelle oder physische Maschinen sein können. Der Master-Knoten übernimmt die Steuerung des Clusters und führt den Scheduler aus, welcher die Bereitstellung der Container basierend auf den Anforderungen und der verfügbaren Rechnerkapazität koordiniert (Lukša, 2018). Sobald ein Entwickler eine Liste von Anwendungen beim Master-Knoten einreicht, werden diese auf den Worker-Knoten des Clusters verteilt. Dabei ist es unerheblich, auf welchem Knoten eine bestimmte Komponente ausgeführt wird.

Der Entwickler kann jedoch angeben, dass bestimmte Anwendungen zusammen ausgeführt werden sollen, sodass sie auf demselben Worker-Knoten bereitgestellt werden. Die restlichen Anwendungen werden zufällig im Cluster verteilt. Unabhängig vom Standort der Anwendung ist die Kommunikation zwischen den Anwendungen im Cluster jederzeit gewährleistet (Lukša, 2018).

Auf den Worker-Knoten werden containerisierte Anwendungen bereitgestellt, ausgeführt und verwaltet. Jeder Worker-Knoten enthält ein Container-Management-Tool wie Docker sowie den Software-Agenten *Kubelet*, der die Befehle des Master-Knotens ausführt und direkt mit der Kubernetes-API kommuniziert (Lukša, 2018). Ein zentrales Konzept von Kubernetes sind Pods, welche einen oder mehreren Container umfassen, die sich Rechenressourcen und Netzwerkverbindungen teilen. Pods bilden die grundlegende Skalierungseinheit in Kubernetes. Bei hoher Auslastung eines Pods wird dieser auf andere Knoten repliziert, um die Last mithilfe eines Load-Balancers zu verteilen (Goniwada, 2022). Der Load-Balancer stellt sicher, dass eingehende Anfragen gleichmäßig auf die verschiedenen Instanzen des Pods verteilt werden.

Kubernetes ermöglicht es CSC, sich auf die Entwicklung von Funktionen und die Verbesserung der Benutzererfahrung zu konzentrieren, ohne sich mit der zugrunde liegenden Infrastruktur befassen zu müssen. Dadurch wird Kubernetes zu einem essenziellen Bestandteil moderner Cloud-Plattformen.

2.2.4 Cloud Object Storage

Die Verwaltung und Speicherung von Daten ist ein wesentlicher Bestandteil moderner Anwendungen. Mit dem Aufkommen von Cloud-nativen Architekturen hat sich die Art und Weise, wie Daten gespeichert und verwaltet werden, grundlegend verändert. Traditionelle Speicherlösungen, die für statische und zentralisierte Systeme ausgelegt sind, stoßen bei dynamischen und verteilten Anwendungen zunehmend an ihre Grenzen. Um diese Herausforderungen zu meistern, sind Speicherlösungen entstanden, die speziell für die Skalierbarkeit, Agilität und Zuverlässigkeit in modernen, verteilten Umgebungen entwickelt wurden. Diese Arbeit konzentriert sich auf Cloud Object Storage (COS), die sich als zentrale Komponente in Cloud-nativen Umgebungen etabliert haben, und betrachtet dabei nicht klassische Datenbanksysteme.

COS sind Speicherarchitekturen, bei denen Daten als Objekte verwaltet und gespeichert werden. Im Gegensatz zu herkömmlichen Dateisystemen, die Daten in hierarchischen Strukturen oder als Blöcke auf Festplatten organisieren, erfolgt die Speicherung in Objektspeichern auf einer objektbasierten Ebene. Jedes Objekt in einem solchen System besteht aus drei wesentlichen Komponenten (Patil et al., 2020):

- **Daten**: Die eigentlichen Nutzdaten, die gespeichert werden sollen, z.B. Dokumente, Bilder oder Videos.
- **Metadaten**: Informationen über die Daten, wie z.B. Erstellungszeit, Dateigröße oder benutzerdefinierte Attribute, die helfen, die Daten besser zu organisieren, zu verwalten und abzufragen.
- **Universell eindeutiger Bezeichner (UUID)**: Eine eindeutige Kennung, die jedes Objekt eindeutig identifiziert, sodass das System die Objekte effizient speichern und abrufen kann, ohne den genauen physischen Speicherort kennen zu müssen.

COS werden primär für die Speicherung unstrukturierter Daten verwendet. Unstrukturierte Daten sind Informationen, die nicht in einer vorgegebenen Datenbankstruktur organisiert sind. Dazu

gehören Multimedia-Dateien, Dokumente, E-Mails und Webinhalte. Die Organisation der Daten erfolgt dabei innerhalb von sogenannten Buckets. Ein Bucket ist eine logische Abstraktion, die zur Bereitstellung eines Datencontainers im Objektspeicher verwendet wird. Die Daten können auf einem beliebigen Server in einem beliebigen Rechenzentrum an verschiedenen geografischen Standorten gespeichert werden. Dabei kann sich der Speicherort der Daten von Zeit zu Zeit ändern, da der CSP die verfügbaren Speicherorte in den Datenzentren dynamisch verwaltet (Odun-Ayo et al., 2017). Aufgrund dieser Abstraktion vom physischen Speicher erfordern Buckets keine vorherige Zuweisung von Speicherkapazität. Die Trennung der Daten wird durch das Eigentum an den Buckets und einer Kombination aus öffentlichen und geheimen Schlüsseln erreicht, die an Objektspeicherkonten gebunden sind. Dieses Sicherheitsmodell gewährleistet, dass die Daten nur für Benutzer und Anwendungen sichtbar sind, die für den Zugriff autorisiert sind. Der Datenzugriff erfolgt über eine Representational State Transfer (REST)-API mittels HTTP-Protokoll, die einen orts- und zeitunabhängigen Zugriff durch einfachen Verweis auf die UUID ermöglicht (Patil et al., 2020). Eine weit verbreitete API für diesen Zweck ist die Amazon Simple Storage Service (S3)-API. Sie ermöglicht Entwicklern, Standardoperationen wie das Hochladen, Herunterladen und Verwalten von Datenobjekten durchzuführen und dabei von der Skalierbarkeit und Flexibilität von COS zu profitieren.

Aufgrund der genannten Eigenschaften bieten COS gewisse Vorteile. Einer der wichtigsten Vorteile ist die nahezu unbegrenzte Skalierbarkeit, die insbesondere für Cloud-native Anwendungen mit großen Datenmengen von entscheidender Bedeutung ist. COS können verschiedene Arten von Daten aufnehmen, einschließlich unstrukturierter Daten wie Multimediadateien, Backups und Logdateien sowie begrenzt strukturierter Daten. Darüber hinaus haben sich spezielle Cloud-native Datenformate entwickelt, die eine ideale Interoperabilität mit COS aufweisen (Abernathey et al., 2021). Durch die flache Architektur und die Möglichkeit, kostengünstige Hardware zu verwenden, sind die Kosten pro gespeicherter Einheit oft niedriger als bei traditionellen Speicherlösungen. Aufgrund der Latenzzeiten von COS kann die Zugriffszeit jedoch höher sein als bei traditionellen Speichermethoden (Patil et al., 2020).

2.2.5 Infrastructure as Code

Cloud-Infrastrukturen können äußerst komplex sein, da sie eine Vielzahl von Servern, Betriebssystemen, Speicherlösungen und andere Ressourcen umfassen. IaC revolutioniert die Art und Weise, wie diese komplexen Cloud-Strukturen bereitgestellt und verwaltet werden, indem es die Automatisierung und Standardisierung der Infrastrukturkonfiguration ermöglicht, ohne dabei manuelle Prozesse zu beinhalten. Dies bedeutet, dass Server, Betriebssysteme, Speicher und andere Infrastrukturelemente mittels maschinenlesbarem Code in Cloud-Umgebungen verwaltet werden (Marinescu, 2023). Dabei orientiert sich IaC stark an den Methoden der Softwareentwicklung, wodurch eine Reihe von Vorteilen entsteht (IBM, 2024f):

- **Zuverlässigkeit und Konsistenz**: Neue Cloud-Umgebungen oder Infrastrukturen werden zuverlässig und konsistent bereitgestellt. IaC ermöglicht die Implementierung derselben Konfiguration ohne manuelle Prozesse, was die Konsistenz zwischen Cloud-Umgebung und Implementierung verbessert.
- **Geschwindigkeit**: IaC ermöglicht die schnelle Einrichtung der gesamten Infrastruktur durch Automatisierung. Dadurch wird weniger Zeit für die Implementierung, Verwaltung und Wartung von Infrastrukturen benötigt, wodurch Kosten gesenkt werden können.
- **Verfolgung und Verantwortlichkeit**: Änderungen an der vorhandenen Infrastruktur werden

im Code vorgenommen und nachverfolgt, wodurch alle Änderungen zurückverfolgbar sind.

- **Umgebungsabweichung erkennen und korrigieren**: Wenn ein Teil der Infrastruktur manuell außerhalb des Codes geändert wird, kann er bei der nächsten Ausführung wieder in den gewünschten Zustand versetzt werden.

IaC unterscheidet zwischen zwei Hauptansätzen: dem deklarativen und dem imperativen Ansatz. Der deklarative Ansatz spezifiziert den gewünschten Endzustand der Infrastruktur, ohne die genauen Schritte zur Erreichung dieses Zustands vorzugeben, während der imperativer Ansatz explizit die einzelnen Schritte definiert, die zur Zielerreichung notwendig sind (Marinescu, 2023). Beide Ansätze haben ihre Vor- und Nachteile, wobei der deklarative Ansatz oft bevorzugt wird, da er eine höhere Abstraktion und Automatisierung bietet (IBM, 2024f).

Ein gängiges Tool, das den deklarativen Ansatz nutzt, ist Terraform. Terraform wurde von HashiCorp entwickelt und ermöglicht Nutzern, Infrastrukturen als Code zu definieren und zu verwalten. Mit Terraform können komplexe Cloud-Infrastrukturen über verschiedene Anbieter hinweg konsistent und wiederholbar bereitgestellt werden. Es verwendet eine deklarative Konfigurationssprache namens HashiCorp Configuration Language, um die gewünschte Infrastruktur zu beschreiben. Terraform stellt dann sicher, dass der aktuelle Zustand der Infrastruktur dem definierten Soll-Zustand entspricht, indem es die notwendigen Schritte automatisiert ausführt (Howard, 2022).

2.3 Cloud-native Architekturen

Die Architektur in der Softwareentwicklung beschreibt die grundlegende Struktur eines Softwaresystems. Sie legt die wesentlichen Komponenten des Systems fest, definiert deren Beziehungen zueinander und gibt die Richtlinien sowie Prinzipien vor, die das Design und die Weiterentwicklung des Systems steuern (Gharbi et al., 2023). Da im Cloud-Umfeld häufig keine klaren Grenzen zwischen System- und Softwarearchitektur existieren, wird in dieser Arbeit der Begriff „Architektur“ verwendet. Dieser umfasst in diesem Kontext das Zusammenspiel von Infrastruktur, Softwarearchitektur und der für die Software notwendigen Datenvorverarbeitung. Entscheidungen hinsichtlich der Softwarearchitektur haben maßgeblich Einfluss auf die Skalierbarkeit, Wartbarkeit, Flexibilität und Leistungsfähigkeit einer Softwarelösung (Tremp, 2021). Die im Folgenden dargestellten Architekturkonzepte konzentrieren sich in erster Linie auf die Softwarearchitektur von Webanwendungen. Dabei steht der Aufbau und die Strukturierung der Softwarekomponenten im Mittelpunkt. Obwohl diese Konzepte zur Softwarearchitektur gehören, haben sie auch wesentliche Auswirkungen auf die zugrunde liegende Systemarchitektur. Insbesondere in modernen Cloud-basierten Anwendungen, bei denen Software und Infrastruktur eng miteinander verbunden sind, beeinflussen Entscheidungen in der Softwarearchitektur oft direkt auch die Systemarchitektur.

Traditionelle monolithische Architekturen gelten heute als überholt. Sie zeichnen sich durch eine stark gekoppelte Struktur aus, bei der alle Komponenten einer Anwendung in eine einzige große Codebasis integriert sind. Diese enge Kopplung erschwert Änderungen und Erweiterungen, da selbst kleine Anpassungen umfangreiche Tests und ein erneutes Deployment der gesamten Anwendung erfordern (Tremp, 2021). Monolithische Architekturen bieten oft geringere Skalierbarkeit und Flexibilität, was die Anpassungsfähigkeit eines Unternehmens oder einer Organisation einschränkt. Zudem verursachen sie erhebliche Herausforderungen in Cloud-basierten, verteilten Systemen, da die Synchronisation bei herkömmlichen Ansätzen den Entwicklern überlassen bleibt (Kratzke, 2018). Dieser monolithische Ansatz führt oft dazu, dass komplette Anwendungen als ein großes Image einer virtuellen Maschine verpackt werden, wie in den Abbildungen 4 - 1 und 4 - 2 dargestellt. Dies kann

zu erheblichen Ausfallzeiten und eingeschränkter Skalierbarkeit führen.

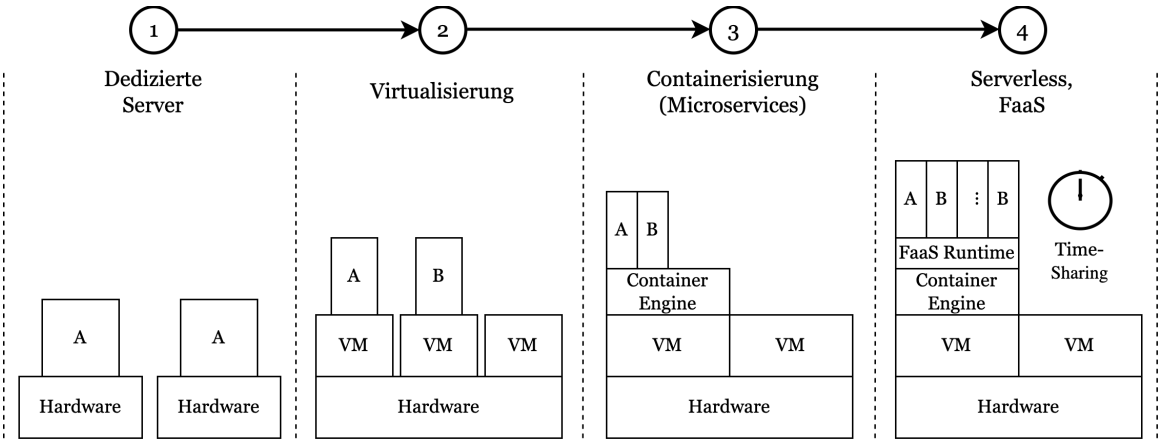


Abbildung 4: *Entwicklung Cloud-Architekturen unter dem Gesichtspunkt der Nutzung von Ressourcen (eigene Darstellung nach Kratzke, 2018)*

Cloud-native Architekturen haben sich dagegen zu einem zentralen Bestandteil der modernen Softwareentwicklung entwickelt. Sie nutzen die Vorteile von Cloud-Umgebungen voll aus und verbessern die Skalierbarkeit und Effizienz von Anwendungen. Dabei basieren diese im Wesentlichen auf den in Kapitel 2.2 vorgestellten Konzepten und Technologien. Diese modernen Architekturen optimieren die Bereitstellung und das Management von Anwendungen und ermöglichen eine flexiblere und schnellere Anpassung an neue Anforderungen. Zu den prominentesten Ansätzen innerhalb der Cloud-nativen Architekturen zählen die Microservice-Architektur und die serverlose Architektur. In den folgenden Kapiteln werden diese daher detailliert beschrieben, um ein vertieftes Verständnis ihrer Funktionsweise und Vorteile zu ermöglichen.

2.3.1 Microservice-Architekturen

Bei der Microservice-Architektur wird eine Anwendung in mehrere kleine, unabhängig voneinander einsetzbare Dienste aufgeteilt, die jeweils einen bestimmten Geschäftsprozess abbilden. Dies ist schematisch in Abbildung 4 - 3 dargestellt. Nach Tremp, 2021 bildet ein Microservice „eine eigenständige fachlich klar fokussierte Teilfunktion einer Applikation ab. Ihm steht ein eigener Ausführungscontainer zur Verfügung. Der Zugriff erfolgt über eine klar definierte synchrone und/oder asynchrone Schnittstelle“ (Tremp, 2021). Dies bedeutet, dass ein Microservice nur eine klar definierte Aufgabe erfüllt, was auch unter dem Prinzip der Single Responsibility bekannt ist. Container eignen sich ideal für Microservices, da sie die nötige Isolation gewährleisten, die für die eigenständige Entwicklung, Bereitstellung und Skalierung dieser kompakten Dienste erforderlich sind (Surianarayanan & Chelliah, 2019).

Jeder Microservice kann unabhängig in einem eigenen Container betrieben werden. Dies ermöglicht eine klare Trennung der einzelnen Funktionalitäten, gewährleistet jedoch gleichzeitig die Interoperabilität zwischen den Diensten. Zudem stellen Container, wie bereits in Kapitel 2.2.2 dargelegt, eine optimale Grundlage für die horizontale Skalierung dar. Bei hoher Auslastung können mehrere Instanzen eines Containers erstellt werden, um die Leistung zu verteilen. Wie bereits in Kapitel 2.2.3 beschrieben, bestehen Anwendungen in der Regel aus einer Vielzahl von Containern. Um diese

Container effizient zu verwalten und zu orchestrieren, sind Tools zur Container-Orchestrierung für Microservice-Architekturen häufig unerlässlich (Lukša, 2018). Kubernetes bietet in diesem Zusammenhang für die in Kapitel 2.2.3 beschriebene Architektur entscheidende Vorteile, wie etwa die Kontrolle über Ressourcen, effektive Planung, Lastenverteilung, Gesundheitsprüfungen, Fehlertoleranz und Autoskalierung. Zudem bietet Kubernetes Funktionen wie Load Balancing, die sicherstellen, dass Anfragen effizient auf die verfügbaren Containerinstanzen verteilt werden und die Microservices jederzeit erreichbar sind (Riti, 2018). Die Kommunikation zwischen den Microservices erfolgt über klar definierte Schnittstellen, die häufig auf leichtgewichtigen Protokollen wie HTTP/HTTPS oder Messaging-Systemen basieren (Surianarayanan & Chelliah, 2019).

Microservice-Architekturen haben nicht nur technische Auswirkungen, sondern beeinflussen auch die Entwicklungsprozesse positiv. So könnten einzelne Microservices in unabhängigen Teams entwickelt werden. Dies passt perfekt zu modernen Softwareentwicklungsmethoden wie Scrum mit DevSecOps, wo kleine, agile Teams autonom arbeiten und kontinuierlich Software liefern (Surianarayanan & Chelliah, 2019). Dadurch wird die Entwicklung flexibler, da Teams unabhängig voneinander arbeiten und ihre bevorzugten Technologien und Methoden verwenden können. Zudem ist keine Einschränkung auf eine einzige Programmiersprache erforderlich, da jeder Microservice in der jeweils am besten geeigneten Sprache entwickelt werden kann. Diese Polyglot-Programmierung führt zu einer besseren Anpassung an die spezifischen Bedürfnisse eines jeden Dienstes und kann die Gesamtleistung und -zuverlässigkeit des Systems erhöhen. So sind nach einer Umfrage von IBM mit über 1200 Entwicklern und IT-Managern 87 % der Nutzer von Microservices der Meinung, dass die Einführung von Microservices die Kosten und den Aufwand wert sind (IBM, 2021).

Trotz ihrer vielen Vorteile bringen Microservices auch erhebliche Herausforderungen mit sich. Ein wesentlicher Nachteil bei der Einführung einer solchen Architektur ist die oft mangelnde Erfahrung des Entwicklungsteams, was eine längere Einarbeitungszeit erfordert. Besonders in großen Unternehmen oder Organisationen wird die Umstellung aufgrund komplexer Strukturen und organisatorischer Herausforderungen zusätzlich erschwert. Außerdem führen die häufigen Dienstaufrufe und die verstärkte Kommunikation zwischen Microservices zu einer erhöhten Netzwerklast, was nicht nur Verzögerungen verursacht, sondern auch Anpassungen der Netzwerkarchitektur erfordern kann. Darüber hinaus entstehen durch die Nutzung separater Datenbanken für jeden Microservice Daten-Duplikationen und zusätzlicher Programmieraufwand, um die Konsistenz zu gewährleisten. Diese steigende Komplexität erschwert auch die Orchestrierung der Dienste. Zudem erhöht die Vielzahl der Microservices die Schwierigkeit beim Debugging und der Fehlerverfolgung erheblich, was ohne Automatisierung zeitaufwändig und teuer sein kann. (Velepucha & Flores, 2023)

Insgesamt bietet die Microservice-Architektur jedoch eine flexible und skalierbare Lösung für moderne Anwendungen, die eine hohe Anpassungsfähigkeit und Fehlertoleranz erfordern. Durch die Nutzung von Containern und modernen Orchestrierungstools wie Kubernetes können Unternehmen und Organisationen ihre Softwareentwicklung und -bereitstellung effizienter gestalten und schneller auf sich ändernde Geschäftsanforderungen reagieren.

2.3.2 Serverlose Architekturen

Serverlose Architekturen, auch als serverloses (engl. serverless) Computing bezeichnet, stellen den neuesten Trend im Bereich Cloud-Computing dar (Kratzke, 2018). Sie ermöglichen Entwicklern, Anwendungen zu erstellen und auszuführen, ohne Server oder Backend-Infrastrukturen verwalten zu müssen (Kratzke, 2018). Die Anwendungen werden auf serverlosen Plattformen ausgeführt, die die erforderliche Umgebung sowie die Tools für Entwicklung, Test und Betrieb bereitstellen. Diese

Plattformen basieren im Wesentlichen auf den Prinzipien von IaaS und PaaS, indem sie die zugrunde liegende Infrastruktur und Plattformdienste nutzen und weiter abstrahieren (Lee et al., 2018). Plattformen wie bspw. AWS Lambda, Azure Functions, Google Cloud Functions oder IBM Code-Engine übernehmen die Verwaltung der zugrunde liegenden Infrastruktur, einschließlich Server, Netzwerke und Speicher, und bieten zudem automatisierte Skalierung, Überwachung und Fehlerbehebung. Der Begriff „serverlos“ kann missverständlich sein, da auch in serverlosen Architekturen Server verwendet werden. Der Unterschied besteht darin, dass Entwickler und Nutzer diese Server nicht selbst verwalten müssen, sondern diese Aufgabe vom CSP übernommen wird. Entscheidungen wie die Anzahl der Server und deren Kapazität werden von der serverlosen Plattform getroffen, wobei die Serverkapazität je nach Auslastung automatisch bereitgestellt wird (Baldini et al., 2017). Dies ermöglicht Entwicklern, sich ausschließlich auf den Code und die Geschäftslogik ihrer Anwendungen zu konzentrieren (Surianarayanan & Chelliah, 2019).

Im Gegensatz zu Microservice-Architekturen wird bei serverlosen Architekturen die Geschäftslogik in zustandslosen, ereignisgesteuerten und ephemeren Rechencontainern ausgeführt, die nur für die Dauer eines einzelnen Aufrufs existieren und vollständig von einem Drittanbieter verwaltet werden (Roberts & Chapin, 2016). Diese Funktionalität wird als FaaS bezeichnet und wurde bereits im Kapitel 2.1.2 beschrieben. Zusammengefasst sind FaaS unabhängige Funktionen, die als Reaktion auf bestimmte Ereignisse ausgeführt werden. FaaS und Serverlos werden oft synonym verwendet, jedoch umfasst Serverlos mehr als nur FaaS. Es handelt sich um einen ganzen Service-Stack, der auf Ereignisse oder Anforderungen reagieren kann und bei Nichtnutzung auf Null skaliert. Neben FaaS umfassen serverlose Dienste auch serverunabhängige Datenbanken und Speicher, Event-Streaming und Messaging sowie API-Gateways (IBM, o.J.).

Die Grundlage einer serverlosen Architektur ist die serverlose Plattform, die als Ereignisverarbeitungssystem fungiert. Diese Kernfunktionalität ist in Abbildung 5 dargestellt. Sie empfängt Ereignisse über HTTP oder andere Cloud-Quellen, ermittelt die registrierten Funktionen zur Verarbeitung, startet bei Bedarf neue Funktionsinstanzen, sendet das Ereignis zur Verarbeitung, sammelt Ausführungsprotokolle und stellt dem Benutzer die Antwort zur Verfügung. Nach der Ausführung kann die Funktion beendet werden, wenn sie nicht mehr benötigt wird. Darüber hinaus erweitert FaaS die gemeinsame Nutzung von Ressourcen auf elastischen Plattformen durch die Anwendung von Time-Sharing-Konzepten. Diese Prinzipien ermöglichen eine deutlich effizientere Auslastung von IT-Infrastrukturen, wodurch teure Always-on-Komponenten vermieden werden (Kratzke, 2018). Dies führt zu erheblichen Kosteneinsparungen. So hat die Studie von Villamizar et al., 2017 gezeigt, dass Time-Sharing bei serverlosen Architekturen die Kosten um bis zu 70 Prozent senken können (Villamizar et al., 2017).

Serverlose Architekturen sind eng mit Microservices-Architekturen verbunden. Wie bereits in Kapitel 2.3.1 beschrieben, besteht eine Microservice-Architektur aus einer Sammlung lose gekoppelter, eigenständiger Dienste, die jeweils eine bestimmte Geschäftsaufgabe erfüllen. Serverlose Architekturen unterstützen dieses Paradigma, indem sie das Infrastruktur weiter abstrahieren und es Entwicklern ermöglichen, sich auf die Implementierung einzelner Services zu konzentrieren. Ein wesentlicher Unterschied besteht darin, dass Microservices in der Regel kontinuierlich laufende Dienste sind, während serverlose Funktionen nur bei Bedarf ausgeführt werden. Durch bestimmte Definitionen der Ressourcen, wie das kontinuierliche Laufen einer Instanz eines Containers, lassen sich auch Microservices über serverlose Plattformen ausführen. Dadurch können containerisierte Microservices neben serverlosen Funktionen innerhalb einer Anwendung koexistieren, was die Bereitstellungsgeschwindigkeit erhöht (Alqaryoutia & Siyamb, 2018). Serverlose Architekturen bringen eine Reihe

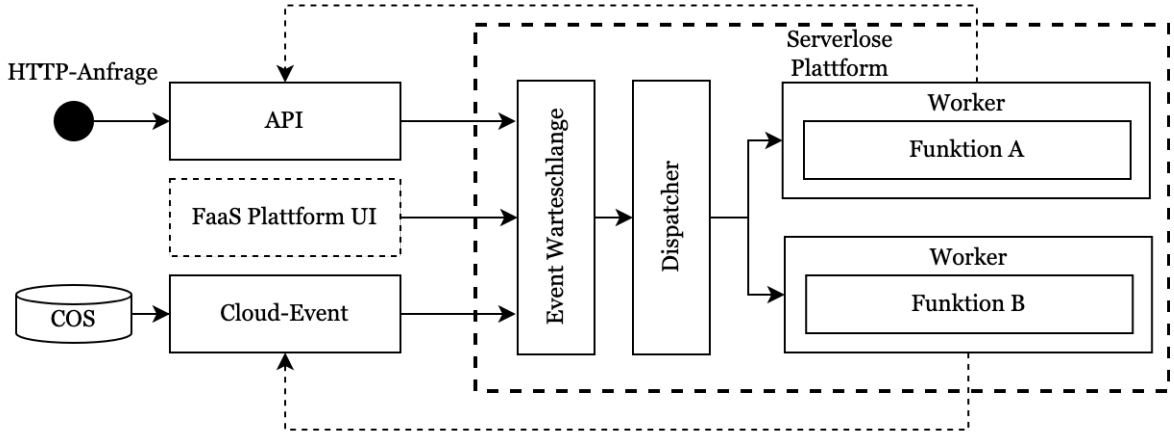


Abbildung 5: *Schematische Darstellung einer serverlosen Architektur (eigene Darstellung nach Kratzke, 2018 und Baldini et al., 2017)*

von Vorteilen mit sich (IBM, o.J.):

- **Integrierte Dienste**: Entwickler können auf eine breite Palette von Diensten zugreifen, die alle serverlos betrieben werden. Dies erleichtert die Entwicklung komplexer Anwendungen.
- **Verkürzte Entwicklungszeiten**: Durch die Nutzung vorgefertigter Dienste und die Möglichkeit, sich auf die Geschäftslogik zu konzentrieren, können Anwendungen schneller entwickelt und auf den Markt gebracht werden.
- **Erhöhte Flexibilität und Agilität**: Unternehmen können schnell auf Marktanforderungen reagieren und ihre Anwendungen skalieren, ohne sich um die zugrunde liegende Infrastruktur kümmern zu müssen.

Allerdings gibt es auch Herausforderungen und potenzielle Nachteile bei der Nutzung von serverlosen Architekturen (Kratzke, 2018):

- **Sicherheitsbedenken**: Die Abhängigkeit von externen Anbietern kann Sicherheitsrisiken mit sich bringen.
- **Anbieterabhängigkeit**: Die Nutzung spezifischer Dienste eines Anbieters kann die Flexibilität einschränken und den Wechsel zu einem anderen Anbieter erschweren.
- **Performance-Einbußen**: Für manche Anwendungen kann die Latenz, die durch die Verwaltung durch den Cloud-Anbieter entsteht, problematisch sein.

In Kombination bilden serverlose Architekturen, Microservices und Container ein zentrales technologisches Fundament, das als Grundlage der modernen Cloud-nativen Anwendungsentwicklung gilt (IBM, o.J.)

2.4 Cloud-optimierte Geodatenformate

Die effektive Verwaltung und Verarbeitung großer Mengen an Geodaten in der Cloud erfordert optimierte Datenformate, die speziell für den Einsatz in der Cloud entwickelt wurden. Grundsätzlich werden Geodaten in Vektor- oder Rasterdaten gespeichert. Rasterdaten bestehen aus einem

regelmäßig angeordneten Gitter von Zellen oder Pixeln, wobei jede Zelle einen bestimmten Wert repräsentiert. Typische Rasterdaten umfassen bspw. Satellitenbilder, digitale Höhenmodelle oder Klimadaten. Vektordaten hingegen bestehen aus Punkten, Linien und Polygonen, die geografische Merkmale abbilden. Zu den Beispielen für Vektordaten zählen Straßen, Flüsse, Grundstücksgrenzen und Gebäudeumrisse (De Lange, 2020). Eine schematische Darstellung beider Modelle ist in Abbildung 6 zu sehen.

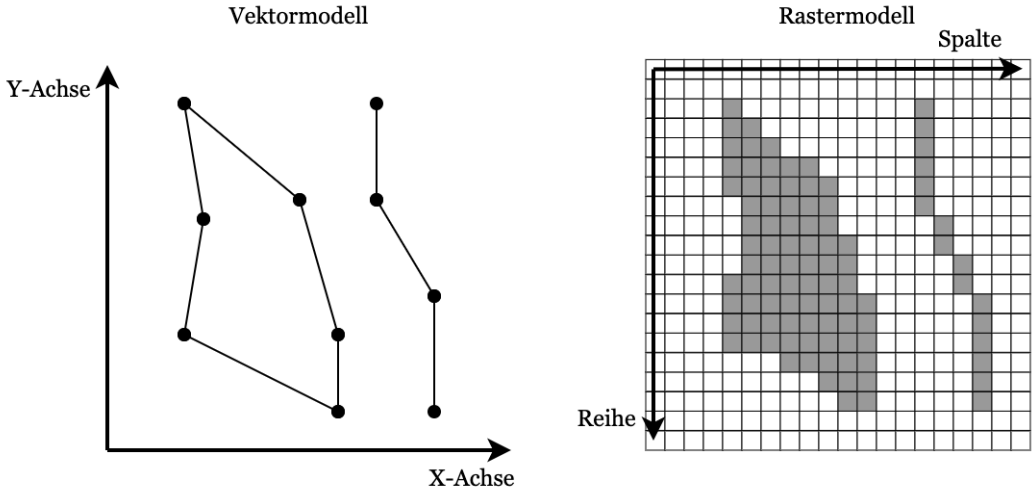


Abbildung 6: *Vektor- und Rastermodell (eigene Darstellung nach De Lange, 2020)*

Sowohl für Vektor- als auch für Rasterdaten existieren eine Vielzahl unterschiedlicher Datenformate. Diese eignen sich jedoch nur bedingt für die Nutzung in der Cloud. Mit der Cloud-nativen Bewegung haben sich Cloud-optimierte Datenformate entwickelt, die eine effiziente Speicherung und Abfrage ermöglichen. Diese Formate verbessern die Zugänglichkeit, Effizienz und Skalierbarkeit von Geodatenoperationen erheblich. Im Folgenden werden die optimierten Formate Cloud-optimiertes GeoTIFF (COG) für Rasterdaten und FlatGeobuf (FGB) für Vektordaten näher beschrieben.

2.4.1 FlatGeobuf

FGB ist ein relativ neues, binäres Dateiformat für räumliche Vektordaten wie Punkte, Linien und Polygone. Es ist für die Nutzung in der Cloud optimiert und unterstützt schnelle Lesezugriffe auf alle Vektorgeometrietypen gemäß der Open Geospatial Consortium (OGC) Simple Features-Spezifikation (Barciauskas, 2023). Es baut auf dem Konzept von FlatBuffers auf, einem plattformübergreifenden Serialisierungsformat, das von Google entwickelt wurde. FlatBuffers ermöglichen schnelle und effiziente Datenserialisierung, was besonders in Anwendungen mit hohen Leistungsanforderungen von Vorteil ist (Google, o.J.). FGB kombiniert diese Vorteile von FlatBuffers mit spezifischen Erweiterungen für die Handhabung von räumlichen Vektorgeometrien.

Im Gegensatz zu traditionellen Vektorformaten wie Shapefiles oder GeoJSON, die oft ineffizient und langsam bei der Verarbeitung großer Datenmengen sind, bietet FGB sowohl höhere Leistung als auch eine geringere Dateigröße. Dies ist vor allem auf das binäre Format zurückzuführen, das FGB kompakter und schneller serialisierbar macht als textbasierte Formate wie GeoJSON oder Well-known Text (WKT). Durch seine flache Struktur ermöglicht es direkte Datenzugriffe ohne vorheriges Deserialisieren, was zu schnelleren Ladezeiten und effizienter Verarbeitung führt (FlatGeoBuf Developers, o.J.). Dadurch können Anwendungen räumliche Daten schneller laden und

verarbeiten, was insbesondere für Echtzeitanwendungen und die Verarbeitung großer Datensätze von entscheidender Bedeutung ist.

Ein wesentlicher Vorteil von FGB liegt in der Unterstützung integrierter räumlicher Indizes, die schnelle räumliche Abfragen ermöglichen und das Remote-Lesen optimieren. Dabei wird ein RTree-Index mit einer kompakten, streamfähigen Kodierung kombiniert, während die Geometriedaten räumlich entlang einer Hilbert-Kurve angeordnet werden, um die Effizienz zu maximieren (Williams, 2022). Diese Eigenschaften machen das Format besonders geeignet für Write-Once-Read-Many (WORM)-Zugriffe in Verbindung mit einem COS. Dies ist vor allem in GIS-Anwendungen nützlich, bei denen schnelle und häufige Abfragen von geografischen Daten erforderlich sind. Zudem wird FGB in gängigen Tools wie Leaflet, Maplibre und OpenLayers unterstützt, was die Benutzerfreundlichkeit weiter erhöht (FlatGeoBuf Developers, o.J.). Neben den Vorteilen identifiziert Williams, 2022 jedoch auch einige Nachteile (Williams, 2022):

- **Komplexität**: Das Format ist aufgrund seiner binären Kodierungen und optimierten Indizierungsstrategien ziemlich kompliziert. Kleine Fehler können Dateien unlesbar machen, und es sind bessere Tools zur Validierung notwendig.
- **Nicht menschenlesbar**: Im Vergleich zu einfacheren Formaten wie GeoJSON und WKT ist FGB schwer lesbar.
- **Nicht für Updates ausgelegt**: Aufgrund der komplexen Datenanordnung und Indizierung sind Aktualisierungen nicht vorgesehen, wodurch es sich am besten für statische Datensätze eignet.
- **Keine nicht-räumliche Indizierung**: Derzeit gibt es keine Möglichkeit, Indizes auf andere Dimensionen als die räumliche hinzuzufügen.

Neben FGB gibt es GeoParquet, welches ebenfalls ein Cloud-optimiertes Vektordatenformat darstellt. Es basiert auf der spaltenorientierten Struktur von Parquet und eignet sich besonders für analytische Abfragen auf großen Datensätzen. Dieses Format wird im Rahmen dieser Arbeit jedoch nicht weiter betrachtet.

2.4.2 Cloud-optimiertes GeoTIFF

Ähnlich wie FGB ist das COG ein relativ neues Datenformat, das jedoch für die Speicherung von Rasterdaten in einem COS optimiert ist. COG basiert auf dem GeoTIFF-Format, das sich als Standard-Rasterdatenformat in der Geoinformatik etabliert hat. Geography Tagged Image File Format (GeoTIFF) ist ein verlustfreies Format, welches im Wesentlichen die Struktur des Tagged Image File Format (TIFF)-Formats verwendet. GeoTIFF nutzt die Eigenschaften des TIFF-Formats, um geografische Informationen in sogenannten Tags zu speichern, ähnlich wie Bildmetadaten in herkömmlichen TIFF-Dateien (Survey, 2020). Diese Tags können Informationen wie Kartenprojektion, Koordinatensysteme, Datumsangaben sowie Transformationsparameter für die Georeferenzierung enthalten. Dadurch wird die korrekte Darstellung der Rasterdaten in einem Geoinformationssystem (GIS) ermöglicht. Da alle erforderlichen Informationen in einer Datei enthalten sind und keine zusätzliche Datei mit bspw. den Projektionsinformationen benötigt wird, erleichtert dies den Austausch und die Nutzung von Geodaten erheblich.

Das COG ist eine Erweiterung des GeoTIFF-Formats, das speziell für die Nutzung in Cloud-Umgebungen entwickelt wurde, insbesondere in Kombination mit einem COS. Beim COG wird das Bild in Kacheln und Pyramiden organisiert (COGEO, 2024):

- **Kachelung:** Bei der Kachelung wird das Bild in kleinere, gleichmäßige Abschnitte, sogenannte Kacheln, unterteilt. Diese ermöglichen es, gezielt nur die benötigten Bildteile zu laden, anstatt das gesamte Bild herunterladen zu müssen, was den Zugriff auf spezifische Bildbereiche beschleunigt.
- **Pyramidenbildung:** Hierbei werden mehrere Versionen des Bildes in unterschiedlichen Auflösungen erstellt. Die höchste Auflösung bildet die Basis, während die niedrigeren Auflösungen sukzessive aus den höher aufgelösten Bildern generiert werden. Diese Pyramidenstruktur ermöglicht eine effiziente Darstellung und Verarbeitung des Bildes, da nur die notwendige Auflösung für die jeweilige Anwendung geladen wird.

Die Kachelung und Pyramidenbildung stellen aktuelle Best Practices dar, die es Software ermöglichen, schnell auf Bilder zuzugreifen (COGEO, 2024). Sie werden jedoch zusätzlich für den HTTP-Range-Request benötigt. Diese Technologie erlaubt es nur bestimmte Teile einer Datei zu übertragen, anstatt die gesamte Datei laden zu müssen. Das bedeutet, dass ein Client gezielt nur die für ihn relevanten Datenbereiche anfordern kann, was die Übertragungszeit erheblich reduziert. Zudem ermöglicht die bessere Komprimierung von COG eine geringere Dateigröße und schnellere Datenübertragung. HTTP-Range-Requests sind nach dem HTTP-Standard optionale Elemente eines Requests, sodass der Zugriff auf die vollständige Datei weiterhin möglich ist (COGEO, 2024).

In herkömmliche GIS kann ein COG genauso genutzt werden, wie ein herkömmliches GeoTIFF (COGEO, 2024). Dies bedeutet, dass keine zusätzlichen Anpassungen oder spezielle Softwareanforderungen erforderlich sind, um ein COG zu nutzen. Dadurch kann das COG nahtlos in bestehende Arbeitsabläufe und Systeme integriert werden können, ohne dass zusätzliche Schulungen oder technische Anpassungen erforderlich sind. Des Weiteren weisen COGs im Vergleich zu herkömmlichen GeoTIFFs keine nennenswerten Nachteile auf. Sie behalten alle Vorteile und Eigenschaften des GeoTIFF-Formats bei und bieten zusätzliche Funktionen für einen effizienteren Datenzugriff und -transfer in Cloud-Umgebungen. Dies macht COGs zu einer vielseitigen und zukunftssicheren Wahl für die Verwaltung und Bereitstellung von Geodaten.

3 Bodenbewegungsdienst Niedersachsen

Die Erdoberfläche ist aufgrund verschiedener anthropogener und geologischer Faktoren ständigen Veränderungen unterworfen. Diese Bodenbewegungen beeinflussen die amtlichen Festpunktfelder, weshalb ein kontinuierliches Monitoring unverzichtbar ist. Das Projekt BOB.NI hat zum Ziel, die vom LGLN aufbereiteten Bodenbewegungsdaten (das BOB.NI-Modell) über eine Webschnittstelle (die BOB.NI-WebApp) sowohl Fachanwendern als auch privaten Nutzern zugänglich zu machen. Dieses Ziel ist im fachlichen Zukunftskonzept VKV 2025 (LGLN, 2017) fest verankert. Der BOB.NI umfasst demnach sowohl die Bodenbewegungsdaten und die dazugehörige Prozesskette (das BOB.NI-Modell) als auch die Präsentationsschnittstelle (die BOB.NI-WebApp).

Um die Bedeutung des BOB.NI und dessen Entwicklung zu verstehen, ist es wichtig, Grundkenntnisse über Bodenbewegungen und deren entsprechenden Datengrundlagen zu vermitteln. Im folgenden Kapitel wird daher zunächst erläutert, was unter Bodenbewegungen zu verstehen ist, wie sie entstehen und mit welchen Messmethoden sie erfasst werden können. Anschließend wird die Prozesskette der Bodenbewegungsdaten beschrieben, deren primäres Ergebnis das BOB.NI-Modell ist. Zusätzlich wird die Aufbereitung dieser Daten zum rasterbasierten Bodenbewegungsmodell erläutert, welches

als Kerndatensatz dieser Arbeit dient.

3.1 Bodenbewegungen

Wie bereits beschrieben unterliegt die Erdoberfläche durch eine Vielzahl anthropogener und geologischer Ursachen ständigen Veränderungen. Nach DIN 21917 sind Bodenbewegungen „die Gesamtheit aller bergbaulich, geologisch oder hydrologisch verursachten Form- und Lageänderungen (Bewegungsvorgang) an der Tagesoberfläche“ (Deutsches Institut für Normung e.V., 1999). Dabei wird der Begriff „Tagesoberfläche“ synonym zur sichtbaren Erdoberfläche verwendet. Um den Begriff der Bodenbewegung besser zu verstehen, ist es hilfreich, die Definition von „Boden“ näher zu betrachten. Yin, 2020 definiert den Boden als die oberste Schicht der Erde. In diesem Zusammenhang können Bodenbewegungen auch als Verformungen der Erdoberfläche betrachtet werden, da sie die sichtbare und direkt zugängliche Erdschicht betreffen. Durch die Kombination der Begriffe „Tagesoberfläche“ und „Boden“ wird deutlich, dass Bodenbewegungen sowohl die gesamte sichtbare Erdoberfläche als auch deren oberste Schicht betreffen (Koppmann, 2020).

Bodenbewegungen können durch verschiedene Ursachen ausgelöst werden, die grundsätzlich in zwei Kategorien unterteilt werden: geogene und anthropogene Ursachen (Brockmeyer et al., 2020). Geogene Ursachen umfassen alle natürlichen Prozesse, die zu Bodenbewegungen führen. Dazu gehören tektonische Aktivitäten wie Erdbeben und Vulkanismus, die durch die Bewegung der Erdplatten verursacht werden. Auch langsame geologische Prozesse wie die Hebung und Senkung von Landmassen sowie Erosion und Sedimentation, die durch Wasser, Wind und Eis verursacht werden, fallen in diese Kategorie. Hydrologische Prozesse, wie Schwankungen im Grundwasserspiegel, können ebenfalls Bodenbewegungen hervorrufen, da sie die mechanischen Eigenschaften des Bodens verändern. In Niedersachsen spielen geogene Ursachen eine eher untergeordnete Rolle. Hier sind die primären Ursachen für signifikante Bodenbewegungen anthropogenen Ursprungs. Diese anthropogenen Bewegungen sind häufig auf die Entnahme von Grundwasser sowie die Gewinnung und Lagerung von Rohstoffen im Untergrund zurückzuführen. Von besonderer Bedeutung sind unterirdische Kavernen zur Speicherung von Erdöl und Erdgas, da 90 % der Gesamtförderung in Deutschland aus Niedersachsen stammen (Heunisch et al., 2017).

Der Abbau von unterirdischen Rohstoffen führt häufig zur Bildung sogenannter Senkungsmulden. Dies geschieht, wenn die Entnahme von Material unter der Erdoberfläche einen Hohlraum hinterlässt, auf den der Druck der darüber liegenden Bodenschichten einwirkt. Laut Yin, 2020 führt die Schließung dieses Hohlraums zu einer Absenkung der Erdoberfläche, die als Senkungsmulde bezeichnet wird (Yin, 2020). Um die daraus resultierenden Bodenbewegungen besser zu verstehen, zeigt Abbildung 7 schematisch das Bewegungsverhalten einer solchen Senkung in einem Querschnitt. Zu sehen ist, wie die Oberflächenbewegung direkt mit der unterirdischen Hohlraumveränderung zusammenhängt. Die horizontale Achse der Abbildung bezieht sich auf die Position über dem Hohlraum, während die vertikale Achse die Veränderung der Erdoberfläche in Form von Absenkungen oder Hebungen darstellt. Zu sehen ist die Höhenänderung (Senkung), die ihr Maximum ($d_{z_{\max}}$) genau in der Mitte der Abbaufläche erreicht und zu den Rändern hin abnimmt. Zusätzlich ist die horizontale Bewegung (Verschiebung) visualisiert. Die maximale horizontale Verschiebung ($d_{hor_{\max}}$) tritt etwa in der Mitte zwischen Senkungsschwerpunkt und Rand des Einflussbereichs auf. Diese horizontale Bewegung lässt sich durch die Deformationsrichtung erklären. Das umliegende Material wird durch den Druck in den entstandenen Hohlraum gedrückt, was zu einer horizontalen Verschiebung führt. Diese Verschiebung ist in der Mitte zwischen Senkungsschwerpunkt und Rand des Einflussbereichs am größten, da hier das Material am stärksten in den Hohlraum gedrückt wird. Im Senkungsschwerpunkt selbst

verschwindet die horizontale Bewegung vollständig, da hier die Bewegung hauptsächlich vertikal nach unten erfolgt (Yin, 2020).

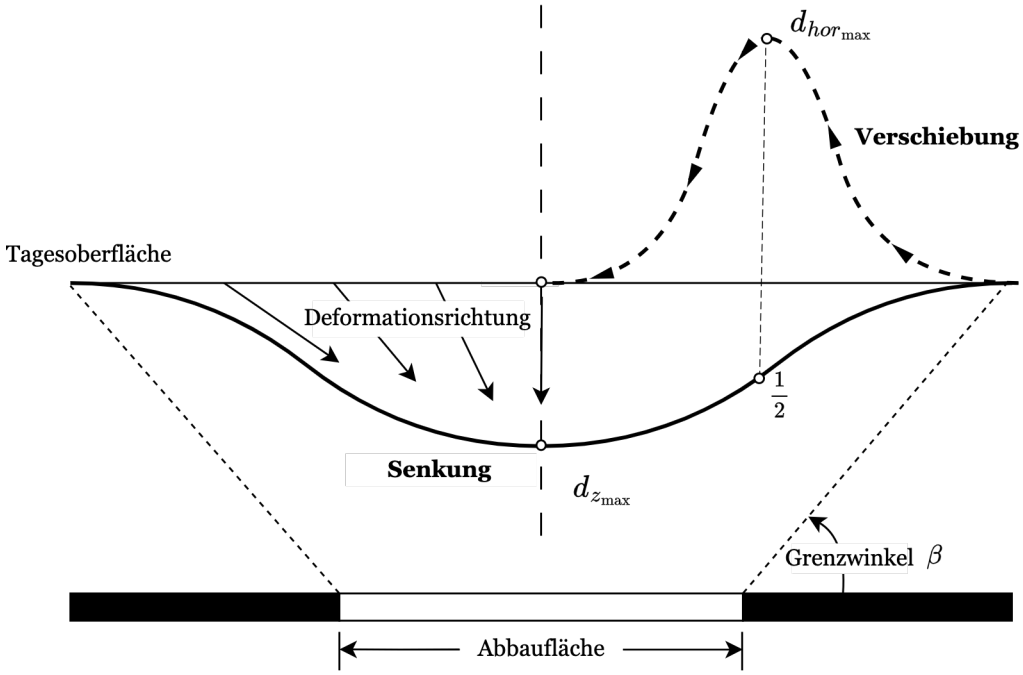


Abbildung 7: *Bodenbewegungsverhalten im Bereich einer Senkungsmulde (eigene Darstellung nach Kratzsch, 2013)*

3.2 Messverfahren zur Erfassung von Bodenbewegungen

Für die Analyse und Interpretation von Bodenbewegungen ist es wichtig, entsprechende Daten zu erfassen und zu visualisieren. Obwohl Bodenbewegungen in der Realität flächenhafte Phänomene sind, erfolgt die Beschreibung oftmals nur punktuell, da die Messdaten auf spezifische Positionen der Erdoberfläche bezogen sind. Die Bewegungen eines Punktes lassen sich in einem dreidimensionalen Koordinatensystem darstellen. Zwei Komponenten beschreiben dabei die horizontale Bewegung (Ost-West und Nord-Süd), während eine Komponente die vertikale Veränderung (Höhenänderung relativ zur Ellipsoidnormale) darstellt. Die Definition der Koordinatenachsen hängt vom verwendeten übergeordneten Koordinatensystem ab. Zusätzlich wird die Zeit als vierte Dimension berücksichtigt, da sich das Bewegungsverhalten über unterschiedliche Zeiträume hinweg verändern kann (Koppmann, 2020). Die Erfassung von Bodenbewegungen kann mittels unterschiedlicher Verfahren erfolgen. Aufgrund der potenziellen rechtlichen und wirtschaftlichen Konsequenzen der Messungen ist eine präzise zeitliche und räumliche Erfassung unerlässlich (Yin, 2020). Zu diesem Zweck werden vornehmlich die etablierten geodätischen Messverfahren wie GNSS, Nivellement und satellitengestützte Radarinterferometrie eingesetzt, um die erforderlichen Genauigkeiten zu gewährleisten (Brockmeyer, 2024). Diese Messmethoden werden im Folgenden näher beschrieben und dargelegt, wie mithilfe dieser Technologien Bodenbewegungen ermittelt werden können. Diese Daten stellen die grundlegenden Datenquellen für die Prozesskette in Niedersachsen dar.

Nivellement
Das Nivellement ist ein geodätisches Verfahren zur Bestimmung von Höhenunterschieden zwischen verschiedenen Punkten der Erdoberfläche. Der Höhenunterschied zwischen zwei Geländepunkten wird durch Messung des vertikalen Abstandes der beiden Punkte von einer horizontalen Ziellinie bestimmt. Dazu wird ein Nivelliergerät verwendet, das eine exakt horizontale Ziellinie erzeugt. Zusätzlich werden zwei Nivellierlatten verwendet, sodass die vertikalen Abstände der Geländepunkte von der Ziellinie exakt bestimmt werden können (Kahmen, 2005). Die Bestimmung der absoluten Höhe eines neuen Punktes erfolgt durch Addition bzw. Subtraktion der Höhendifferenz der Punkte von einer bekannten Ausgangshöhe. In Deutschland wird zur Höhenbestimmung das Deutsches Haupthöhennetz (DHHN) verwendet. Das DHHN ist ein Netz geodätischer Punkte mit genau bekannten Höhen, das als Bezugsrahmen für ein physikalisches Höhenbezugssystem dient. Um eine einheitliche Höhenbestimmung des DHHN zu gewährleisten, wurden in bundesweiten Messkampagnen über mehrere Jahre Doppelnivellements durchgeführt und die Beobachtungen mit einem statischen Modellansatz ausgeglichen (Brockmeyer, 2024). Im Rahmen der Pflege des Gesamtnetzes werden zwischen den Wiederholungsmessungen ausgefallene Festpunkte durch lokale Messungen ersetzt und neu bestimmt. Die Aktualisierung der Höhen einzelner Netzabschnitte oder Strecken erfolgt nach Bedarf. Die ermittelten Höhen des DHHN werden für die Datenhaltung im Amtliches Festpunktinformationssystem (AFIS) als Zeitreihe gespeichert und jeder Höhenfestpunkt mit einer Lagekoordinate zur Georeferenzierung versehen. Aus dem Vergleich der Höhen verschiedener Epochen lassen sich bereits Höhenänderungen im Nivellementnetz ableiten. Durch eine Weiterverarbeitung der beobachteten Höhendifferenzen können systematische Einflüsse vermieden werden, sodass vertikale Bodenbewegungen mit höchster Genauigkeit erfasst werden (Brockmeyer, 2024). Allerdings eignet sich das Nivellement aufgrund der geringen Anzahl an Messpunkten und des erheblichen Aufwands nur bedingt, um flächenhaft kontinuierliche Bodenbewegungen zu detektieren.

Global Navigation Satellite System (GNSS)
Das Global Navigation Satellite System (GNSS) umfasst verschiedene Satellitennavigationssysteme wie Global Positioning System (GPS), Global'naya Navigatsioannaya Sputnikovaya Sistema (GLONASS), Galileo (EU) und BeiDou (China). Diese Systeme ermöglichen eine präzise Positionsbestimmung auf der Erdoberfläche. GNSS operiert mittels eines Netzwerks von Satelliten, die kontinuierlich Signale zur Erde senden. Ein GNSS-Empfänger nimmt diese Signale auf und ermittelt die Distanz zu den Satelliten anhand der Zeitdifferenz zwischen Aussendung und Empfang. Diese Messungen basieren auf der genauen Synchronisation der Uhren an Bord der Satelliten und des Empfängers. Durch Triangulation der berechneten Entfernungen zu mehreren Satelliten kann die exakte Position des Empfänger bestimmt werden. Um jedoch sicherzustellen, dass Nutzer des GNSS-Verfahrens mit nur einem Empfänger bei Bedarf in Echtzeit präzise Koordinaten für ihren Standort ermitteln können, sind zusätzliche Korrekturdaten für die registrierten Beobachtungen erforderlich. Diese Korrekturdaten können von einem Positionierungsdienst bezogen werden, der auf einem Netz von einheitlich koordinierten Referenzstationen basiert. Ein solcher Positionierungsdienst in Deutschland ist Satellitenpositionierungsdienst der deutschen Landesvermessung (SAPOS). SAPOS wird einheitlich von den Bundesländern betrieben und basiert auf einem Netz von Referenzstationen, die im Durchschnitt etwa 50 Kilometer voneinander entfernt sind (Brockmeyer, 2024). Diese Referenzstationen empfangen kontinuierlich GNSS-Signale und sorgen durch die Bereitstellung von Korrekturdaten für eine hohe Genauigkeit der Positionsbestimmung. Um die Zuverlässigkeit der Positionsbestimmung über die amtlichen SAPOS-Dienste zu gewährleisten, führen die Länder kontinuierlich Monitoringmaßnahmen zur Qualitätssicherung durch. Im Rahmen des Koordinatenmonitorings wird bspw. die Stabilität der Referenzstationen kontinuierlich überwacht. Dazu werden die aufgezeichneten GNSS-Signale eines ganzen Tages im Postprocessing kontinuierlich ausgewertet und

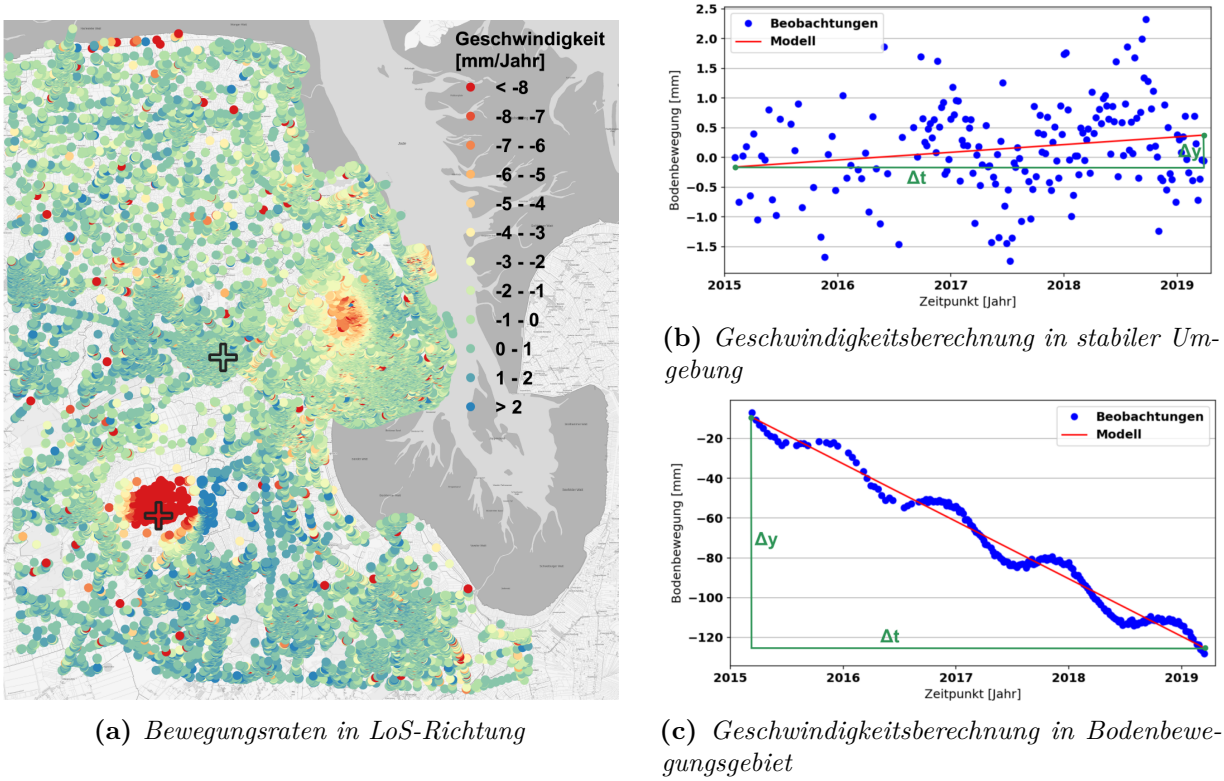
die resultierenden Tageslösungen zu wöchentlichen Koordinatensätzen zusammengefasst. Mit den so gewonnenen Koordinatenzeitreihen kann die dreidimensionale Bewegung der Referenzstationen mit hoher Genauigkeit und zeitlicher Auflösung beschrieben werden, sodass Bodenbewegungen bestimmt werden können (Brockmeyer, 2024). Aufgrund der Abstände zwischen den Referenzstationen ist jedoch auch mit GNSS keine optimale flächenhafte Bestimmung von Bodenbewegungen möglich.

Radarinterferometrie
Radar steht für Radio Detection and Ranging und ist eine Technik zur Fernmessung und Erkennung von Objekten mittels elektromagnetischer Wellen. Diese Systeme senden gebündelte Impulse im Mikrowellenbereich aus, die von den Zielen teilweise absorbiert, teilweise reflektiert und wieder empfangen werden. Dieses Messverfahren wird als Echoprinzip bezeichnet (Klausing & Holpp, 1999). Eine spezialisierte Anwendung der Radartechnologie ist die satellitengestützte InSAR. InSAR gehört zu den aktiven Fernerkundungssystemen und spielt eine entscheidende Rolle in der Fernerkundung. Die Synthetic Aperture Radar (SAR)-Technik nutzt die Vorwärtsbewegung der Satelliten in Kombination mit den ausgesendeten Impulsen, um zweidimensionale Bilder der Erdoberfläche zu erzeugen. Dabei werden bei nahezu identischen Satellitenpositionen wiederholt Bilder desselben Gebiets aufgenommen. Durch den Einsatz der Interferometrie werden diese wiederholten Aufnahmen genutzt, um präzise Höheninformationen der Erdoberfläche zu erhalten. Interferometrie ist im Allgemeinen ein Messverfahren, das die Überlagerung von Wellen nutzt, um physikalische Größen wie Entfernungen oder Formänderungen mit hoher Genauigkeit zu bestimmen. Dabei werden die Phasendifferenzen im Wertebereich von $-\pi$ bis $+\pi$ zwischen zwei SAR-Szenen gebildet, die eine Abhängigkeit zu den Punktpositionsänderungen der reflektierenden Objekte aufweisen (Yin, 2020). Das resultierende Differenzbild wird als Interferogramm bezeichnet und bildet die Grundlage für die Erfassung von Bodenbewegungen mittels InSAR. Um die Deformationsänderungen der Erdoberfläche aus dem Interferogramm zu extrahieren, müssen verschiedene Fehlerquellen eliminiert werden. Dies geschieht durch die Auswertung mehrerer Interferogramme eines Untersuchungsgebietes, die einen sogenannten SAR-Stack bilden. Hierzu hat sich das von Ferretti et al., 2001 entwickelte Verfahren der Persistent Scatterer Interferometry (PSI) etabliert, bei dem ausschließlich zeitlich kohärente Pixel von Persistent Scatterern (PS) verwendet werden. Die erfassten Punktpositionsänderungen beziehen sich auf eine Referenzzeit und einen als stabil angenommenen Referenzpunkt, sodass nur die relative Bewegung von PS innerhalb eines SAR-Stacks als Zeitreihe beschrieben wird. Typischerweise werden infrastrukturelle Einrichtungen wie Gebäude oder Verkehrswege als PS detektiert, was zu einer besonders hohen Informationsdichte in urbanen Gebieten führt. InSAR eignet sich gut zur Erfassung von Bodenbewegungen, da durch die Umlaufbahn der Satelliten eine hohe Messfrequenz erreicht wird, die Daten eine gute Genauigkeit aufweisen und eine große Anzahl von Messpunkten erfasst wird.

3.3 Prozesskette zur flächenhaften Modellierung von Bodenbewegungen

Neben dem Ziel, die Bodenbewegungen in einer Webanwendung zur Verfügung zu stellen, wird im Zukunftskonzept VKV2025 festgelegt, dass das Festpunktfeld von der punkt- und linienhaften Darstellung um die flächenhafte Betrachtung sowie die zeitliche Komponente erweitert wird. Darauf aufbauend wurde eine Prozesskette entwickelt, mittels der die punktweisen PSI-Daten in ein flächenhaftes Modell überführt werden. Dieses flächenhafte Modell spiegelt das BOB.NI-Modell wider. Die gesamte Prozessierungskette ist in Abbildung 9 dargestellt und wird im Folgenden vereinfacht beschrieben. Für eine detaillierte Beschreibung sei auf Koppmann, 2020 und Brockmeyer, 2024 verwiesen. Die Grundlage für die flächenhafte Modellierung bilden die InSAR PSI-Zeitreihen, die von der BGR bereitgestellt werden (Brockmeyer, 2024). Zum aktuellen Bearbeitungsstand (Oktober

2024) decken diese den Beobachtungszeitraum von 2014 bis 2021 ab. Die hierfür notwendigen SAR-Aufnahmen stammen von den Satelliten Sentinel-1a und Sentinel-1b (Koppmann, 2020). Die gelieferten Bodenbewegungen beziehen sich auf die sogenannte Line of Sight (LoS), welche die Bewegungen in Richtung des Satelliten beschreibt. Die vom BGR gelieferten Daten werden in Abbildung 8 dargestellt. Die Farbe der Punkte gibt die von der BGR angegebene Geschwindigkeit in $mm/Jahr$ an: Rot für starke Senkungen, blau für leichte Hebungen und grün für stabile Bereiche. Zusätzlich sind die PSI-Zeitreihen in einem stabilen Gebiet (Abbildung 8b, grünes Kreuz) und in einem Bodenbewegungsgebiet (Abbildung 8c, rotes Kreuz) dargestellt.



(a) *Bewegungsraten in LoS-Richtung*

(b) *Geschwindigkeitsberechnung in stabiler Umgebung*

(c) *Geschwindigkeitsberechnung in Bodenbewegungsgebiet*

Abbildung 8: *PSI-Zeitreihendaten und deren Geschwindigkeitsberechnung (Koppmann, 2020)*

Zunächst wird innerhalb der Prozessierungskette eine zeitliche und räumliche Analyse der Daten durchgeführt. Bei der zeitlichen Analyse wird zunächst jeder PSI-Punkt eines SAR-Stacks einzeln betrachtet und Ausreißer herausgefiltert. Anschließend wird eine Modellfunktion berechnet, die den Bewegungstrend möglichst gut beschreibt. Aus dem Modell wird die Geschwindigkeit v in der Einheit $mm/Jahr$ abgeleitet. Dazu wird die Differenz der Abstandsänderungen Δy, die zum ersten Zeitpunkt y_{Beginn} und zum letzten Zeitpunkt y_{Ende} der Modellfunktion gehören, auf den gesamten Beobachtungszeitraum Δt bezogen (Koppmann, 2020). Die entsprechende Formel ist in Gleichung (1) angegeben. Zusätzlich wird die Berechnung in den Abbildungen 8 (b) und (c) veranschaulicht.

$$v = \frac{y_{Ende} - y_{Anfang}}{t_{Ende} - t_{Anfang}} = \frac{\Delta y}{\Delta t} \tag{1}$$

Nach der zeitlichen Analyse werden bei der räumlichen Analyse die berechneten Geschwindigkeiten der PSI-Punkte mit deren Nachbarschaft in Relation gesetzt. Dabei wird angenommen, dass umliegende Punkte eine ähnliche Geschwindigkeit aufweisen, da sich die Erdoberfläche an einem Ort

und dessen Umgebung ähnlich verhält (Koppmann, 2020). Bei signifikanten Abweichungen wird eine Ausreißerfilterung nach Brockmeyer et al., 2020 angewendet, um entsprechende Ausreißer zu entfernen.

Auf Basis der gefilterten Daten erfolgt eine flächenhafte Approximation mittels Ordinary Kriging nach Brockmeyer et al., 2020. Als Eingangsdaten werden die berechneten Geschwindigkeiten und deren Standardabweichungen verwendet. Das resultierende Modell weist eine regelmäßige Gitterstruktur mit einer Gitterweite von 200 m × 200 m auf. Die berechneten Geschwindigkeiten beziehen sich jedoch noch auf die LoS, sodass diese nur eine Relativbewegung aufweisen. Zur Umrechnung der relativen Bewegung in absolute Deformationen werden die im Kapitel 3.2 verwendeten GNSS- und Nivellementdaten verwendet. Diese Daten erfassen großräumige Deformationen, bezogen auf stabile Gebiete in Deutschland und Niedersachsen. Diese hochgenauen Daten kalibrieren die Bewegungsmodelle der SAR-Stacks und sorgen für deren konsistente Referenzierung. Abschließend werden die kalibrierten Bewegungsmodelle der einzelnen SAR-Stacks kombiniert und die Bodenbewegungskomponenten abgeleitet. Da sich die Sentinel-1-Satelliten aufgrund der Erdrotation auf polaren Bahnen in auf- und absteigender Richtung um die Erde bewegen, entstehen Aufnahmen eines Untersuchungsgebietes aus annähernd entgegengesetzten Blickwinkeln (Yin, 2020). Unter Verwendung dieser Informationen können sowohl vertikale als auch horizontale Bewegungen (Ost-West-Richtung) aus dem Modell abgeleitet werden. Um die Genauigkeit der vorhergesagten Deformationen abschätzen zu können, wird als zusätzliche Information für jeden Modellwert die zugehörige Standardabweichung berechnet. Zusätzlich erfolgt eine Modellvalidierung durch Fachexperten. Anzumerken ist, dass die gesamte Prozesskette ein manueller Prozess ist, der mit verschiedenen Skripten durchgeführt wird.

Der Ausgangsdatensatz der Prozesskette ist das BOB.NI-Modell, wobei die letzte Verarbeitung der Daten im Januar 2024 durchgeführt wurde (Stand Oktober 2024). Dabei handelt es sich um einen Punktdatensatz, der die Bodenbewegungen im Bereich der Landesfläche Niedersachsens in einem 200 m × 200 m-Punktraster beschreibt. Die einzelnen Punkte beinhalten unterschiedliche Attribute, die in Tabelle 1 zusammengefasst sind.

Tabelle 1: *Attribute BOB.NI-Modell*

X	X-Koordinate des Punktes
Y	Y-Koordinate des Punktes
vUp	Vertikale Bodenbewegungsgeschwindigkeit
s_vUp	Standardabweichung Vertikalgeschwindigkeit
vEast	Horizontale (Ost-West) Bodenbewegungsgeschwindigkeit
s_vEast	Standardabweichung Horizontalgeschwindigkeit

3.4 Rasterbasiertes Bodenbewegungsmodell

Die Anwendung des BOB.NI-Modells in Form einer punktförmigen Nutzung ist sowohl für die Verarbeitung als auch für die Visualisierung nicht geeignet. In der Konsequenz werden die vektoriellen Punktrasterdaten des BOB.NI-Modells in Rasterobjekte mittels der flächenhaften Triangulated Irregular Network (TIN)-Interpolation überführt. Im Rahmen dieses Prozesses werden insgesamt vier Rasterdateien auf Basis der Attribute (siehe Tabelle 1) erstellt. Jede dieser Dateien enthält ausschließlich Informationen über *vUp*, *s_vUp*, *vEast* oder *s_vEast*, welche über den entsprechenden Pixelwert repräsentiert werden. Als Datenformat wird GeoTIFF verwendet. Zusätzlich wird die

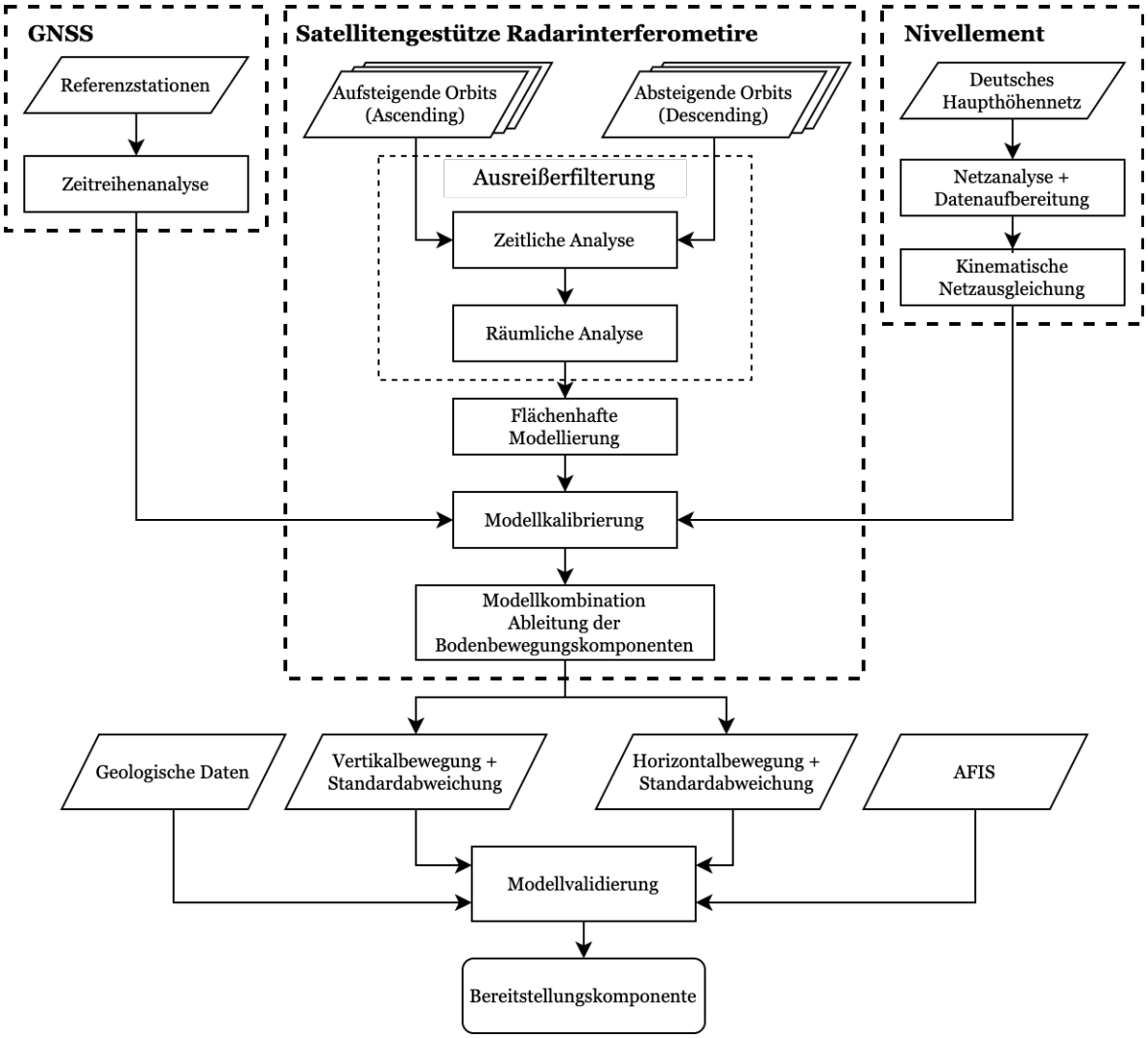
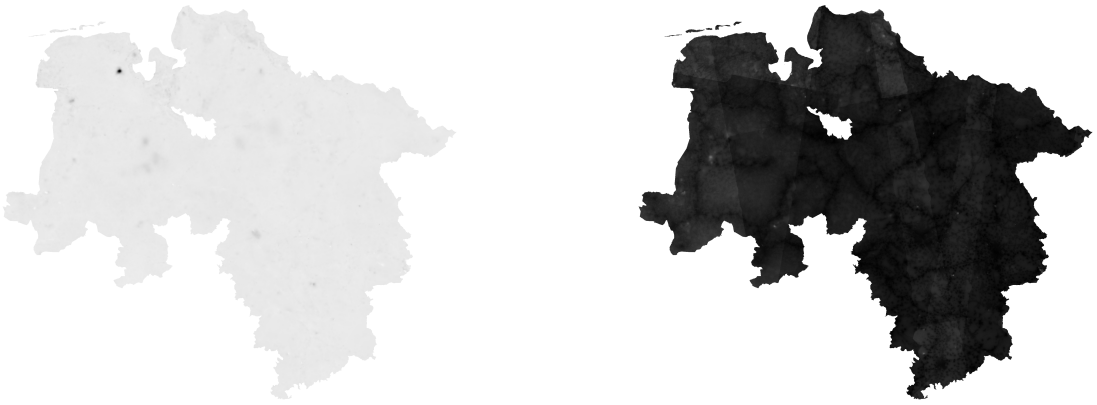


Abbildung 9: *Prozesskette zur flächenhaften Modellierung von Bodenbewegungen (eigene Darstellung nach Brockmeyer, 2024)*

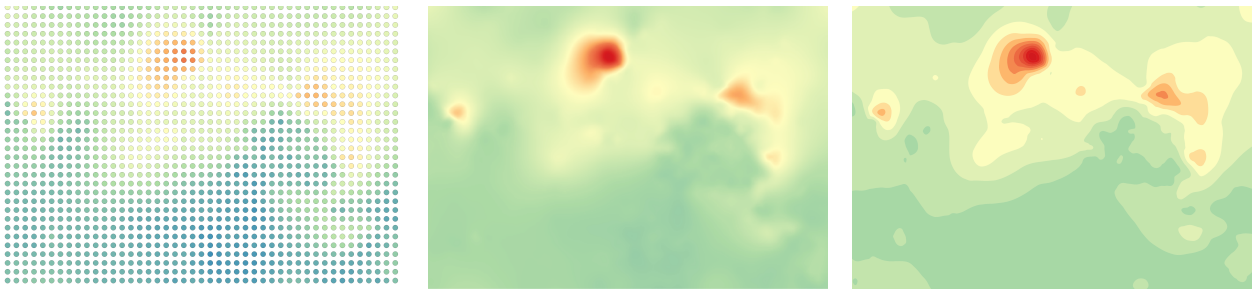
Auflösung der Rasterdateien bei der TIN-Interpolation derart modifiziert, dass die Auflösung des punktbasierten Eingangsdatensatzes (200 m × 200 m) mittels einer bikubischen Interpolation in ein 20 m × 20 m-Raster überführt wird. Dadurch findet eine Verdichtung der Modelldaten statt, was sowohl den Detaillierungsgrad erhöht als auch die Möglichkeit zur Ableitung feinerer Strukturen in Folgeprodukten eröffnet (Borchers, 2022). Inwieweit diese Hochskalierung für die Abfrage der Daten notwendig ist, wird in Kapitel 5.2 analysiert. Die vier resultierenden Rasterdateien werden im weiteren Verlauf dieser Arbeit als *rasterbasiertes Bodenbewegungsmodell* bezeichnet und dienen im Rahmen dieser Arbeit als primäre Datenquelle. Ihre primäre Funktion besteht in der Abfrage von Bodenbewegungen innerhalb der BOB.NI-WebApp. Damit bilden sie die Grundlage für Einzelpunktmessungen oder eine Folge von Einzelpunktmessungen in Form von Profilableitungen. Das rasterbasierte Bodenbewegungsmodell für die Vertikalgeschwindigkeit ist in Abbildung 10 dargestellt. Abgebildet werden die Vertikalgeschwindigkeiten (10a) und die zugehörigen Standardabweichungen (10b). Die Pixelwerte repräsentieren diskrete Werte, wobei die Graustufen in der Abbildung einen Verlauf vom Minimum (schwarz) zum Maximum (weiß) des jeweiligen Attributs im Datensatz darstellen.



(a) *Vertikalgeschwindigkeit* **(b)** *Standardabweichung der Vertikalgeschwindigkeit*

Abbildung 10: *Rasterbasiertes Bodenbewegungsmodell für Niedersachsen*

Neben der Abfrage von Bodenbewegungen bildet das rasterbasierte Bodenbewegungsmodell die Grundlage für die Generierung weiterer relevanter Datensätze wie Konturlinien und Verdachtsgebiete. Bei den Konturlinien werden auf Basis des rasterbasierten Bodenbewegungsmodell über die Festlegung der Klassenbreite (0,5 mm für Bodenbewegungen und 0,2 mm für Standardabweichung) ausgehend vom Wert 0 an den Isolinien getrennte Ringobjekte gebildet. Diese werden entsprechend einer Lookup-Table einheitlich farblich dargestellt und dienen als Basiskarte für die BOB.NI-WebApp (Borchers, 2022). Zur besseren Handhabung werden diese in sogenannte Vector-Tiles umgewandelt, die eine effizientere Verarbeitung und Darstellung in webbasierten Anwendungen ermöglichen. Die Ergebnisse der verschiedenen Datenaufbereitungen zur Analyse und Visualisierung des BOB.NI-Modells sind in Abbildung 11 mit der einheitlichen farblichen Codierung dargestellt.



(a) *Punktbasiertes Modell* **(b)** *Rasterbasiertes Modell* **(c)** *Abgeleitete Konturen*

Abbildung 11: *Datenaufbereitung für Analyse und Visualisierung des BOB.NI-Modells*

Die automatisierte Generierung und Speicherung der Vector-Tiles aus dem punktbasierten BOB.NI-Modell ist bereits durch den sogenannten *bob-tiling-service* umgesetzt, der in Kapitel 6.3 näher beschrieben wird.

4 Anforderungsanalyse

Die Anforderungsanalyse stellt einen zentralen Bestandteil dieser Masterarbeit dar. Zu Beginn wird der aktuelle IST-Zustand beschrieben, wobei die derzeitige Handhabung sowie der Entwicklungsstand des BOB.NI innerhalb des LGLN detailliert dargestellt wird. Darauf aufbauend werden konkrete Anforderungen an den BOB.NI auf Basis von Stakeholdern, Dokumenten und Bestandssystemen ermittelt. Diese fokussieren sich im Rahmen dieser Arbeit auf die Architektur und das Backend. Die identifizierten Anforderungen werden in funktionale und qualitative Anforderungen kategorisiert. Jede Anforderung wird mit F für funktionale Anforderungen und Q für Qualitätsanforderungen gekennzeichnet und nummeriert. Zum Teil stellen die unterschiedlichen Quellen gleiche Anforderungen. Diese werden nicht mehrfach aufgelistet. Aufgrund der Umsetzung als Minimum Viable Product (MVP) werden alle Anforderungen als Muss-Anforderung deklariert. Alle identifizierten Anforderungen sind im Anhang A strukturiert aufgelistet.

4.1 IST-Zustand

Derzeit existiert in Niedersachsen kein dedizierter Bodenbewegungsdienst, der flächendeckend Bodenbewegungsdaten bereitstellt. Zwar stehen in Deutschland und Europa entsprechende Dienste zur Verfügung, jedoch bieten diese keine flächendeckenden Informationen, sondern präsentieren die Bodenbewegungen punktuell. Um diese Lücke zu schließen, wurde eine Prozesskette entwickelt (siehe Kapitel 3.3), die nun flächendeckende Daten liefert, deren Bereitstellung jedoch noch nicht realisiert ist. Um diese Lücke zu schließen, wurde ein Prototyp zur Bereitstellung dieser Daten entwickelt. Dieser entspricht jedoch nicht der IT-Strategie des LGLN und auch sonst nicht der einer modernen IT-Infrastruktur. Eine detaillierte Analyse dieses Prototyps erfolgt in Kapitel 5.1. Vor diesem Hintergrund hat sich das LGLN das Ziel gesetzt, einen ganzheitlichen BOB.NI zu schaffen. Dieser soll sowohl die Datenverwaltung und -verarbeitung als auch die Bereitstellung übernehmen und mit der IT-Strategie des LGLN sowie modernen Anforderungen an Cloud-native Architekturen konform sein. Zur Entwicklung solcher Anwendungen wurden im LGLN sogenannte geoLabs eingerichtet. Diese agil operierenden Teams entwickeln Cloud-basierte Lösungen, die als Webanwendungen bereitgestellt werden. Langfristig sind die geoLabs als DevSecOps-Teams für den Betrieb und die Einhaltung von Sicherheits- und Datenschutzanforderungen zuständig (LGLN, 2022). Das geoLabs-Team „Geodätische Services“ widmet sich der Entwicklung einer Cloud-nativen Lösung für den BOB.NI. Erste Erfolge wurden bereits mit dem sogenannten *bob-tiling-service* erzielt, der die Daten des BOB.NI-Modells automatisiert für die Visualisierung aufbereitet. Allerdings muss die Prozesskette bei neuen Daten manuell gestartet werden, und es gibt noch kein Frontend für die BOB.NI-WebApp. Um eine ganzheitliche Cloud-native Architektur zur automatisierten Verwaltung und Verarbeitung von Bodenbewegungen zu schaffen, ist eine detaillierte Anforderungsanalyse erforderlich, die im Folgenden durchgeführt wird.

4.2 Anforderungen von Stakeholdern

Eine wichtige Quelle für Anforderungen sind Stakeholder. Stakeholder sind in der Regel Personen oder Personengruppen, die einen direkten oder indirekten Einfluss auf die Anforderungen an die zu entwickelnde Software haben. Sie sind sowohl Quelle der Anforderungen als auch Teil der Systemumgebung. Dabei definieren die Stakeholder häufig spezifische Anforderungen an die Software. Daher ist es notwendig, die wichtigsten Eigenschaften der Stakeholder im Nutzungskontext zu beschreiben,

um die gestellten Anforderungen vollständig zu verstehen (Hermann, 2022). Im Folgenden werden zunächst relevante Stakeholder identifiziert und durch sogenannte Personas dargestellt. Personas sind fiktive Charaktere, die sich an den realen Eigenschaften und Verhaltensweisen tatsächlicher Personen orientieren. Anschließend werden die von den Stakeholdern geäußerten Anforderungen an die Software in Form von User Stories aufgelistet, ggf. zusammengefasst und zu konkreten Anforderungen formuliert.

4.2.1 Personas

Im konkreten Anwendungsfall werden grundsätzlich zwei Arten von Stakeholdern unterschieden: das Entwicklungsteam und die Nutzer des BOB.NI. Das Entwicklungsteam stellt die zentrale Umsetzungseinheit des BOB.NI dar. Es besteht aus Entwicklern, die gemeinsam als geoLab-Team an der Realisierung der Software arbeiten. Dieses Team trägt die Verantwortung für die Implementierung und den Betrieb der Anwendung und sorgt dafür, dass die Software den festgelegten Spezifikationen entspricht. Die Nutzer des BOB.NI werden in zwei Hauptgruppen unterteilt: Fachnutzer und Privatnutzer. Fachnutzer umfassen professionelle Anwender, die den BOB.NI im Rahmen ihrer beruflichen Tätigkeit nutzen. Dazu zählen bspw. interne Angestellte im LGLN sowie Personen in anderen Behörden oder Unternehmen. Privatnutzer hingegen bestehen aus individuellen Anwendern, welche den BOB.NI für persönliche Zwecke verwenden.

Entwicklungsteam
Lisa Müller ist die Product Ownerin des neu gegründeten SCRUM-Entwicklungsteams. Mit ihren 37 Jahren bringt sie bereits zehn Jahre Erfahrung in der Softwareentwicklung und vier Jahre als Product Ownerin mit. Sie hat einen Masterabschluss in Informatik und legt großen Wert darauf, dass die Qualität der gesamten Architektur an erster Stelle steht. In ihrer Rolle als Product Ownerin ist Lisa dafür verantwortlich, dass die Architektur des Projekts auf einem sehr aktuellen technologischen Stand ist. Sie versteht, dass eine solide und moderne Architektur die Grundlage für zukünftige funktionale Erweiterungen bildet. Daher ist es ihr Ziel, zunächst die Qualität der Architektur sicherzustellen, bevor sukzessive weitere funktionale Anforderungen integriert werden. Lisa arbeitet eng mit ihrem Team zusammen, um sicherzustellen, dass die Implementierungen sowohl den aktuellen technologischen Standards entsprechen als auch den langfristigen Anforderungen des Projekts gerecht werden. Gleichzeitig sucht sie stets nach Möglichkeiten, Kosten zu sparen, ohne die Qualität zu beeinträchtigen.

Fachnutzer
Markus Schneider ist ein erfahrener Vermessungsingenieur im Katasteramt, Mitte 40, mit mehr als 15 Jahren Berufserfahrung. Markus hat einen Abschluss in Geodäsie und ist auf Präzisionsvermessungen und Nivellements spezialisiert. Bei seiner täglichen Arbeit setzt er verschiedene Messinstrumente und -techniken ein, um Grundstücke und Gebäude zu erfassen und zu aktualisieren. In letzter Zeit ist er jedoch immer wieder auf inkonsistente Messdaten gestoßen, vor allem bei Lage- und Höhenfestpunkten. Diese treten vor allem im Bereich einer neu erschlossenen Gaskaverne auf. Nun vermutet er, dass diese Gaskaverne sich auf den Boden und entsprechend auch auf die Festpunkte auswirkt.

Privatnutzer
Sabine Fischer ist eine 52-jährige Hausbesitzerin, deren Gebäude kürzlich Risse in den Wänden und im Fundament bekommen hat. Sie wohnt in der Nähe einer Gaskaverne und macht sich Sorgen, dass die Bodenbewegungen durch die Aktivitäten in der Kaverne verursacht werden könnten. Sabine

hat keinen technischen Hintergrund, aber sie ist sehr besorgt um die Sicherheit und den Wert ihres Eigentums. Ihr Haus ist ihr größtes finanzielles Investment und sie möchte sicherstellen, dass es strukturell sicher bleibt. Nachdem sie die Risse bemerkt hat, sucht Sabine nach einer Möglichkeit, die Ursache zu bestimmen und entsprechende Maßnahmen zu ergreifen.

4.2.2 User-Stories

Im Folgenden werden die geäußerten Anforderungen der Stakeholder mittels User-Stories beschrieben und entsprechende Anforderungen abgeleitet.

- **Q1.)** Die Software muss zunächst als MVP entwickelt werden, wobei eine hochwertige Architektur sicherstellt, dass bestehende Komponenten und neue Anforderungen von Nutzern nahtlos integriert werden können.
  - Als Lisa Müller ist es mir wichtig, zunächst eine qualitativ hochwertige Architektur zu schaffen, in die ich weitere funktionale Anforderungen leicht integrieren kann.
  - Als Lisa Müller möchte ich möglichst schnell den unterschiedlichen Stakeholdern ein lauffähiges Produkt vorzeigen.
  - Als Lisa Müller möchte ich bereits entwickelte Komponenten nahtlos in die bestehende Architektur integrieren können.
- **Q2.)** Die Software muss möglichst ausschließlich kostengünstige Cloud-native Technologien verwenden, um die Betriebskosten niedrig zu halten und gleichzeitig eine effiziente Entwicklung sicherzustellen.
  - Als Lisa Müller möchte ich kostengünstige Technologien verwenden, um die Betriebskosten zu Beginn der Entwicklung niedrig zu halten.
  - Als Lisa Müller möchte ich moderne Technologien nutzen, um stets auf dem neuesten Stand der Technik zu bleiben.
- **F1.)** Die Software muss die Möglichkeit bieten, Bodenbewegungsstatistiken für benutzerdefinierte Gebiete zu berechnen, um einen umfassenden Überblick über die vertikalen und horizontalen Verschiebungen zu erhalten.
  - Als Markus Schneider möchte ich Statistiken für definierte Bereiche berechnen lassen können, um beobachtete Abweichungen meiner Messungen mit Bodenbewegungen evaluieren zu können.
  - Als Markus Schneider möchte ich sowohl Statistiken zur vertikalen als auch zu horizontalen Verschiebungen bekommen.
- **Q3.)** Die Bodenbewegungsdaten müssen eine hohe Genauigkeit aufweisen, um präzise und verlässliche Informationen bereitzustellen.
  - Als Markus Schneider möchte ich, dass die Bodenbewegungsdaten eine hohe Genauigkeit aufweisen, um diese mit meinen hochgenauen Messungen evaluieren zu können.
  - Als Sabine Fischer möchte ich, dass die Bodenbewegungsdaten die Bewegung in der Nähe meines Hauses in Zentimetern präzise abfragen können, um einen genauen Nachweis dieser Bewegungen vorlegen zu können.
- **Q4.)** Die Software muss möglichst aktuelle Daten visualisieren und bei neuen Daten diese schnell und automatisiert integrieren, um stets die neuesten Informationen darzustellen.

- Als Lisa Müller möchte ich eine möglichst automatisierte Datenverarbeitungspipeline, um den manuellen Aufwand so gering wie möglich zu halten.
- Als Markus Schneider möchte ich stets die aktuellsten Daten visualisiert bekommen, um meine geplanten Messungen in der Nähe einer Gaskaverne effizient planen zu können.
- Als Sabine Fischer möchte ich möglichst aktuelle Bodenbewegungen sehen, um zu gucken, ob mein Haus davon betroffen ist.

4.3 Anforderungen aus Dokumenten

Neben den Stakeholdern sind Dokumente eine wichtige Quelle für Anforderungen. Sie dienen oft als Grundlage für allgemeine Anforderungen an die Software, die den Stakeholdern oft nicht bekannt sind. So dienen Normen und Standards, Strategiepapiere oder auch Unternehmens- / Behördenrichtlinien als Quelle für Anforderungen (Hermann, 2022). Im Rahmen dieser Arbeit werden das Zukunftskonzept derVKV mit dem Zieljahr 2025, die IT-Strategie des LGLN sowie die ISO/IEC 25010, 2011, die Kriterien für die Qualität von Software festlegt, herangezogen.

Zukunftskonzept VKV2025
Um die zukünftigen Herausforderungen der VKV effektiv zu meistern und die Verwaltung zeitgemäß zu gestalten, hat das LGLN das Zukunftskonzept VKV2025 entwickelt. Dieses Konzept bildet die Grundlage für eine moderne Verwaltung bis zum Jahr 2025. Im Rahmen des Zukunftskonzepts wurde ein Thesenpapier veröffentlicht, das die Zielsetzungen und Handlungsfelder in sechs allgemeinen sowie 23 aufgabenspezifischen Thesen darlegt. Sowohl eine These als auch ein Handlungsfeld beziehen sich auf das Thema der Bodenbewegungen, weshalb sich folgende Anforderung ableiten lässt:

- **F2.)** Die Software muss flächendeckenden Bodenbewegungen unter Berücksichtigung der zeitlichen Komponente visualisieren.

IT-Strategie des LGLN
Die IT-Strategie des LGLN ist ein umfassendes Strategiepapier, das die Umsetzung und Neuausrichtung der IT des LGLN bis zum Jahr 2025 skizziert. Es beschreibt eine wegweisende neue IT-Referenzarchitektur, die den Übergang von bisher weitgehend monolithischen Fachanwendungen zu Microservices vorsieht. Ein zentrales Ziel dieser einheitlichen Referenzarchitektur ist es, die bestehenden komplexen Abhängigkeiten zwischen den Fachanwendungen zu beseitigen und somit eine medienbruchfreie Weitergabe von Daten zu ermöglichen. Die Kommunikation soll dabei über eine REST-API erfolgen, die idealerweise dem *OGC API Feature Standard* entspricht. Darüber hinaus wird der Einsatz von Cloud-nativen Technologien angestrebt, um eine hohe Skalierbarkeit zu gewährleisten. Bei Bedarf sollen auch Services von Drittanbietern integriert werden können, um die Architektur zu ergänzen und zu erweitern. (LGLN, 2022)

Aus der IT-Strategie lassen sich demnach folgende Qualitätsanforderungen ableiten:

- **Q5.)** Die Architektur muss als Cloud-basierte Microservice-Architektur konzipiert werden.
- **Q6.)** Die Software muss, wenn erforderlich, verfügbare Komponenten von Drittanbietern als Service nutzen.
- **Q7.)** Die Architektur muss eine medienbruchfreie Weitergabe der Bodenbewegungsdaten ermöglichen.
- **Q8.)** Die Kommunikation mit der Software muss über eine generische REST-API erfolgen.

ISO/IEC 25010 - Standard für Softwarequalität
Neben internen Dokumenten des LGLN dient die ISO/IEC 25010 als Quelle für Anforderungen. Dieser international anerkannte Standard definiert Qualitätsmodelle für Software- und Systemprodukte. Dieser Standard, veröffentlicht von der International Organization for Standardization (ISO) und der International Electrotechnical Commission (IEC), dient als Leitfaden zur Bewertung der Softwarequalität und bildet eine Grundlage für die Qualitätsanforderungen an Softwareprodukte. Die ISO/IEC 25010 umfasst zwei Hauptmodelle: das Produktqualitätsmodell und das Qualitätsmodell für die Nutzung (ISO/IEC 25010, 2011). Im Rahmen dieser Arbeit wird nur das Produktqualitätsmodell betrachtet, das acht Qualitätsmerkmale enthält, mit denen die internen und externen Eigenschaften eines Softwareprodukts bewertet werden. Diese Merkmale sind Funktionalität, Leistungseffizienz, Kompatibilität, Gebrauchstauglichkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit und Portabilität (ISO/IEC 25010, 2011). Aufgrund der Anforderung, dass zunächst ein MVP entwickelt werden soll und der Fokus der Arbeit auf dem Backend liegt, werden sowohl die Funktionalität, welche die Vollständigkeit hinsichtlich der Softwarefunktionalität beschreibt, als auch die Gebrauchstauglichkeit, welche die Benutzbarkeit der Software beschreibt, nicht weiter betrachtet.

- **Q9.) Leistungsfähigkeit:** Die Architektur muss eine hohe Leistungseffizienz gewährleisten, indem sie kurze Reaktionszeiten und minimalen Ressourcenverbrauch unter verschiedenen Lastbedingungen sicherstellt.
- **Q10.) Kompatibilität:** Die Software muss eine hohe Kompatibilität aufweisen, indem sie nahtlos mit anderen Systemen, Plattformen und Anwendungen zusammenarbeitet.
- **Q11.) Zuverlässigkeit:** Die Architektur muss eine hohe Zuverlässigkeit sicherstellen, indem sie kontinuierlich und fehlerfrei arbeitet und Ausfallzeiten minimiert.
- **Q12.) Sicherheit:** Die Software muss höchste Sicherheitsstandards erfüllen, indem sie Daten vor unbefugtem Zugriff schützt und sichere Datenübertragungen gewährleistet.
- **Q13.) Wartbarkeit:** Die Architektur muss eine hohe Wartbarkeit unterstützen, indem sie leicht zu aktualisieren und zu modifizieren ist.
- **Q14.) Übertragbarkeit:** Die Software muss eine hohe Übertragbarkeit aufweisen, indem sie problemlos auf verschiedene Hardware- und Softwareplattformen migriert werden kann.

4.4 Anforderungen aus Bestandssystemen

Verschiedene Institutionen und Behörden haben in den letzten Jahren webbasierte Anwendungen zur Bereitstellung von Bodenbewegungsinformationen veröffentlicht. Diese decken unterschiedliche Bereiche ab und reichen von der kontinentalen bis zur nationalen Ebene. Diese Anwendungen zeigen wesentliche Unterschiede in der Art der Darstellung, insbesondere in den Daten- und Visualisierungskomponenten, sowie im Umfang und der Komplexität der zur Verfügung stehenden Analysetools. Im Rahmen der Entwicklung des Prototyps wurden eine Vielzahl dieser Systeme eingehend untersucht. Aufgrund dessen wird im Rahmen dieser Arbeit auf eine ausführliche Analyse der Bestandssysteme verzichtet und auf Borchers, 2022 verwiesen. Basierend auf der Analyse wurden folgende wesentliche funktionale Anforderungen abgeleitet, die für die Entwicklung des MVP als wichtig erachtet werden:

- **F3.)** Die Software muss eine Funktion zur Ableitung von Profilen bieten, bei der im Kartenteil der Webanwendung ein freies Profil durch Zeichnen definiert wird, das automatisch mit

den Daten des Bodenbewegungsmodells verschnitten wird, um im abbildenden Teil den kontinuierlichen Verlauf der Geschwindigkeit entlang dieses Profils zu präsentieren.

- **F4.)** Die Software muss Gebiete, in denen Bodenbewegungen zu erwarten sind, auf einer Karte visualisieren und entsprechende Statistiken zu diesen Gebieten liefern.

5 Analyse

Das Kapitel der Analyse widmet sich der eingehenden Untersuchung verschiedener Bestandteile, die maßgeblich Einfluss auf die Architektur sowie die Verwaltung und Verarbeitung der rasterbasierten Bodenbewegungsdaten haben. Zunächst wird der entwickelte Prototyp der BOB.NI-WebApp analysiert und mit den im vorherigen Kapitel erhobenen Anforderungen abgeglichen. Anschließend werden die rasterbasierten Bodenbewegungsdaten untersucht, um eine möglichst präzise und effiziente Abfrage der Daten zu ermöglichen. Abschließend wird die Eignung Cloud-optimierter Geodaten hinsichtlich verschiedener Gesichtspunkte wie Kosten und Laufzeit untersucht. Der Fokus liegt primär auf dem rasterbasierten Bodenbewegungsmodell und der Verwaltung und Speicherung mittels COGs. Da im Rahmen dieser Arbeit jedoch auch Vektordaten benötigt werden, wird zusätzlich das Format FGB auf dessen Eignung kurz untersucht.

5.1 Analyse Prototyp

Wie bereits beschrieben, wurde im LGLN ein Prototyp der BOB.NI-WebApp entwickelt, der in Borchers, 2022 dokumentiert ist. Diese BOB.NI-WebApp bildet in Kombination mit der Prozesskette zur Generierung des BOB.NI-Modells sowie der entsprechenden Vorprozessierung den gesamten BOB.NI. Die BOB.NI-WebApp steht derzeit nur einem eingeschränkten Nutzerkreis im internen Netzwerk des LGLN zur Verfügung. Im folgenden Kapitel wird dieser Prototyp im Detail beschrieben, wobei insbesondere auf die Architektur und die Funktionsweise eingegangen wird, und anschließend auf Basis der erhobenen Anforderungen bewertet. Da die aktuelle Entwicklung gemäß der Anforderung *Q1* als MVP erfolgen soll, liegt der Schwerpunkt der Analyse auf den Qualitätsanforderungen. Zunächst erfolgt eine detaillierte Beschreibung der Architektur mit den verwendeten Technologien. Anschließend wird diese detailliert analysiert und abschließend ein Zwischenfazit für den Prototyp gezogen.

5.1.1 Architektur

Die Architektur des Prototyps basiert im Wesentlichen auf dem klassischen Client-Server-Modell und ist als Single-Page-Applikation (SPA) konzipiert. Sie setzt sich hauptsächlich aus vier Komponenten zusammen: der Infrastruktur, dem Frontend, dem Backend und der manuellen Datenvorverarbeitung der Bodenbewegungsdaten. Der grundlegende Aufbau sowie das vereinfachte Zusammenspiel der einzelnen Komponenten ist in Abbildung 12 dargestellt.

Die dazugehörige Funktionsweise und das Zusammenspiel der einzelnen Komponenten ist vereinfacht als Sequenzdiagramm in Abbildung 13 dargestellt. Demnach lässt sich die Funktionsweise in zwei wesentliche Bereiche aufteilen: die Vorverarbeitung der Daten durch die Entwickler und die Nutzung der Anwendung durch den Anwender. Im Folgenden werden die einzelnen Komponenten kurz beschrieben und die verwendeten Technologien erläutert.

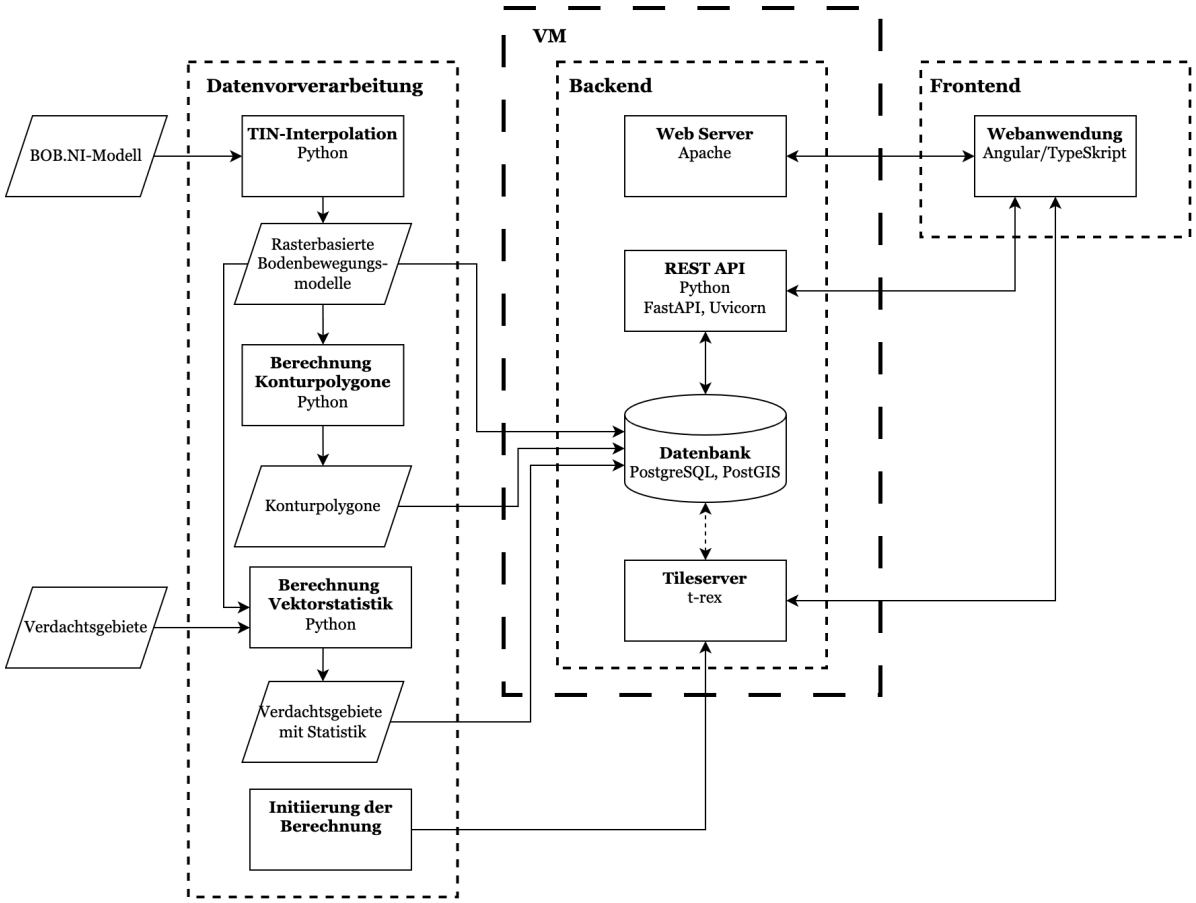


Abbildung 12: *Schematische Darstellung der Architektur des Prototyps*

Infrastruktur

Der Prototyp läuft auf einer VM (Windows Server 2016), die über die erforderlichen Ressourcen für den Betrieb der Anwendung verfügt: eine CPU vom Typ Intel Xeon Gold mit 2,8 GHz und 4 Kernen sowie 16 GB Arbeitsspeicher. Diese VM wird on-premise im Landesnetz des LGLN betrieben.

Datenvorverarbeitung

Die für den Prototyp notwendige Datenvorverarbeitung setzt auf dem BOB.NI-Modell auf und erfolgt manuell durch die Nutzung unterschiedlicher Python-Skripte. Dabei umfasst die Datenvorverarbeitung im Wesentlichen die Umwandlung des BOB.NI-Modells in rasterbasierte Bodenbewegungsmodelle und die Generierung der Konturpolygone, was bereits in Kapitel 3.4 beschrieben wurde. Zusätzlich werden vorhandene Verdachtsgebiete aufbereitet, indem auf Basis des rasterbasierten Bodenbewegungsmodells Statistiken für diese Gebiete berechnet werden.

Backend

Das Backend der Architektur ist ein zentraler Bestandteil und besteht aus mehreren wesentlichen Komponenten und Technologien, die für die Datenverarbeitung und -bereitstellung verantwortlich sind. Zunächst werden die manuell aufbereiteten Bodenbewegungsdaten, einschließlich des BOB.NI-Rastermodells, der BOB.NI-Konturpolygone und der Verdachtsgebiete, in eine PostgreSQL-Datenbank gespeichert. Diese Datenbank wird durch PostGIS, eine Erweiterung für räumliche Daten, ergänzt, die es ermöglicht, Raster- und Vektordaten effizient zu speichern und zu verwalten.

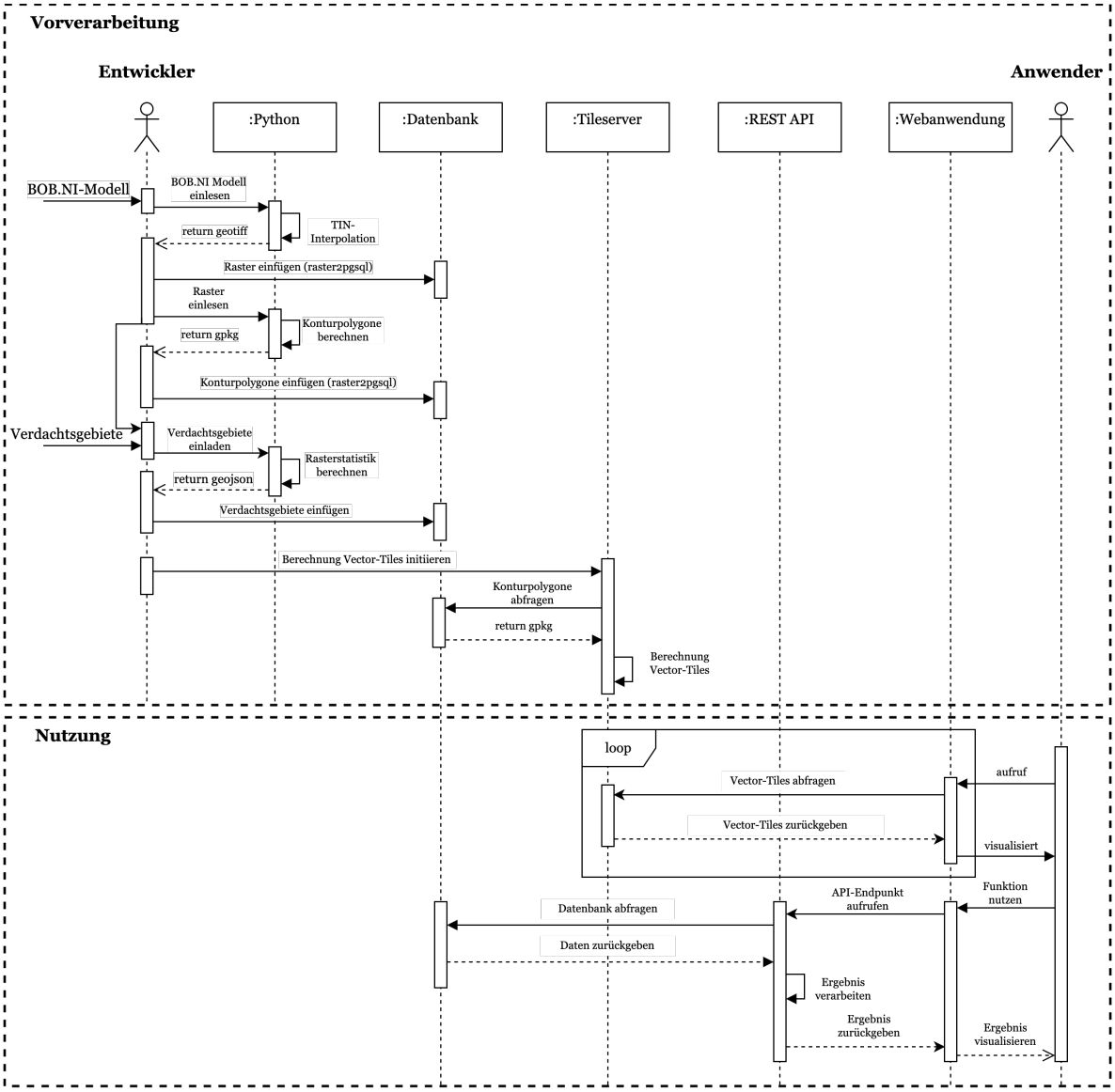


Abbildung 13: *Funktionsweise des Prototyps*

Innerhalb der Architektur dient die Datenbank als primäre Datenquelle.

Zur effizienten Darstellung der Konturpolygone im Frontend werden die Konturpolygone in Vector-Tiles umgewandelt. Dafür kommt der Service *t-rex TileServer* zum Einsatz, der sowohl die Berechnung als auch die Bereitstellung der Vector-Tiles übernimmt. Über eine Konfigurationsdatei werden die Parameter für die Einbindung der Datenbank, das maßstabsabhängige Laden der Datensätze entsprechend dem Generalisierungsgrad sowie weitere Einstellungen zur Kachel-Erzeugung festgelegt (Kalberer, 2021). Die vorberechneten Kacheln werden schließlich über den im *t-rex TileServer* integrierten Webserver statisch für das Frontend bereitgestellt.

Die Abfrage spezifischer Informationen durch den Benutzer über das Frontend erfolgt über eine REST-API, die als Python-Service mit dem Web-Framework *FastAPI* entwickelt wurde. Diese REST-API ermöglicht vordefinierte Datenbankabfragen und übernimmt wesentliche Funktiona-

litäten und Datenverarbeitungsprozesse. Als primäre Funktionalität ist hier die in Anforderung *F3* beschriebene Profilerstellung zu nennen, die im Kapitel 5.2 näher behandelt wird. Neben der primären Funktionalität sind eine Vielzahl weiterer API-Endpunkte für unterschiedliche Aufgaben vorhanden, wie bspw. das Abfragen von Capabilities. Die API ist auf eine vordefinierte Datenquelle festgelegt, sodass eine Änderung der Datenquelle z.B. über die Payload nicht möglich ist. Die Integration des Asynchronous Server Gateway Interface (ASGI) Dienstes *uvicorn* ermöglicht die Bereitstellung der Schnittstelle auf dem Server sowie den Empfang und die Beantwortung von Anfragen (Tiangolo, 2024).

Frontend
Das Frontend der BOB.NI-WebApp wurde als SPA mit dem Webentwicklungsframework Angular entwickelt. Durch die Umsetzung als SPA wird initial nur ein einziges HTML-Dokument geladen, dessen Inhalt durch die lokale Prozesslogik dynamisch angepasst wird, wodurch eine besonders schnelle Interaktion mit dem Frontend (Tremp, 2021) ermöglicht wird. Die Bereitstellung der Frontend-Anwendung erfolgt über einen Apache-Webserver.

5.1.2 Bewertung Architektur

Im Folgenden wird der Prototyp anhand der im vorherigen Kapitel beschriebenen Komponenten und der erhobenen Anforderungen untersucht. Dabei konzentriert sich die Analyse auf die Infrastruktur, die Datenvorverarbeitung und das Backend, da der Schwerpunkt dieser Arbeit nicht auf dem Frontend liegt. Es wird geprüft, inwieweit die Komponenten des Prototyps den in der Anforderungsanalyse definierten Anforderungen entsprechen, wobei primär die Qualitätsanforderungen im Fokus stehen. Da jedoch nicht alle Anforderungen zur Bewertung des Prototyps geeignet sind, wird eine passende Auswahl getroffen. Für jede Komponente werden nur die spezifischen Anforderungen herangezogen, die sich auf diese Komponente beziehen. Funktionale Anforderungen bleiben unberücksichtigt, da sie oft keinen direkten Einfluss auf die Architektur haben.

Untersuchung Infrastruktur
Für die Analyse und Bewertung der Infrastruktur lassen sich die Qualitätsanforderungen aus der ISO/IEC 25010 heranziehen.

Q9.) Leistungsfähigkeit: Die Architektur soll eine hohe Leistungseffizienz gewährleisten, indem sie möglichst kurze Antwortzeiten und minimalen Ressourcenverbrauch unter verschiedenen Lastbedingungen sicherstellt. Die vorhandenen Rechenressourcen sind für den derzeit eingeschränkten Nutzerkreis ausreichend. Es fehlt jedoch die Skalierbarkeit der Ressourcen unter verschiedenen Lastbedingungen, was bei stark schwankenden Anforderungen im öffentlichen Betrieb zu Engpässen führen könnte. Dadurch wird auch diese Anforderung vom Prototyp nicht erfüllt.

Q10.) Kompatibilität: Die Software soll eine hohe Kompatibilität aufweisen, indem sie nahtlos mit anderen Systemen, Plattformen und Anwendungen zusammenarbeitet. Die Lösung des Prototyps ist eher monolithisch und fachspezifisch für das konkrete Frontend konzipiert, wodurch eine Nutzung für andere Daten oder Datenquellen nicht möglich ist. Zudem läuft die Anwendung gekapselt auf einer VM, was die Integration mit anderen Systemen zusätzlich erschwert. Dadurch ist die Anforderung der Kompatibilität nicht erfüllt.

Q11.) Zuverlässigkeit: Die Architektur muss ein hohes Maß an Zuverlässigkeit gewährleisten, indem sie einen kontinuierlichen und fehlerfreien Betrieb sicherstellt und Ausfallzeiten minimiert. Diese

Anforderung wird nicht vollständig erfüllt. Es gibt keine Ausfallsicherung für die VM, was bedeutet, dass im Falle eines Fehlers kein redundantes System zur Verfügung steht, das automatisch übernehmen könnte. Darüber hinaus erfordern Änderungen oder Updates der Software ein Herunterfahren des Systems. Dies führt zu Ausfallzeiten, welche die Zuverlässigkeit der Architektur beeinträchtigen.

Q12.) Sicherheit: Die Software soll höchste Sicherheitsstandards erfüllen, indem sie Daten vor unbefugtem Zugriff schützt und sichere Datenübertragungen gewährleistet. Der aktuelle Prototyp ist nur für einen eingeschränkten Nutzerkreis innerhalb des LGLN-internen Netzes zugänglich. Aufgrund dessen spielt die Sicherheit keine primäre Rolle, weshalb Sicherheitsmechanismen wie Zero-Trust nicht berücksichtigt wurden. Dennoch befindet sich die Anwendung auf einer VM innerhalb des internen Netzwerks, was ein gewisses Maß an Sicherheit gewährleistet, wodurch die Anforderung der Sicherheit zum Teil erfüllt ist, in der Form jedoch nicht für einen öffentlichen Einsatz geeignet ist.

Q13.) Wartbarkeit: Die Architektur soll eine hohe Wartbarkeit unterstützen, indem sie leicht zu aktualisieren und zu modifizieren ist. Diese Anforderung ist nicht erfüllt, da Änderungen oder Updates am Quellcode der jeweiligen Bestandteile einen aufwendigen Austausch auf der VM erfordern. Während dieser Zeit ist die Anwendung nicht erreichbar, was sich zusätzlich negativ auf die Zuverlässigkeit auswirkt.

Q14.) Übertragbarkeit: Die Software soll eine hohe Übertragbarkeit aufweisen, indem sie problemlos auf verschiedene Hardware- und Softwareplattformen migriert werden kann. Aufgrund der Nutzung einer VM ist die Übertragbarkeit jedoch eingeschränkt, da VMs oft an bestimmte Hypervisoren und Konfigurationen gebunden sind, was die Migration auf andere Systeme erschwert.

Insgesamt lässt sich daher sagen, dass die Infrastruktur für eine prototypische Umsetzung für einen kleinen, internen Nutzerkreis durchaus ausreichend ist, ein produktiver Einsatz mit öffentlichem Zugang jedoch nicht empfohlen werden kann, da grundlegende, standardisierte Softwarequalitätsanforderungen nicht erfüllt sind.

Untersuchung Datenvorverarbeitung

Für die Bewertung der Datenvorverarbeitung innerhalb des Prototyps sind die Anforderungen *Q4* und *Q7* von Bedeutung. Die Anforderung *Q4* verlangt, dass die Daten möglichst aktuell visualisiert und neue Daten schnell und automatisiert integriert werden. Bei Betrachtung des Sequenzdiagramms in Abbildung 13, ist der sehr aufwändige, manuelle Workflow gut zu erkennen. Sowohl die Datenaufbereitung als auch die Datenpflege müssen manuell durchgeführt werden. Das bedeutet, dass bei einem neuen BOB.NI-Modell die gesamte Datenvorverarbeitung erneut manuell durchgeführt werden muss. Bei einer ganzheitlichen Betrachtung des BOB.NI müsste die gesamte Prozesskette aus Kapitel 3.3 erneut manuell durchlaufen werden, falls neue Bodenbewegungsdaten vom BGR vorliegen. Durch diesen sehr aufwendigen, zeitintensiven Prozess kann diese Anforderung nicht erfüllt werden. Auch die Anforderung *Q7*, die eine medienbruchfreie Weitergabe der Bodenbewegungsdaten fordert, wird nicht erfüllt. Die gesamte Vorverarbeitung der Bodenbewegungsdaten, einschließlich der Generierung des BOB.NI-Modells, erfordert den Einsatz zahlreicher unterschiedlicher Skripte und Applikationen. Darüber hinaus ist ein umfangreiches manuelles Datenmanagement sowie der manuelle Austausch einzelner Daten und Dateien erforderlich, was die Effizienz der Datenverarbeitung stark beeinträchtigt. Zusätzlich sind zum Teil Skripte vorhanden, die auch als Funktionalitäten innerhalb der BOB.NI-WebApp Anwendung finden könnten. Ein konkretes Beispiel ist die Berechnung der Rasterstatistik, die derzeit händisch nur für die Verdachtsgebiete durchgeführt wird. Diese Berechnung könnte jedoch als Funktionalität in die BOB.NI-WebApp integriert werden, sodass der

Anwender bspw. ein Polygon zeichnet und für dieses Gebiet automatisch die entsprechenden Statistiken berechnet werden. Diese Umsetzung würde innerhalb der neuen Architektur die Anforderung *F1* umsetzen.

Zusammenfassend lässt sich sagen, dass der aktuelle Prototyp eine zu aufwendige manuelle Vorverarbeitung der Daten erfordert, die durch zahlreiche Skripte und händische Datenmanagementprozesse durchgeführt wird. Dies entspricht nicht einer modernen Cloud-nativen Architektur und auch nicht der IT-Strategie des LGLN.

Untersuchung Backend
Die Anforderung *Q2* fordert die Implementierung der Software als Cloud-basierte Microservice-Architektur. Das Backend des Prototyps weist eine eher monolithische Struktur auf, in der die Komponenten eng miteinander verknüpft sind. Zudem ist, wie bereits aus der Betrachtung der Infrastruktur ersichtlich, keine Implementierung in der Cloud vorgesehen.

Die REST-API, die als Python-Service mit dem Web-Framework FastAPI entwickelt wurde, übernimmt wesentliche Funktionalitäten und Datenverarbeitungsprozesse. Obwohl FastAPI eine moderne und effiziente Lösung für die Entwicklung von APIs und auch Microservices darstellt, entspricht die Implementierung nicht den Eigenschaften eines Microservices. Nach Tremp, 2021 ist ein Microservice „eine eigenständige fachlich klar fokussierte Teilfunktion einer Applikation [...]. Ihm steht ein eigener Ausführungscontainer zur Verfügung. Der Zugriff erfolgt über eine klar definierte synchrone und/oder asynchrone Schnittstelle“ (Tremp, 2021). Vergleicht man diese Definition mit der Implementierung der REST-API, werden einige Diskrepanzen deutlich. Ein zentraler Aspekt von Microservices ist das Single Responsibility Prinzip, also die Konzentration auf eine einzige Aufgabe oder Funktion. Die REST-API des Prototyps hingegen übernimmt mehrere Funktionen, was diesem Prinzip widerspricht und sie monolithisch macht. Ein weiteres Kriterium ist das generische Schnittstellendesign. Microservices kommunizieren über klar definierte, generische Schnittstellen, die unabhängig von spezifischen Implementierungen oder Frontends sind. Die REST-API hingegen hat Schnittstellen, die stark auf das Frontend zugeschnitten sind. So kann bspw. die Datenquelle nicht über die Payload geändert werden. Dadurch steht die REST-API des Prototyps im direkten Widerspruch zur Anforderung *Q2*. Zusätzlich existiert mit der PostgreSQL-Datenbank nur eine primäre Datenquelle. Dadurch entsteht eine starke Abhängigkeit zwischen den Komponenten, was zu Engpässen bei einer Vielzahl an Anfragen führen kann und die Skalierung erschwert. Dies wirkt sich auch negativ auf die Zuverlässigkeit aus, da beim Ausfall einer Komponente, z.B. der Datenbank, das gesamte System nicht mehr funktioniert, was sich negativ auf die Erfüllung der Anforderung *Q11* auswirkt. Des Weiteren erschwert diese enge Koppelung von Komponenten das Erweitern der Funktionalitäten. So ist eine automatisierte Umsetzung der beschriebenen Datenvorverarbeitung nur unter einer signifikanten Anpassung der gesamten Architektur möglich, was zusätzlich die Anforderung *Q4* und *Q7* erschwert.

Die Nutzung zur Erzeugung und Bereitstellung der Vector-Tiles über den *t-rex Tileserver* stellt einen Schritt in die Modulare Architektur dar, allerdings besteht ähnlich wie bei der REST-API auch hier eine starke Abhängigkeit zur zentralen Datenbank.

Zusammenfassend lässt sich sagen, dass das Backend des Prototyps aufgrund der starken Abhängigkeiten der einzelnen Komponenten und der generellen Implementierung in seiner Form einer monolithischen Architektur ähnelt. Eine direkte Nutzung der REST-API für eine Microservice-Architektur ist daher nicht möglich und erfordert ein Neugestaltung des Backends, um den Anforderungen gerecht zu werden und die Anwendung Cloud-fähig zu machen.

5.1.3 Zwischenfazit Prototyp

Der untersuchte Prototyp der BOB.NI-WebApp zeigt in Bezug auf Infrastruktur, Datenvorverarbeitung und Backend deutliche Schwächen, die einen produktiven Einsatz mit öffentlichem Zugang verhindern. Diese Defizite stehen im deutlichen Widerspruch zu den erhobenen Qualitätsanforderungen.

Obwohl die Infrastruktur für eine prototypische Umsetzung ausreichend ist, erfüllt sie grundlegende, standardisierte Softwarequalitätsanforderungen aus der ISO/IEC 25010 nicht, wodurch diese nicht für den produktiven Betrieb geeignet ist. Darüber hinaus erfordert der Prototyp eine aufwändige manuelle Vorverarbeitung der Datengrundlage. Eine Automatisierung dieser Prozesse ist aufgrund der bestehenden Architektur ohne Anpassung weiterer Komponenten nur schwer möglich. Zusätzlich weisen die einzelnen Komponenten starke Abhängigkeiten untereinander auf, was monolithischen Architekturen ähnelt.

Für einen produktiven Einsatz in der Cloud ist daher ein Umstieg auf Cloud-native Technologien und Architekturen unumgänglich. Der bestehende Prototyp eignet sich aufgrund seiner monolithischen Struktur und der aufwendigen manuellen Datenverarbeitung nicht für eine direkte Migration in die Cloud. Stattdessen bedarf es einer neuen Architektur, die auf modernen Cloud-nativen Ansätzen basiert. Dies umfasst die Implementierung von Microservices, die Automatisierung der Datenverarbeitungsprozesse und den Einsatz skalierbarer und flexibler Infrastrukturtechnologien wie Kubernetes, Docker und serverlosen Architekturen. Nur so kann sichergestellt werden, dass die Anwendung den Anforderungen einer modernen, cloud-basierten Umgebung gerecht wird und langfristig erfolgreich betrieben werden kann.

5.2 Analyse Datenabfrage

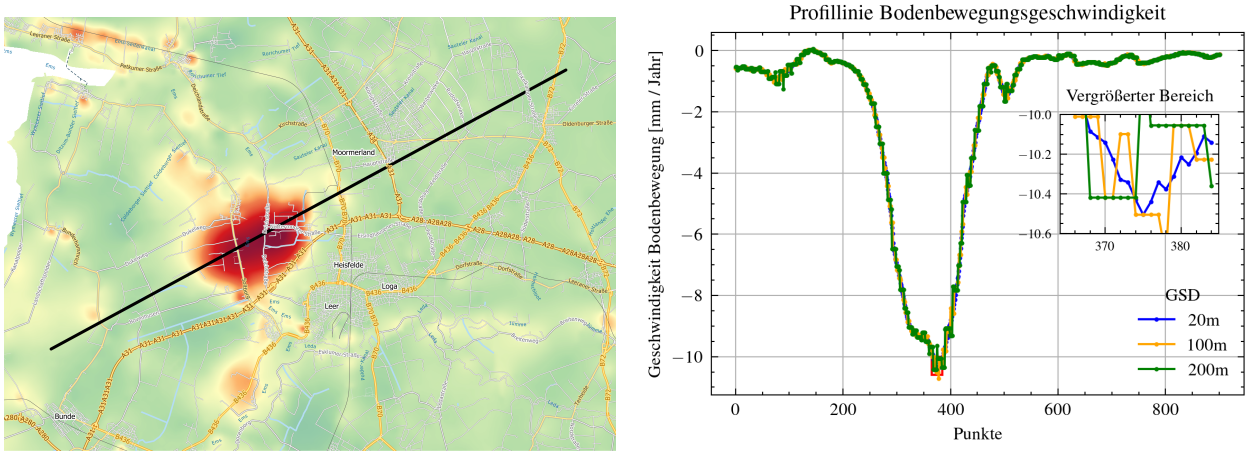
Das rasterbasierte Bodenbewegungsmodell dient als primäre Datenquelle dieser Arbeit. Innerhalb des folgenden Kapitels wird untersucht, wie die Bodenbewegungsinformationen basierend auf dem rasterbasierten Modell effizient und präzise abgefragt werden können. Dazu werden zunächst Funktionalitäten zur Abfrage der Bodenbewegungen auf Grundlage der gestellten Anforderungen identifiziert, die später innerhalb der Cloud-nativen Architektur umgesetzt werden. Neben der architektonischen Umsetzungsmöglichkeit erfolgt zusätzlich eine Untersuchung des rasterbasierten Bodenbewegungsmodell in Kombination mit den ermittelten Funktionalitäten. Dabei wird ermittelt, welche Auflösung hinsichtlich Genauigkeit, Effizienz und Speicherbedarf am besten für die Datenabfrage geeignet ist, was wiederum Auswirkungen auf die Architektur und Umsetzung des Funktionalitäten hat. Zusätzlich wird in Kapitel 5.3 die Laufzeit der Funktionalitäten in Verbindung mit Cloud-nativen Geodatenformaten untersucht.

5.2.1 Funktionalitäten

Innerhalb der Anforderungsanalyse wurden drei funktionale Anforderungen ermittelt, welche die Abfrage des rasterbasierten Bodenbewegungsmodells benötigen: Die Anforderung *F1* verlangt, dass die Anwendung in der Lage ist, Bodenbewegungsstatistiken für benutzerdefinierte Gebiete zu berechnen. Die Anforderung *F3* fordert die Fähigkeit, Bodenbewegungsprofile zu erstellen. Die Anforderung *F4* fordert die Verfügbarkeit von Bodenbewegungsstatistiken für Verdachtsgebiete. Basierend auf den Anforderungen lassen sich zwei konkrete Funktionalitäten ableiten: die Erstellung von Bodenbewegungsprofilen (Anforderung *F3*) und die Berechnung von Rasterstatistiken (Anforderung *F1* und *F4*). Dabei ist festzustellen, dass die Anforderungen *F1* und *F4* dieselbe Funktionalität benötigen,

nämlich die Berechnung von Bodenbewegungsstatistiken für spezifische Gebiete. Aus diesem Grund können diese Anforderungen zusammengefasst und durch eine einheitliche Funktionalität abgedeckt werden, wodurch die Komplexität der Implementierung reduziert werden kann.

Bei der Erzeugung von Profillinien auf Basis des rasterbasierten Bodenbewegungsmodells wird eine festgelegte Anzahl von Punkten zwischen einem Start- und Endpunkt berechnet. In dieser Arbeit wird eine Punktanzahl von 900 zwischen dem Anfangs- und Endpunkt verwendet, basierend auf der Anzahl der Punkte, die im Prototyp eingesetzt wurden. Diese Punkte sind gleichmäßig entlang der Linie verteilt und dienen als Abfragepunkte für die jeweiligen Bodenbewegungsinformationen. Für jeden Punkt wird die Bodenbewegungsinformation aus der Rasterzelle entnommen, in der sich der Punkt befindet. Durch das Plotten der Bodenbewegungsinformationen entlang der jeweiligen Position entsteht schließlich ein Bodenbewegungsprofil, welches in Abbildung 14b zu sehen ist. Die Abbildung zeigt ein beispielhaftes Profil (schwarze Linie) im Bodenbewegungsgebiet Jemgum/Nüttermoor.



(a) *Profillinie im Bereich Jemgum/Nüttermoor* **(b)** *Bodenbewegungen entlang Profillinie*

Abbildung 14: *Ergebnis eines Bodenbewegungsprofils mit Nächster Nachbar*

Das Verfahren, bei dem der Pixelwert des nächstgelegenen Punktes übernommen wird, wird als *Nächster Nachbar* bezeichnet. Dieses Verfahren führt dazu, dass der nächstgelegene Rasterwert als Repräsentant für den tatsächlichen Punktwert herangezogen wird, ohne eine Interpolation zwischen benachbarten Rasterzellen vorzunehmen. Diese Vorgehensweise kann zu Treppeneffekten führen, die nicht nur von der Realität abweichen, sondern auch die Genauigkeit der Abfragen beeinträchtigen. Eine Grafik zur Entstehung des Treppeneffekts ist in Anhang B1 zu finden. Dabei zeigt sich der Treppeneffekt insbesondere im vergrößerten Bereich der Abbildung 14b. Diese Ungenauigkeit steht im direkten Widerspruch mit der Anforderung *Q3*, welche eine hohe Genauigkeit der Bodenbewegungsdaten fordert. Um einen kontinuierlichen Verlauf bei der Profillinienbildung zu gewährleisten, ist eine Subpixelinterpolation für die Bildung von Profilen unumgänglich. Da verschiedene Interpolationsverfahren für die Subpixelinterpolation infrage kommen und die Theorie wichtig für das Verständnis der späteren Umsetzung ist, wird in Kapitel 5.2.2 zunächst die Theorie hinter der Subpixelinterpolation erläutert und gängige Interpolationsmethoden vorgestellt. Anschließend erfolgt in Kapitel 5.2.3 eine Analyse der Subpixelinterpolation im Kontext der Profillinienbildung. Innerhalb der Untersuchung findet zusätzlich die Betrachtung der unterschiedlichen Auflösungen des rasterbasierten Bodenbewegungsmodells statt, wodurch die beste Kombination aus Subpixelinterpolation und Auflösung ermittelt wird.

Neben der Subpixelinterpolation zeigen die Anforderungen *F1* und *F4* die Notwendigkeit der Berechnung von Rasterstatistiken für die Bodenbewegungen. Bei der Rasterstatistik werden basierend auf einem Raster Statistiken für definierte Bereiche, wie z.B. Polygone berechnet. Innerhalb des Prototyps wurde die Rasterstatistik lediglich in der manuellen Vorverarbeitung für die Verdachtsgebiete verwendet. Angesichts der Erfordernis, sowohl für vordefinierte als auch benutzerdefinierte Polygone Rasterstatistiken zu berechnen, ist eine generische Implementierung dieser Funktionalität notwendig. Dadurch kann diese Rasterstatistik zur automatisierten Datenvorverarbeitung und zur benutzerdefinierten Abfrage herangezogen werden. Ähnlich wie bei der Subpixelinterpolation ist eine Analyse der Auflösung des rasterbasierten Bodenbewegungsmodells auch hier sinnvoll. Diese Analyse findet in Kapitel 5.2.4 statt.

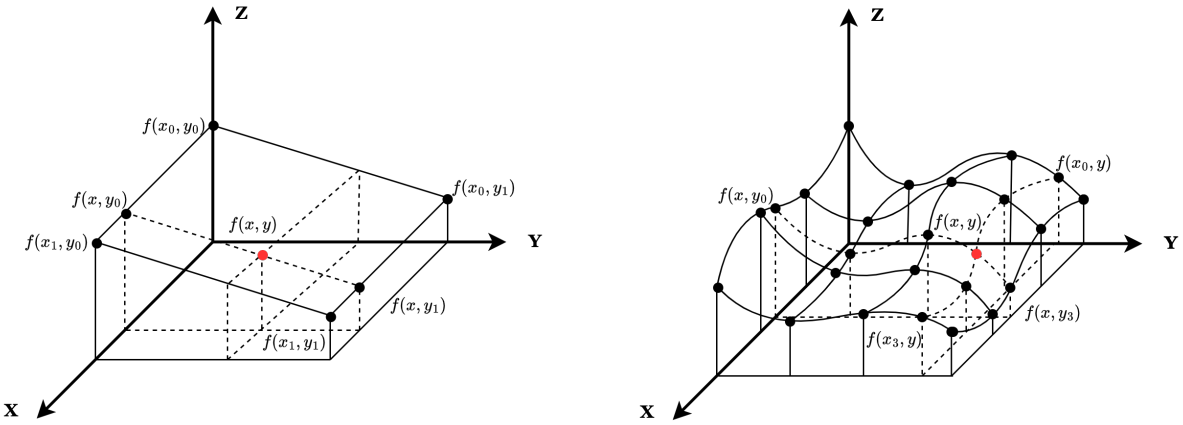
Für die Umsetzung beider Funktionalitäten bieten sich Microservices an, die im Kapitel 2.3.1 beschrieben wurden. Die Funktionalitäten besitzen damit die grundsätzlichen Vorteile von Microservices, wie eine verbesserte Skalierbarkeit und Wartbarkeit. Darüber hinaus haben Microservices aufgrund ihrer generischen Schnittstellen den Vorteil, dass sie leicht in andere Anwendungen integriert werden können oder für unterschiedliche Anwendungsbereiche, wie der Datenvorverarbeitung und der benutzerdefinierten Abfrage, genutzt werden können. Da sowohl die Subpixelinterpolation als auch die Berechnung von Rasterstatistiken viele Anwendungsgebiete in der Geoinformatik haben, können diese leicht in anderen Kontexten, wie z.B. der Abfrage von Höheninformationen aus einem Digitalen Oberflächenmodell (DOM), verwendet werden.

5.2.2 Einschub Subpixelinterpolation

Für ein tieferes Verständnis der Subpixelinterpolation sowie der unterschiedlichen Interpolationsmethoden werden im Folgenden die theoretischen Grundlagen und gängigen Verfahren vorgestellt. Die Interpolation ist eine wichtige Technik in der Mathematik und der Bildverarbeitung. Sie wird verwendet, um Werte an Positionen zu schätzen, die zwischen bekannten Datenpunkten liegen. Dazu existieren unterschiedliche Interpolationsmethoden, die sich stark in ihrer Genauigkeit aber auch Effizienz unterscheiden. Diese Interpolationsmethoden werden bei der Subpixelinterpolation verwendet, um Werte innerhalb eines Pixels auf Basis der benachbarten Pixel zu schätzen. Im Folgenden werden gängige Interpolationsmethoden vorgestellt und deren Anwendung im Kontext der Subpixelinterpolation erläutert. Die unterschiedlichen Ergebnisse der Interpolationsverfahren für eine beispielhafte mathematische Funktion sind in Abbildung 16 veranschaulicht.

Die einfachste Form der Interpolation ist die nächste Nachbar-Interpolation (engl. Nearest Neighbor). Dabei wird der Wert eines unbekannten Punktes als der Wert der nächstgelegenen Rasterzelle angenommen. Aufgrund dessen, dass lediglich eine Rasterzelle abgefragt werden muss, ist diese Methode sehr schnell. Allerdings führt diese zu blockartigen Artefakten und ist für viele Anwendungen nicht ausreichend genau. Ein Beispiel für ein Ergebnis der Nächster Nachbar-Interpolation ist in Abbildung 16a bzw. im Kontext der Profillinien in Abbildung 14b dargestellt.

Die bilineare Interpolation ist eine Erweiterung der linearen Interpolation in zwei Dimensionen. Sie basiert auf der Interpolation in zwei Richtungen. Die Funktionsweise ist in Abbildung 15a dargestellt. Gegeben sind vier benachbarte Punkte in einem Raster, $f(x_0, y_0)$, $f(x_0, y_1)$, $f(x_1, y_0)$ und $f(x_1, y_1)$. Die bilineare Interpolation schätzt den Wert $f(x, y)$ an einem Punkt (x, y) innerhalb des Rasters. Dabei wird zunächst eine lineare Interpolation in der x-Richtung durchgeführt, um die Werte an den Punkten (x, y_0) und (x, y_1) zu bestimmen. Anschließend wird die lineare Interpolation in der y-Richtung angewendet, um den endgültigen interpolierten Wert $f(x, y)$ zu berechnen (Lv et al.,



(a) *Bilineare Interpolation* **(b)** *Bikubische Interpolation*

Abbildung 15: *Funktionsweise der Subpixelinterpolation*

2009). Die Formel zur Berechnung ist in Formel 2 zu sehen.

$$f(x,y) = \frac{(x_1 - x)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} f(x_0, y_0) + \frac{(x - x_0)(y_1 - y)}{(x_1 - x_0)(y_1 - y_0)} f(x_1, y_0) + \frac{(x_1 - x)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} f(x_0, y_1) + \frac{(x - x_0)(y - y_0)}{(x_1 - x_0)(y_1 - y_0)} f(x_1, y_1) \tag{2}$$

Die vereinfachte Matrixschreibweise ist in Formel 3 dargestellt.

$$f(x,y) = \frac{1}{(x_1 - x_0)(y_1 - y_0)} \begin{bmatrix} x_1 - x & x - x_0 \end{bmatrix} \begin{bmatrix} f(x_0, y_0) & f(x_0, y_1) \\ f(x_1, y_0) & f(x_1, y_1) \end{bmatrix} \begin{bmatrix} y_1 - y \\ y - y_0 \end{bmatrix} \tag{3}$$

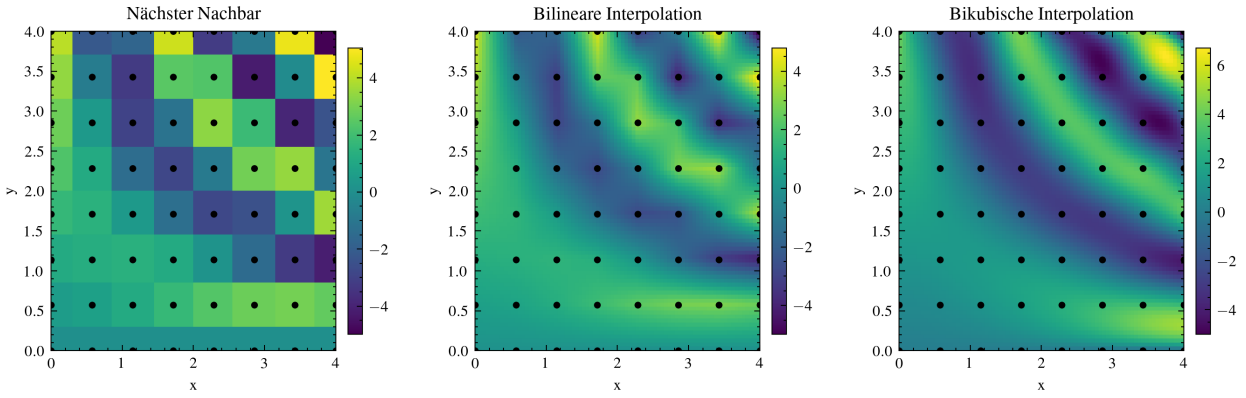
Das Ergebnis der bilinearen Interpolation ist in Abbildung 16b zu sehen. Zu erkennen ist, dass die interpolierten Werte eine glatte Übergangsfläche zwischen den diskreten Datenpunkten bilden und im Gegensatz zur nächstgelegenen Nachbarschaft keine abrupten, blockartigen Übergänge aufweisen.

Die bikubische Interpolation ist eine fortgeschrittene Methode zur Schätzung von Werten. Diese Methode verwendet mindestens 16 benachbarte Punkte, die in einer 4×4-Matrix angeordnet sind, um den interpolierten Wert zu berechnen. Eine vereinfachte Darstellung der allgemeinen Funktionsweise ist in Abbildung 15b zu sehen. Dabei werden kubische Polynomfunktionen basierend auf den Datenpunkten gebildet. Die Berechnungsformel für den Wert am Punkt (x, y), der von einem 4×4-Gitter umgeben ist, ist in Formel 4 dargestellt.

$$f(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j \tag{4}$$

Dabei sind a_{ij} die Koeffizienten, die aus den bekannten Werten und deren Ableitungen berechnet werden. Die bikubische Interpolation berücksichtigt sowohl die Werte als auch die ersten und zweiten Ableitungen der Funktion an den bekannten Punkten (Gao & Gruev, 2011). Dadurch können glattere Übergänge erzielt werden, was insbesondere bei der Bildskalierung zu weniger sichtbaren Artefakten führt. Um die Koeffizienten a_{ij} zu berechnen, wird zunächst ein lineares

Gleichungssystem gelöst. In diesem Prozess wird auch ein Gewichtungs-Kernel angewendet, der festlegt, wie stark die benachbarten Pixel in die Berechnung der Koeffizienten eingehen (Keys, 1981). Da die verschiedenen Gewichtungsfunktionen (Kernels) in dieser Arbeit nicht weiter behandelt werden, sei an dieser Stelle auf die detaillierte Darstellung in Keys, 1981 verwiesen. Nachdem die Koeffizienten a_{ij} unter Berücksichtigung der Gewichtungsfunktion bestimmt wurden, wird die obige Formel verwendet, um den interpolierten Wert $f(x, y)$ zu berechnen. Das Ergebnis der bikubischen Interpolation ist in Abbildung 16c zu sehen. Es zeigt sich, dass die Oberfläche gegenüber der bilinearen Interpolation eine noch höhere Glätte aufweist, insbesondere in der Nähe der Datenpunkte.



(a) *Nächster Nachbar* **(b)** *Bilineare Interpolation* **(c)** *Bikubische Interpolation*

Abbildung 16: *Ergebnisse unterschiedlicher Interpolationsmethoden*

Die Wahl der Interpolationsmethode hängt von den Anforderungen an die Glätte und Genauigkeit der Ergebnisse ab. Während die bilineare Interpolation aufgrund der kleineren Nachbarschaft einfacher und schneller ist, liefert die bikubische Interpolation qualitativ hochwertigere Ergebnisse. Eine Evaluierung der besten Methode im Kontext der Bodenbewegungen mit der passenden Auflösung erfolgt im folgenden Kapitel.

5.2.3 Vergleich Ergebnisse Subpixelinterpolation

Um zu ermitteln, welche Kombination aus Auflösung und Subpixelinterpolation die optimale Grundlage zur Bildung der Bodenbewegungsprofile darstellt, wird zunächst festgelegt, welche Auflösungen des rasterbasierten Bodenbewegungsmodells betrachtet werden. Konkret wird im Folgenden der Datensatz mit der vertikalen Geschwindigkeit betrachtet, die entsprechenden Grafiken für die horizontalen Bodenbewegungen sind im Anhang B4 zu finden. Als obere Grenze kann eine Auflösung von 200 m × 200 m (200 m) gewählt werden, da dies der Ausgangsgröße des BOB.NI-Modells entspricht. Zusätzlich wird eine mittlere Auflösung von 100 m × 100 m (100 m) gewählt, um eine höhere Genauigkeit zu gewährleisten, ohne den Speicherbedarf stark zu erhöhen. Die höhere Auflösung von 20 m × 20 m (20 m) bietet die Möglichkeit, besonders feine Details zu erfassen, die für spezifische Analysen entscheidend sein können. Das rasterbasierte Bodenbewegungsmodell mit einer Auflösung von 20 m bildete die Datengrundlage für den Prototyp. Es ist zu beachten, dass die Auflösungen von 20 m und 100 m durch Interpolation des ursprünglichen rasterbasierten Bodenbewegungsmodells mit einer Auflösung von 200 m berechnet wurden. Für die Bildung der Profile wurden die in Kapitel 5.2.2 beschriebenen bilinearen und bikubischen Interpolationen verwendet. Die konkrete Umsetzung dieser Funktionalitäten wird jedoch erst in Kapitel 7.2.2 detailliert erläutert. Im Rahmen dieses Kapitels erfolgt die Evaluierung primär basierend auf der Genauigkeit und der Größe der Daten.

Zusätzlich wird der Aufwand für die Abfragen betrachtet, wobei eine detaillierte Analyse der Laufzeit im Kapitel 5.3 erfolgt, in dem die Kombination mit Cloud-optimierten Datenformaten untersucht wird.

Die Tabelle 2 zeigt die Dateigrößen der jeweiligen Auflösungen des rasterbasierten Bodenbewegungsmodells im normalen TIFF-Format. Es ist deutlich erkennbar, dass die Dateigröße der 20 m Auflösung mit Abstand die größte ist und die Dateigröße der 100 m und 200 m Auflösung erheblich übersteigt. Da das rasterbasierte Bodenbewegungsmodell aus vier separaten Dateien besteht, die alle die gleiche Dateigröße haben, ist die Gesamtgröße viermal so groß wie die Größe der einzelnen Dateien.

Tabelle 2: *Dateigröße des rasterbasierten Bodenbewegungsmodells bei unterschiedlichen Auflösungen*

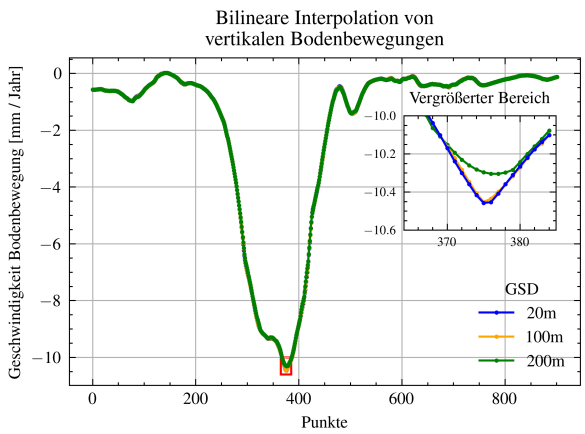
20 m Rasterdatei	**100 m Rasterdatei**	**200 m Rasterdatei**
907,6 MB	36,3 MB	9,1 MB

Basierend auf den unterschiedlichen Auflösungen werden im Folgenden die einzelnen Profile mit der bilinearen und der bikubischen Subpixelinterpolation betrachtet. Zusätzlich werden die Differenzen, welche die einzelnen Auflösungen zueinander aufweisen, dargestellt. Als repräsentatives Profil wird das Profil aus Abbildung 14a verwendet. Da es keine festen Vorgaben hinsichtlich der Genauigkeit gibt, muss ein Kompromiss zwischen Effizienz und Genauigkeit gefunden werden. Dies erfordert eine Abwägung der Gewährleistung ausreichender Präzision, ohne dabei die Ressourcen unnötig zu belasten.

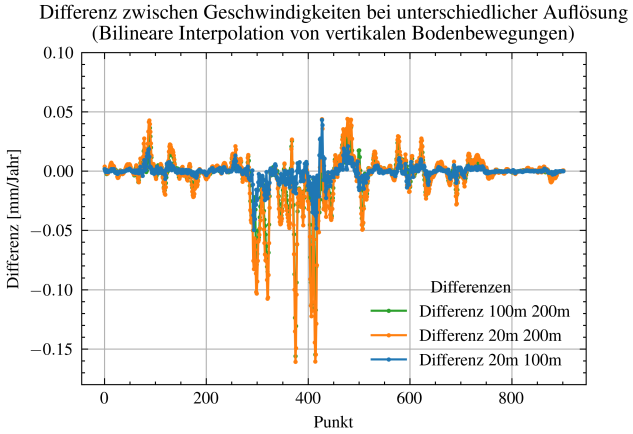
Die Abbildung 17 visualisiert die berechneten Profile mittels bilinearer Interpolation (Abbildung 17a und 17b) und bikubischer Interpolation (Abbildung 17c und 17d) für die unterschiedlichen Auflösungen. Zusätzlich sind die entsprechenden Differenzen der Profile dargestellt. Da keine Ground-Truth-Daten zur Verfügung stehen, wird sowohl bei der bilinearen als auch bei der bikubischen Interpolation die 20 m Auflösung als Referenz verwendet. Dabei unterscheiden sich die Profile bei einer Auflösung von 20 m bei beiden Verfahren nur minimal, was in Anhang B2 abgebildet ist. Zusätzlich werden für beide Verfahren die mittlere Abweichung ($\overline{x}$), der Betrag der maximalen Abweichung ($|\max|$) sowie die Standardabweichung (σ) berechnet, um die Ergebnisse zur quantifizieren.

Die Ergebnisse der bilinearen Interpolation für die verschiedenen Auflösungen in den Abbildungen 17a und 17b zeigen, dass die Profile größtenteils nur geringe Abweichungen voneinander aufweisen. Größere Unterschiede treten jedoch insbesondere an den lokalen Extremen auf, vor allem zwischen der gröberen Auflösung von 200 m und den feineren Auflösungen von 20 m und 100 m. Zwischen den Auflösungen von 20 m und 100 m sind die Unterschiede dagegen minimal. Diese Beobachtungen werden durch die statistischen Kennzahlen in Tabelle 3 weiter verdeutlicht. Zu sehen ist, dass die 200 m Auflösung die größten Abweichungen zum 20 m Modell aufweist, wie durch die maximale Abweichung von 0,161 mm/Jahr und der mittleren Abweichung von 0,006 mm/Jahr belegt wird. Gleichzeitig zeigt die Standardabweichung von 0,025 mm/Jahr, dass die Streuung der Abweichungen bei dieser Kombination ebenfalls am höchsten ist. Die Abweichungen und Streuung zwischen 20 m und 100 m sind dagegen deutlich geringer.

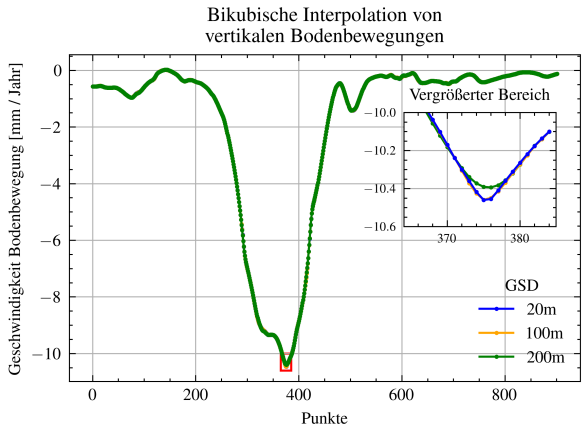
Zwischen der 20 m und der 100 m Auflösung bestehen bei der Nutzung der bilinearen Interpolation kaum Unterschiede. Da die Speichergröße durch die 100 m Auflösung jedoch um rund 97 % reduziert werden kann, sollte diese als optimaler Kompromiss zwischen Genauigkeit und Speicherbedarf für die bilineare Interpolation bevorzugt verwendet werden.



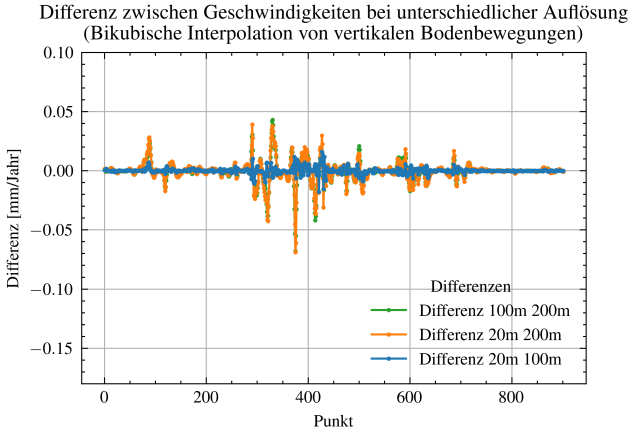
(a) *Beispielprofil bilineare Interpolation*



(b) *Differenzen bilineare Interpolation*



(c) *Beispielprofil bikubische Interpolation*



(d) *Differenzen bikubische Interpolation*

Abbildung 17: *Bodenbewegungsprofile mit bilinearer und bikubischer Interpolation bei unterschiedlichen Auflösungen und deren Abweichungen*

Als alternativer Ansatz wird im Folgenden die bikubische Subpixelinterpolation im Kontext der Bodenbewegungen näher betrachtet. Die Ergebnisse der bikubischen Interpolation, einschließlich der zugehörigen Abweichungen, sind in den Abbildungen 17c und Abbildung 17d dargestellt. Aufgrund der größeren Nachbarschaft liefert die bikubische Interpolation in der Regel präzisere Ergebnisse als die bilineare Interpolation, insbesondere bei höheren Auflösungen. Dies zeigt sich in Abbildung 17c, in der das Profil der bikubischen Interpolation näher an den Profilen mit höherer Auflösung liegt als bei der bilinearen Interpolation (siehe Abbildung 17a). Die statistischen Kennzahlen in Tabelle 4 unterstreichen diese Ergebnisse. So liegt die mittlere Abweichung zwischen der 20 m und der 100 m Auflösung nahezu bei Null (-0,00005 mm/Jahr), ebenso wie die Standardabweichung (0,002 mm/Jahr), was auf eine hohe Übereinstimmung hindeutet. Auch die maximale Abweichung von 0,018 mm/Jahr ist äußerst gering. Zwischen der 20 m und der 200 m Auflösung bleibt die mittlere Abweichung mit -0,00026 mm/Jahr weiterhin klein, jedoch sind sowohl die maximale Abweichung (0,069 mm/Jahr) als auch die Standardabweichung (0,009 mm/Jahr) höher. Diese Werte verdeutlichen, dass die bikubische Interpolation auch bei niedrigeren Auflösungen weiterhin präzise arbeitet, jedoch nicht die Genauigkeit der 100 m Auflösung erreicht. Ein Vergleich zwischen

Tabelle 3: *Statistische Werte der Abweichung bei einer bilinearen Interpolation*

	$\overline{x}$	**|max|**	σ
20 m - 100 m	-0,001 mm/Jahr	0,050 mm/Jahr	0,007 mm/Jahr
20 m - 200 m	-0,006 mm/Jahr	0,161 mm/Jahr	0,025 mm/Jahr
100 m - 200 m	-0,005 mm/Jahr	0,156 mm/Jahr	0,020 mm/Jahr

der 100 m und 200 m Auflösung bestätigt diesen Trend. Die mittlere Abweichung liegt bei -0,00027 mm/Jahr und die maximale Abweichung bei 0,068 mm/Jahr, was ebenfalls auf eine geringere Präzision im Vergleich zur 100 m Auflösung hindeutet.

Tabelle 4: *Statistische Werte der Abweichung bei einer bikubischen Interpolation*

	$\overline{x}$	**|max|**	σ
20 m - 100 m	-0,00005 mm/Jahr	0,018 mm/Jahr	0,002 mm/Jahr
20 m - 200 m	-0,00026 mm/Jahr	0,069 mm/Jahr	0,009 mm/Jahr
100 m - 200 m	-0,00027 mm/Jahr	0,068 mm/Jahr	0,008 mm/Jahr

Grundsätzlich zeigt die Analyse, dass der Datensatz mit einer Auflösung von 100 m sowohl bei der bilinearen als auch bei der bikubischen Interpolation hinsichtlich Speicherbedarf und Genauigkeit als die effizienteste Option angesehen werden kann. Die bilineare Interpolation erweist sich als effizient, da insgesamt weniger Pixel abgefragt werden müssen, was den Rechenaufwand reduziert. Allerdings führt die kleinere Nachbarschaft zu teilweise scharfen Artefakten in den Profilen, die nicht der Realität entsprechen. Ein Beispiel hierfür ist im Anhang B3 dargestellt. Die bikubische Interpolation hingegen glättet solche Artefakte durch die Verwendung einer größeren Nachbarschaft und bietet damit in vielen Fällen eine höhere Genauigkeit. Aufgrund der größeren Nachbarschaft ist sie jedoch mit einem höheren Rechenaufwand verbunden. Das bedeutet, dass beide Interpolationsmethoden im Zusammenhang mit Bodenbewegungen ihre Vor- und Nachteile haben. Aus diesem Grund sollen sowohl die bilineare als auch die bikubische Interpolation innerhalb des Microservices implementiert werden, um je nach Anwendungsfall flexibel auf die Anforderungen an Effizienz und Genauigkeit reagieren zu können. Im nächsten Kapitel wird zusätzlich die Laufzeit der beiden Verfahren in Kombination mit der Datenverwaltung analysiert, insbesondere im Hinblick auf Cloud-optimierte Datenformate.

Es ist zu beachten, dass im Rahmen dieses Kapitels nur ein repräsentatives Profil für die vertikalen Bodenbewegungen untersucht wurde und andere Profile zu abweichenden Ergebnissen führen können. Jedoch haben weitere Untersuchungen gezeigt, dass ähnliche Ergebnisse für die verschiedene Profile und horizontalen Bodenbewegungen zu erwarten sind. Darüber hinaus hängt die Genauigkeit der Subpixelinterpolation auch von der Verteilung der Punkte entlang des Profils ab. Die bisherige Methode der Erzeugung von 900 Punkten stellt keinen optimalen Ansatz dar, da unabhängig von der Länge der Strecke 900 Punkte erzeugt werden. Dadurch entsteht bei einer kurzen Strecke eine deutlich höhere Punktdichte, was zu einer unnötigen Erhöhung des Rechenaufwands führt, während bei längeren Strecken die Punktdichte unzureichend sein kann, um Details der Bodenbewegungen akkurat zu erfassen. Zukünftig sollte eine Metrik entwickelt werden, welche die Verteilung der Punkte entlang der Strecke besser berücksichtigt. Eine mögliche Verbesserung wäre die Implementierung eines auf dem Abtasttheorem basierenden Ansatzes, welcher die Länge der Strecke berücksichtigt, um eine gleichmäßige Punktdichte bei unterschiedlichen Strecken zu gewährleisten. Die Berechnung könnte auf der Seite des Backends durchgeführt werden, wobei lediglich die Übergabe der Parameter Start- und Endpunkt erforderlich wäre. Im Rahmen dieser Arbeit wird jedoch keine detaillierte

Diskussion oder Implementierung dieser Methode vorgenommen.

5.2.4 Vergleich Ergebnisse Rasterstatistik

Im folgenden Kapitel wird die Funktionalität der Rasterstatistik mit den verschiedenen Auflösungen des rasterbasierten Bodenbewegungsmodells untersucht, die zur Bildung von Bodenbewegungsstatistiken innerhalb von Polygonen verwendet wird. Die detaillierte Beschreibung der Implementierung der für die Analyse verwendeten Rasterstatistik erfolgt in Kapitel 7.2.3. Genau wie bei der Untersuchung der Subpixelinterpolation wird das rasterbasierte Bodenbewegungsmodell mit den Auflösungen 20 m, 100 m und 200 m betrachtet. Als Polygone der Eingangsdaten werden die Verdachtsgebiete verwendet, welche 84 einzelne Polygone darstellen. Als statistischer Wert der Rasterstatistik wird lediglich der Mittelwert betrachtet.

Die Abweichungen des Mittelwerts der einzelnen Polygone ist in Abbildung 18 dargestellt. Zu sehen ist, ähnlich wie bei der Subpixelinterpolation, dass die Abweichungen zwischen der 20 m und 100 m Auflösung sehr gering sind, während die Differenz zwischen 20 m und 200 m deutlich höher ist.

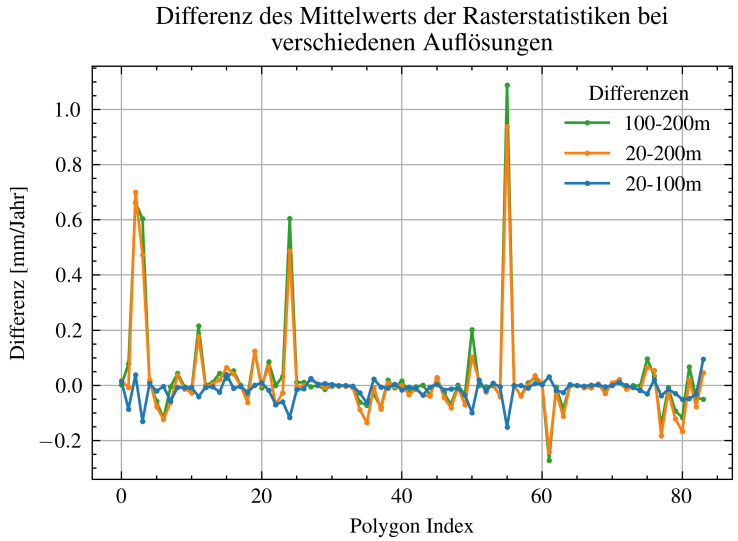


Abbildung 18: *Abweichung des Mittelwerts der Rasterstatistik in Abhängigkeit der verschiedenen Auflösungen*

Die statistischen Kennzahlen in Tabelle 5 zeigen, dass die Abweichungen zwischen der 20 m und 100 m Auflösung mit einer mittleren Abweichung von -0,014 mm/Jahr und einer maximalen Abweichung von 0,151 mm/Jahr gering sind. Zusätzlich weisen die Abweichungen eine geringe Standardabweichung auf, was auf eine geringe Streuung der Abweichungen schließen lässt. Im Vergleich dazu sind die Abweichungen zwischen der 200 m und der 20 m Auflösung deutlich stärker und gut in Abbildung 18 zu erkennen. Zwar ist die mittlere Abweichung mit 0,016 mm/Jahr im Vergleich zur 100 m Auflösung nur geringfügig höher, jedoch deuten die größere maximale Abweichung von 0,938 mm/Jahr und die höhere Standardabweichung darauf hin, dass es Ausreißer gibt, welche innerhalb der Abbildung 18 auch zu erkennen sind.

In Anbetracht der deutlich geringeren Dateigröße (siehe Tabelle 2) sowie der geringen Abweichung zur hohen Auflösung von 20 m bietet das rasterbasierte Bodenbewegungsmodell mit einer Auflösung

Tabelle 5: *Statistische Werte der Abweichungen der Rasterstatistik für die mittlere Bodenbewegung innerhalb der Verdachtsgebiete*

	$\overline{x}$	**|max|**	σ
20 m - 100 m	-0,014 mm/Jahr	0,151 mm/Jahr	0,035 mm/Jahr
20 m - 200 m	0,016 mm/Jahr	0,938 mm/Jahr	0,161 mm/Jahr
100 m - 200 m	0,030 mm/Jahr	1,089 mm/Jahr	0,177 mm/Jahr

von 100 m auch für die Rasterstatistik einen optimalen Kompromiss. Dies gilt sowohl hinsichtlich der Genauigkeit als auch des Speicherbedarfs. Zusätzlich wird die Abfrage der Pixel reduziert, sodass die Anzahl der abgefragten Zellen um das 25-fache reduziert wird. Dies verringert den Aufwand der Abfrage erheblich. Eine detaillierte Analyse der Laufzeiten findet sich in Kapitel 5.3.

5.2.5 Zwischenfazit Datenabfrage

Die Analyse der Datenabfrage hat gezeigt, dass die Subpixelinterpolation und die Rasterstatistik wesentliche Funktionalitäten für die Erfüllung der Anforderungen bei der Abfrage der rasterbasierten Bodenbewegungsdaten darstellen. Es wird empfohlen, die jeweiligen Funktionalitäten in Microservices zu implementieren und gemäß der Anforderung *Q8* über eine REST-API zugänglich zu machen. Diese können durch eine entsprechende Cloud-native Architektur einfach eingebunden werden. Zusätzlich wird bei der Subpixelinterpolation empfohlen, sowohl die bilineare als auch die bikubische Interpolation aufgrund ihrer spezifischen Vorteile bei unterschiedlichen Genauigkeiten beizubehalten.

Darüber hinaus wurde in diesem Kapitel gezeigt, dass sich eine Auflösung von 100 m für das rasterbasierte Bodenbewegungsmodell als ideale Datengrundlage für die erwiesen hat. Durch die Nutzung der 100 m Auflösung für die Bildung von Bodenbewegungsprofilen sowie für die Berechnung von Rasterstatistiken kann der Speicherbedarf im Vergleich zum Prototyp um 97 % reduziert werden, ohne wesentliche Genauigkeitseinbußen hinnehmen zu müssen. Zudem werden im nachfolgenden Kapitel auch die Laufzeiten des Modells betrachtet, um eine umfassende Bewertung der Effizienz sicherzustellen. Dadurch bietet das Modell eine effiziente und ressourcenschonende Alternative, die den Anforderungen an Präzision und Speicherplatz optimal gerecht wird. Ein weiterer Handlungsbedarf wurde hinsichtlich der Punkte entlang der Profillinie identifiziert, jedoch wird dieser Aspekt im weiteren Verlauf der Arbeit nicht weiter vertieft.

5.3 Analyse der Eignung von Cloud-optimierten Geodatenformaten

Wie in Kapitel 5.1 beschrieben, wurde im Prototyp eine PostgreSQL-Datenbank mit der Erweiterung PostGIS als zentrale Speichereinheit verwendet. In dieser Datenbank werden das rasterbasierte Bodenbewegungsmodell sowie weitere Vektorlayer, wie die Verdachtsgebiete, gespeichert und für Abfragen und Visualisierungen bereitgestellt. Eine Migration dieser Datenbank in eine Cloud-Umgebung kann jedoch sehr teuer sein. Die Kosten für eine PostgreSQL-Datenbank-Service in der IBM Cloud betragen zum Zeitpunkt der Bearbeitung (06.08.2024) etwa 75 € pro Monat für 5 GB Speicher, 0,5 vCPU und 4 GB RAM. Dieser Preis basiert auf einer durchschnittlichen Nutzung, ohne Steuern (IBM, 2024d). Eine standardmäßige PostgreSQL-Datenbank ist zudem nicht Cloud-native, was bedeutet, dass sie in ihrer herkömmlichen Form nicht optimal auf die Vorteile einer Cloud-Umgebung, wie automatische Skalierbarkeit oder dynamisches Ressourcenmanagement, ausgelegt ist. Dieser Ansatz erfüllt daher nicht die Anforderung *Q2*, die kostengünstige, Cloud-native Technologien fordert.

Eine kostengünstige, Cloud-native Alternative zur Datenbank sind Cloud-optimierte Geodatenformate in Kombination mit einem COS. Wie in Kapitel 2.4 beschrieben, sind diese Formate für die Speicherung und effiziente Abfrage in einer Cloud-Umgebung entwickelt und bieten optimale Kompatibilität mit einem COS. Zudem bieten diese Formate eine bessere Datenkompression und schnellere Zugriffszeiten, was die Speicherkosten weiter senkt und die Cloud-Ressourcen effizienter nutzt. Im Gegensatz zur PostgreSQL-Datenbank, deren Betrieb in der IBM-Cloud kostenintensiv sein kann, belaufen sich die Kosten für 5 GB Speicherplatz in einem COS auf nur wenige Cent pro Monat.

In diesem Kapitel wird die Eignung des COG für das rasterbasierte Bodenbewegungsmodell und des FGB für Vektorlayer wie die Verdachtsgebiete untersucht, wobei der Fokus auf dem rasterbasierten Bodenbewegungsmodell liegt. Der Fokus der Untersuchung liegt auf den Laufzeiten der in Kapitel 5.2.1 identifizierten Funktionalitäten und dem Speicherbedarf der jeweiligen Formate. Die Umsetzung der verwendeten Funktionalitäten erfolgt in Kapitel 7.2.2 und 7.2.3. Ziel der Untersuchung ist es, festzustellen, ob die Cloud-optimierten Formate die Datenbank vollständig ersetzen können und eine effizientere Lösung für die Verwaltung und Verarbeitung der Bodenbewegungsdaten hinsichtlich der Anforderung *Q2* darstellen.

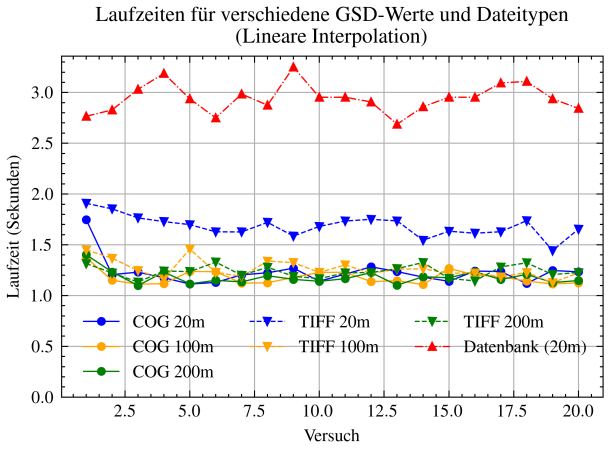
Für die Laufzeittests wurden spezifische Hardwarekonfigurationen eingesetzt, um die Performance der Cloud-optimierten Formate im Vergleich zu herkömmlichen GeoTIFFs zu bewerten. Eine Testumgebung mit 4 GB RAM und einer virtuelle CPU (vCPU) in der IBM-Cloud wurde für die Analyse des COG-Formats eingerichtet, wobei die entsprechenden Dateien in einem COS gespeichert wurden. Der Zugriff der Testumgebung auf die Daten erfolgte über einen direkten Endpunkt innerhalb der IBM-Cloud, um unnötige Latenzzeiten zu vermeiden. Anfragen wurden von einem lokalen PC über eine 100.000 Kbit/s-Leitung durchgeführt, was zusätzliche Latenzen verursachen kann, die jedoch zu realistischen Laufzeiten führen. Da eine direkte Implementierung der Datenbank in der Cloud nicht möglich war, wurden die Laufzeiten der Datenbanken auf der Hardware des Prototyps durchgeführt. Diese bestand aus einer Intel Xeon Gold CPU mit 2,8 GHz und 4 Kernen sowie 16 GB RAM. Die Laufzeittests für die Datenbank wurden direkt auf der VM durchgeführt, was den Datentransfer innerhalb des Netzwerks eliminierte. Dadurch sind die Laufzeitergebnisse nur eingeschränkt vergleichbar, da die Prototyp-Hardware und der Wegfall der Datenübertragung im lokalen Netzwerk eine deutlich höhere Performance ermöglichen.

5.3.1 Cloud-optimiertes GeoTIFF

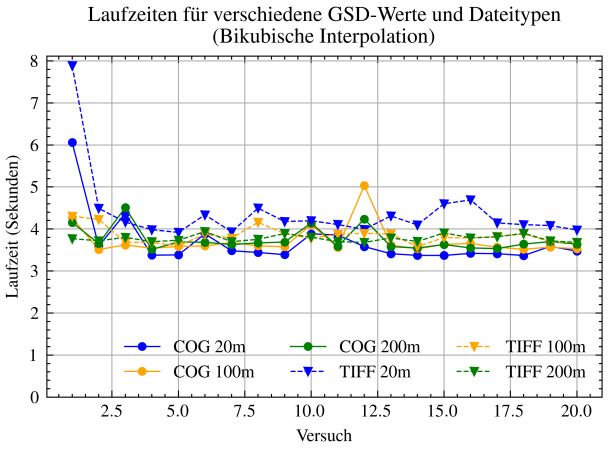
Im folgenden Kapitel wird untersucht, ob das COG-Format in Kombination mit einem COS für die Verwaltung und Verarbeitung des rasterbasierten Bodenbewegungsmodells geeignet ist. Im Rahmen der vorliegenden Untersuchung werden die Laufzeiten für die Erstellung von Bodenbewegungsprofilen sowie für die Erstellung von Rasterstatistiken analysiert, welche auf dem rasterbasierten Bodenbewegungsmodell basieren.

Für die Untersuchung werden die Profillinien und die Rasterstatistik auf Basis eines herkömmlichen GeoTIFFs, des COG und der Datenbank erstellt. Da die Datenbank nicht in der Cloud implementiert werden konnte und im Prototyp nur die bilineare Interpolation verfügbar ist, wird für die Datenbank lediglich die Laufzeit der linearen Interpolation analysiert. Jedes Experiment wurde 20-mal durchgeführt, um valide Ergebnisse zu gewährleisten und Ausreißer, die durch zufällige Schwankungen oder Latenzen in der Datenübertragung verursacht werden könnten, zu minimieren. Neben der Laufzeitanalyse wird zusätzlich der Speicherbedarf der einzelnen Formate untersucht, um zu bewerten, wie effizient die Daten in verschiedenen Formaten gespeichert werden können.

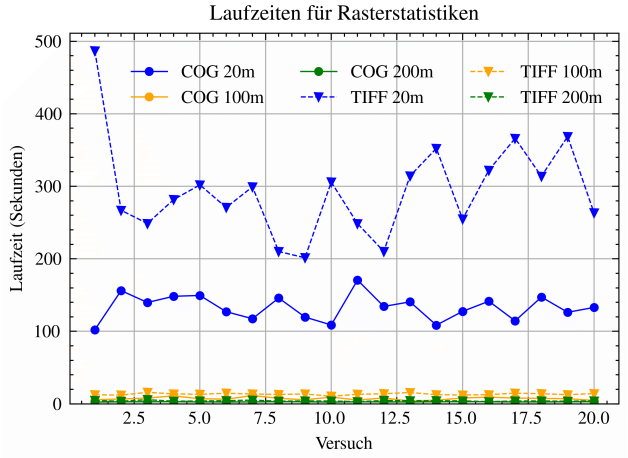
Abbildung 19 zeigt die Laufzeiten für die Erstellung eines Profils mit der bilineare Interpolation (Abbildung 19a) und die bikubische Interpolation (Abbildung 19b) sowie für die Erstellung des Rasterstatistik (Abbildung 19c). Zusätzlich sind die mittleren Laufzeiten in Tabelle 6 zusammengefasst.



(a) *Laufzeiten unterschiedlicher Formate bei bilinearer Interpolation*



(b) *Laufzeiten unterschiedlicher Formate bei bikubische Interpolation*



(c) *Laufzeiten unterschiedlicher Formate bei Rasterstatistik*

Abbildung 19: *Laufzeitvergleich der Funktionen für verschiedene Dateiformate des rasterbasierten Bodenbewegungsmodells*

Bei der bilinearen Interpolation (Abbildung 19a) sind die Laufzeiten für COG-Dateien im Allgemeinen kürzer als für TIFF-Dateien und Datenbankzugriffe. Die COG-Dateien zeigen unabhängig von der Auflösung konsistente Ergebnisse mit Laufzeiten von ca. 1,176 Sekunden bis 1,238 Sekunden. Im Vergleich dazu sind die Laufzeiten für TIFF-Dateien etwas höher, insbesondere für TIFF 20 m mit ca. 1,683 Sekunden. Die längsten Laufzeiten wurden für Datenbankzugriffe mit ca. 2,942 Sekunden gemessen, wobei hier aufgrund der Testumgebung keine Latenzen vorhanden sind und zusätzlich die Tests auf einer leistungsfähigeren Hardware durchgeführt wurden. Dies zeigt, dass die Verwaltung und Abfrage von rasterbasierten Bodenbewegungsdaten in der Datenbank erheblich weniger effizient ist als die direkte Nutzung der Dateien in Verbindung mit einem COS.

Tabelle 6: *Mittelwerte der Laufzeiten für bilineare und bikubische Interpolation sowie Rasterstatistik beim rasterbasierten Bodenbewegungsmodell*

	COG 20 m	**COG 100 m**	**COG 200 m**	**TIFF 20 m**	**TIFF 100 m**	**TIFF 200 m**	**Datenbank 20 m**
Bilinear	1,228 s	1,176 s	1,179 s	1,683 s	1,262 s	1,235 s	2,942 s
Bikubisch	3,685 s	3,714 s	3,748 s	4,384 s	3,843 s	3,773 s	/
Raster-statistik	139,178 s	6,708 s	2,769 s	317,187 s	13,462 s	4,292 s	/

Ein ähnliches Muster zeigt sich auch bei der bikubischen Interpolation (Abbildung 19b). Hierbei sind die Laufzeiten etwa dreimal höher, da bei der bikubischen Interpolation 16 Pixel berücksichtigt werden müssen, im Gegensatz zu nur 4 Pixeln bei der bilinearen Interpolation. Auch bei der bikubischen Interpolation schneiden die COG-Dateien ebenfalls besser ab als die TIFF-Dateien, mit Laufzeiten zwischen 3,685 Sekunden und 3,748 Sekunden, während die Laufzeiten für TIFF-Dateien zwischen 3,843 Sekunden und 4,383 Sekunden liegen. Für diese Interpolationsmethode sind aufgrund der fehlenden Umsetzung innerhalb des Prototyps keine Messwerte für Datenbankabfragen verfügbar. Insgesamt lässt sich festhalten, dass die COG sowohl bei der bilinearen als auch bei der bikubischen Interpolation durchweg geringfügig bessere Laufzeiten aufweisen. Dies könnte auf die Optimierung des COG für das Streaming zurückzuführen sein. Zudem lässt sich annehmen, dass sich die unterstütze Kachelung und Pyramidenbildung, wie in Kapitel 2.4.2 beschrieben, primär bei den für die Subpixelinterpolation verwendeten *HTTP-Range-Request* effizienter auswirkt. Auffällig ist, dass das COG mit einer Auflösung von 20 m trotz anfänglich längerer Laufzeiten, die durch die Aufwärmphase der Hardware oder anfängliches Caching und Dateimanagement verursacht werden können, am Ende die beste Laufzeit erreicht. Dies deutet darauf hin, dass die Vorteile des COG insbesondere bei größeren Dateien zum Tragen kommen.

Ein vergleichbares Bild ergibt sich auch bei der Berechnung der Rasterstatistik (Abbildung 19c). Auch hier schneidet das COG bei allen untersuchten Auflösungen am besten ab. Besonders auffällig ist der deutliche Vorteil des COG bei größeren Dateien wie dem 20m-Raster. Während das COG bei der 20 m-Auflösung eine Laufzeit von 139,178 Sekunden aufweist, benötigt es beim TIFF-Format mit 317,187 Sekunden mehr als doppelt so lange. Dies unterstreicht, dass das COG durch seine Optimierung für Streaming und die integrierte Kachelung bei rechenintensiven Abfragen wie der Rasterstatistik deutlich effizienter arbeitet.

Neben der besseren Performance der Abfrage in Kombination mit der Subpixelinterpolation zeigt Tabelle 7 zudem, dass die Cloud-optimierten Datenformate aufgrund ihrer optimierten Datenkompression eine geringere Speichergröße aufweisen. Dabei wird in der Tabelle die zusammengesetzte Speichergröße des gesamten rasterbasierten Bodenbewegungsmodells, also der vier einzelnen Dateien, dargestellt.

Insgesamt lässt sich festhalten, dass der Einsatz von COG in Kombination mit einem COS erhebliche Vorteile für die Verwaltung, Verarbeitung und Abfrage von rasterbasierten Bodenbewegungsdaten bietet. Ein wesentlicher Vorteil dieser Kombination ist die deutliche Kostenersparnis gegenüber dem Einsatz von PostgreSQL in der Cloud, da die hohen Betriebskosten einer Datenbanklösung vermieden werden können. Diese Kostenreduktion wird zusätzlich durch die kleineren Dateigrößen der COG-Dateien begünstigt, was zu weiteren, geringfügigen Einsparungen führt. Darüber hinaus sind

Tabelle 7: *Gesamtgröße des rasterbasierten Bodenbewegungsmodell bei unterschiedlichen Dateiformaten und Auflösungen*

	COG 20 m	**COG 100 m**	**COG 200 m**	**TIFF 20 m**	**TIFF 100 m**	**TIFF 200 m**	**Datenbank 20 m**
Dateigröße	2,6932 GB	0,105 GB	0,0173	2,8111 GB	0,1129 GB	0,0214 GB	/

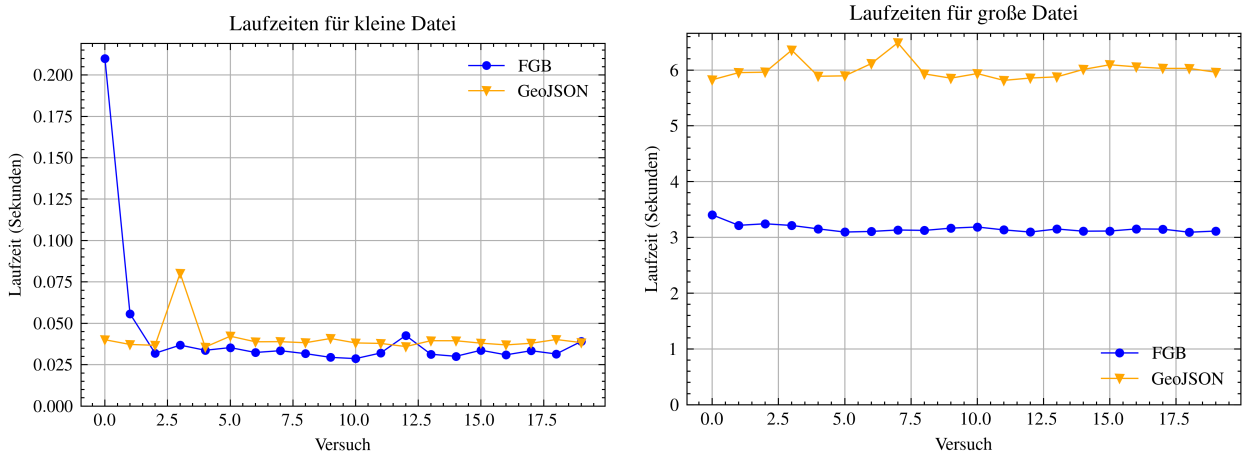
die Laufzeiten bei Verwendung von COS deutlich performanter als bei der im Prototyp verwendeten PostgreSQL-Datenbank und nicht Cloud-optimierten GeoTIFFs. Zusätzlich hat die Analyse der Laufzeiten gezeigt, dass die 100 m Auflösung, welche im Kapitel 5.2 als primäre Datenquelle identifiziert wurde, eine gute Wahl ist. Primär bei der Rasterstatistik führt die Auflösungen von 20 m zu extrem langen Laufzeiten, die für den Anwender nicht akzeptabel sind. Im Gegensatz dazu sind die Zeiten bei einer Auflösung von 100 m deutlich besser und bieten einen guten Kompromiss zwischen Genauigkeit und Effizienz.

5.3.2 FlatGeobuf

Neben den rasterbasierten Bodenbewegungsdaten existieren ergänzende Layer, wie bspw. die Verdachtsgebiete, welche Gebiete darstellen, in denen Bodenbewegungen erwartbar sind. Diese werden im Vektordatenformat bereitgestellt. Innerhalb des Prototyps wurden diese Verdachtsgebiete als Vector-Tiles berechnet. Aufgrund dessen, dass sich diese Gebiete regelmäßig aktualisieren und die Berechnung von Vector-Tiles sehr aufwendig ist, ist die Überlegung, diese mithilfe von FGB darzustellen. FGB wurde in Kapitel 2.4.1 vorgestellt und stellt ein Cloud-optimierten Datenformat für Vektordaten dar, welches sich insbesondere für Visualisierungen eignet. Obwohl FGB weniger gut für häufige Aktualisierungen geeignet ist, ist die Erzeugung einer neuen Datei deutlich einfacher und weniger aufwendig als die erneute Berechnung von Vector-Tiles. Im Folgenden werden die Zugriffszeiten, die erforderlich sind, um die Datei vollständig aus dem COS zu laden, sowie die Speichergröße betrachtet. Als Vergleich zum FGB wird das herkömmliche GeoJSON-Format verwendet, da es sich hierbei um ein gängiges Vektordatenformat handelt, welches häufig in Webanwendungen Anwendung findet.

In Abbildung 20 sind die Zugriffszeiten der jeweiligen Datei zu sehen. Dabei wird zwischen einer kleinen Datei (Abbildung 20a), welche im GeoJSON-Format 1,2 MB groß ist, und einer großen Datei (Abbildung 20b), welche im GeoJSON-Format 73,4 MB groß ist, unterschieden. Zusätzlich werden in Tabelle 8 die mittleren Zugriffszeiten betrachtet.

Es zeigt sich, dass die durchschnittliche Zugriffszeit bei kleineren Dateien für GeoJSON 0,038 Sekunden beträgt, während sie für FGB mit 0,0506 Sekunden etwas langsamer ausfällt. Zu beachten ist jedoch, dass beim ersten Laden vom FGB eine deutlich längere Zugriffszeit gemessen wurde, was auf Latenzen oder das Hochfahren der Hardware zurückzuführen sein könnte. Danach waren die Zugriffszeiten von FGB in den meisten Fällen schneller als die von GeoJSON. Dieser Ausreißer beim ersten Zugriff erhöht den Mittelwert der Zugriffszeiten für FGB, was den Eindruck erweckt, dass es langsamer ist, obwohl es bei den meisten Zugriffen effizienter arbeitet. Bei der großen Datei gibt es einen deutlichen Unterschied in den Zugriffszeiten. Während beim Laden des GeoJSON eine durchschnittliche Zugriffszeit von 5,900 Sekunden benötigt wird, ist das Laden des FGB mit 3,252 Sekunden fast doppelt so schnell. Durch diese deutliche Zeitersparnis beim Laden des FGB ergeben sich klare Vorteile bei der Visualisierung der Daten, da die schnellere Zugriffszeit eine flüssigere und



(a) *Kleine Datei (1,2 MB GeoJSON)* **(b)** *Große Datei (73,4 MB GeoJSON)*

Abbildung 20: *Vergleich der Zugriffszeiten von GeoJSON und FGB bei unterschiedlichen Dateigrößen*

reaktionsschnellere Darstellung ermöglicht. Darüber hinaus ist die Generierung eines FGB wesentlich einfacher und weniger ressourcenintensiv als die aufwendige Berechnung von Vektorkacheln.

Tabelle 8: *Mittelwert der Zugriffszeiten bei GeoJSON und FGB*

	GeoJSON	**FGB**
Kleine Datei (1,2 MB GeoJSON)	0,038 s	0,0506 s
Große Datei (73,4 MB GeoJSON)	5,900 s	3,252 s

Ein weiterer Vorteil des FGB-Formats ist der geringere Speicherbedarf im Vergleich zu GeoJSON. Die entsprechenden Dateigrößen der Testdaten in den Formaten sind in der Tabelle 9 angegeben. Aufgrund der binären Speicherstruktur ist das FGB zwar nicht mehr menschenlesbar, jedoch reduziert sich der Speicherbedarf im Gegensatz zum GeoJSON um rund die Hälfte. Dies führt zu einer effizienteren Speichernutzung sowie zu einer weiteren Beschleunigung der Übertragung und Verarbeitung der Daten. Insgesamt erweist sich das FGB-Format als besonders geeignet für die Speicherung ergänzender Layer, wie bspw. die Verdachtsgebiete. Da die Verdachtsgebiete eine klein Dateigröße (1,2 MB) aufweisen, fallen die Unterschiede in der Laufzeit hier nicht stark ins Gewicht. Zukünftig könnten jedoch auch größere Vektordaten beim BOB.NI hinzukommen, bei denen das FGB-Format aufgrund des geringeren Speicherbedarfs und der schnelleren Zugriffszeiten signifikantere Vorteile bieten würde.

Tabelle 9: *Vergleich Dateigrößen von GeoJSON und FGB*

	GeoJSON	**FGB**
Kleine Datei	1,2 MB	0,6 MB
Große Datei	73,4 MB	39,0 MB

5.3.3 Zwischenfazit Cloud-optimierte Datenformate

Dieses Kapitel zeigt, dass sowohl das COG-Format als auch das FGB-Format in Kombination mit einem COS vielversprechende Alternativen zu herkömmlichen Dateiformaten und Datenbanken für die Verwaltung und Verarbeitung von Bodenbewegungsdaten darstellen. Das COG-Format bietet deutliche Vorteile gegenüber traditionellen GeoTIFFs und PostgreSQL-Datenbanken, insbesondere in Bezug auf Speicher- und Laufzeiteffizienz. Laufzeittests belegen, dass COG in Verbindung mit einem COS eine leistungsstarke Methode zur Abfrage der rasterbasierter Bodenbewegungsdaten ist. Die Nutzung eines COS bietet zusätzliche Vorteile wie nahezu unbegrenzte Skalierbarkeit und die flexible Verwaltung verschiedener Datentypen, was insbesondere für Cloud-native Anwendungen von großem Vorteil ist (siehe Kapitel 2.2.4). Zudem eignet sich ein COS ideal für rasterbasierte Daten, da diese als Dateien vorliegen und effizient als Objekte gespeichert und verwaltet werden können, was eine schnelle Speicherung ohne Umwege über eine Datenbank ermöglicht. Auch das FGB-Format erweist sich als effizienter als das traditionelle GeoJSON-Vektorformat und ist einfacher zu erzeugen als die im Prototyp verwendeten Vector-Tiles, was es ermöglicht, Verdachtsgebietgeometrien bei Aktualisierungen schneller auszutauschen.

Die Kombination aus COG, FGB und COS ermöglicht eine effiziente Datenhaltung, die den Einsatz einer Datenbank überflüssig macht und somit Kosteneinsparungen ermöglicht. Diese Komponenten bieten zudem wesentliche Vorteile, wie eine verbesserte Abfragegeschwindigkeit, nahezu unbegrenzte Skalierbarkeit und eine erhöhte Flexibilität bei der Verwaltung unterschiedlicher Datentypen. Dadurch eignet sie sich besonders gut für Cloud-native Anwendungen. Es wird daher empfohlen, diese Architektur auf Basis der Kombination von COG, FGB und COS zu nutzen.

6 Konzept und Architektur

Im folgenden Kapitel werden das grundlegende Konzept sowie die Architektur der Lösung beschrieben, die als Basis für die konkrete Umsetzung in Kapitel 7 dient. Zunächst wird die allgemeine Idee der neuen Cloud-nativen Architektur anhand eines Konzepts erläutert. Darauf aufbauend wird ein detailliertes Architekturmodell entwickelt, das sowohl die theoretischen Grundlagen als auch die erhobenen Anforderungen berücksichtigt. Wie bereits in Kapitel 2.3 beschrieben wurde, versteht sich der Begriff Architektur im Rahmen dieser Arbeit als eine Komposition aus Infrastruktur, Softwarearchitektur und der notwendigen Datenvorverarbeitung. Für jede dieser Komponenten wird innerhalb dieses Kapitels ein eigenes Konzept entwickelt, welches die erhobenen Anforderungen aus Kapitel 4 erfüllt. Diese Modelle fügen sich in der Umsetzung zu einer ganzheitlichen Architektur zusammen, die eine effiziente Cloud-native Architektur zu Verwaltung und Verarbeitung der rasterbasierten Bodenbewegungsdaten als Ergebnis hat.

Die umfassende Betrachtung der verschiedenen Aspekte und Komponenten zielt darauf ab, ein tiefes Verständnis der Gesamtarchitektur des BOB.NI zu bekommen. Das Ziel ist es, die theoretischen Grundlagen mit praxisorientierten Umsetzungsstrategien zu verbinden, um eine robuste, zukunftsfähige Anwendung zu schaffen, die den Anforderungen gerecht wird und flexibel auf zukünftige Entwicklungen reagieren kann.

6.1 Konzept

Das Ziel der neuen Architektur besteht darin, den BOB.NI als umfassendes Gesamtsystem zu betrachten. Dabei wird nicht nur das BOB.NI-Modell und die BOB.NI-WebApp berücksichtigt, sondern auch die gesamte Prozesskette zur Generierung des Modells sowie die Datenvorverarbeitung, die für die Bereitstellung der erforderlichen Daten in der BOB.NI-WebApp notwendig ist. Aufgrund dieser ganzheitlichen Betrachtung bedarf es einer Architektur, die sowohl eine Umsetzung und Nutzung von Microservices gemäß der Anforderung *Q5* ermöglicht, als auch eine effiziente und automatisierte Datenverarbeitung nach Anforderung *Q4* unterstützt.

Um diesen Anforderungen gerecht zu werden, wird eine hybride Architektur auf Basis einer serverlosen Plattform gewählt, die eine Kombination aus Microservices und serverlosen Funktionen umfasst. Die Wahl dieser Architektur beruht auf verschiedenen Vorteilen, die zum Teil in Kapitel 2.3.2 erläutert wurden. Zu diesen Vorteilen gehören die einfache Skalierbarkeit, die hohe Flexibilität, die Kompatibilität mit COS sowie die Möglichkeit einer schnellen Implementierung. Zudem ermöglicht eine serverlose Plattform Kosteneinsparungen, da Funktionen auf null skaliert werden können, wenn sie nicht benötigt werden, was sich auf die Erfüllung der Anforderung *Q2* auswirkt.

Die Datenspeicherung und Verwaltung baut auf den Ergebnissen aus Kapitel 5.3 auf, indem als zentraler Speicher ein COS in Kombination mit Cloud-optimierten Geodatenformaten genutzt wird. So wird das rasterbasierte Bodenbewegungsmodell als COG innerhalb eines Buckets gespeichert. Ergänzende Vektordaten wie z.B. die Verdachtsgebiete werden als FGB gespeichert. Eine Datenbank wird demnach innerhalb der neuen Cloud-nativen Architektur nicht mehr benötigt.

Zur Erfüllung der Anforderungen *Q4* und *Q7* an eine effiziente, automatisierte und medienbruchfreie Datenverarbeitung wird eine automatisierte Verarbeitungspipeline entwickelt, die auf einer Kombination aus serverlosen Funktionen, Microservices und einem COS basiert. Diese Architektur verwendet Ereignis-Subskriptionen, um auf spezifische Ereignisse wie das Laden oder Ändern von Objekten in einem Bucket zu reagieren. Entsprechende serverlose Funktionen oder Microservices werden auf der serverlosen Plattform ausgeführt. Diese Architektur ist auch mit dem bereits beschriebenen *bob-tiling-service* kombinierbar.

Die resultierenden Daten werden letztlich über die BOB.NI-WebApp bereitgestellt. Das Frontend ist als containerisierte Anwendung auf der serverlosen Plattform implementiert. Visualisierungsdaten, wie Konturpolygone als Vector-Tiles und Verdachtsgebiete als FGB, sowie weitere zukünftige Layer sind in einem Bucket gespeichert, der direkt vom Frontend angesteuert wird. Die Abfrage der Daten erfolgt mittels der in Kapitel 5.2.1 ermittelten Microservices. Diese sind über entsprechende REST-APIs ansteuerbar und verfügen über generische Schnittstellen, wodurch diese auch in anderen rasterbasierten Kontexten Anwendung finden können.

6.2 Infrastruktur

Für die Infrastruktur wird die IBM-Cloud verwendet, bei der es sich um eine Public Cloud handelt. Der Grund für die Wahl der IBM-Cloud ist, dass diese die am häufigsten genutzte Cloud innerhalb des LGLN ist und ein Wechsel zu einem anderen Cloud-Anbieter keine signifikanten Vorteile bringen würde. Die Infrastruktur besteht aus fünf Komponenten: der IBM-Code-Engine, der Container-Registry, der IBM Cloud Schematics, dem Quellcode Repository und dem IBM COS mit unterschiedlichen Buckets. Diese Komponenten werden im Folgenden näher erläutert und ihre

Verwendung begründet.

IBM Cloud Code Engine
Als serverlose Plattform wird die IBM Cloud Code Engine (IBM CE) gewählt, die auf Kubernetes basiert und die wesentlichen Vorteile dieser Technologie ohne deren Komplexität bietet (siehe Kapitel 2.3.2). Sie dient als zentrale Laufzeitumgebung der Architektur und unterstützt die Ausführung von Jobs, Funktionen und Anwendungen. Trotz der vereinfachten Handhabung erlaubt die IBM CE den Zugriff auf die vollständige Kubernetes-Konfiguration und die Verwendung vertrauter Werkzeuge wie *kubectl* (IBM, 2024e). Durch die Kubernetes-Basis verfügt die IBM CE über wesentliche Merkmale wie Ressourcenkontrolle, Lastenverteilung, Gesundheitsprüfung und Autoskalierung, die in Kapitel 2.2.3 beschrieben sind. Zusätzlich unterstützt die Plattform die zentralen Cloud-Computing-Charakteristika wie On-Demand-Self-Service, Ressourcenkonsolidierung, Broad Network Access und Rapid Elasticity (siehe Kapitel 2.1.1), wodurch eine effiziente, flexible und bedarfsgerechte Nutzung der Ressourcen sichergestellt wird. Diese Merkmale tragen wesentlich dazu bei, die grundlegenden Qualitätsanforderungen der ISO/IEC 25010 (Anforderung *Q9* - *Q14*) zu erfüllen.

In Bezug auf die in Kapitel 4 definierten Anforderungen unterstützt die IBM CE insbesondere die Anforderung *Q5*, die eine Cloud-basierte Microservice-Architektur fordert. Durch ihre flexible und skalierbare Struktur ermöglicht die IBM CE eine einfache Bereitstellung und Anpassung von Microservices. Darüber hinaus trägt sie zur automatisierten Datenverarbeitung bei, indem ereignisgesteuerte Batch-Jobs unterstützt werden, was der Erfüllung der Anforderung *Q4* zugutekommt. Die Anforderungen an die Leistungsfähigkeit (*Q9*) werden durch die automatische Skalierung, die kurzen Reaktionszeiten und den minimalen Ressourcenverbrauch unter variierenden Lastbedingungen unterstützt. Zudem trägt die robuste Infrastruktur der IBM CE zur Zuverlässigkeit bei, indem sie Mechanismen zur automatischen Wiederherstellung im Falle von Störungen bietet, was die Anforderung *Q11* adressiert.

Grundsätzlich werden in der IBM Cloud Code Engine drei Arten von Entitäten unterschieden, die verschiedene Aufgaben und Anforderungen erfüllen (IBM, 2024a):

- **Anwendung:** Eine Anwendung oder App führt den Code zur Verarbeitung von HTTP-Anforderungen aus. Sie verfügt über eine spezifische URL für eingehende Anfragen. Die Anzahl der aktiven Instanzen einer Anwendung wird automatisch entsprechend des eingehenden Workload skaliert, einschließlich der Skalierung auf null, wenn keine Anfragen vorliegen, um eine effiziente Ressourcennutzung sicherzustellen.
- **Funktion:** Eine Funktion ist ein zustandsloser Codeausschnitt, der Aufgaben als Reaktion auf eine HTTP-Anforderung ausführt. Im Gegensatz zu Anwendungen speichert eine Funktion keinen Zustand zwischen den Ausführungen, was sie besonders geeignet für einfache, wiederkehrende Aufgaben macht.
- **Job:** Ein Job führt eine oder mehrere Instanzen eines ausführbaren Codes parallel aus. Im Unterschied zu Anwendungen, die auf die kontinuierliche Verarbeitung von HTTP-Anforderungen ausgelegt sind, werden Jobs einmalig gestartet und nach der Ausführung beendet. Sie eignen sich besonders für Batch-Verarbeitungen oder andere Aufgaben, die nur bei Bedarf durchgeführt werden müssen.

Die Entitäten können entweder durch containerisierte Anwendungen oder direkt durch Quellcode implementiert werden. Jobs und Anwendungen werden hierbei in der Regel als Container ausgeführt.

Damit bietet die IBM CE auch die Möglichkeit, Microservices als Anwendungen zu betreiben. Funktionen hingegen können direkt als Quellcode bereitgestellt werden, ohne dass ein kompletter Container benötigt wird. Dadurch sind diese primär für kleine Aufgaben geeignet.

Quellcode-Repository
Das Quellcode-Repository dient als zentrale Plattform für die Verwaltung des gesamten Quellcodes der Anwendung. Hier werden alle Änderungen am Code durchgeführt und versioniert, um eine konsistente und nachvollziehbare Entwicklung zu gewährleisten. Neben der Verwaltung des Anwendungscodes wird im Repository auch die gesamte Infrastruktur der Anwendung verwaltet. Dies geschieht durch den Einsatz von IaC (siehe Kapitel 2.2.5), wobei Terraform als zentrale deklarative Sprache dient. Darüber hinaus dient das Repository zukünftig als Grundlage für die Continuous Integration/Continuous Deployment (CI/CD)-Pipeline der Anwendung. Diese Pipeline automatisiert das Bauen, Testen und Bereitstellen der Anwendung, um eine schnelle und zuverlässige Softwareauslieferung zu ermöglichen und sicherzustellen, dass Änderungen nahtlos und ohne Unterbrechungen in die Produktionsumgebung übernommen werden können. Zu beachten ist, dass die CI/CD-Pipeline im Rahmen dieser Arbeit nicht umgesetzt wird. Dennoch ist ihre zukünftige Implementierung, insbesondere im Hinblick auf die Anforderung *Q13* zur Wartbarkeit von großer Bedeutung.

IBM Cloud Schematics
IBM Cloud Schematics ist ein Dienst für IaC, der die automatische Erstellung, Konfiguration und Verwaltung von IBM Cloud-Ressourcen mithilfe von Skripten ermöglicht (IBM, 2024b). Wie bereits erwähnt, wird Terraform als zentrale deklarative Sprache eingesetzt. Die erstellten Skripte werden anschließend von IBM Cloud Schematics genutzt, um die Infrastruktur auf Basis des Terraform-Codes zu provisionieren und zu verwalten. Ein besonderer Vorteil der Nutzung von IaC mittels Terraform ist die Plattformunabhängigkeit. Durch die Nutzung von IaC kann die Infrastruktur auch auf anderen Cloud-Plattformen implementiert werden, etwa im Falle eines Wechsels von der IBM Cloud. Dadurch bleibt ein Großteil der Infrastruktur portabel, was die Anforderung *Q14* erfüllt. Zudem lässt sich der gesamte Prozess durch die Integration in CI/CD-Pipelines weiter automatisieren, was die Effizienz und Skalierbarkeit der Lösung zusätzlich erhöht.

IBM Cloud Container Registry
Innerhalb der IBM Cloud Container Registry (IBM CR) werden die Container-Images, welche die Anwendung bilden, zentral verwaltet. Diese Registry ermöglicht es, Container-Images sowohl aus privaten Registry als auch aus lokalen Entwicklungsumgebungen zu speichern (siehe Kapitel 2.2.2). Zusätzlich führt die IBM Cloud Container Registry Sicherheitsüberprüfungen durch, bei denen die genutzten Bibliotheken auf Schwachstellen untersucht werden. Die Umsetzung dieser Sicherheitsüberprüfungen trägt zur Erfüllung der Anforderung *Q12* bei, welche die Sicherheit der Anwendung gewährleistet. Zukünftig soll die Aktualisierung der Anwendungen, die auf den Container-Images basieren, durch die CI/CD-Pipeline automatisiert werden. Dadurch wird sichergestellt, dass immer die neuesten Versionen der Images in der Registry vorhanden sind und die Anwendungen stets auf dem aktuellsten Stand bleiben.

IBM Cloud Object Storage
Der IBM COS wird als zentraler Datenspeicher verwendet und bietet ein S3-kompatibles Interface, das den Zugriff und die Verwaltung der Daten erleichtert. Die konkreten Daten werden in unterschiedliche Buckets gespeichert.

Der grundsätzliche Aufbau und das Zusammenspiel der beschriebenen Komponenten ist in Abbildung 21 dargestellt.

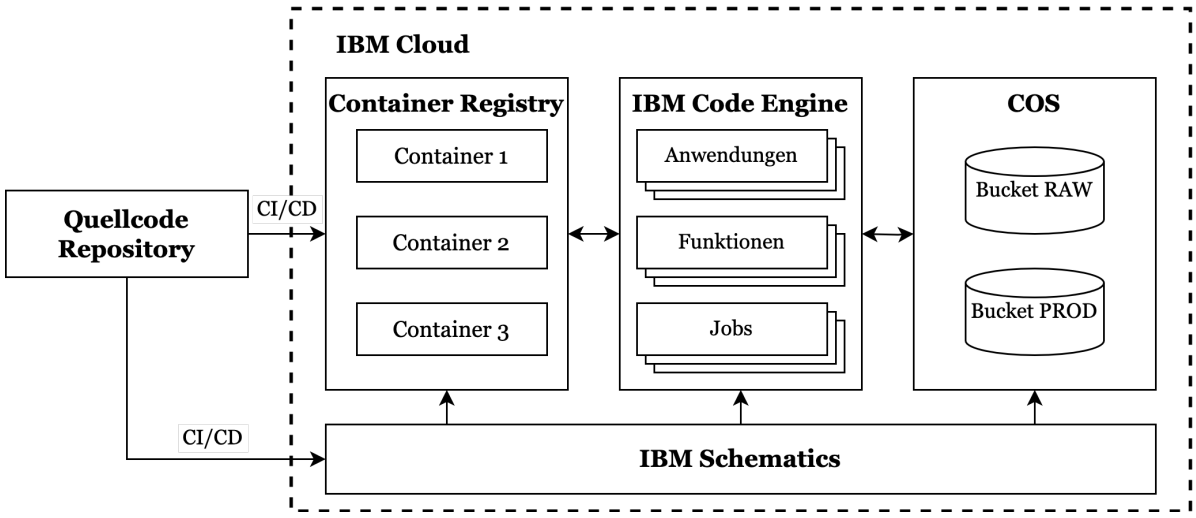


Abbildung 21: *Aufbau der Infrastruktur*

Die Infrastruktur nutzt verschiedene Dienste zur Automatisierung und Verwaltung von Anwendungen und Daten. Der Quellcode wird von einem Repository über CI/CD-Pipelines in die IBM Cloud übertragen. Dort werden Container in der Container Registry gespeichert, die von der IBM Code Engine zur Ausführung von Anwendungen, Funktionen und Jobs genutzt werden. Dabei kommunizieren die einzelnen Entitäten in der IBM CE mit den Daten innerhalb des COS. Über IBM Schematics wird die Infrastruktur mithilfe von Terraform-Skripten aufgebaut. Hier werden die Container Registry, die IBM CE und der COS mit den verschiedenen Buckets definiert und initialisiert.

6.3 Automatisierte Datenverwaltung

Die automatisierte Datenverwaltung ist ein zentraler Bestandteil der Cloud-nativen Architektur. Aufgrund der Anforderung *Q7*, welche eine medienbruchfreie Weitergabe der Bodenbewegungsdaten fordert, sowie der Anforderung *Q4*, welche die schnelle und automatisierte Integration neuer Daten verlangt, ist die Umsetzung einer automatisierten Datenverarbeitung unumgänglich. Sie ersetzt die bisher verwendeten manuellen Skripte und Anwendungen, die im Prototyp zur Datenvorverarbeitung eingesetzt wurden. Im Rahmen einer ganzheitlichen Betrachtung des BOB.NI wird das gesamte Konzept der automatisierten Datenverwaltung, einschließlich der Prozesskette zur Generierung des rasterbasierten Bodenbewegungsmodells (siehe Kapitel 3.3) in seiner Gesamtheit betrachtet. Aufgrund der hohen Komplexität wird jedoch nicht die gesamte Prozesskette in dieser Arbeit umgesetzt. Stattdessen konzentriert sich die Umsetzung auf die automatisierte Datenverarbeitung für neu generierte, rasterbasierte Bodenbewegungsmodelle und aktualisierte Verdachtsgebiete. Diese Umsetzung verdeutlicht das Potenzial der automatisierten Datenverwaltung mittels IBM CE und COS, ohne den Rahmen dieser Arbeit zu überschreiten.

Das ganzheitliche Konzept zur automatisierten Datenverwaltung ist in Abbildung 22 dargestellt. Zusätzlich wird in Abbildung 23 die Funktionsweise des später umgesetzten Teilbereichs als Sequenzdiagramm veranschaulicht. Im Allgemeinen basiert das Konzept auf zwei Buckets und mehreren

Jobs, welche innerhalb der IBM CE ausgeführt werden. Der erste Bucket (*Bucket RAW)* dient der Speicherung der untransformierten Rohdaten. Hier werden neue Datensätze, wie z.B. neue Verdachtsgebiete, über eine definierte Schnittstelle hochgeladen. Über einen sogenannten *Listener* wird die Änderung innerhalb des Buckets erkannt. Mithilfe einer Ereignis-Subskription wird basierend auf dem Dateisuffix bestimmt, welcher Job innerhalb der IBM CE ausgeführt wird. Der entsprechende Job transformiert die Rohdaten und speichert diese nach Abschluss im zweiten Bucket (*Bucket PROD*). Dieser Bucket dient als zentraler Speicherort für die verarbeiteten Daten, die als Grundlage für die BOB.NI-WebApp dienen. Die Jobs der IBM CE sind für diese Aufgabe besonders geeignet, da sie optimal auf die Ausführung von Batch-Jobs abgestimmt sind. Sie werden nur dann aktiv, wenn sie tatsächlich benötigt werden, wodurch Ressourcen effizient genutzt und unnötige Kosten durch ungenutzte Kapazitäten vermieden werden.

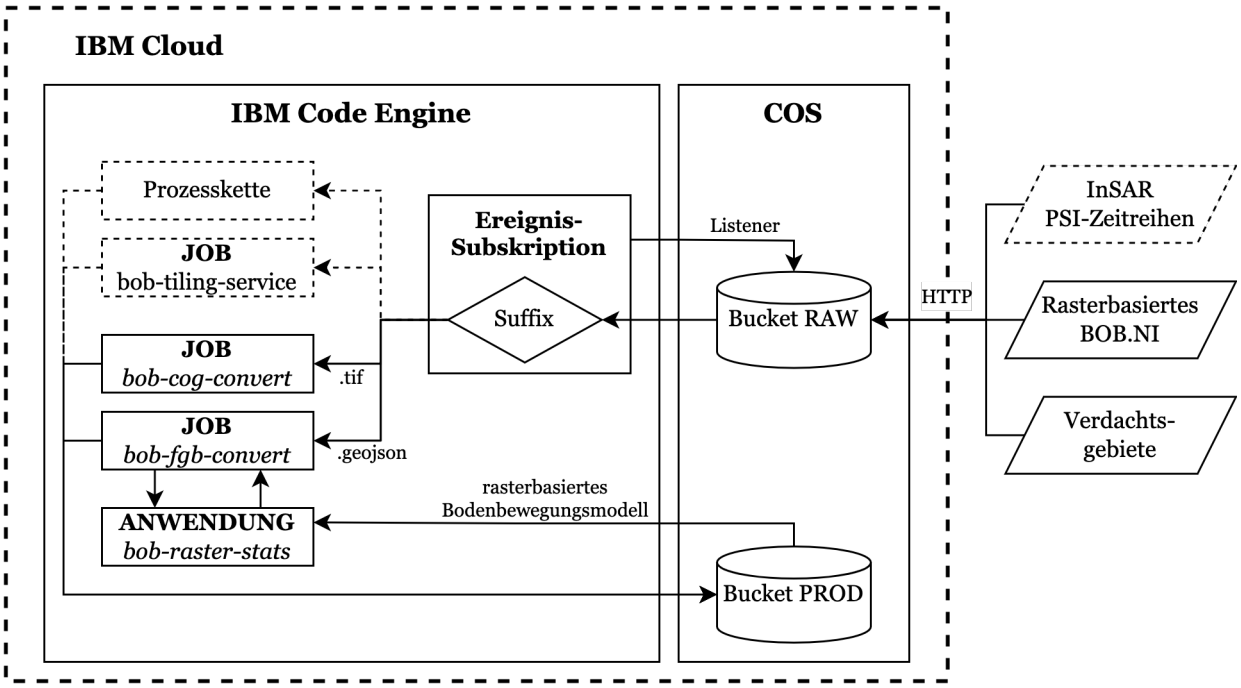


Abbildung 22: *Konzept der automatisierten Datenverwaltung*

Die Automatisierung der gesamten Prozesskette sowie die Nutzung des *bob-tiling-service* werden in dieser Arbeit nur konzeptionell betrachtet, weshalb diese Aspekte in Verbindung mit den InSAR-PSI-Zeitreihen gestrichelt dargestellt sind. Zukünftig sollte das Speicherkonzept so gestaltet werden, dass das BGR, welche die InSAR-PSI-Zeitreihen bereitstellt, eine definierte Schnittstelle zur Verfügung stehen hat, über welche die neuen Daten in den *Bucket RAW* hochgeladen werden können. Eine Ereignis-Subskription erkennt diese Änderung automatisch und stößt basierend auf dem festgelegten Dateisuffix die Prozesskette an. Diese besteht aus mehreren einzelnen Jobs, die auf Basis der bestehenden Skripte implementiert und nacheinander ausgeführt werden können. Aufgrund dessen, dass bestimmte Parameter erst nach der Sichtung der neuen Daten festgelegt werden können, sieht der Ansatz vor, diese Parameter manuell über Umgebungsvariablen zu setzen. Dies ermöglicht eine flexible Anpassung der Werte, ohne die Automatisierung stark einzuschränken. Das Ergebnis dieser Verarbeitungspipeline sind Konturpolygone der Bodenbewegungen als Vector-Tiles für die Visualisierung sowie ein rasterbasiertes Bodenbewegungsmodell im COG-Format zur Abfrage von Bodenbewegungsdaten.

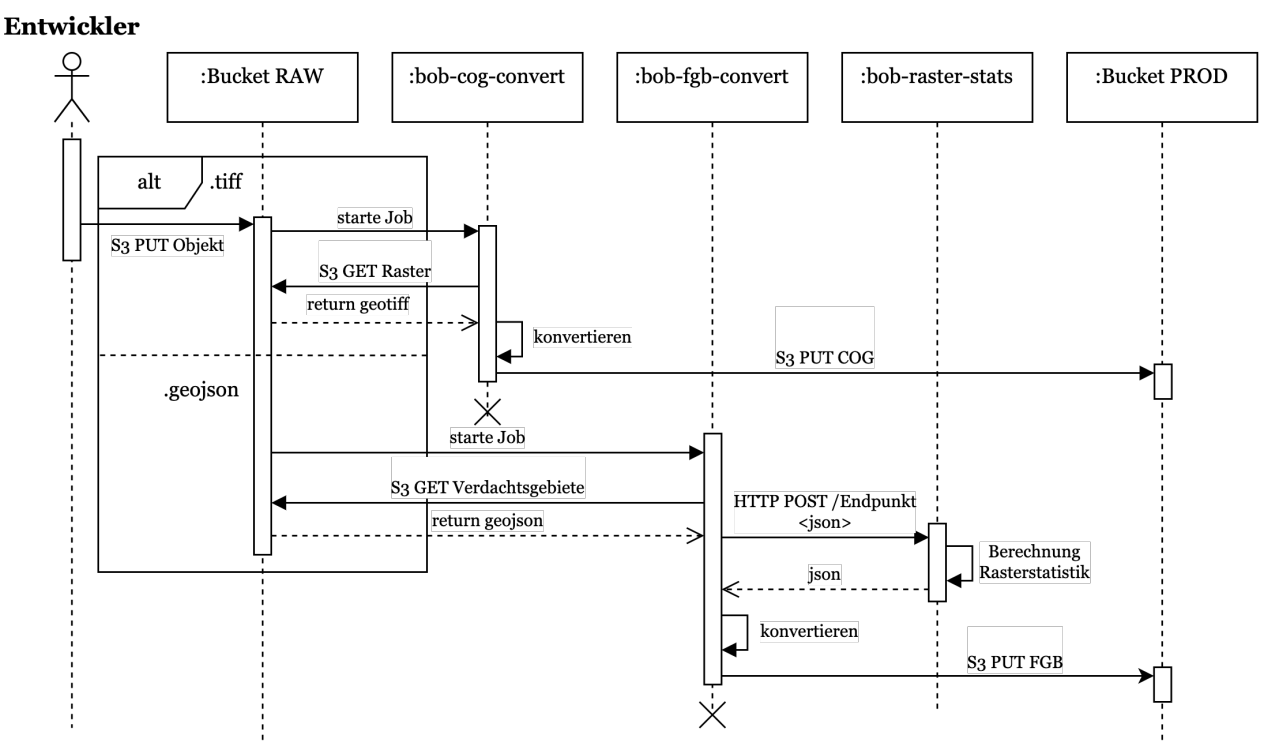


Abbildung 23: *Funktionsweise der implementierten automatisierten Datenverwaltung*

Der *bob-tiling-service* ist ein bereits implementierter Job zur automatisierten Datenverarbeitung. Er wurde als containerisierte Anwendung entwickelt und kann als Batch-Job in einer serverlosen Architektur ausgeführt werden. Dabei übernimmt der Service die Aufgabe, das punktuelle Bodenbewegungsmodell (siehe Kapitel 3.4) in Konturpolygone zu konvertieren. Diese Polygone werden anschließend als Vector-Tiles gespeichert, um eine effiziente Visualisierung der Daten zu ermöglichen. Eine detaillierte Darstellung des Aufbaus des Services findet sich in Anhang C. Da die Polygone, wie in Kapitel 3.4 beschrieben, aus dem rasterbasierten Bodenbewegungsmodell abgeleitet werden, erzeugt dieser Job bereits das rasterbasierte Bodenbewegungsmodell. Dieses wird nur während der Laufzeit des Jobs benötigt, weshalb es nach Abschluss des Jobs gelöscht. Zukünftig könnte der Job so angepasst werden, dass er sowohl die Vector-Tiles als auch das rasterbasierte Bodenbewegungsmodell generiert und speichert. Hierbei kann für das rasterbasierte Bodenbewegungsmodell direkt die Konvertierung oder der Export in ein COG durchgeführt werden. Dadurch ist bei einem neuen Bodenbewegungsmodell lediglich ein Job notwendig.

Im Kontext dieser zukünftigen Ausrichtung wird ein konkretes Konzept für die automatisierte Datenverwaltung entwickelt, das im Rahmen dieser Arbeit umgesetzt wird. Während das langfristige Ziel die vollständige Automatisierung der gesamten Prozesskette ist, konzentriert sich das umgesetzte Speicherkonzept zunächst auf die automatische Verarbeitung eines neuen rasterbasierten Bodenbewegungsmodells, das durch die manuelle Ausführung der Prozesskette aus Kapitel 3.3 entsteht, sowie auf die Integration aktualisierter oder neu identifizierter Verdachtsgebiete. Basierend auf den Ergebnissen der Analyse in Kapitel 5.3 wurde festgestellt, dass die Formate COG und FGB besonders gut für die Anforderungen des BOB.NI geeignet sind. Daher sollten die derzeit verwendeten herkömmlichen Datenformate automatisiert in die entsprechenden Cloud-optimierten

Formate COG und FGB konvertiert werden. Zusätzlich sollen für die Verdachtsgebiete automatisch neue Bodenbewegungsstatistiken berechnet werden. Die Vorgehensweise ist im Sequenzdiagramm in Abbildung 23 dargestellt.

Bei neu erkannten Dateien im Bucket *Bucket RAW*, wird anhand der Suffix der Dateien überprüft, welcher Job ausgeführt werden muss. Für Dateien mit dem Suffix `.tif` wird der Job *bob-cog-convert* ausgeführt, welche die Konvertierung eines normalen GeoTIFF zu einem COG durchführt. Da das rasterbasierte Bodenbewegungsmodell aus insgesamt vier Dateien besteht, wird der Job für jede Datei separat gestartet. Die konvertierten Dateien werden anschließend im zweiten Bucket (*Bucket PROD*) mit einem festgelegten Dateinamen gespeichert. Sofern bereits Daten unter dem Namen vorhanden sind, werden diese Dateien mit dem neuen rasterbasierten Bodenbewegungsmodell überschrieben. Dies stellt sicher, dass immer das neueste rasterbasierte Bodenbewegungsmodell im *Bucket PROD* vorhanden ist. Die Rohdaten im *Bucket RAW* bleiben erhalten und werden erst überschrieben, wenn neue Daten eingehen. So wird gewährleistet, dass die ursprünglichen Rohdaten als Backup dienen, falls während der automatisierten Datenverarbeitung Fehler auftreten sollten. Ein ähnlicher Ansatz wie beim rasterbasierten Bodenbewegungsmodell erfolgt bei den Verdachtsgebieten. Wenn die Ereignis-Subskription eine neue `.json`-Datei im *Bucket RAW* erkennt, wird der Job *bob-fgb-convert* gestartet. Aufgrund der Anforderung *F4*, die verlangt, dass Bodenbewegungsstatistiken für die Verdachtsgebiete verfügbar sind, wird neben der Konvertierung in das FGB-Format auch die Berechnung der Rasterstatistiken für die Verdachtsgebiete durchgeführt. Hierfür wird der Microservice zur Berechnung der Rasterstatistiken genutzt, der direkt innerhalb des Jobs aufgerufen wird. Die Ergebnisse der Rasterstatistiken werden anschließend in die neue Datei integriert. Die Umsetzung der beschriebenen automatisierten Datenverwaltung mit den entsprechenden Jobs erfolgt in Kapitel 7.3.

6.4 Softwarearchitektur

Das folgende Kapitel beschreibt den Aufbau der Softwarearchitektur, welche sich im Wesentlichen auf die BOB.NI-WebApp bezieht. Diese Webanwendung basiert auf der zuvor beschriebenen Infrastruktur sowie der automatisierten Datenvorverarbeitung. Als Architekturkonzept wird eine Microservice-Architektur verwendet, die in der serverlosen Umgebung der IBM CE ausgeführt wird. Diese Architektur erfüllt dabei die Anforderung an eine Cloud-basierte Microservice-Architektur nach Anforderung *Q5*.

Nach der Anforderung *Q1* soll die BOB.NI-WebApp zunächst als MVP entwickelt werden, welches die in Kapitel 5.2.1 identifizierten Funktionalitäten beinhaltet. Die Anwendung nutzt eine Microservice-Architektur in Kombination mit IBM CE, wodurch sie flexibel und leicht erweiterbar bleibt. Nach erfolgreicher Implementierung des MVPs kann die Funktionalität mühelos durch zusätzliche Microservices oder serverlose Funktionen erweitert werden, um zukünftige Anforderungen zu erfüllen. Zusätzlich unterstützt die Microservice-Architektur die Leistungsfähigkeit gemäß Anforderung *Q9* und die Wartbarkeit nach Anforderung *Q13*. Durch die Nutzung der Microservices können diese separat voneinander skaliert als auch gewartet werden.

Eine schematische Darstellung der Softwarearchitektur ist in Abbildung 24 dargestellt. Diese besteht im Kern aus dem Frontend sowie dem Backend, welche im Folgenden detaillierter beschrieben werden. Neben der schematischen Darstellung ist in Abbildung 25 die grundsätzliche Funktionsweise mithilfe eines Sequenzdiagramms dargestellt.

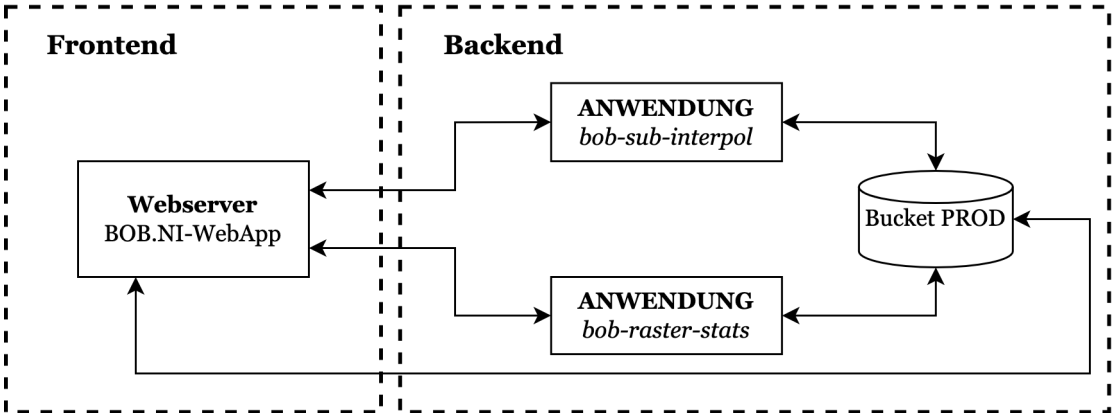


Abbildung 24: *Schematische Darstellung der Softwarearchitektur für die BOB.NI-WebApp*

6.4.1 Backend

Das Backend der Softwarearchitektur stellt die Grundlage der Geschäftslogik dar und verwaltet die Funktionalitäten der Anwendung. Hier erfolgt die Verarbeitung der Benutzereingaben aus dem Frontend. Das Backend besteht aus den Microservices für die Subpixelinterpolation *bob-sub-interpol* und der Berechnung von Rasterstatistiken *bob-raster-stats*, welche in Kombination die Geschäftslogik bilden. Zusätzlich dient, wie bereits in Kapitel 6.3 beschrieben, der Bucket *Bucket PROD* als zentraler Datenspeicher für die Vector-Tiles der Konturpolygone, des rasterbasierten Bodenbewegungsmodells und der Verdachtsgebiete. Die Kommunikation mit dem *Bucket PROD*, der im IBM COS liegt, erfolgt dabei über das S3-Protokoll.

Die Interaktion zwischen dem Frontend und den einzelnen Microservices wird standardisiert über HTTP-Anfragen realisiert. Hierbei kommen nach der Anforderung *Q8* REST-APIs zum Einsatz, die dem REST-Architekturstil folgen. Diese APIs ermöglichen den Zugriff auf Funktionen und Daten über das Internet, wobei die Interaktion durch die Nutzung von HTTP-Methoden wie GET, POST, PUT und DELETE erfolgt. Jede Ressource im System wird dabei an einen Uniform Resource Identifier (URI) adressiert, wodurch spezifische Funktionen oder Datenquellen bereitgestellt werden (Ehsan et al., 2022). Die Nutzung unterschiedlicher Endpunkte kommt primär bei der Subpixelinterpolation zum Einsatz. Dabei bietet der Microservice *bob-sub-interpol* jeweils einen dedizierten Endpunkt für die bilineare und bikubische Subpixelinterpolation. Über diese Endpunkte kann der Nutzer im Frontend die gewünschte Methode zur Subpixelinterpolation auswählen.

Der jeweilige Microservice wird über HTTP POST angesprochen, wobei die notwendigen Parameter, wie in Kapitel 7.2.1 beschrieben, übergeben werden. Der entsprechende Microservice lädt daraufhin das rasterbasierte Bodenbewegungsmodell aus dem dafür vorgesehenen Bucket, führt die Subpixelinterpolation oder Rasterstatistik durch und gibt das verarbeitete Ergebnis an das Frontend zurück. Die Umsetzung des Backends erfolgt in Kapitel 7.2.

6.4.2 Frontend

Das Frontend stellt keinen zentralen Kern der Arbeit dar, allerdings bietet die Implementierung eine wertvolle Möglichkeit, die Funktionalitäten der einzelnen Microservices zu testen und zu veranschaulichen. Zudem bietet die Umsetzung eine Übersicht über eine spätere mögliche Umsetzung der Visualisierung.

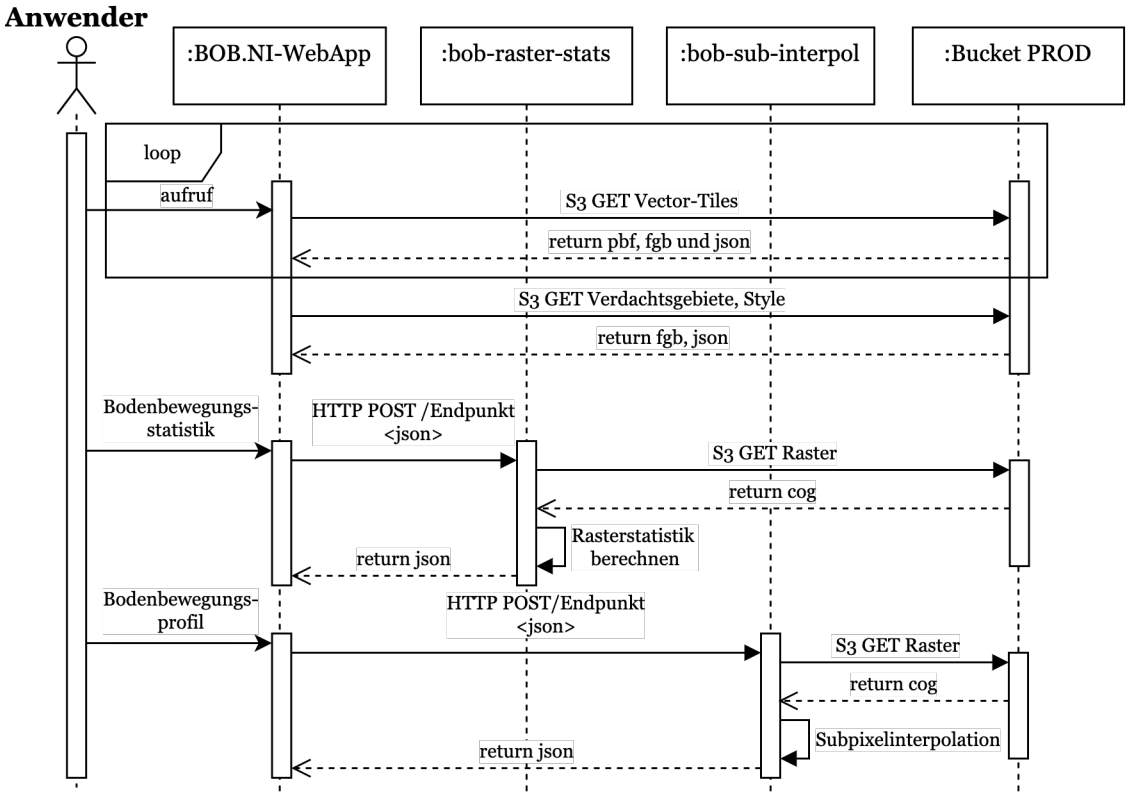


Abbildung 25: *Funktionsweise der Softwarearchitektur*

In Bezug auf die Anforderung *Q6*, welche die potenzielle Nutzung von Services dritter fordert, besteht eine Möglichkeit zur Umsetzung des Frontends durch das Masterportal. Das Masterportal bietet eine zentrale Plattform, die bereits über viele Funktionalitäten und Visualisierungsmöglichkeiten für Geodaten verfügt, welche für die BOB.NI-WebApp Anwendung finden könnte. Allerdings ist das Masterportal relativ neu und komplex, was die schnelle und einfache Implementierung von spezifischen Funktionen erschwert. Daher wird eine prototypische Eigenentwicklung für das Frontend erstellt, um mehr Flexibilität und eine schnellere Implementierung der gewünschten Funktionalitäten zu gewährleisten.

Die BOB.NI-WebApp wird über die IBM CE betrieben und bereitgestellte. Dabei werden zunächst die für die Visualisierung notwendigen Daten aus dem Bucket *Bucket PROD* geladen, darunter die Konturpolygone als Vector-Tiles sowie deren Stildefinition und die Verdachtsgebiete im Format FGB. Frontendseitig stehen dem Anwender Funktionalitäten zur Ermittlung von Bodenbewegungsprofilen und zur Berechnung von Rasterstatistiken innerhalb der Polygone zur Verfügung. Nach entsprechender Nutzung der Funktionalitäten werden die notwendigen Informationen (siehe Kapitel 7.2.1) über die REST-API an die jeweiligen Microservices gesendet. Über den Endpunkt des Microservices für die Subpixelinterpolation kann dabei die jeweilige Interpolationsmethode ausgewählt werden. Die entsprechenden Ergebnisse werden mittels JSON zurückgegeben und visualisiert. Die Umsetzung des Frontends erfolgt in Kapitel 7.4.

7 Umsetzung

Im folgenden Kapitel wird die vollständige prototypische Umsetzung und Implementierung der Cloud-nativen Architektur zur Verwaltung und Verarbeitung rasterbasierter Bodenbewegungsdaten ausführlich beschrieben. Dabei wird auf das in Kapitel 6 vorgestellte Konzept und die entwickelte Architektur zurückgegriffen. Zunächst wird die Implementierung der Infrastruktur mithilfe von Terraform beschrieben. Im Anschluss erfolgt die Realisierung der Microservice-Architektur, beginnend mit einer Beschreibung des grundlegenden Aufbaus der einzelnen Microservices, gefolgt von einer detaillierten Erläuterung der spezifischen Logik jedes Microservices. Daraufhin wird die automatisierte Datenverwaltung umgesetzt, wobei der Fokus auf der Implementierung der jeweiligen Jobs liegt. Zusätzlich wird die Umsetzung der Ereignis-Subskription innerhalb der IBM-Cloud erläutert. Abschließend wird die prototypische Umsetzung des Frontends vorgestellt, das sich auf die Visualisierung der verarbeiteten Daten konzentriert. Eine Auflistung der für die Umsetzung verwendeten Technologien ist im Anhang E zu finden.

7.1 Implementierung Cloud-Infrastruktur

Die Cloud-Infrastruktur in der IBM Cloud wird mithilfe von IaC unter Einsatz der deklarativen Sprache Terraform (siehe Kapitel 2.2.5) und IBM Schematics implementiert. Terraform wurde ausgewählt, da es besonders gut für die modulare Gestaltung der Infrastruktur geeignet ist. Die in dieser Arbeit verwendeten Module wurden bereits vom geoLab-Team „Geodätische Services“, das für die Entwicklung von BOB.NI zuständig ist, erstellt. Dazu gehören unter anderem Module für die IBM CE, IBM COS und dem IBM CR. Auf Grundlage dieser Module wird die in Kapitel 6.2 beschriebene Infrastruktur mit IBM Schematics umgesetzt.

Die modulare Struktur der Terraform-Konfiguration für die Infrastruktur dieser Arbeit ist in Abbildung 26 abgebildet. Grundsätzlich enthalten die Module eine Reihe von Standarddateien, die eine konsistente Struktur zur Definition und Verwaltung der Infrastruktur bereitstellen (HasiCorp, o.J.):

- `main.tf`: Enthält die Definitionen der Infrastruktur, also die konkreten Ressourcen, die angelegt werden.
- `variables.tf`: Definiert die Variablen, die zur Parametrisierung des Moduls genutzt werden.
- `locals.tf`: Definiert lokale Variablen, die innerhalb des Moduls verwendet werden.
- `outputs.tf`: Definiert die Ausgaben, die nach der Ausführung des Moduls zurückgegeben werden.

Als zentrale Komponente dient das Root Module *student-project*, das die gesamte Infrastruktur definiert und IBM als Cloud-Provider festlegt. Es koordiniert das Zusammenspiel der verschiedenen Child Modules und ermöglicht eine unabhängige Verwaltung sowie Wiederverwendbarkeit einzelner Infrastrukturkomponenten.

Die Struktur des Root Module ist darauf ausgelegt, spezifische Child Modules zu referenzieren, die jeweils dedizierte Ressourcen bereitstellen. So konfiguriert bspw. das *code-engine-project*-Modul die IBM CE und verwendet dazu die Dateien `main.tf`, `variables.tf` und `locals.tf`, um die notwendigen Ressourcen zu definieren. Nach diesem modularen Prinzip werden auch die Module

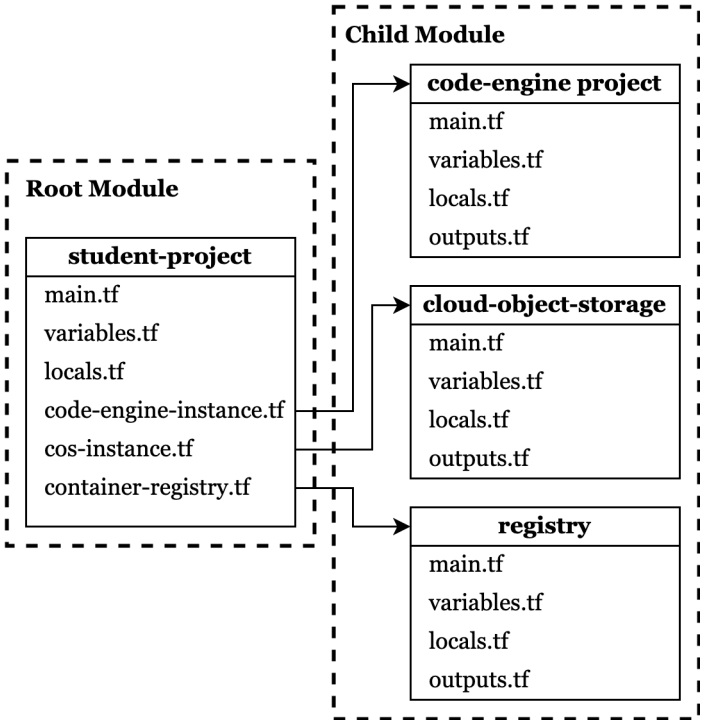


Abbildung 26: *Terraform-Konfiguration*

cloud-object-storage und *container-registry*, welche für das Management von COS und der Container Registry verantwortlich sind, eingebunden.

Die Umsetzung der Infrastruktur wird im Folgenden beispielhaft anhand der Erstellung der COS-Instanz mittels Terraform beschrieben. Ein ähnlicher Aufbau gilt für die IBM CE und die Container Registry.

Der Quelltext 1 zeigt einen Codeausschnitt der Datei `cos-instance.tf`, in der eine COS-Instanz erstellt wird. Dabei wird auf das über einen relativen Pfad eingebundene Modul *cloud-object-storage* verwiesen. Dieses Modul übernimmt die Bereitstellung der COS-Instanz und die Konfiguration der zugehörigen Buckets. Dazu werden entsprechende Parameter definiert, die festlegen, wie die COS-Instanz und die Buckets konfiguriert werden sollen. Dies umfasst unter anderem die Angabe der Region, in der die COS-Instanz betrieben wird. Diese wird über die `variables.tf` auf „eu-de“ festgelegt, wodurch die COS-Instanz in den Rechenzentren in Frankfurt angesiedelt wird. Dies trägt dazu bei, die Latenzzeiten zu verkürzen, da die Daten in geografischer Nähe zu den Nutzern in Europa verarbeitet und gespeichert werden. Zusätzlich werden Parameter für die Buckets initialisiert, wie deren Name, Endpunkttyp, Speicherklasse, Zugriffsrechte und die Konfiguration für Cross-Origin Resource Sharing (CORS). Es ist zu beachten, dass der *Bucket-RAW* nicht öffentlich zugänglich ist, da er nur untransformierte Rohdaten enthält, die intern verarbeitet werden. Im Gegensatz dazu ist der *Bucket-PROD* der zentrale Datenspeicher für die BOB.NI-WebApp. Da dieser Bucket die aufbereiteten Daten für die öffentliche Nutzung bereitstellt, ist er öffentlich zugänglich, um den Zugriff durch die BOB.NI-WebApp zu ermöglichen.

Quelltext 1: *Einbindung des COS-Moduls mit Terraform*

```
# Cloud object storage instance and buckets
module "cos_bobni" {
  source = "../modules/cloud-object-storage"
  region = var.region
  ...
  # Initialization of bucket attributes, specifying name, endpoint type,
    storage
  #class, public access settings, and CORS configuration for each bucket
  buckets = [
    {
      name            = "bucket-raw"
      endpoint_type   = "public"
      storage_class   = "smart"
      public_access   = false
      set_cors_config = true
    },
    {
      name            = "bucket-prod"
      endpoint_type   = "public"
      storage_class   = "smart"
      public_access   = true
      set_cors_config = false
    }
  ]
}
```

Die definierten Parameter werden im Modul *cloud-object-storage* verwendet, um die konkreten Ressourcen in der IBM Cloud anzulegen. Ein Ausschnitt aus der Datei `main.tf` des Moduls ist in Quelltext 2 zu sehen. Hier wird zunächst eine IBM COS-Instanz mit der Ressource `cos_instance` erstellt. Zusätzlich werden weitere, hier nicht näher behandelte Attribute gesetzt. Im zweiten Abschnitt des Codes wird für jedes Bucket-Element in der übergebenen Liste eine COS-Bucket-Ressource (`ibm_cos_bucket`) erstellt. Diese wird dynamisch für jede Bucket-Konfiguration angelegt, indem der Name des Buckets, die Speicherklasse, die Region und der Endpunkttyp für jeden Eintrag in der Liste festgelegt werden.

Quelltext 2: *Modul zur Implementierung eines COS*

```
...
# Creates an IBM cloud object storage instance
resource "ibm_resource_instance" "cos_instance" {
      name              = var.multi_environment ? local.name :
      var.project_name
      resource_group_id = var.resource_group_id
      service           = "cloud-object-storage"
      plan              = var.plan
      location          = "global"
      tags              = var.labels
```

```
11 }
12 ...
13 # Creates a COS bucket for every list item
14 resource "ibm_cos_bucket" "cos_bucket" {
15   for_each             = { for k, v in var.buckets : k => v }
16   bucket_name          = var.multi_environment ? "${var.scope}
17   -${each.value.name}-${var.resource_suffix}" :
       "${var.scope}-${each.value.name}"
18   resource_instance_id = ibm_resource_instance.cos_instance.id
19   storage_class        = each.value.storage_class
20   region_location      = var.region
21   endpoint_type        = each.value.endpoint_type
22   ...
23 }
24 ...
```

Für die Instanz der IBM CE und des IBM CR werden lediglich andere Parameter in den Referenzdateien definiert, die dann im jeweiligen Modul über entsprechende Ressourcen, wie `ibm_code_engine_project` und `ibm_container_registry`, verarbeitet werden. Diese Parameter umfassen spezifische Konfigurationen für die jeweiligen Dienste, die nach dem gleichen modularen Prinzip wie bei der COS-Instanz implementiert werden.

7.2 Implementierung Microservice-Architektur

Im Folgenden wird die Umsetzung der Microservice-Architektur der BOB.NI-WebApp vorgestellt, die speziell für die Abfrage und Verarbeitung rasterbasierter Bodenbewegungsdaten entwickelt wurde. Wie in Kapitel 6.4.1 beschrieben, besteht die Architektur hauptsächlich aus zwei zentralen Microservices, deren Implementierung im weiteren Verlauf detailliert erläutert wird.

7.2.1 Allgemeiner Aufbau Microservices

Die Microservices für die Subpixelinterpolation und die Rasterstatistik werden als containerbasierte Anwendungen entwickelt. Dadurch wird eine flexible und skalierbare Bereitstellung der Services ermöglicht. Da die Microservices statuslos sind, ist zudem keine zusätzliche Datenbank erforderlich. Zudem wird die Kompatibilität gemäß der Anforderung *Q10* verbessert, da die Microservices auch in anderen Systemen eingesetzt werden können (siehe Kapitel 2.2.2).

Die Container-Images der Microservices werden im IBM CR verwaltet und innerhalb der IBM CE als Anwendungen ausgeführt, wobei der Deployment manuell über die Benutzeroberfläche der IBM Cloud erfolgte. Dabei werden die Containerinstanzen in der IBM CE nicht auf Null skaliert, wodurch mindestens eine Instanz kontinuierlich erreichbar bleibt. Dies stellt sicher, dass die Microservices jederzeit verfügbar sind und ohne Verzögerung auf Anfragen reagieren können. Eine Umsetzung mittels der IBM CE Funktionen wurde nicht getestet, könnte jedoch eine Alternative darstellen. Jeder Container ist über einen spezifischen Endpunkt und Port erreichbar. Als Rechenressourcen wird jeder Container-Instanz eine vCPU mit 4 GB Arbeitsspeicher zugewiesen, da in der Entwicklungsphase nur geringe Zugriffszahlen erwartet werden. Ein wesentlicher Vorteil der Cloud-Umgebung besteht darin, dass die Rechenressourcen bei Bedarf problemlos skaliert werden können. Die Anwendung ist aufgrund der Nutzung der IBM CE so konzipiert, dass sie horizontal skalierbar ist. Das bedeutet,

dass bei steigender Last automatisch zusätzliche Instanzen der Container bereitgestellt werden können, um eine optimale Leistung sicherzustellen. Die Anzahl der Instanzen wird flexibel zwischen einer und zehn Instanzen skaliert, abhängig von der aktuellen Anfragelast. Dabei bleibt mindestens eine Instanz stets aktiv, um Verzögerungen durch das Hochfahren neuer Container zu vermeiden. Die automatische Skalierung wird ausgelöst, wenn die gleichzeitige Anzahl von Anfragen 100 erreicht, was als Standardwert innerhalb der IBM CE definiert ist.

Beide Microservices basieren auf einem nahezu identischen Dockerfile. Das Dockerfile des Microservices für die Subpixelinterpolation ist in Quelltext 3 dargestellt. Es verwendet ein Geospatial Data Abstraction Library (GDAL)-Image als Basis, da die für die Subpixelinterpolation und Rasterstatistik benötigten Bibliotheken auf den Funktionalitäten von GDAL aufbauen. Im Dockerfile wird eine Python-Umgebung im Container eingerichtet. Zunächst wird das System aktualisiert, anschließend werden Python 3.10 und Pip installiert. Danach erfolgt die Installation der in der `requirements.txt` definierten Abhängigkeiten. Der Anwendungscode wird in das Arbeitsverzeichnis des Containers kopiert und Umgebungsvariablen für den Docker-Host, den Port sowie die Entwicklungsumgebung werden konfiguriert. Anschließend wird der Port 8000 im Container freigegeben, um die externe Kommunikation zu ermöglichen. Zum Schluss wird die Anwendung gestartet, sodass sie innerhalb des Containers betriebsbereit ist und über den definierten Port auf Anfragen reagieren kann.

Quelltext 3: *Dockerfile für entwickelte Microservices*

```
# Use the GDAL image as a base
FROM ghcr.io/osgeo/gdal:ubuntu-small-3.8.5

# Set the working directory in the container
WORKDIR /code

# Install Python and necessary packages
RUN apt-get update \
    && apt-get install -y python3.10 python3-pip \
    && rm -rf /var/lib/apt/lists/*

# Copy the requirements.txt to the working directory
COPY ./requirements.txt /code/requirements.txt

# Install the Python dependencies
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

# Copy the entire app directory to the working directory
COPY ./app /code/app

# Set environment variables for Docker host, port, and stage
ENV DOCKER_HOST="0.0.0.0"
ENV DOCKER_PORT=8000
ENV STAGE="DEVELOPMENT"

# Expose the port that the container will run on
EXPOSE $DOCKER_PORT
```

```
28
29 # Command to run when the container starts
30 CMD ["python", "/code/app/main.py"]
```

Für die Umsetzung der Microservices wird primär das Framework FastAPI in Kombination mit der Programmiersprache Python 3.10 verwendet. FastAPI bietet eine effiziente und schnelle Verarbeitung von HTTP-Anfragen und ermöglicht die Bereitstellung einer REST-konformen API-Struktur (Tiangolo, 2024). Dadurch wird die Anforderung *Q8* erfüllt, die eine Kommunikation über eine REST-API fordert. Der nachfolgende Quelltext 4 definiert das notwendige Datenmodell sowie einen beispielhaften Endpunkt.

Quelltext 4: *Definition des Datenmodells mit Pydantic und Implementierung eines exemplarischen FastAPI-Endpunkts*

```
1  ...
2  # Define a Pydantic model 'ItemTin' that represents the input data structure
3  class ItemTin(BaseModel):
4      file_name: str
5      file_name_std: Optional[str] = Field(
6          None, description="Optional for std-value at the given coordinate")
7      bucket_name: str
8      band: int
9      epsg: int
10     data: dict
11 ...
12 # Define a POST API endpoint '/tinDataBicubic' that accepts 'ItemTin' as
       input data
13 @app.post("/tinDataBicubic")
14 async def tinDataCubic(item: ItemTin, r: Request):
15 ...
16 return geojson
```

Die Klasse *ItemTin* dient als Datenschema für die HTTP-Anfragen und legt fest, welche Parameter in den Requests enthalten sein müssen. Sie nutzt die Bibliothek *Pydantic*, um die Validierung der Daten zu gewährleisten und stellt sicher, dass nur korrekt formatierte Eingaben akzeptiert werden. Diese Parameter werden in Form einer JSON-Payload übermittelt, die als primäres Austauschformat zwischen dem Frontend und den Microservices verwendet wird. Die JSON-Payload wird mittels HTTP POST an die entsprechende URL und den definierten Endpunkt des Microservices gesendet, der mithilfe von FastAPI implementiert ist. Im Quelltext 4 ist beispielhaft die Initialisierung des Endpunkts `tinDataBicubic` dargestellt. Der Decorator `@app.post("/tinDataBicubic")` verbindet den Endpunkt mit der Funktion `tinDataCubic`, die durch den POST-Request aufgerufen wird. In diesem Fall empfängt die Funktion ein JSON-Objekt vom Typ `ItemTin` sowie eine Anfrage `Request` und verarbeitet diese asynchron. Dadurch kann die Funktion im Hintergrund arbeiten, was es dem System ermöglicht, während der Verarbeitung parallel weitere Anfragen zu bearbeiten. Die für die Anfragen erforderlichen Elemente der Klasse `ItemTin` werden im Folgenden näher beschrieben:

- **Dateiname der Bodenbewegung**: Dieser Parameter ermöglicht es, den Namen der entsprechenden Rasterdatei anzugeben. Da das rasterbasierte Bodenbewegungsmodell sowohl vertikale als auch horizontale Bodenbewegungsinformationen enthält, kann über den Da-

teinamen festgelegt werden, welche dieser beiden Informationen abgefragt werden soll. Die Entscheidung, ob die vertikale oder horizontale Komponente verwendet wird, wird somit anhand des Dateinamens getroffen.

- **Dateiname der Standardabweichung**: Dieser Parameter erlaubt die Auswahl der Rasterdatei, welche die Standardabweichung der Bodenbewegung repräsentiert. Sollte keine Standardabweichung benötigt werden oder sie ist in anderen Kontexten nicht vorhanden, kann dieser Parameter leer bleiben.
- **Bucket-Name**: Der Bucket-Name gibt an, in welchem Bucket sich die benötigten Rasterdateien befinden. Aktuell wird nur auf Buckets zugegriffen, die innerhalb der bestehenden Architektur liegen und über definierte Zugangsdaten (Credentials) erreichbar sind. Zukünftig besteht jedoch die Möglichkeit, diese Struktur generischer zu gestalten, sodass auch externe Buckets, die außerhalb der aktuellen Architektur liegen, eingebunden und genutzt werden können.
- **Rasterband**: Über diesen Parameter wird das spezifische Rasterband ausgewählt, das für die Anfrage relevant ist. In einem Multi-Band-Raster kann jedes Band unterschiedliche Informationen enthalten, daher ist es wichtig, das passende Rasterband zu spezifizieren.
- **EPSG-Code**: Der EPSG-Code definiert das räumliche Bezugssystem, in dem die Daten vorliegen und verarbeitet werden.
- **Daten**: Dieser Parameter enthält die spezifischen Informationen, die für die jeweilige Funktionalität benötigt werden. Für die Subpixelinterpolation werden hier die einzelnen Koordinaten sowie die Distanzwerte (x, y, s) angegeben, an denen der interpolierte Subpixelwert berechnet werden soll. Für die Berechnung der Rasterstatistik enthält dieser Parameter Polygone im GeoJSON-Format.

Eine beispielhafte Anfrage an den Microservice der Subpixelinterpolation ist in Anhang F dargestellt. Die Ergebnisrückgabe erfolgt im GeoJSON-Format. Dabei werden die jeweiligen Geometrien (Polygone oder Punkte) als Feature Collection mit den entsprechenden Ergebnisattributen zurückgegeben. Der Vorteil der Rückgabe der Ergebnisse mittels GeoJSON liegt zum einen in der schnellen Kompatibilität mit modernen Mapping-Bibliotheken wie Leaflet und MapLibre, zum anderen kann das Ergebnis direkt in ein GIS-System wie QGIS geladen werden, wodurch eine zukünftige direkte Anbindung an ein GIS anzudenken ist. Zudem wird durch die Nutzung von GeoJSON versucht, eine OGC-konforme Rückgabe der Daten zu gewährleisten. Dies ermöglicht eine effiziente Integration der Ergebnisse in bestehende GIS-Workflows.

7.2.2 Microservice Subpixelinterpolation

Im Folgenden wird die konkrete Logik der Subpixelinterpolation beschrieben. Der Microservice stellt dafür zwei Endpunkte bereit: einen für die bilineare Interpolation (`/tinDataBilinear`) und einen weiteren für die bikubische Interpolation (`/tinDataBicubic`). Obwohl es zwei separate Endpunkte gibt, greifen beide größtenteils auf dieselben Funktionen zurück. Die Hauptunterschiede liegen in der spezifischen Interpolationsmethode, die jeweils zur Anwendung kommt. Anzumerken ist, dass die Logik der bilinearen Interpolation zum Teil auf dem Prototyp basiert, jedoch aufgrund der veränderten Struktur und der Nutzung eines COS anstelle einer Datenbank nahezu vollständig neu konzipiert werden musste.

Die Eingabe basiert auf der in Kapitel 7.2.1 beschriebenen Payload, wobei eine einzelne Koordinaten oder eine Liste von mehrere Koordinaten als konkrete Daten übermittelt werden. Diese Koordinaten repräsentieren bspw. Punkte entlang der Profillinie, an denen die Bodenbewegungsdaten abgerufen werden sollen. Diese Koordinaten können entweder im WGS84- oder UTM32-Format vorliegen. Falls die Koordinaten im WGS84-Format angegeben werden, erfolgt eine automatische Umwandlung in das metrische UTM-Koordinatensystem mit der Bibliothek *pyproj*, um die Berechnungen zu vereinfachen und effizienter durchführen zu können. Als Datengrundlage zur Abfrage dient die COG-Datei, deren Name über die Payload übergeben wird. Zusätzlich wird für jeden Punkt die Standardabweichung aus dem zugehörigen Rasterdatensatz abgefragt, um die Unsicherheit der Bodenbewegungsinformation abschätzen zu können. Dies ist insbesondere bei der Darstellung der Profillinien wichtig, um das Unsicherheitsband berechnen und visualisieren zu können. Für jeden Punkt müssen demnach zwei COG-Dateien abgefragt und subinterpoliert werden, um sowohl die Bodenbewegungsinformation als auch die Standardabweichung an der entsprechenden Koordinate zu ermitteln.

Die Subpixelinterpolation erfolgt sukzessiv für jede Koordinate des Profils. Zunächst wird, abhängig von der Auflösung des Rasters und der gegebenen Koordinate, das Zentrum der jeweiligen Nachbarschaft mittels der Funktion in Quelltext 5 ermittelt.

Quelltext 5: *Ermittlung des Mittelpunktes der Nachbarschaft*

```
1  def get_cell_center(coordinate_component, resolution):
2      """Get center of a rastercell
3      Arguments:
4              coordinate_component --  X, Y komponent of the coordinate
5              resolution -- Size of a pixel
6      """
7      modulo = coordinate_component % resolution
8      if modulo >= resolution / 2:
9          return coordinate_component - coordinate_component % resolution +
       resolution
10     else:
11         return coordinate_component - coordinate_component % resolution
```

Dabei ist `coordinate_component` die gegebene X- oder Y-Komponente der Koordinate und `resolution` die Auflösung des Rasters. Die Auflösung des Rasters wird dabei über eine entsprechende Abfrage der Metadaten der angefragten Datei bestimmt. Die Funktion stellt sicher, dass der Mittelpunkt jeder Nachbarschaft so festgelegt wird, dass er möglichst nahe an der vorgegebenen Koordinate liegt. Dadurch wird gewährleistet, dass die korrekten umliegenden Pixel für die Nachbarschaft ausgewählt werden. Ausgehend von diesem berechneten Zentrum lassen sich sowohl die 2×2- als auch die 4×4-Nachbarschaften ableiten. Der entsprechende Quelltext für diese Berechnung ist in Quelltext 6 dargestellt. Dafür werden ausgehend vom berechneten Zentrum (`coord_center_e` und `coord_center_n`) die Koordinaten der benachbarten Zellen bestimmt, indem die Auflösung (`resolution`) genutzt wird, um die jeweiligen Abstände in X- und Y-Richtung festzulegen. Diese Abstände werden dann entsprechend der Größe der Nachbarschaft (2×2 oder 4×4) multipliziert, um die genauen Positionen der benachbarten Zellen zu berechnen. Zunächst werden die westlich und östlich des Zentrums liegenden Koordinaten berechnet und in der Liste `bbox_e` gespeichert. Anschließend erfolgt die Berechnung der südlich und nördlich des Zentrums befindlichen Koordinaten, welche in der Liste `bbox_n` abgelegt werden. Diese Koordinaten definieren die benachbarten Zellen,

die in jeder Richtung die Nachbarschaft um das Zentrum abdecken. Die daraus resultierende Liste enthält somit die X- und Y-Koordinaten, die die Nachbarschaft vollständig beschreiben.

Quelltext 6: *Ermittlung der Nachbarschaft für die Interpolation*

```
def get_bboxN(coord, coord_center_e, coord_center_n, resolution,
    neighborhood_size):
    """Get bounding box of a raster cell (n x n)
    Arguments:
            coord -- Center coordinate (midpoint)
            resolution -- Size of a pixel
            coord_center_e -- Center coordinate in the east-west direction
            coord_center_n -- Center coordinate in the north-south direction
            neighborhood_size -- Size of the neighborhood (1: 2x2, 2: 4x4)
    """
    bbox_e = []
    # Start west of the center
    e_local = coord_center_e - neighborhood_size * resolution
    # Loop through east direction to find bounding box (East-direction)
    while e_local <= coord_center_e + neighborhood_size * resolution:
        if coord[0] - resolution * neighborhood_size < e_local < coord[0] +
    resolution * neighborhood_size:
            bbox_e.append(e_local)
        e_local += resolution

    bbox_n = []
    # Start south of the center
    n_local = coord_center_n - neighborhood_size * resolution
    # Loop through north direction to find bounding box (North-direction)
    while n_local <= coord_center_n + neighborhood_size * resolution:
        if coord[1] - resolution * neighborhood_size < n_local < coord[1] +
    resolution * neighborhood_size:
            bbox_n.append(n_local)
        n_local += resolution

    return [bbox_e, bbox_n]
```

Die resultierende Nachbarschaft dient zur Abfrage der Bodenbewegungsinformationen aus dem rasterbasierten Bodenbewegungsmodell für die Subpixelinterpolation. Diese Abfrage erfolgt über eine Methode der separaten Klasse *S3instance*. In dieser Klasse wird mithilfe der Bibliothek *boto3* ein S3-Client eingerichtet, der die Kommunikation mit dem COS ermöglicht. Dabei wird der direkte Endpunkt des COS genutzt, um eine direkte Verbindung herzustellen, wodurch Umwege über das öffentliche Internet vermieden werden und die COG-Dateien effizient geladen werden können.

Für die Abfrage der Pixelinformationen wird zunächst mittels der *rasterio*-Bibliothek die benötigte Datei aus dem entsprechenden Bucket geöffnet. Dabei wird die Funktion `rasterio.open()` verwendet, die es ermöglicht, auf die Datei zuzugreifen, ohne sie vollständig herunterladen zu müssen. Der dazugehörige Quelltext ist in Quelltext 7 dargestellt.

Quelltext 7: *Öffnen einer Rasterdatei mittels rasterio*

```
...
# Open the primary raster file if not already opened
url = f's3://{bucket_name}/{file_name}'
        if file_name not in self.src:
            self.src[file_name] = rasterio.open(url)
...
```

Im Anschluss wird ein *Window* für die berechnete Nachbarschaft innerhalb der COG-Datei erzeugt, das mithilfe eines HTTP-Range-Requests nur die relevanten Datenbereiche abruft, um eine effiziente Abfrage zu ermöglichen. Über dieses *Window* werden die Pixelinformationen gezielt aus der COG-Datei abgegriffen. Dadurch müssen, abhängig von der gewählten Interpolationsmethode, lediglich 4 Pixel bei der bilinearen Interpolation oder 16 Pixel bei der bikubischen Interpolation pro Punkt geladen werden, was die Effizienz der Datenverarbeitung erheblich steigert. Der entsprechende Quelltext für die Bildung des *Window* sowie der Abfrage der Pixelinformationen ist in 8 dargestellt.

Quelltext 8: *Abfrage der Pixelinformationen über ein Window*

```
...
# Convert coordinates to row and column indices and determine the bounding box
indices = []
for east, north in coordinates:
    indices.append(src.index(east, north))
rows, cols = zip(*indices)
min_row, max_row = min(rows), max(rows)
min_col, max_col = min(cols), max(cols)

# Calculate the window
window = Window(min_col, min_row, max_col -
                min_col + 1, max_row - min_row + 1)

# Reading only the window for better performance
pixel_values = src.read(band, window=window)

# Create an array of the same shape as pixel_values to map coordinates to
    pixel values
results = {}
for (east, north), (row, col) in zip(coordinates, indices):
    pixel_value = pixel_values[row - min_row, col - min_col]
    results[(east, north)] = pixel_value

return results
...
```

Hierbei werden zunächst die Koordinaten der ermittelten Nachbarschaft in Rasterindizes konvertiert. Anschließend wird auf Basis der Nachbarschaftskoordinaten die Bounding Box bestimmt, welche das *Window* definiert. Schließlich werden die Pixelwerte des eingelesenen Fensters den entsprechenden Koordinaten zugeordnet und gespeichert.

Die abgefragten Pixelinformationen werden für die Subpixelinterpolation mittels des *RegularGridInterpolator* verwendet. Hierbei ist zu beachten, dass Sonderfälle auftreten können, die mittels einer Fallunterscheidung behandelt werden müssen. Diese sind in Abbildung 27 dargestellt und im Folgenden beschrieben:

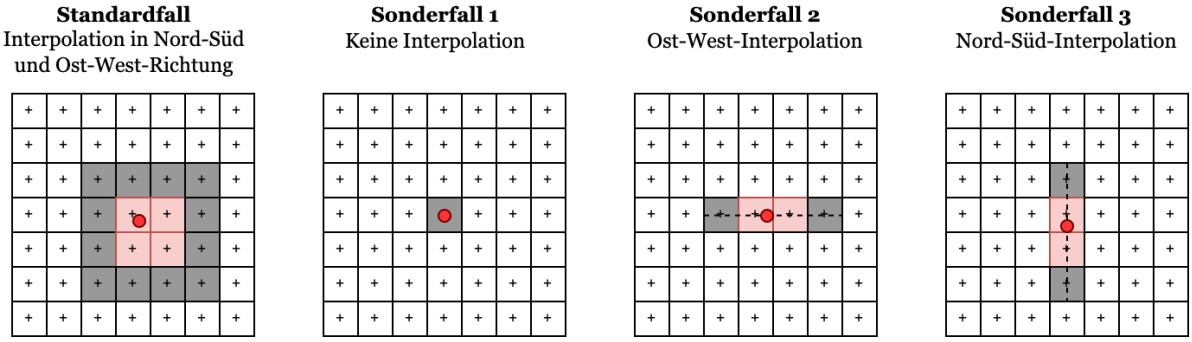


Abbildung 27: *Sonderfälle bei bilinearer und bikubischer Subpixelinterpolation*

- **Standardfall:** Die Koordinate liegt innerhalb einer Rasterzelle. In diesem Fall erfolgt die Interpolation sowohl in Nord-Süd- als auch in Ost-West-Richtung.
- **Sonderfall 1:** Die Koordinate liegt direkt auf dem Mittelpunkt der Rasterzelle. Hier ist keine Interpolation erforderlich, und der Pixelwert wird direkt übernommen.
- **Sonderfall 2:** Die Koordinate liegt auf derselben Y-Achse wie der Mittelpunkt der Rasterzelle. In diesem Fall ist nur eine Interpolation in Ost-West-Richtung notwendig.
- **Sonderfall 3:** Die Koordinate liegt auf derselben X-Achse wie der Mittelpunkt der Rasterzelle. In diesem Fall ist nur eine Interpolation in Nord-Süd-Richtung notwendig.

Es ist wichtig zu betonen, dass das Auftreten der genannten Sonderfälle in der Praxis äußerst selten ist und nahezu gegen Null tendiert. Dies hängt hauptsächlich mit der Koordinatentransformation und der Darstellung von Float-Werten in Python zusammen. Ein Sonderfall tritt nur dann ein, wenn eine Koordinatenkomponente exakt mit der Halbierenden eines Pixels übereinstimmt, was unter realen Bedingungen kaum vorkommt. Sollte die gesamte Nachbarschaft, bspw. in Randbereichen, nicht vollständig vorhanden sein, wird zur Vermeidung fehlerhafter Werte ein Nullwert zurückgegeben.

Die Nachbarschaften der jeweiligen Interpolationsmethode, einschließlich der extrahierten Pixelwerte und der Koordinaten, an denen der interpolierte Pixelwert bestimmt werden soll, bilden die Grundlage für die Subpixelinterpolation. Abhängig vom verwendeten Endpunkt (bilineare oder bikubische Interpolation) wird die entsprechende Funktion aufgerufen, die sowohl die beschriebenen Sonderfälle berücksichtigt als auch die Subpixelinterpolation durchführt. Die Interpolation selbst wird mithilfe des *RegularGridInterpolator* aus der *SciPy*-Bibliothek umgesetzt, da diese die in Kapitel 5.2.2 beschriebenen Interpolationsmethoden bereitstellt. Dabei wird je nach Interpolationsmethode ein passender Interpolator auf Basis der definierten Nachbarschaft initialisiert, welcher zusätzlich die Sonderfälle einbezieht. Anschließend wird der Interpolator genutzt, um den Subpixelwert an der gewünschten Koordinate zu berechnen. Der Quelltext 9 zeigt den relevanten Codeausschnitt für die bikubische Interpolation im Standardfall.

Quelltext 9: *Nutzung des RegularGrindInterpolator für die Subpixelinterpolation*

```
# e_in -- East direction coordinates, representing the positions to the left
    and right of the target pixel
# n_in -- North direction coordinates, representing the positions above and
    below the target pixel
...
# Initialize the cubic interpolator with the provided grid and values
interpolator = RegularGridInterpolator((e_in, n_in), v, method='cubic')
# Define the point of interest (POI) for interpolation using the target
    coordinates
poi = np.array([coords_in[0], coords_in[1]])
...
# Perform cubic interpolation at the specified point of interest
result = interpolator(poi)
```

Das Ergebnis der Subpixelinterpolation umfasst sowohl die interpolierte Bodenbewegungsinformation als auch die zugehörige Standardabweichung an der spezifischen Koordinate. Diese Koordinate wird zusammen mit den entsprechenden Pixelinformationen und der Standardabweichung in einem GeoJSON-Feature gespeichert und in eine Liste eingefügt. Dieses Verfahren wird iterativ für alle Punkte der Eingabe durchgeführt. Abschließend werden alle Punkte im GeoJSON-Format als Feature Collection in eine GeoJSON-Datei überführt und über HTTP zurückgegeben.

Ein derzeitiger Kritikpunkt des Microservices ist, dass die Rückgabe-Koordinaten immer im UTM32-Format vorliegen, unabhängig vom Eingabeformat. Zukünftig könnte die Rückgabe flexibel an den in der Payload angegebenen EPSG-Code angepasst werden, um die Koordinaten im gewünschten bzw. urpsünglichen Referenzsystem bereitzustellen.

7.2.3 Microservice Rasterstatistik

Die Rasterstatistik stellt einen dedizierten Endpunkt (`/getRasterStatistics`) zur Verfügung. Dabei dient, wie beim Microservice der Subpixelinterpolation, die in Kapitel 7.2.1 beschriebene Payload als Eingabe. In diesem Fall wird jedoch eine Feature Collection im GeoJSON-Format mit Polygonen als Daten übergeben. Auch hier können die Koordinaten im WGS84- oder UTM32-Format vorliegen, wobei sie vor der Verarbeitung mittels *pyproj* in das metrische UTM-Koordinatensystem umgewandelt werden.

Ein Ausschnitt des Quelltextes für die Berechnung ist in Quelltext 10 dargestellt. Als Eingabe für die konkrete Berechnung der Rasterstatistik dienen die einzelnen Polygone (Features) der übergebenen Feature Collection. Diese Features werden zunächst in *GeoDataFrame* der Bibliothek *GeoPandas* umgewandelt, da sich Polygone in Python mit dieser Struktur leichter verarbeiten lassen. Für die Berechnungen werden sowohl das COG mit den Bodenbewegungsinformationen (horizontal oder vertikal) als auch das entsprechende COG mit den zugehörigen Standardabweichungen verwendet. Für beide Rasterdaten werden jeweils getrennt statistische Analysen durchgeführt, die anschließend zusammengeführt werden.

Die Berechnung der Rasterstatistiken erfolgt mithilfe der Python-Bibliothek *rasterstats* und der Methode `zonal_stats()`. Zunächst wird die Rasterdatei, welche innerhalb der Payload als Datenquelle angegeben wird, mittels `rasterio.open()` geöffnet. Anschließend wird das gesamte Rasterbild

eingelesen und wichtige Informationen wie NoData-Werte und die Affin-Transformation des Rasters abgerufen. Im nächsten Schritt wird die Methode `zonal_stats()` verwendet, um die gewünschten Rasterstatistiken zu berechnen. Dabei werden die einzelnen Polygone (Features) der Eingangsdaten, das eingelesene Rasterbild sowie die affine Transformation und die NoData-Werte als Parameter übergeben. Zusätzlich wird das Ausgabeformat auf GeoJSON festgelegt und die zu berechnenden statistischen Kennzahlen wie Mittelwert, Minimum und Maximum werden spezifiziert. Die Methode `zonal_stats()` berücksichtigt standardmäßig bei der Rasterisierung alle Zellen, deren Mittelpunkt innerhalb des jeweiligen Polygons liegt, um eine präzise Berechnung sicherzustellen (Perry, 2015).

Quelltext 10: *Berechnung der Rasterstatistik*

```
input_obj = gpd.GeoDataFrame.from_features(polygon['features'])
...
url_v = f's3://{bucket_name}/{file_name}'
...
with rasterio.open(url_v) as dst:
    nodata = src.nodata
    affine = src.transform
    img_v = dst.read(band)
...
stats_v = zonal_stats(vectors=input_obj, raster=img_v, affine=affine,
    nodata=nodata,
                                geojson_out=True, stats="mean min max")
...
```

Nach der Berechnung der Rasterstatistiken für die Bodenbewegung und der zugehörigen Standardabweichung werden die Ergebnisse in ein *GeoDataFrame* zusammengeführt und für die weitere Verarbeitung aufbereitet. Dieses *GeoDataFrame* wird anschließend in das GeoJSON-Format konvertiert, welches als Rückgabe dient. Das GeoJSON enthält die einzelnen Polygone mit den zugehörigen berechneten Statistiken.

Während des Entwicklungsprozesses wurde eine mögliche Optimierung identifiziert, die jedoch nicht implementiert wurde. Diese Verbesserung besteht darin, vor der Berechnung die Bounding Box der gesamten Polygone zu ermitteln und als *Window* zu verwenden. Dadurch könnte vermieden werden, das gesamte Rasterbild einzulesen, was die Effizienz und Performance der Berechnungen weiter steigern würde. Diese Verbesserung bleibt als potenzielles Optimierungspotenzial für zukünftige Implementationen bestehen. Zudem werden die Ausgabekoordinaten, genau wie bei der Subpixelinterpolation, standardmäßig im UTM32-Format bereitgestellt. Zukünftig könnte auch hier die Rückgabe flexibel an den in der Payload angegebenen EPSG-Code angepasst werden.

7.3 Implementierung automatisierte Datenverwaltung

Die in Kapitel 6.3 beschriebene automatisierte Datenverwaltung wird in Form von zwei containerisierten Jobs umgesetzt. Der erste Job wandelt das rasterbasierte Bodenbewegungsmodell, das als herkömmliches GeoTIFF vorliegt, in COG-Dateien um, während der zweite Job die Verdachtsgebiete im GeoJSON-Format in FGB-Dateien umwandelt und zusätzlich mit dem entsprechenden Microservice die Rasterstatistik für die enthaltenen Polygone berechnet. Diese Anwendungen werden innerhalb der IBM CE als ereignisgesteuerte Jobs ausgeführt.

Das Dockerfile ist in Quelltext 11 dargestellt und für beide Jobs identisch aufgebaut. Aus diesem Grund wird im Folgenden lediglich die Umsetzung des Jobs zur Konveriertung von GeoTIFF-Dateien beschrieben. Als Basis dient das GDAL-Image, da GDAL für die Datenkonvertierung innerhalb des Python-Skripts benötigt wird. Im nächsten Schritt wird die Bibliothek *boto3* installiert. Abschließend wird das Python-Skript `convert.py`, welches die Logik der Konvertierung enthält, in das Docker-Image integriert, sodass es beim Start des Containers direkt ausgeführt werden kann.

Quelltext 11: *Dockerfile für Datenkonvertierungs-Jobs*

```
#Use the GDAL image as a base
FROM ghcr.io/osgeo/gdal:ubuntu-small-3.8.5

# Update the package list and install Python 3.10 and pip
RUN apt-get update && \
    apt-get install -y python3.10 python3-pip && \
    rm -rf /var/lib/apt/lists/*

# Install boto3 Python package using pip
RUN pip3 install boto3

# Copy the convert.py script into the container
COPY convert.py /convert.py

# Set the entry point to run the convert.py script using Python 3
ENTRYPOINT ["python3", "/convert.py"]
```

Die erstellten Docker-Images der jeweiligen Jobs werden anschließend in die IBM CR hochgeladen. Basierend auf den Container-Images werden im Anschluss die konkreten Jobs in der IBM CE erstellt. Die Erstellung kann entweder über die grafische Benutzeroberfläche, Terraform oder die zugehörige CLI erfolgen. Im Rahmen der Arbeit wurde die Umsetzung über die IBM-CLI realisiert, wobei zukünftig Bereitstellung und Einrichtung der Jobs durch die Einbettung in Terraform automatisiert werden kann. Der Befehl zur Erstellung des Jobs aus dem Container-Image ist im Quelltext 12 abgebildet. Da keine Angaben zu den Ressourcen gemacht wurden, werden dem Job automatisch die Standardressourcen (eine vCPU und 4 GB RAM) zugewiesen.

Quelltext 12: *Erstellung eines Jobs mit der IBM-CLI*

```
ibmcloud ce job create --name bob-cog-convert --image
   private.de.icr.io/geodserv-student-projects/bob-cog-convert:latest
```

Die Erstellung der Ereignis-Subskription über die IBM CLI ist in Quelltext 13 dargestellt. Dabei wird eine Subskription mit dem Namen *bob-sub-cog* angelegt, die bei einem *write*-Ereignis (Hinzufügen eines Objekts) mit dem Suffix `.tiff` ausgelöst wird. Sobald ein solches Objekt im Bucket *Bucket RAW* gespeichert wird, startet die Subskription automatisch den zugeordneten Job *bob-cog-convert*, um die entsprechende Verarbeitung durchzuführen. Äquivalent dazu wird der Job und die Subskription für die GeoJSON-Konvertierung mit Rasterstatistik erstellt.

Quelltext 13: *Erstellung einer Ereignis-Subskription mit der IBM-CLI*

```
ibmcloud ce subscription cos create --name bob-sub-cog --destination
    bob-cog-convert --bucket Bucket RAW --destination-type job --suffix
    ".tif" --event-type write
```

Nach Auslösung der Ereignis-Subskription werden der Dateiname, der Bucket und weitere Attribute als Umgebungsvariablen an den Job übergeben. Auf Basis der Umgebungsvariablen wird die entsprechende Konvertierung bzw. Berechnung ausgeführt. Die Logik der einzelnen Jobs wird im nachfolgend näher beschrieben.

bob-cog-convert
Der Job *bob-cog-convert* konvertiert GeoTIFF-Dateien in COG, um die Abfrage des rasterbasierten Bodenbewegungsmodells zu optimieren. Ausschnitte aus dem Konvertierungsskript sind in Quelltext 14 dargestellt. Dabei werden die Variablen `input_file` und `source_bucket` aus den Umgebungsvariablen der Ereignis-Subskription geladen. Über die Bibliothek *boto3* wird eine Verbindung zum COS hergestellt, um auf die Objekte zugreifen zu können. Im Anschluss wird über *boto3* die angegebene Datei lokal aus dem *source_bucket* (*Bucket RAW*) geladen und mittels `gdal_translate` in ein COG konvertiert. Anschließend wird die konvertierte Datei mit dem Präfix „COG“ in den Ziel-Bucket hochgeladen und die lokale Datei nach erfolgreichem Hochladen gelöscht.

Quelltext 14: *Automatisierte Konvertierung von GeoTIFF zu COG mittels GDAL*

```
cos = boto3.resource('s3',
                     aws_access_key_id=os.getenv('COS_API_KEY'),
                     aws_secret_access_key=os.getenv('COS_SECRET_KEY'),
                     config=Config(signature_version='s3v4'),
                     endpoint_url=os.getenv('COS_ENDPOINT'))
...
output_key = 'COG_' + input_file
...
bucket = cos.Bucket(source_bucket)
bucket.download_file(input_file, local_input_path)
...
 # Convert GeoTIFF to Cloud-Optimized GeoTIFF using GDAL
subprocess.run(['gdal_translate', '-of', 'COG',
                local_input_path, local_output_path], check=True)
...
bucket = cos.Bucket(destination_bucket)
        bucket.upload_file(local_output_path, output_file)
```

bob-fgb-convert
Der Job *bob-fgb-convert* ist größtenteils identisch zum Job *bob-cog-convert* aufgebaut, erweitert jedoch die Funktionalität der Konvertierung um die Berechnung der Rasterstatistik über den entsprechenden Microservice. Zunächst wird die GeoJSON-Datei heruntergeladen, deren Name in den Umgebungsvariablen steht und von der Ereignis-Subskription gesetzt wurde. Mithilfe der weiteren festgelegten Umgebungsvariablen wird daraus die Payload für die API-Anfrage an den Microservice erstellt, der die Rasterstatistik berechnet. Der entsprechende Quelltextausschnitt ist in Quelltext 15 zu sehen.

Quelltext 15: *Payload-Initialisierung und API-Anfrage zur Berechnung der Rasterstatistiken*

```
with open(local_input_path, 'r') as file:
            geojson_data = json.load(file)
...
# Initialize payload
wrapped_geojson = {"file_name": os.getenv('FILE_VALUE'),
                   "file_name_std": os.getenv('FILE_STD'),
                   "bucket_name": os.getenv('TAR_BUCKET'),
                   "band": os.getenv('RASTER_BAND'),
                   "data": geojson_data}
api_url = os.getenv('API_URL')
...
response = requests.put(api_url, json=wrapped_geojson)
with open(temp_geojson_path, 'w') as temp_file:
                json.dump(response_data, temp_file)
...
```

Das resultierende GeoJSON mit den berechneten Statistiken wird anschließend mit dem GDAL-Tool *ogr2ogr*, das zur Konvertierung von Dateiformaten dient, in das FGB-Format umgewandelt (siehe Quelltext 16). Zusätzlich wird das Ausgabekoordinatensystem auf WGS84 festgelegt, um die Datei für die Visualisierung in Kapitel 7.4 vorzubereiten. Abschließend wird die konvertierte Datei mit dem Präfix „FGB“ in den Bucket *Bucket PROD* hochgeladen und der Container wird von den temporären Dateien bereinigt.

Quelltext 16: *Umwandlung des GeoJSON in FGB mittels GDAL*

```
subprocess.run(['ogr2ogr', '-f', 'FlatGeobuf', local_output_path,
    temp_geojson_path, '-t_srs', 'EPSG:4326'], check=True)
...
destination_bucket.upload_file(local_output_path, fgb_output_key)
```

Anzumerken ist, dass aufgrund der prototypischen Umsetzung keine umfassende Fehlerbehandlung in den Jobs implementiert wurde. So wird bspw. beim Job *bob-fgb-convert* keine Validierung der Geometrie durchgeführt. Dadurch könnte es passieren, dass auch eine GeoJSON-Datei mit Punktfeatures verarbeitet wird, für die jedoch keine Rasterstatistiken berechnet werden können. Zukünftig sollte eine ausführliche Fehlerbetrachtung implementiert werden, die sicherstellt, dass potenzielle Fehlerquellen umfassend erfasst und abgefangen werden.

7.4 Implementierung Frontend

Da das Frontend nicht den Schwerpunkt dieser Arbeit bildet, wird dessen Umsetzung hier nur knapp beschrieben. Für die Entwicklung des Frontends wird das moderne Build-Tool *Vite.js* verwendet. Durch den Einsatz von nativen ES-Modulen im Browser bietet *Vite.js* eine äußerst performante Entwicklungsumgebung (Vite, o.J.). Dies ermöglicht eine schnelle und effiziente Erstellung prototypischer Frontends, was es besonders geeignet für die Implementierung des Frontends der BOB.NI-WebApp macht. Die Anwendung läuft in einem Container innerhalb der IBM CE, wodurch während der Entwicklungsphase kein separater Webserver erforderlich ist. Als Kartenbasis dient *MapLibre*, welches sowohl zur Visualisierung der Vector-Tiles als auch der Verdachtsgebiete verwendet wird. Hierbei

werden die eindeutigen URLs der Datensätze genutzt, um diese direkt aus dem Bucket *Bucket PROD* zu laden. Das Initialisieren der Karte sowie Laden der Vector-Tiles und Verdachtsgebiete im GeoJSON-Format ist in Quelltext 17 dargestellt.

Quelltext 17: *Einladen der Verdachtsgebiete aus dem COS*

```
...
// Initialize the map
var map = new maplibregl.Map({
  container: "map",
  style:
    "https://geodserv-vector-tiles-dev.s3.eu-de.cloud-object-storage.
     appdomain.cloud/bavi-sn.json",
  center: [9, 52.6],
  zoom: 7,
});

var verdachtsgebiete_url =
"https://geodserv-cog.s3.eu-de.cloud-object-storage.
 appdomain.cloud/GeoJSON_Verdachtsgebiete.geojson";

// Add GeoJSON data source
map.addSource("verdachtsgebiete_src", {
    type: "geojson",
    data: verdachtsgebiete_url,
});
...
```

Zwar ist aufgrund der Ergebnisse der Analyse aus Kapitel 5.3.2 vorgesehen, dass für die Verdachtsgebiete das Format *FGB* verwendet wird, allerdings ist *MapLibre* nicht nativ mit dem Format kompatibel. Ein dafür vorgesehenes Plugin (*mapbox-gl-flatgeobuf*) funktioniert ebenfalls nicht. Aus diesem Grund wurde das Tool *bob-fgb-convert* angepasst, um die Verdachtsgebiete samt Statistiken neben FGB auch als GeoJSON exportieren zu können. Für die Visualisierung in MapLibre ist es notwendig, dass die Daten im WGS84-Koordinatensystem vorliegen, weshalb dieses für den Export verwendet wird. Nach dem erfolgreichen Laden der GeoJSON-Daten werden diese mit einer festgelegten Stildefinition visualisiert. Es ist jedoch davon auszugehen, dass das neue FGB-Format in Zukunft eine bessere Kompatibilität mit MapLibre und auch anderen Bibliotheken bieten wird, wodurch der Export in GeoJSON möglicherweise nicht mehr notwendig sein wird. Die Benutzeroberfläche des Frontends ist in Abbildung 28 dargestellt. Weitere Abbildung sind in Anhang D zu finden.

Als MVP bietet das Frontend lediglich die beschriebenen Funktionalitäten der Berechnung von Profillinien sowie der Berechnung von Rasterstatistiken. Diese Funktionen basieren auf den vordefinierten Tools von *MapLibre* zum Zeichnen von Linienzügen und Polygonen. Die Erstellung von Bodenbewegungsprofilen erfolgt durch das Zeichnen einer Linie, wobei mehrere Stützpunkte gesetzt werden können. Zwischen dem Anfangs- und Endpunkt der Linie werden mithilfe der Bibliothek *turf.js* 900 Punkte zum Abfragen der Bodenbewegungsdaten berechnet. Basierend auf den Koordinaten der Punkte und den weiteren Parametern wird die in Kapitel 7.2.1 beschriebene Payload erstellt und an die Microservice übermittelt. Sobald die berechneten Werte im Frontend verfügbar sind, erscheint eine Sidebar, in der die Ergebnisse mithilfe von *chart.js* visualisiert werden.

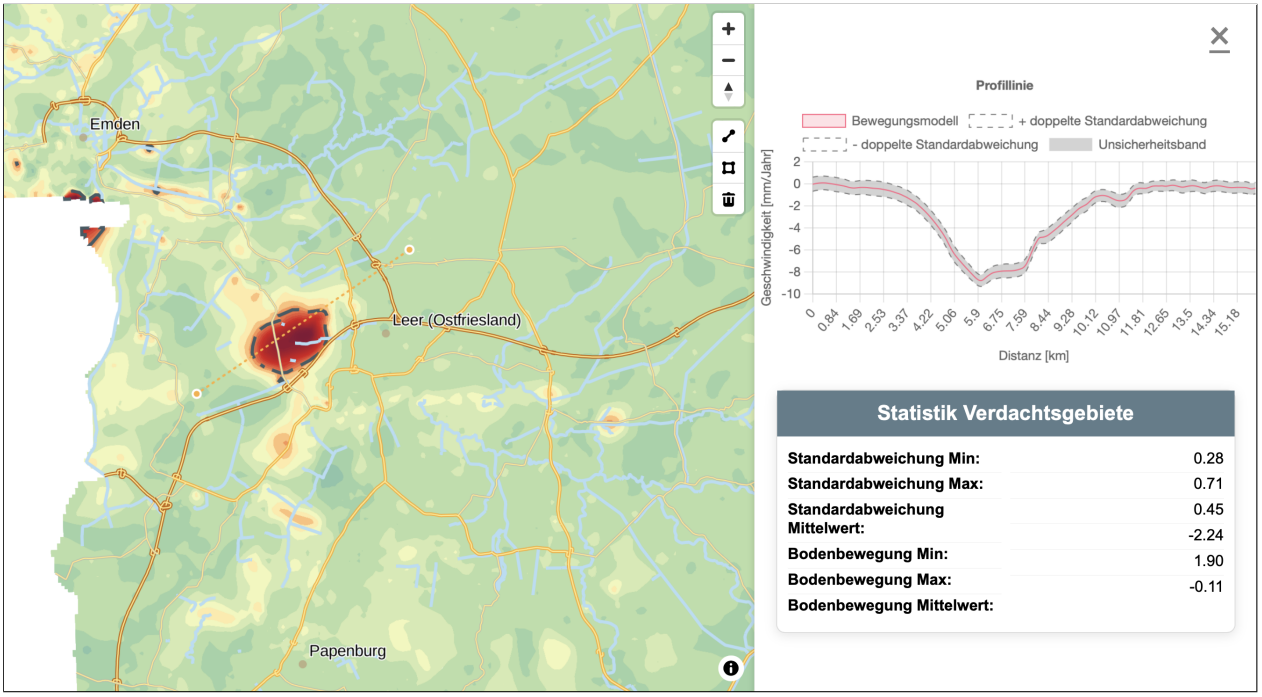


Abbildung 28: *Umgesetztes Frontend zur Datenvisualisierung und Datenabfrage*

Dadurch, dass für jeden Punkt sowohl die Bodenbewegungsinformation als auch die dazugehörige Standardabweichung durch den entsprechenden Microservice abgefragt werden, wird ein zusätzliches Unsicherheitsband der Bodenbewegungen berechnet. Dieses entsteht, indem die doppelte positive und negative Standardabweichung auf die jeweiligen Werte angewendet wird. Das Ergebnis der Visualisierung eines Bodenbewegungsprofils ist in der Abbildung 28 oben rechts, sowie in Anhang D2 dargestellt.

Bei der Berechnung der Rasterstatistik wird das vom Benutzer mit dem *MapLibre*-Polygon-Tool gezeichnete Polygon in das GeoJSON-Format umgewandelt. Dieses GeoJSON wird, ähnlich wie bei der Erstellung von Profillinien, zusammen mit weiteren Parametern in die Payload integriert und an den Microservice zur Berechnung der Rasterstatistik übermittelt. Die Ergebnisse werden anschließend in einer Tabelle in der Sidebar dargestellt, wie in Abbildung 28 unterhalb des Bodenbewegungsprofils und im Anhang D3 ersichtlich. Die Rasterstatistiken der Verdachtsgebiete lassen sich durch einfaches Anklicken innerhalb des Verdachtsgebiets über ein Pop-up visualisieren (siehe Anhang D4).

Grundsätzlich ist zu erwähnen, dass das Frontend bewusst sehr einfach gehalten wurde, um primär einen Überblick über die Implementierung der Funktionalitäten und deren Visualisierung zu geben. Es verfügt bspw. über keine Navigationsleiste oder ähnliche Benutzeroberflächenelemente. Zudem wurde bislang keine Option zur Auswahl der Interpolationsmethode integriert, mit der der Benutzer zwischen bilinearer und bikubischer Interpolation bei der Abfrage der Bodenbewegungen wählen könnte. Diese Funktion lässt sich jedoch in Zukunft leicht ergänzen, da lediglich der Endpunkt des Microservices angepasst werden muss.

8 Evaluierung

Das folgende Kapitel widmet sich der Evaluierung der implementierten Cloud-nativen Architektur. Im Mittelpunkt steht die Überprüfung der in Kapitel 4 definierten Anforderungen. Darüber hinaus werden die Anforderungen an Leistungsfähigkeit und Zuverlässigkeit gemäß ISO/IEC 25010 gesondert betrachtet und durch entsprechende Untersuchungen validiert. Abschließend erfolgt eine zusammenfassende Bewertung der Ergebnisse.

8.1 Umsetzung der Anforderungen

Die Umsetzung der Anforderungen wird separat für die funktionalen Anforderungen, die spezifischen Qualitätsanforderungen an die BOB.NI sowie die allgemeinen Qualitätsanforderungen gemäß ISO/IEC 25010 überprüft.

Funktionale Anforderungen
Bei der Überprüfung der Umsetzung der funktionalen Anforderungen zeigt sich, dass die Anforderungen *F1* und *F4* erfolgreich durch den Microservice für die Rasterstatistik und die Visualisierung der Verdachtsgebiete umgesetzt wurden. Eine Besonderheit liegt darin, dass dank der generischen Schnittstelle derselbe Microservice sowohl in der automatisierten Datenvorverarbeitung zur Berechnung der Statistik für die Verdachtsgebiete, als auch für benutzerdefinierte Polygone eingesetzt werden kann, wodurch der Entwicklungsaufwand erheblich reduziert wurde. Darüber hinaus wurde die Anforderung *F3* durch den Microservice für Subpixelinterpolation vollständig erfüllt. Dieser ermöglicht sowohl die präzise Erstellung von Bodenbewegungsprofilen als auch die genaue Abfrage einzelner oder mehrerer Bodenbewegungen an spezifischen Koordinaten, wodurch er vielseitig einsetzbar ist. Allerdings muss für den produktiven Einsatz bei beiden Microservices noch eine verbesserte Fehlerbehandlung implementiert werden.

Nur die Anforderung *F2* aus dem Zukunftskonzept VKV2025 konnte nicht umgesetzt werden, da die notwendigen zeitlichen Daten nicht vorliegen. Dies liegt daran, dass beim rasterbasierten Bodenbewegungsmodell ausschließlich die gemittelte Bodenbewegungsgeschwindigkeit pro Jahr für den gesamten Beobachtungszeitraum berechnet wird, wodurch die zusätzliche zeitliche Komponente verloren geht. Aufgrund der Tatsache, dass für jeden PSI-Punkt eine entsprechende Modellfunktion (siehe Kapitel 3.3) berechnet wird, stehen die zeitlichen Informationen jedoch zur Verfügung. Aus diesen Daten könnten zukünftig rasterbasierte Bodenbewegungsmodelle in unterschiedlichen zeitlichen Intervallen abgeleitet werden, um die zeitliche Variation der Bodenbewegungen mit einfließen zu lassen. Entsprechende Funktionalitäten zur Abfrage sowie Visualisierung müssten nach dieser Umsetzung entwickelt werden. Diese Daten bieten Potenzial für zukünftige Erweiterungen und die weitere Detaillierung der Bodenbewegungen.

Spezifische Qualitätsanforderungen
Die spezifischen Qualitätsanforderungen umfassen die Anforderungen *Q1* bis *Q8*, welche verschiedene Vorgaben an die Architektur des BOB.NI aus unterschiedlichen Quellen stellen. Diese wurden größtenteils bereits vollständig umgesetzt. Die Software wurde zunächst als MVP entwickelt, was bedeutet, dass sie momentan als Minimalprodukt vorliegt. Dank der Umsetzung der Softwarearchitektur als flexible Microservice-Architektur ist eine künftige Erweiterung problemlos möglich. Dadurch konnten die Anforderungen *Q1* und *Q5* vollständig erfüllt werden. Zusätzlich verfügen die Microservices über generische REST-APIs, was zur Erfüllung der Anforderung *Q8* beiträgt.

Durch die automatisierte Datenverwaltung wurde auch die Anforderung *Q7* vollständig erfüllt, während die Anforderung *Q4* teilweise umgesetzt ist. Anforderung *Q4* fordert die automatische und schnelle Integration neuer Daten. Derzeit bezieht sich dies jedoch lediglich auf neue rasterbasierte Bodenbewegungsmodelle, die aktuell noch manuell erzeugt werden müssen. Die entwickelte Infrastruktur ermöglicht es zukünftig, die gesamte Prozesskette vollständig zu automatisieren, sodass neue InSAR-Daten vom BGR direkt verarbeitet und als Grundlage für den BOB.NI genutzt werden können. Damit wäre auch diese Anforderung in vollem Umfang erfüllt. Durch die Nutzung von COS in Kombination mit den Cloud-nativen Geodatenformaten konnte eine kostengünstige, moderne Alternative zur Datenspeicherung realisiert werden. Zudem werden die Betriebskosten der Architektur durch den sparsamen Einsatz von Ressourcen für die Microservices und Jobs gering gehalten. Eine genaue Berechnung der Kosten ist jedoch schwierig, da dies in einer Cloud-Umgebung aufgrund dynamischer Preisgestaltung und Ressourcenverteilung komplex ist. Es ist jedoch davon auszugehen, dass die Kosten deutlich geringer ausfallen als bei der Nutzung der virtuellen Maschine des Prototyps. Dadurch ist die Anforderung *Q2* erfüllt, allerdings können sich die geringen Ressourcen auf die Anforderungen der Leistungsfähigkeit und Zuverlässigkeit auswirken, was in Kapitel 8.2 näher betrachtet wird. Die Anforderung nach hoher Datenqualität (*Q3*) gilt als teilweise erfüllt, da präzise Funktionalitäten zur Abfrage der Bodenbewegungsdaten implementiert wurden und ein Kompromiss zwischen Genauigkeit, Effizienz und Speicherkapazität für die Auflösung der Datengrundlage identifiziert wurde. Um diese Anforderung jedoch vollständig zu erfüllen, muss zukünftig ein fundierter Ansatz zur Erzeugung von Zwischenpunkten bei der Erstellung von Profillinien implementiert werden. Dieser sollte sich am Abtasttheorem orientieren und die Anzahl der Punkte dynamisch anhand der Auflösung und der Länge der Strecke festlegen, anstatt pauschal eine feste Anzahl von Zwischenpunkten zu erzeugen.

Die Anforderung *Q6* wurde nicht umgesetzt, da derzeit alle Komponenten eigenständig entwickelt wurden. In Kapitel 6.4.2 wird erläutert, dass in Zukunft das Masterportal als Frontend fungieren könnte, jedoch hätte die Implementierung dieser Funktionalität den Rahmen der Arbeit überschritten.

Allgemeine Qualitätsanforderungen der ISO/IEC 25010

Die allgemeinen Qualitätsanforderungen umfassen die Anforderungen *Q10* bis *Q14*, welche sich auf die ISO/IEC 25010 beziehen. Diese Norm legt allgemeine Kriterien für qualitative Software fest. Die Anforderungen aus der ISO/IEC 25010 werden gesondert betrachtet, da sie allgemeiner Natur sind und nicht speziell auf den BOB.NI zugeschnitten sind.

Die Anforderung der Kompatibilität (*Q10*) wird durch den Einsatz von Docker-Containern und generischen Schnittstellen vollständig erfüllt. Die generischen Schnittstellen ermöglichen eine standardisierte Kommunikation zwischen den verschiedenen Komponenten der Software, was die Integration in bestehende Umgebungen und die Erweiterbarkeit erheblich erleichtert. Hinsichtlich der Sicherheit (*Q12*) wurde im Rahmen der bisherigen Arbeit festgestellt, dass diese noch nicht umfassend genug für einen produktiven Einsatz untersucht und implementiert wurde. Bisher wurden lediglich grundlegende Sicherheitsmechanismen umgesetzt, wie die Verwendung von Access-Keys beim Zugriff auf den COS oder die Überprüfung der verwendeten Bibliotheken auf bekannte Sicherheitslücken. Wichtige Sicherheitskonzepte, wie das „Shared-Nothing“-Prinzip, wurden hingegen bislang nicht ausreichend berücksichtigt. Für den Produktivgang der Anwendung ist es daher unerlässlich, diese Sicherheitsaspekte weiter zu analysieren und entsprechende Maßnahmen umzusetzen. Die Wartbarkeit (*Q13*) der Anwendung ist derzeit nur eingeschränkt gegeben. Zwar können die Versionen der Microservices manuell im IBM CR aktualisiert werden, jedoch sollte dieser Prozess zukünftig auto-

matisiert ablaufen. Eine Automatisierung könnte durch den Einsatz einer CI/CD-Pipeline erreicht werden, was die Wartung erleichtert und gleichzeitig die Zuverlässigkeit und Konsistenz bei Updates sicherstellt. Die Übertragbarkeit (*Q14*) der Software wird grundsätzlich durch die Verwendung von IaC gewährleistet, was eine einfache Migration auf verschiedene Hardware- und Softwareplattformen ermöglicht. Allerdings werden in der aktuellen Architektur spezifische IBM-Komponenten wie die IBM CE genutzt, die bei anderen Cloud-Anbietern unterschiedlich implementiert sein könnten. Dies könnte bei einer Migration auf alternative Cloud-Umgebungen zu Herausforderungen führen, weshalb eine detaillierte Analyse der Zielplattform erforderlich wäre, um die Übertragbarkeit vollständig sicherzustellen. Die Anforderungen an Zuverlässigkeit und Leistungsfähigkeit können nicht allein durch den Einsatz einer bestimmten Technologie als erfüllt angesehen werden. Diese Aspekte werden daher im folgenden Kapitel 8.2 genauer untersucht.

8.2 Zuverlässigkeit und Leistungsfähigkeit

Die Umsetzung der Anforderungen an Zuverlässigkeit (*Q11*) und Leistungsfähigkeit (*Q9*) wird im Folgenden näher betrachtet. Dies ist notwendig, da deren Erfüllung nicht ausschließlich auf theoretischen Annahmen oder der bloßen Auswahl einer bestimmten Technologie basieren kann. Stattdessen müssen diese Anforderungen durch Experimente unter realistischen Lastbedingungen überprüft und validiert werden. Um diese Untersuchungen durchzuführen, wurde das Tool *Grafana k6* verwendet. *Grafana k6* ist ein Open-Source-Tool zur Durchführung von Performancetests. Es ermöglicht, die Leistung und Belastbarkeit von Anwendungen unter realen Bedingungen zu testen (Grafana Labs, 2024). Dabei können Benutzer simuliert werden, welche gleichzeitig innerhalb einer bestimmten Zeitspanne auf entsprechende Ressourcen zugreifen.

Im Folgenden wird *Grafana k6* verwendet, um Performancetests für die implementierten Microservices unter verschiedenen Bedingungen durchzuführen. Die Jobs der automatisierten Datenverwaltung werden nicht betrachtet, da hier die Leistungsfähigkeit nicht von Bedeutung ist und die Zuverlässigkeit aufgrund des geringen Rechenaufwandes gegeben ist. Ziel ist es, zu überprüfen, ob die Anforderungen *Q11* und *Q9* erfüllt werden. Dazu wird der gleichzeitige Zugriff von 10, 20 und 50 Personen innerhalb eines Zeitraums von 100 Sekunden auf die Microservices simuliert. Diese sind, wie in Kapitel 7.2.1 beschrieben, standardmäßig mit jeweils einer vCPU und 4 GB RAM ausgestattet. Dabei werden die Laufzeiten sowie die Anzahl an erfolgreichen und fehlgeschlagenen Anfragen (Checks) gemessen, um die Leistungsfähigkeit und Zuverlässigkeit zu bewerten.

Die Tabelle 10 zeigt die Ergebnisse des Performancetests für den Microservice der Subpixelinterpolation. Hierbei wurde die bilineare Interpolation mit einem beispielhaften Bodenbewegungsprofil gewählt. Bei 10 gleichzeitigen Benutzern sind die Laufzeiten mit einer durchschnittlichen Dauer von 2,6 Sekunden und einer Erfolgsrate von 100 % bei den Checks durchweg stabil, was auf eine gute Leistungsfähigkeit hindeutet. Bei 20 Benutzern steigt die durchschnittliche Laufzeit auf 9,59 Sekunden und die Erfolgsquote der Checks sinkt auf 97,27 %, da bei 3 von 107 Anfragen die Zeitüberschreitung von 30 Sekunden überschritten wurde, wodurch die Zuverlässigkeit der Anwendung leicht sinkt. Bei 50 gleichzeitigen Benutzern erreicht die durchschnittliche Dauer 24,12 Sekunden und die Checks fallen auf nur 73,75%. Dies verdeutlicht, dass das System ab dieser Lastgrenze deutlich an Zuverlässigkeit und Leistung verliert, was bereits bei 20 Benutzern spürbar ist. Aufgrund des höheren Rechenaufwands der bikubischen Interpolation ist davon auszugehen, dass die Systemressourcen bereits bei einer geringeren Benutzeranzahl an ihre Leistungsgrenze stoßen würden.

Benutzer	Dauer	min	max	Checks
10	2,60 s	0,99 s	10,58 s	100,00 %
20	9,59 s	0,16 s	29,46 s	97,27 %
50	24,12 s	1,12 s	60,00 s	73,75 %

Tabelle 10: *Zusammenfassung des Performancetests für den Microservice der Subpixelinterpolation*

Ein ähnliches Ergebnis zeichnet sich bei dem Performancetest der Rasterstatistik ab. Hier wurde ein beispielhaftes Polygon mit einer Größe von rund 800 km^2 als Payload verwendet. Bei 10 Benutzern bleibt die Performance mit einer durchschnittlichen Laufzeit von 3,64 Sekunden und einer Erfolgsrate der Checks von 100,00 % stabil. Bei 20 Benutzern erhöht sich die durchschnittliche Dauer auf 13,07 Sekunden, und die Checks sinken auf 93,68 %, was auf beginnende Performanceprobleme hinweist. Bei 50 Benutzern verschlechtert sich die Situation deutlich, mit einer durchschnittlichen Laufzeit von 23,80 Sekunden und einer Erfolgsrate von nur noch 70,07 %.

Benutzer	Dauer	min	max	Checks
10	3,64 s	1,02 s	26,71 s	100,00 %
20	13,07 s	0,18 s	59,90 s	93,68 %
50	23,80 s	1,30 s	60,00 s	70,07 %

Tabelle 11: *Zusammenfassung des Performancetests für den Microservice der Rasterstatistik*

Zusammenfassend lässt sich sagen, dass die Ressourcen von einer vCPU und 4 GB RAM für den Betrieb mit wenigen Nutzern (rund 10 gleichzeitig) ausreichend sind. Bei steigenden Nutzerzahlen sind die definierten Ressourcen jedoch nicht mehr ausreichend, um eine leistungsfähige und zuverlässige Anwendung zu gewährleisten. Um die Ressourcen der einzelnen Instanz zu entlasten, steht eine automatisierte horizontale Skalierung zur Verfügung. Diese greift jedoch erst bei 100 gleichzeitigen Anfragen, wofür die bestehenden Ressourcen einer Instanz, wie die Ergebnisse der Performancetests zeigen, nicht ausreichend sind. Um eine optimale Ressourcenverteilung zu gewährleisten, müssen zukünftig sowohl die Rechenressourcen pro Nutzer als auch die durchschnittlichen Nutzerzahlen ermittelt werden. Auf dieser Grundlage sollten die Ressourcen einer Instanz sowie die Parameter für die horizontale Skalierung entsprechend angepasst werden. Trotz der Limitierung der Ressourcen auf wenige Benutzer können die Anforderungen an Leistungsfähigkeit (*Q9*) und Zuverlässigkeit (*Q11*) als erfüllt betrachtet werden, da die Cloud-Infrastruktur eine einfache und schnelle Anpassung der Ressourcen durch On-demand self-service ermöglicht. Die Tests haben zudem gezeigt, dass bei fehlerhaften Container-Instanzen ein automatisierter Neustart der Instanz erfolgt, was die Ausfallsicherheit und Verfügbarkeit der Anwendung zusätzlich gewährleistet.

8.3 Bewertung

Die vorgestellte Cloud-native Architektur zur Verwaltung und Verarbeitung rasterbasierter Bodenbewegungsdaten stellt eine moderne und zukunftsfähige Lösung dar, die sich an dem neuesten Stand der Technik und etablierten Cloud-native Technologien orientiert. Ein Großteil der aufgestellten Anforderungen wurde bereits erfolgreich umgesetzt. Bei den nicht oder nicht vollständig erfüllten Anforderungen wurden die zugrunde liegenden Probleme identifiziert und konkrete Handlungsempfehlungen dargelegt. Die Architektur wurde primär für den BOB.NI entwickelt, doch aufgrund ihrer generischen Gestaltung lässt sie sich auch auf andere Anwendungen im Bereich der rasterbasierten Bodenbewegungsdaten übertragen. Sowohl die Speicherung als auch die Abfrage der Daten sind so

flexibel gestaltet, dass das Konzept über den BOB.NI hinaus Anwendung finden kann, was einen wesentlichen Vorteil hinsichtlich der zukünftigen Nutzung in anderen Bereichen darstellt.

Besonders positiv hervorzuheben ist die entwickelte Infrastruktur basierend auf einem COS und der IBM CE. Diese Kombination aus Datenspeicher und serverloser Plattform ermöglicht eine vollautomatisierte Datenverwaltung über Jobs und bietet zusätzlich die Flexibilität, die Microservices für die Abfrage der Bodenbewegungen effizient zu betreiben. Die Wahl der Microservice-Architektur erweist sich dabei als strategisch sinnvoll, da sie die Erweiterbarkeit der BOB.NI-WebApp erheblich vereinfacht. Ein weiterer Vorteil ist, dass die Microservices unabhängig voneinander betrieben werden, wodurch sie individuell skaliert und gewartet werden können, ohne die Gesamtarchitektur zu beeinträchtigen. Durch die Verwendung generischer REST-APIs können diese Dienste zudem problemlos in anderen rasterbasierten Anwendungen integriert werden, was ihre Flexibilität und Wiederverwendbarkeit weiter steigert. Auch der Einsatz von IBM COS und Cloud-optimierten Geodatenformaten erweist sich als äußerst vorteilhaft, da dadurch entweder die Kosten für einen zusätzlichen Datenbankdienst oder der hohe Implementierungsaufwand reduziert und gleichzeitig die Effizienz bei der Abfrage von rasterbasierten Bodenbewegungsdaten deutlich verbessert wird. Zudem bietet der IBM COS eine optimale Grundlage für eine automatisierte Datenverwaltung, da dieser Ereignis-Subskriptionen unterstützt, wodurch Prozesse automatisch ausgelöst und verwaltet werden können. Erste Implementierungen zur Automatisierung wurden bereits erfolgreich durchgeführt, was die Möglichkeiten der automatisierten Datenverwaltung über diese Architektur unterstreicht. Ein weiterer Ausbau, etwa zur Erweiterung der Prozesskette für die flächenhafte Erzeugung eines Bodenbewegungsmodells, ist bereits in Planung und könnte in die bestehende Infrastruktur integriert werden.

Obwohl es sich bei der vorliegenden Implementierung um einen Prototyp handelt, der in seiner aktuellen Form noch nicht für den produktiven Einsatz geeignet ist, gibt es dennoch zentrale Aspekte, die kritisch betrachtet werden müssen. Besonders hervorzuheben ist das Fehlen einer durchgängigen CI/CD-Pipeline, welche die Effizienz der Entwicklungs- und Bereitstellungsprozesse sowohl für die Software als auch für die Infrastruktur erheblich beeinträchtigt. Eine vollständige Implementierung dieser Pipeline wäre entscheidend, um einen umfassenden DevSecOps-Ansatz zu ermöglichen, der nicht nur die kontinuierliche Bereitstellung und Verbesserung der Software sicherstellt, sondern auch die Sicherheit über den gesamten Lebenszyklus hinweg gewährleistet. Ein weiterer Schwachpunkt ist das Fehlen einer umfassenden Sicherheitsanalyse, was ein potenzielles Risiko für die Cloud-native Architektur darstellt. Eine gründliche Sicherheitsüberprüfung und die Implementierung geeigneter Schutzmaßnahmen sind daher unerlässlich. Zusätzlich besteht noch Verbesserungspotenzial in der Fehlerbehandlung der einzelnen Jobs und Microservices, um die Zuverlässigkeit der Lösung im laufenden Betrieb weiter zu erhöhen. Auch die Ressourcenoptimierung der Microservice-Instanzen ist ein Bereich, der noch verbessert werden muss, wie in Kapitel 8.2 aufgezeigt wurde. Um eine optimale Leistung sicherzustellen, sollten in Zukunft genaue Werte für die Ressourcenzuweisung pro Instanz ermittelt werden, basierend auf der Anzahl der Nutzer. Ebenso sollte der Schwellenwert für gleichzeitige Zugriffe so festgelegt werden, dass frühzeitig eine horizontale Skalierung eingeleitet wird. Dies würde eine effektive Lastverteilung gewährleisten und Leistungseinbußen vorbeugen.

9 Fazit und Ausblick

Im Rahmen dieser Masterarbeit wurde eine progressive Cloud-native Architektur entwickelt, die eine effiziente Verwaltung und Verarbeitung rasterbasierter Bodenbewegungsdaten ermöglicht und erfolgreich in der IBM-Cloud implementiert wurde. Obwohl die Architektur primär auf den BOB.NI ausgerichtet ist, zeichnet sie sich durch ihre generischen Komponenten aus, die auch für die Verarbeitung anderer rasterbasierter Daten oder in ähnlichen Anwendungsbereichen genutzt werden können. Darüber hinaus wurde die Architektur so konzipiert, dass sie vollständig den Vorgaben der IT-Strategie des LGLN entspricht und sich an den Qualitätsmerkmalen der ISO/IEC 25010 orientiert, was ihre Qualität und Zukunftsfähigkeit unterstreicht. Durch die Integration moderner Cloud-nativer Technologien und die flexible Struktur bietet die Architektur eine skalierbare und anpassungsfähige Lösung, die langfristig einen nachhaltigen und effizienten Betrieb gewährleistet.

Die IST-Analyse hat gezeigt, dass in Niedersachsen kein dedizierter Bodenbewegungsdienst existiert, der flächendeckende Informationen bereitstellt. Um diese Lücke zu schließen, wurde ein Prototyp entwickelt, welcher jedoch nicht der IT-Strategie des LGLN entspricht. Dadurch ist eine Cloud-native Neuentwicklung unumgänglich. Auf Grundlage dieser Erkenntnis wurde eine umfassende Anforderungsanalyse durchgeführt, die auf den Anforderungen der Stakeholder, verschiedenen Dokumenten sowie bestehenden Systemen basiert und dabei unterschiedliche funktionale und Qualitätsanforderungen für die Neuentwicklung identifizierte. Neben der Anforderungsanalyse wurden unterschiedliche Komponenten analysiert, die für die Entwicklung von zentraler Bedeutung sind. Dabei wurden der entwickelte Prototyp sowie die Funktionalitäten zur effizienten Datenabfrage und die Eignung von Cloud-optimierten Geodatenformaten als zentrales Speicherformat untersucht. Es zeigte sich, dass der Prototyp nicht nur den strategischen Anforderungen des LGLN widerspricht, sondern auch wesentliche Qualitätsmerkmale nach ISO/IEC 25010 nicht erfüllte. Zusätzlich wurde festgestellt, dass der Prototyp einen hohen manuellen Aufwand für die Datenvorverarbeitung erfordert und eine Automatisierung aufgrund der monolithischen Struktur nur erschwert möglich ist. Für die präzise und effiziente Abfrage der Bodenbewegungsdaten wurden die Funktionalitäten der Subpixelinterpolation und Rasterstatistik als zentrale Komponenten identifiziert, die als Microservices implementiert werden sollen. Dadurch können sie unabhängig voneinander skaliert werden und lassen sich leicht in anderen Kontexten wiederverwenden. Zudem wurde die optimale Auflösung der rasterbasierten Bodenbewegungsdaten für diese Funktionalitäten ermittelt, um ein ideales Gleichgewicht zwischen Speicherplatz, Genauigkeit und Effizienz sicherzustellen. Abschließend wurde die Eignung von Cloud-optimierten Geodatenformaten in Kombination mit einem COS als zentrale Speicherlösung für die Bodenbewegungsdaten ermittelt, da diese Kombination kostengünstiger, effizienter und einfacher in der Handhabung ist als herkömmliche Datenbanken wie PostgreSQL.

Die Erkenntnisse aus der Analyse und die aufgestellten Anforderungen bildeten die Grundlage für das Konzept und die entwickelte Architektur sowie deren prototypische Umsetzung, die in dieser Arbeit als Zusammenspiel von Infrastruktur, Softwarearchitektur und notwendiger Datenvorverarbeitung verstanden wird. Dabei wurde ein ganzheitliches Architekturkonzept für BOB.NI entwickelt, das sowohl die vollständige Datenvorverarbeitung als auch die Bereitstellungskomponente der BOB.NI-WebApp integriert. Zur Umsetzung dieser Architektur wurde in der IBM-Cloud eine Infrastruktur aufgebaut, die sowohl die automatisierte Datenverwaltung als auch die notwendige Softwarearchitektur unterstützt. Wesentliche Bestandteile dieser Lösung sind ein COS mit mehreren Buckets zur Speicherung der Cloud-optimierten Geodatenformate sowie die IBM CE, eine serverlose Plattform, die sowohl für die Ausführung der einzelnen Jobs der automatisierten Datenverwaltung als

auch als Grundlage für die Microservice-Architektur der BOB.NI-WebApp dient. Die Microservices sind so konzipiert, dass sie REST-API-Spezifikationen erfüllen, was eine effiziente und flexible Kommunikation sowie eine nahtlose Integration zwischen den verschiedenen Komponenten ermöglicht. Durch den Einsatz bewährter Cloud-nativer Technologien wird zudem eine hohe Zuverlässigkeit, Skalierbarkeit und Flexibilität der Lösung sichergestellt.

Nach der erfolgreichen Umsetzung dieser Architektur, wurde eine umfassende Evaluierung durchgeführt, um die Erfüllung der gestellten Anforderungen zu überprüfen und Optimierungspotenziale zu identifizieren. Auf Grundlage dieser Evaluierung lässt sich feststellen, dass fast alle Anforderungen erfüllt wurden und für nicht vollständig erfüllte Anforderungen eine Umsetzungsoption dargelegt wurde. Im Vergleich zum ursprünglichen Prototyp stellt die neue Lösung eine deutlich modernere und flexiblere Alternative dar, die sich leichter erweitern und an neue Anforderungen anpassen lässt. Das automatisierte Speicherkonzept, das auf Cloud-optimierte Geodatenformate setzt, sowie die generische Abfragestruktur der Bodenbewegungsdaten steigern nicht nur die Effizienz des Systems erheblich, sondern tragen zugleich maßgeblich zur Senkung der Betriebskosten bei. Zusätzlich zu den spezifischen Anforderungen des BOB.NI wurden bei der Entwicklung auch standardisierte Anforderungen an hochwertige Software berücksichtigt, die sich an der ISO/IEC 25010 orientieren. Durch die konsequente Umsetzung dieser Anforderungen wurde sichergestellt, dass die entwickelte Architektur nicht nur den spezifischen Anforderungen des BOB.NI entspricht, sondern auch den allgemeinen Standards für hochwertige Software gerecht wird. Gleichzeitig gibt es jedoch noch einige Bereiche, die optimiert werden müssen, insbesondere in Bezug auf die effiziente Ressourcennutzung, die Implementierung einer CI/CD-Pipeline sowie eine umfassende Sicherheitsanalyse. Darüber hinaus bedarf es eines fundierten Ansatzes zur Erzeugung von Zwischenpunkten bei der Erstellung von Profillinien, um eine präzise Darstellung zu gewährleisten. Auch die Umsetzung des Frontends ist derzeit noch unklar. Es muss entschieden werden, ob eine Integration in das Masterportal erfolgt oder ob eine eigenständige Lösung entwickelt wird.

Zukünftig sollten zunächst die beschriebenen Kritikpunkte vorrangig adressiert werden, um die Architektur in eine produktionsreife und langfristig stabile Lösung zu überführen. Dabei sollte der Fokus auf der effizienten Ressourcennutzung liegen, indem die benötigten Rechenressourcen genauer analysiert und durch dynamische Lastverteilung optimiert werden. Dies würde nicht nur die Betriebskosten weiter senken, sondern auch die Skalierbarkeit der Anwendung verbessern. Ein weiterer wichtiger Schritt ist die Implementierung einer CI/CD-Pipeline, um die kontinuierliche Integration und Bereitstellung von Updates zu erleichtern. Dies wird die Wartbarkeit der Plattform erheblich verbessern und eine schnelle Reaktion auf zukünftige Anforderungen ermöglichen. Die umfassende Sicherheitsanalyse sollte parallel erfolgen, um mögliche Schwachstellen zu identifizieren und die Anwendung gegen potenzielle Bedrohungen zu schützen. In Bezug auf das Frontend der Anwendung sollte die konkrete Umsetzung geklärt werden. Hierbei könnten Standards für die Gebrauchstauglichkeit, wie die DIN EN ISO 9241-110, herangezogen werden, um die Eignung des Masterportals in Hinblick auf Benutzerfreundlichkeit zu analysieren. Anhand dieser Kriterien ließe sich beurteilen, ob das Masterportal den Anforderungen gerecht wird oder ob die Entwicklung einer eigenständigen Lösung sinnvoller wäre. Darüber hinaus bietet es sich an, die bestehenden Microservices noch generischer zu gestalten und als Service über die noch in die in der Entwicklung befindliche geoPlattform des LGLN bereitzustellen. Dies würde es Nutzern ermöglichen, spezifische Funktionalitäten wie die Berechnung von Rasterwertstatistiken und die Subpixelinterpolation flexibel für ihre eigenen Anwendungen zu nutzen. Ein solcher Schritt würde die Wiederverwendbarkeit und Flexibilität der Komponenten weiter steigern. Darüber hinaus können die entwickelten Microservices zur automatisierten Fehlerprüfung von Messdaten eingesetzt werden, indem sie direkt von

der verwendeten Messsoftware angesprochen werden. Dabei könnte automatisch geprüft werden, ob fehlerhafte Messwerte mit den Bodenbewegungen an der spezifischen Messkoordinate übereinstimmen oder korrelieren. Langfristig könnte dieser Ansatz weiterentwickelt werden, sodass die Microservices als Korrekturdienst fungieren, der Messwerte automatisch unter Berücksichtigung der Bodenbewegungen anpasst. Ein solches Verfahren würde jedoch zunächst umfassend untersucht und validiert werden müssen, um sicherzustellen, dass die Anpassungen zuverlässig und präzise sind.

Literaturverzeichnis

Abernathey, R. P., Augspurger, T., Banihirwe, A., Blackmon-Luca, C. C., Crone, T. J., Gentemann, C. L., Hamman, J. J., Henderson, N., Lepore, C., McCaie, T. A., et al. (2021). Cloud-native repositories for big scientific data. *Computing in Science & Engineering*, *23*(2), 26–35.

Alqaryoutia, O., & Siyamb, N. (2018). Serverless Computing and Scheduling Tasks on Cloud: A. *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, *40*(1), 235–247.

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Isahagian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless Computing: Current Trends and Open Problems.

Barciauskas, A. e. a. (2023). Cloud Optimized Geospatial Formats Guide - FlatGeoBuf. https://guide.cloudnativegeo.org/flatgeobuf/intro.html. Abgerufen am 11.07.2024

BITKOM. (2009). *Cloud Computing - Evolution in der Technik, Revolution im Business*. Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.

Borchers, C. (2022). Projektbereicht - Konzeption einer webbasierten Plattform zur Visualisierung und zum Processing von Daten des Bodenbewegungsdienstes Niedersachsen [Dokument zur internen Verwendung].

Brockmeyer, M. (2024). *Modellierung von Bodenbewegungen anhand heterogener Messverfahren am Beispiel der niedersächsischen Landesfläche* (Diss.). Leibniz Universität Hannover. Bayerische Akademie der Wissenschaften.

Brockmeyer, M., Schnack, C., & Jahn, C.-H. (2020). Datenanalyse und flächenhafte modellierung der psi-informationen des bodenbewegungsdienst deutschlands für die landesfläche niedersachsens. *Zfv-Zeitschrift für Geodäsie, Geoinformation und Landmanagement*, (zfv 3/2020).

Casalicchio, E. (2019). Container orchestration: A survey. *Systems Modeling: Methodologies and Tools*, 221–235.

Cloud Native Computing Foundation, C. (2023). CNCF Cloud Native Definition v1.0. https://github.com/cncf/toc/blob/main/DEFINITION.md. Abgerufen am 2024-07-01

COGEO. (2024). Cloud Optimized GeoTIFF (COG). https://www.cogeo.org. Abgerufen am 15.07.2024

De Lange, N. (2020). *Geoinformatik: in Theorie und Praxis* (4. Aufl.). Springer-Verlag.

Deutsches Institut für Normung e.V. (1999). *DIN 21917: 1999-02 - Bergmännisches Risswerk Gebirgs- und Bodenbewegungen*. Beuth Verlag GmbH.

Diaby, T., & Rad, B. B. (2017). Cloud computing: a review of the concepts and deployment models. *International Journal of Information Technology and Computer Science*, *9*(6), 50–58.

Ehsan, A., Abuhaliqa, M. A. M., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*, *12*(9), 4369.

Feil, N., Bögelsack, A., Schulz, R., & Abrantes, G. (2023). *Public Cloud Potenzial in einem Unternehmensumfeld*. Springer.

Ferretti, A., Prati, C., & Rocca, F. (2001). Permanent scatterers in SAR interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, *39*(1), 8–20.

FlatGeoBuf Developers. (o.J.). FlatGeobuf. https://github.com/flatgeobuf/flatgeobuf. Abgerufen am 11.07.2024

Gao, S., & Gruev, V. (2011). Bilinear and bicubic interpolation methods for division of focal plane polarimeters. *Optics express*, *19*(27), 26161–26173.

Gharbi, M., Koschel, A., Rausch, A., & Starke, G. (2023). *Basiswissen für Softwarearchitekten: Aus- und Weiterbildung nach iSAQB-Standard zum Certified Professional for Software Architecture–Foundation Level*. dpunkt. verlag.

Goniwada, S. R. (2022). *Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples*. Springer.

Google. (o.J.). FlatBuffers. https://github.com/google/flatbuffers. Abgerufen am 11.07.2024

Grafana Labs. (2024). *k6 Documentation*. https://grafana.com/docs/k6/latest/. Abgerufen am 24.09.2024

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

HasiCorp. (o.J.). Standard Module Structure. https://developer.hashicorp.com/terraform/language/modules/develop/structure. Abgerufen am 13.09.2024

Hentschel, R., & Leyh, C. (2018). Cloud Computing: Status quo, aktuelle Entwicklungen und Herausforderungen. In S. Reinheimer (Hrsg.), *Cloud Computing: Die Infrastruktur der Digitalisierung* (S. 3–20). Springer Fachmedien Wiesbaden.

Hermann, A. (2022). *Grundlagen der Anforderungsanalyse - Standardkonformes Requirements Engineering*. Springer Vieweg.

Heunisch, C., Caspers, G., Elbracht, J., Langer, A., Röhling, H.-G., Schwarz, C., & Streif, H. (2017). Erdgeschichte von Niedersachsen. Geologie und Landschaftsentwicklung. *1864-7529*, *6*, 3–84.

Howard, M. (2022). Terraform–Automating Infrastructure as a Service. *arXiv preprint arXiv:2205.10676*.

IBM. (2021). Microservices in the enterprise, 2021: Real benefits, worth the challenges - How organizations are finding speed, agility and resiliency through microservices.

IBM. (2024a). Code Engine. https://cloud.ibm.com/docs/codeengine. Abgerufen am 04.09.2024

IBM. (2024b). Code Schematics. https://www.ibm.com/products/schematics. Abgerufen am 04.09.2024

IBM. (2024c). Containers. https://www.ibm.com/de-de/topics/containers. Abgerufen am 02.07.2024

IBM. (2024d). Hosting models pricing. https://cloud.ibm.com/docs/cloud-databases?topic=cloud-databases-hosting-pricing. Abgerufen am 27.08.2024

IBM. (2024e). Serverless in der IBM Cloud. https://www.ibm.com/de-de/products/code-engine. Abgerufen am 02.07.2024

IBM. (2024f). Was ist 'Infrastruktur als Code' (Infrastructure as Code)? https://cloud.ibm.com/docs/schematics?topic=schematics-infrastructure-as-code. Abgerufen am 04.07.2024

IBM. (o.J.). Was ist Serverless Computing? https://www.ibm.com/de-de/topics/serverless.

ISO/IEC 25010. (2011). *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models (ISO/IEC 25010:2011(E))*.

Jahn, C.-H., Feldmann-Westendorff, U., Grüner, D., Kulle, U., & Lembrecht, P. (2011). Die Erneuerung des Deutschen Haupthöhennetzes in Niedersachsen. *Nachrichten der Niedersächsischen Vermessungs- und Katasterverwaltung*, *04/2011*, 3–26.

Kahmen, H. (2005). *Angewandte Geodäsie: Vermessungskunde*. De Gruyter.

Kalberer, P. (2021). t-rex Configuration Reference. https://t-rex.tileserver.ch/doc/reference/#configuration-reference. Abgerufen am 06.08.2024

Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, *29*(6), 1153–1160.

Klausing, H., & Holpp, W. (1999). *Radar mit realer und synthetischer Apertur*. Oldenbourg Wissenschaftsverlag.

Koppmann, V. (2020). *Erweiterung der raum-zeitlichen Analysen von InSAR-Daten zur getrennten Ableitung von Bodenbewegungen in vertikaler und horizontaler Richtung* (Masterarbeit). Leibniz Universität Hannover.

Kratzke, N. (2018). A brief history of cloud application architectures. *Applied Sciences*, *8*(8).

Kratzsch, H. (2013). *Bergschadenkunde*. Springer-Verlag.

Lee, H., Satyam, K., & Fox, G. (2018). Evaluation of Production Serverless Computing Environments. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 442–450. https://doi.org/10.1109/CLOUD.2018.00062

LGLN. (2017). Fachliches Zukunftskonzept für die Vermessungs- und Katasterverwaltung - Zieljahr 2025 [Dokument zur internen Verwendung].

LGLN. (2022). *IT-Strategie des LGLN* (1.0) [Dokument zur internen Verwendung].

Lins, S., & Sunyaev, A. (2018). Klassifikation von Cloud-Services. In H. Krcmar, C. Eckert, A. Roßnagel, A. Sunyaev & M. Wiesche (Hrsg.), *Management sicherer Cloud-Services: Entwicklung und Evaluation dynamischer Zertifikate* (S. 7–13). Springer Fachmedien Wiesbaden.

Lukša, M. (2018). *Kubernetes in Action: Anwendungen in Kubernetes-Clustern bereitstellen und verwalten*. Carl Hanser Verlag GmbH Co KG.

Lv, Y., Feng, Q., & Qi, L. (2009). A study of sub-pixel interpolation algorithm in digital speckle correlation method. *2008 International Conference on Optical Instruments and Technology: Optoelectronic Measurement Technology and Applications*, *7160*, 740–748.

Marinescu, D. C. (2023). *Cloud computing: theory and practice*. Katey Birtcher.

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing* (Techn. Ber. Nr. 800-145). National Institute of Standards und Technology (NIST). Gaithersburg, MD.

Münzl, G., Pauly, M., & Reti, M. (2015). *Cloud Computing als neue Herausforderung für Management und IT*. Springer.

Odun-Ayo, I., Ajayi, O., Akanle, B., & Ahuja, R. (2017). An overview of data storage in cloud computing. *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, 29–34.

Patil, A., Rangarao, D., Seipp, H., Lasota, M., dos Santos, R. M., Markovic, R., Casey, S., Bollers, S., Gucer, V., Lin, A., et al. (2020). *Cloud Object Storage as a Service: IBM Cloud Object Storage from Theory to Practice-For developers, IT architects and IT specialists*. IBM Redbooks.

Perry, M. T. (2015). Zonal Statistics. https://pythonhosted.org/rasterstats/manual.html#zonal-statistics. Abgerufen am 19.09.2024

Quix, C. (2021). Big-Data-Technologien. *Data Science: Konzepte, Erfahrungen, Fallstudien und Praxis*, 133–148.

Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, *17*(3), 228.

Riti, P. (2018). *Pro DevOps with Google Cloud Platform: With Docker, Jenkins, and Kubernetes*. Springer.

Roberts, M., & Chapin, J. (2016). *What Is Serverless?* O'Reilly Media.

Scheuner, J., & Leitner, P. (2020). Function-as-a-Service performance evaluation: A multivocal literature review. *Journal of Systems and Software*, *170*, 110708.

Sirtl, H. (2010). Cloud Computing - Stabiles Fundament für IT as a Service. *Z Serv Manag*, *13*, 3–7.

Surianarayanan, C., & Chelliah, P. R. (2019). Essentials of Cloud Computing. *Cham: Springer International Publishing*.

Survey, U. G. (2020). Landsat Cloud Optimized GeoTIFF (COG): DFCB Version 2.0. https://d9-wret.s3.us-west-2.amazonaws.com/assets/palladium/production/s3fs-public/atoms/files/LSDS-1388-Landsat-Cloud-Optimized-GeoTIFF_DFCB-v2.0.pdf. Abgerufen am 15.07.2024

Team Geodätische Services. (2024). Architekturdokumentation BOB.NI WebApp [Dokument zur internen Verwendung].

Tiangolo. (2024). FastAPI. https://fastapi.tiangolo.com. Abgerufen am 06.08.2024

Tremp, H. (2021). *Architekturen Verteilter Softwaresysteme* (1. Aufl.). Springer.

Velepucha, V., & Flores, P. (2023). A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access*, *11*, 88339–88358.

Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A., et al. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *Service Oriented Computing and Applications*, *11*, 233–247.

Vite. (o.J.). Why Vite. https://vitejs.dev/guide/why.html. Abgerufen am 21.09.2024

Williams, H. (2022). Kicking the Tires: Flatgeobuf. https://worace.works/2022/02/23/kicking-the-tires-flatgeobuf/. Abgerufen am 11.07.2024

Yin, X. (2020). *Einflüsse geometrischer Radar- Aufnahmekonstellationen auf die Qualität der kombinativ berechneten Bodenbewegungskomponenten* (Diss.). TU Clausthal.

A Anforderungsverzeichnis

A1 Funktionale Anforderungen

- **F1.)** Die Software muss die Möglichkeit bieten, Bodenbewegungsstatistiken für benutzerdefinierte Gebiete zu berechnen, um einen umfassenden Überblick über die vertikalen und horizontalen Verschiebungen zu erhalten.
- **F2.)** Die Software muss flächendeckende Bodenbewegungen unter Berücksichtigung der zeitlichen Komponente visualisieren.
- **F3.)** Die Software muss eine Funktion zur Ableitung von Profilen bieten, bei der im Kartenteil der Webanwendung ein freies Profil durch Zeichnen definiert wird, das automatisch mit den Daten des Bodenbewegungsmodells verschnitten wird, um im abbildenden Teil den kontinuierlichen Verlauf der Geschwindigkeit entlang dieses Profils zu präsentieren.
- **F4.)** Die Software muss Gebiete, in denen Bodenbewegungen zu erwarten sind, auf einer Karte visualisieren und entsprechende Statistiken zu diesen Gebieten liefern.

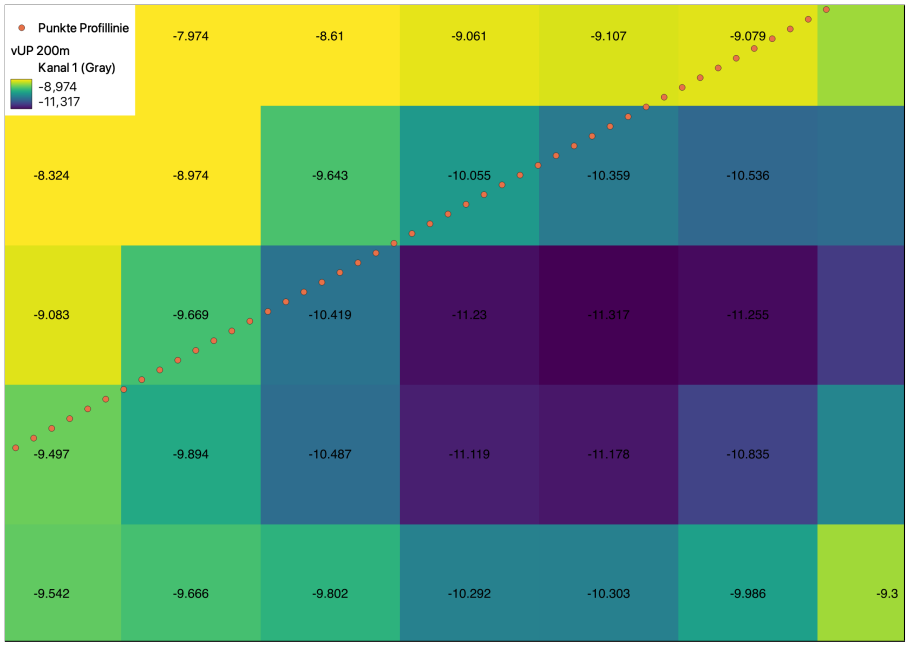
A2 Qualitätsanforderungen

- **Q1.)** Die Software muss zunächst als MVP entwickelt werden, wobei eine hochwertige Architektur sicherstellt, dass bestehende Komponenten und neue Anforderungen von Nutzern nahtlos integriert werden können.
- **Q2.)** Die Software muss möglichst ausschließlich kostengünstige Cloud-native Technologien verwenden, um die Betriebskosten niedrig zu halten und gleichzeitig eine effiziente Entwicklung sicherzustellen.
- **Q3.)** Die Bodenbewegungsdaten müssen eine hohe Genauigkeit aufweisen, um präzise und verlässliche Informationen bereitzustellen.
- **Q4.)** Die Software muss möglichst aktuelle Daten visualisieren und bei neuen Daten diese schnell und automatisiert integrieren, um stets die neuesten Informationen darzustellen.
- **Q5.)** Die Architektur muss als Cloud-basierte Microservice-Architektur konzipiert werden.
- **Q6.)** Die Software muss, wenn erforderlich, verfügbare Komponenten von Drittanbietern als Service nutzen.
- **Q7.)** Die Architektur muss eine medienbruchfreie Weitergabe der Bodenbewegungsdaten ermöglichen.
- **Q8.)** Die Kommunikation mit der Software muss über eine generische REST-API erfolgen.
- **Q9.) Leistungsfähigkeit:** Die Architektur muss eine hohe Leistungseffizienz gewährleisten, indem sie kurze Reaktionszeiten und minimalen Ressourcenverbrauch unter verschiedenen Lastbedingungen sicherstellt.
- **Q10.) Kompatibilität:** Die Software muss eine hohe Kompatibilität aufweisen, indem sie nahtlos mit anderen Systemen, Plattformen und Anwendungen zusammenarbeitet.
- **Q11.) Zuverlässigkeit:** Die Architektur muss eine hohe Zuverlässigkeit sicherstellen, indem sie kontinuierlich und fehlerfrei arbeitet und Ausfallzeiten minimiert.

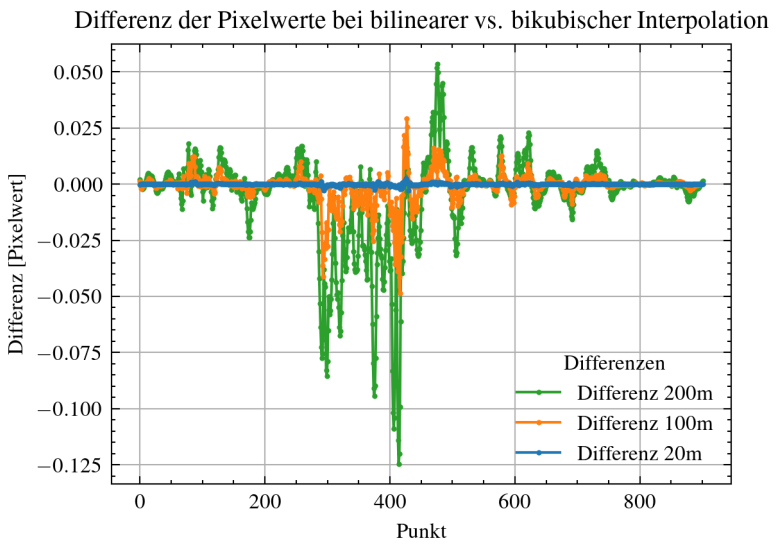
- **Q12.) Sicherheit:** Die Software muss höchste Sicherheitsstandards erfüllen, indem sie Daten vor unbefugtem Zugriff schützt und sichere Datenübertragungen gewährleistet.
- **Q13.) Wartbarkeit:** Die Architektur muss eine hohe Wartbarkeit unterstützen, indem sie leicht zu aktualisieren und zu modifizieren ist.
- **Q14.) Übertragbarkeit:** Die Software muss eine hohe Übertragbarkeit aufweisen, indem sie problemlos auf verschiedene Hardware- und Softwareplattformen migriert werden kann.

B Grafiken Vergleich Subpixelinterpolation

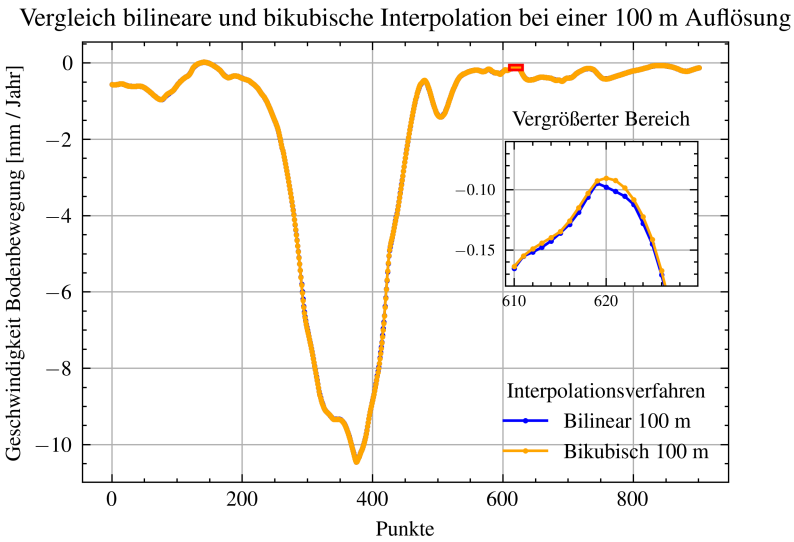
B1 Entstehung Treppeneffekt



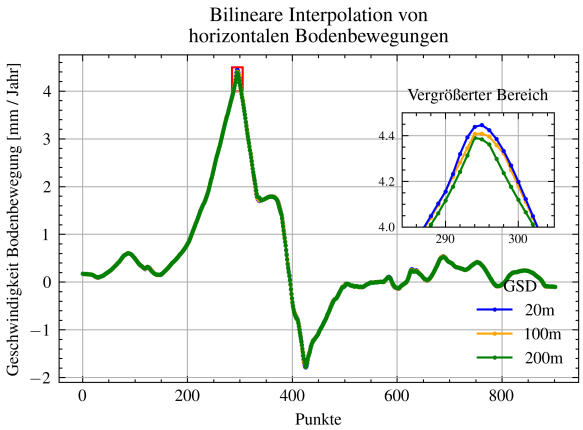
B2 Vergleich Differenzen bilineare und bikubische Interpolation

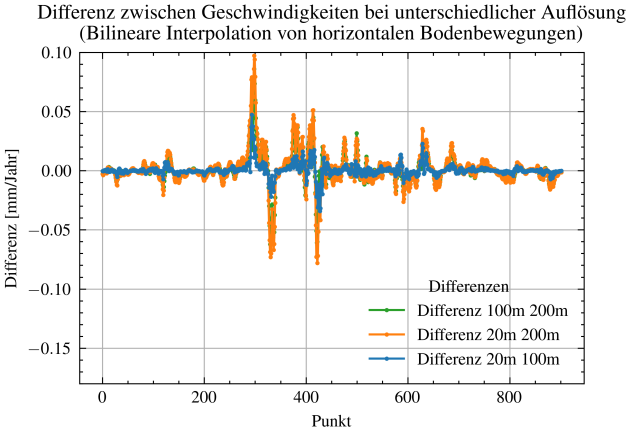


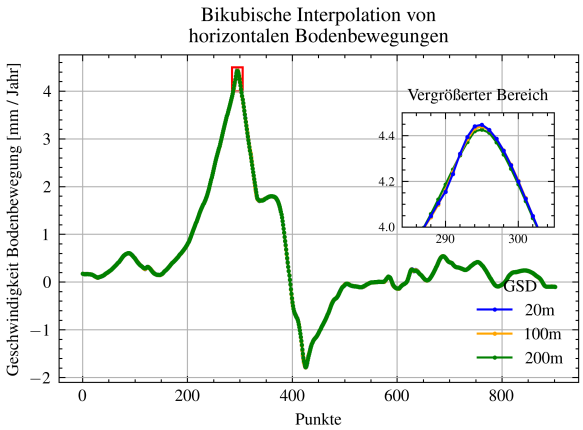
B3 Spitze Artefakte bei bilinearer Interpolation

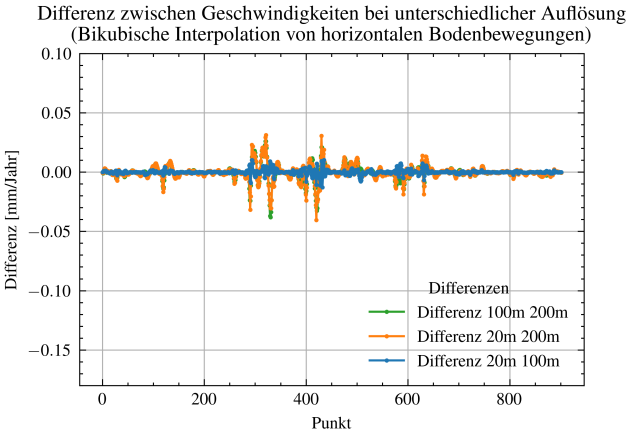


B4 Profile und Abweichungen für horizontale Bodenbewegungen

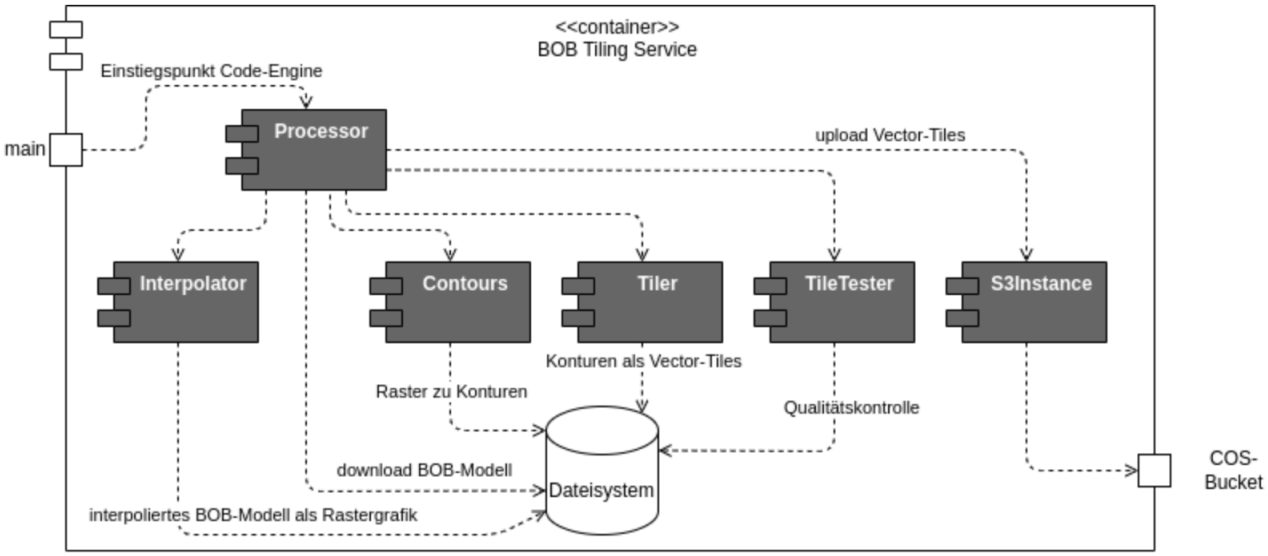








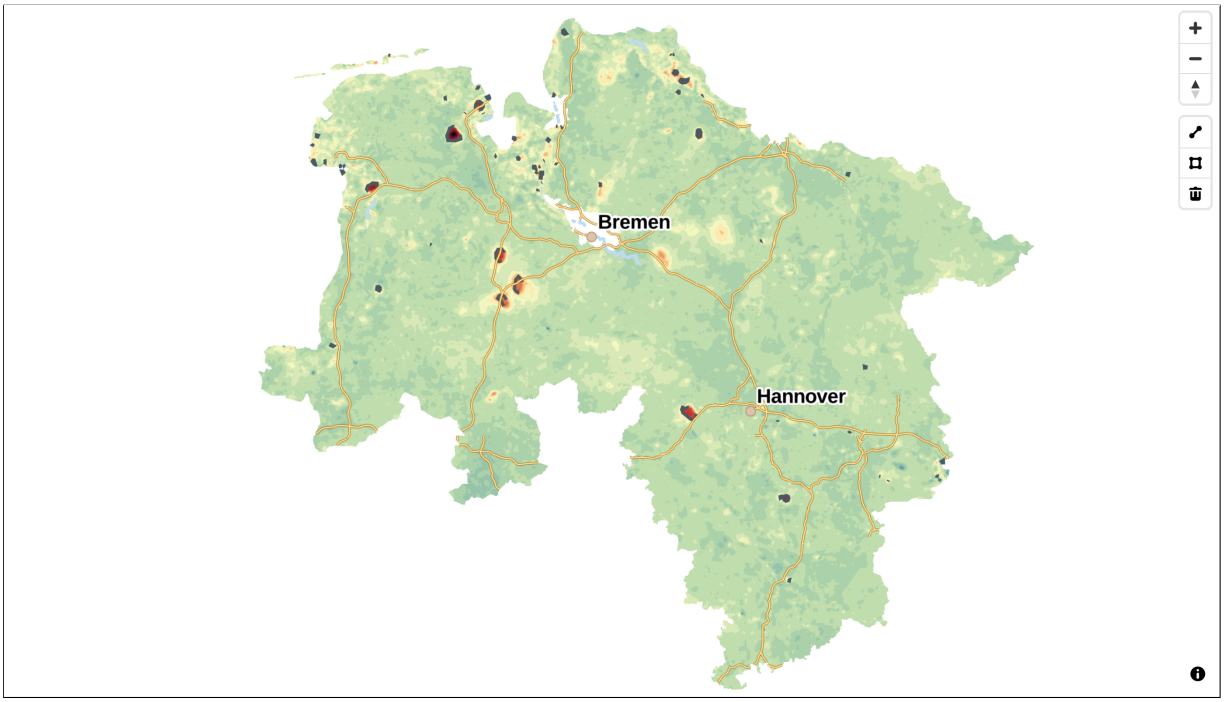
C Aufbau bob-tiling-service



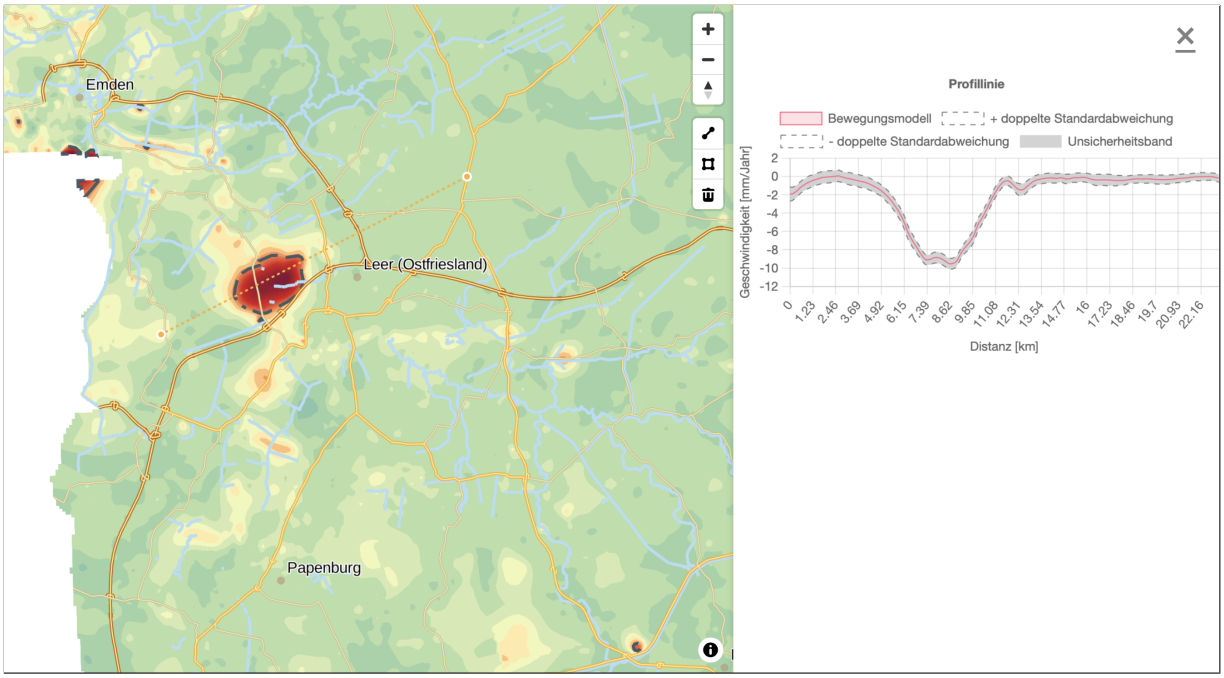
(Team Geodätische Services, 2024)

D Frontend

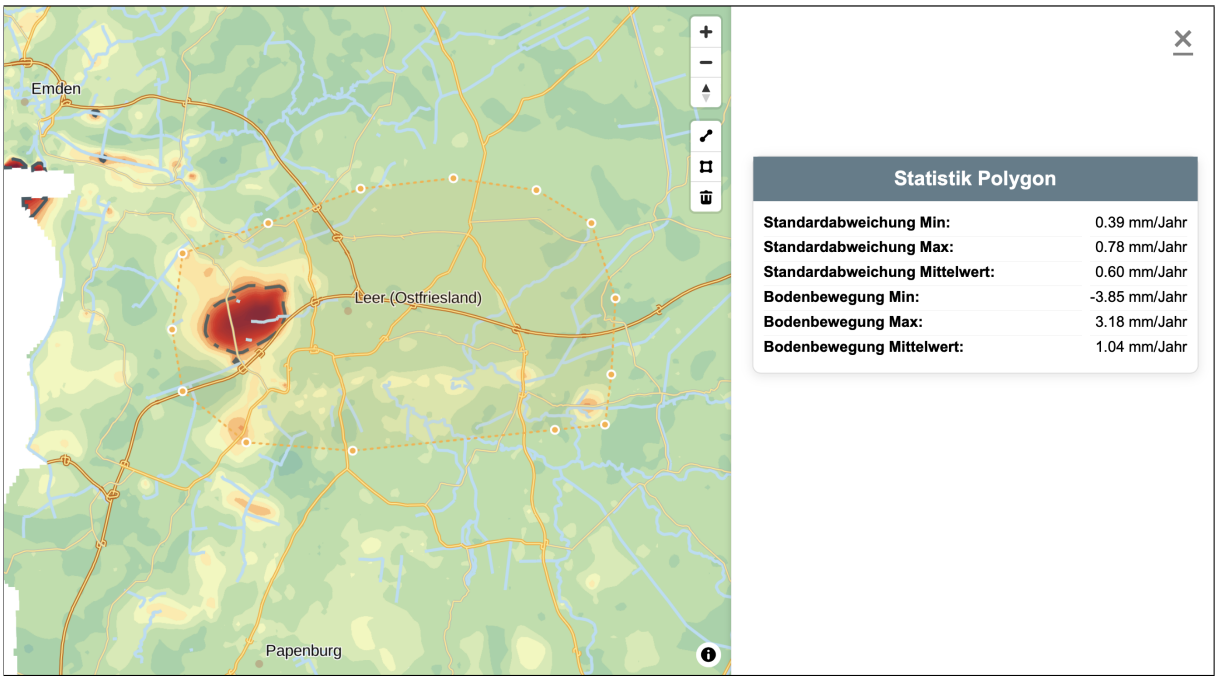
D1 Startansicht



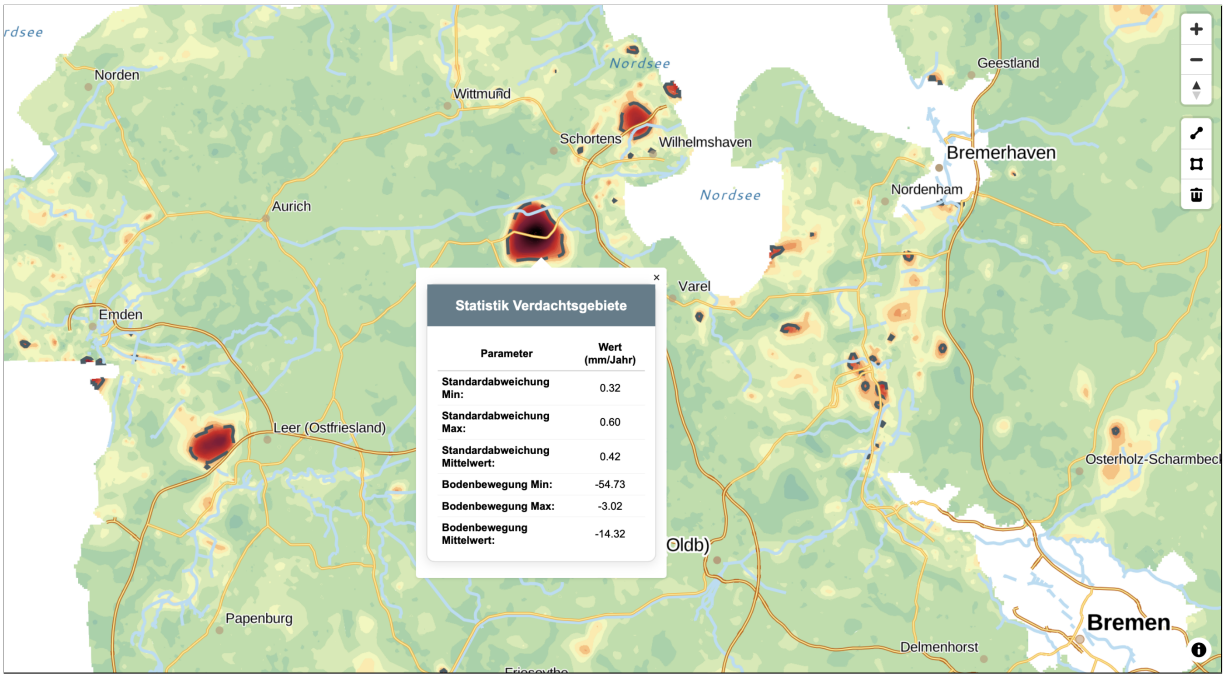
D2 Bodenbewegungsprofil



D3 Benutzerdefinierte Rasterstatistik



D4 Bodenbewegungsstatistik in Verdachtsgebiet



E Tabellarische Auflistung der verwendeten Technologien und Bibliotheken

E1 Technologien

Technologie (Version)	Beschreibung
Docker (25.0.2)	Containerisierungsplattform zur Bereitstellung und Skalierung der Microservices und Jobs
GDAL (3.8.5)	Geospatial Data Abstraction Library zur Konvertierung der Datenformate und Grundlage für verschiedene Python-Bibliotheken
GitLab	Quellcodeverwaltung
IBM Cloud	Cloud-Plattform zur Bereitstellung der Infrastruktur und Ressourcen für die Anwendung
Insomnia (v2023.4.0)	Tool zum Testen und Debuggen der REST-APIs der Microservices
JavaScript	Programmiersprache zur Implementierung des Frontends
Python (3.10)	Primäre Programmiersprache zur Umsetzung der Microservices
QGIS (3.22)	Geoinformationssystem zur Visualisierung schnellen Visualisierung der Daten
Terraform	IaC-Sprache zur automatisierten Bereitstellung der Cloud-Ressourcen
VS Code (1.88.1)	Entwicklungsumgebung zur Programmierung und Debugging der Anwendung
Vite.js	Build-Tool zur Entwicklung und Optimierung des Frontends der Anwendung

E2 Python-Bibliotheken

Bibliothek	Beschreibung
fastapi	Umsetzung der REST-API der Microservices
uvicorn	ASGI-Server zum Ausführen von FastAPI-Anwendungen
pydantic	Datenvalidierung und -konvertierung durch benutzerdefinierte Python-Modelle
boto3	AWS SDK für Python zur Interaktion mit dem COS
python-dotenv	Lesen von Umgebungsvariablen aus einer .env-Datei
rasterio	Lesen von Rasterdateien
pyproj	Projektionen und Transformationen der Eingangskoordinaten
scipy	Bibliothek für den RegularGridInterpolator für die Subpixelinterpolation
numpy	Grundlegende Bibliothek für grundlegende Berechnungen sowie Nutzung der numpy Arrays
shapely	Erzeugung von Polygonen aus GeoJSON für Rasterstatistiken
geopandas	Bibliothek zur räumlichen Analyse und Handhabung von Geodaten
rasterstats	Berechnung von Zonenstatistiken der Bodenbewegungsdaten

E3 JavaScript-Module

Bibliothek	Beschreibung
maplibre-gl	Open-Source-Bibliothek für die Darstellung von Vektordaten.
mapbox/mapbox-gl-draw	Erweiterung zum Zeichnen und Bearbeiten von Geometrien .
turf/turf	Bibliothek für räumliche Analyse (Berechnung der Zwischenpunkte für Profillinie)
chart.js	Bibliothek zur Erstellung von interaktiven Diagrammen (Visualisierung der Profillinien).
chartjs-plugin-zoom	Plugin zum Hinzufügen von Zoom- und Schwenkfunktionen zu Diagrammen.
mapbox-gl	Bibliothek zur Visualisierung von interaktiven Karten mit Vektordaten.
mapbox-gl-flatgeobuf	Plugin für FlatGeobuf-Format-Unterstützung in Mapbox GL JS.

F Beispielhafte Payload

```
{
    "file_name": "COG_100m_2024-fake_TIN_vUp.tif",
    "file_name_std":"COG_100m_2024-fake_TIN_s_vUp.tif",
    "band": 1,
    "bucket_name": "bucket_prod",
    "epsg": 4326,
    "data": {
        "coordinates":[
            [ 8.163245645097046, 53.153411533753044,1 ],
            [ 8.159881340844974, 53.135028014089933,2 ],
            [ 8.17441994136286, 53.132624939624172,3 ],
            [ 8.182350087099888, 53.124094025270701,4 ],
            [ 8.201574682826019, 53.124454486440563,5 ]
        ]
    }
}
```

G Beschreibung des elektronischen Anhangs

Dem USB-Stick, der dieser Masterarbeit beigefügt ist, liegt der digitale Anhang der Arbeit bei. Dieser enthält den implementierten Quellcode, relevante Datensätze sowie Beispielvideos der Umsetzung. Die Ordnerstruktur ist wie folgt aufgebaut:

- **bob-cog-convert** – Quellcode des Jobs zur Konvertierung eines GeoTIFF in ein Cloud Optimized GeoTIFF
- **bob-fgb-convert** – Quellcode des Jobs Job zur Konvertierung einer GeoJSON-Datei in FlatGeobuf inklusive der Berechnung von Rasterstatistiken für die Polygone
- **bob-frontend-cos** – Quellcode des Frontends der Anwendung
- **bob-infrastructure** – Terraform-Skripte zur Erstellung der Infrastruktur in der IBM Cloud
- **bob-raster-stats** - Quellcode für den Microservice zur Berechnung von Rasterstatistiken
- **bob-sub-interpol** - Quellcode für den Microservice der Subpixelinterpolation
- **Rasterbasiertes Bodenbewegungsmodell** – Daten des rasterbasierten Bodenbewegungsmodells im GeoTIFF- und COG-Format mit einer Auflösung von 100 m
- **Verdachtsgebiete** – Verwendete Verdachtsgebiete im GeoJSON-Format (ohne Statistik) und GeoJSON sowie FlatGeobuf (mit Statistik)
- **Videos** – Videos zur Demonstration der entwickelten automatisierten Datenverarbeitung und der Funktionalitäten des Frontends sowie eine textliche Erklärung
- **Eigenständigkeitserklärung JadeHS.pdf** – Dokument zur Eigenständigkeitserklärung sowie Nuzung von KI-Systemen der Jade Hochschule
- **Dokumentation der Verwendung generativer KI-Systeme.md** – Selbe Bemerkung wie in Anhang H
- **README.md** – Beschreibung des digitalen Anhangs sowie weitere Anmerkungen
- **Masterarbeit_Wykhoff_Jannes.pdf** – Masterarbeit

Die Untersuchungen, die im Rahmen dieser Masterarbeit durchgeführt wurden, wurden mit Python 3.10 und Jupyter-Notebooks umgesetzt. Viele der Diagramme wurden mit der Bibliothek Matplotlib erstellt. Diese Diagramme und dazugehörigen Skripte sind nicht im digitalen Anhang enthalten, können aber auf Anfrage nachgereicht werden.

H Dokumentation der Verwendung generativer KI-Systeme

Das vollständige Dokument „Erklärung gemäß dem für Ihren Studiengang gültigen Allgemeinen Teil (Teil A) der Prüfungsordnung der Jade Hochschule Wilhelmshaven/Oldenburg/Elsfleth sowie zur Nutzung von KI-Systemen“ ist im digitalen Anhang enthalten. Nachfolgend findet sich eine Übersicht der in dieser Arbeit eingesetzten generativen KI-Systeme:

Im Rahmen der vorliegenden Masterarbeit wurde das KI-System ChatGPT 4.0 unterstützend eingesetzt. Der Einsatz erfolgte in den folgenden Bereichen:

- **Erstellung von Quellcode-Kommentaren**: ChatGPT 4.0 wurde zur Generierung von erklärenden Kommentaren im Quellcode verwendet.
- **Fehlerbehandlung und Debugging**: Das KI-System unterstützte bei der Identifikation und Behebung von Fehlern im Quellcode. Dabei wurden Lösungsvorschläge zum Teil übernommen.
- **Frontend-Entwicklung**: Für die Entwicklung bestimmter Funktionalitäten im Frontend der WebApp wurden generative KI-Technologien eingesetzt. Diese Funktionalitäten sind im Code entsprechend gekennzeichnet, sodass die KI-basierten Abschnitte klar nachvollziehbar sind.
- **Verwendung von Python**: ChatGPT 4.0 wurden genutzt, um das Verständnis bestimmter Python-Konzepte, wie z.B. die Nutzung von `zip`, zu verbessern. Diese Erklärungen wurden anschließend angewendet, um entsprechende Funktionalitäten im Projekt zu implementieren.

Zudem wurden DeepL für Übersetzungen und DeepL Write für Textüberarbeitungen verwendet.

Es wird ausdrücklich darauf hingewiesen, dass alle wesentlichen Entwicklungsentscheidungen, insbesondere die Konzeption und Implementierung der Hauptfunktionen, eigenständig durchgeführt wurden. Der Einsatz von ChatGPT 4.0 beschränkte sich auf unterstützende Maßnahmen, die keine Einflussnahme auf die inhaltliche Tiefe oder wissenschaftliche Eigenständigkeit dieser Arbeit hatten.