



Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“

(UNIGIS MSc) am Fachbereich Geoinformatik (Z_GIS)
der Paris Lodron Universität Salzburg

zum Thema

„Performancevergleich von Open-Source- Vector-Tiles-Serverlösungen zur Bereitstellung von Geodaten aus PostGIS-Datenbanken“

eingereicht von

Fabian Rechsteiner

107112, UNIGIS MSc Jahrgang 2022

GutachterIn:

Dr. Christian Neuwirth

Zur Erlangung des Grades

„Master of Science“, abgekürzt „MSc“

Frauenfeld, 24.05.2024

ZUSAMMENFASSUNG

Die Einführung der neuen OGC API-Familie markiert einen Meilenstein im Austausch von Geodaten über das Web. Mit dem OGC API-Tiles Standard können nun sowohl Raster- als auch Vektordaten als Kacheln bereitgestellt werden, wobei Vektordaten nicht mehr als Bilder, sondern als Geometrie an den Client übergeben werden. In dieser Masterarbeit werden sechs Open-Source-Vector-Tiles-Server (BBOX, Ldproxy, Martin, pg_tileserv, Tegola und TiPg) mithilfe von Docker in einer Public Cloud aufgesetzt und konfiguriert. Aus den Vektordaten der PostGIS-Datenbank werden für jeden Server Vector Tiles erstellt. Durch verschiedene Testszenarien mit Apache JMeter wird ermittelt, welcher Server die Vector Tiles am schnellsten bereitstellen kann. Die Ergebnisse des Performancetests zeigen deutliche Unterschiede in der Leistungsfähigkeit der Server. Ein Server erwies sich dabei als der mit Abstand schnellste, während ein anderer eindeutig der langsamste über alle Testszenarien hinweg war. Um die Ergebnisse visuell zu veranschaulichen und zu vergleichen, wird zusätzlich mithilfe von Maplibre GL JS eine Webseite erstellt, auf der die Daten der verschiedenen Server visualisiert und verglichen werden können. Diese Masterarbeit unterstreicht erneut das enorme Potenzial und die Vorteile der neuen OGC API Standards sowie von Vector Tiles und soll Geodatenanbieter dazu motivieren, ihre Daten zukünftig ebenfalls nach diesen Standards anzubieten.

ABSTRACT

The introduction of the new OGC API family marks a milestone in the exchange of geodata over the web. With the OGC API - Tiles standard, both raster and vector data can now be provided as tiles, with vector data no longer being transmitted as images, but as geometry to the client. In this master thesis, six open-source-vector-tiles-servers (BBOX, Ldproxy, Martin, pg_tileserv, Tegola, and TiPg) are set up and configured using Docker in a public cloud. Vector tiles are created for each server from the vector data of the PostGIS database. Various test scenarios with Apache JMeter are used to determine which server can deliver the vector tiles the fastest. The results of the performance test show clear differences in the performance of the servers. One server proved to be by far the fastest, while another was clearly the slowest across all test scenarios. Additionally, a website is created using Maplibre GL JS to visually illustrate and compare the results. This master thesis underscores the enormous potential and advantages of the new OGC API standards as well as vector tiles and aims to motivate geodata providers to offer their data according to these standards in the future.

INHALTSVERZEICHNIS

Zusammenfassung.....	I
Abstract	I
Inhaltsverzeichnis.....	II
Abkürzungsverzeichnis.....	III
Abbildungsverzeichnis.....	III
1 Einleitung.....	1
2 OGC API	2
2.1 OGC API – Tiles	5
2.1.1 Raster Tiles	5
2.1.2 Vector Tiles.....	6
3 Methodik.....	7
3.1 Grundlagedaten.....	8
3.2 Public Cloud.....	9
3.2.1 Varianten Cloud-Umgebung.....	10
3.2.2 Docker Compose	12
3.3 Vector-Tiles-Server	13
3.3.1 Übersicht Vector Tiles Server	15
4 Implementierung der Vector Tiles Server	16
4.1 BBOX.....	17
4.2 Ldproxy	20
4.3 Martin.....	25
4.4 pg_tileserv	27
4.5 Tegola	29
4.6 TiPg.....	32
5 Performanceuntersuchung	34
5.1 Test-Design.....	35
5.1.1 Testziele.....	35
5.1.2 Testszenarien.....	35
5.1.3 Testdaten.....	39
5.1.4 Testumgebung.....	39
5.2 Performancetest mit Apache JMeter	41
6 Ergebnisse	44
6.1 Resultate mit einem Benutzer.....	46
6.2 Resultate mit mehreren Benutzer.....	52
6.3 Visueller Performancevergleich	55

7	Diskussion.....	58
8	Schlussfolgerung und Ausblick	58
9	Literaturverzeichnis.....	59

ABKÜRZUNGSVERZEICHNIS

AGI	<i>Amt für Geoinformation</i>
API	<i>Application Programming Interface</i>
AV	<i>Amtliche Vermessung</i>
COG.....	<i>Cloud Optimized Geotiff</i>
CSW	<i>Catalog Service for the Web</i>
DGGS.....	<i>Discret Global Grid System</i>
EDR	<i>Environmental Data Retrieval</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IoT.....	<i>Internet of Things</i>
LOD	<i>Level of Details</i>
MVT	<i>Mapbox Vector Tile</i>
OGC.....	<i>Open Geospatial Consortium</i>
PBF.....	<i>Protocolbuffer Binary Format</i>
SLD.....	<i>Styled Layer Descriptor</i>
SOAP	<i>Simple Object Access Protocol</i>
SQL.....	<i>Structured Query Language</i>
SSH.....	<i>Secure Shell</i>
SWE.....	<i>Sensor Web Enablement</i>
TMS.....	<i>Tile Matrix Set</i>
UAV.....	<i>unmanned aerial vehicle</i>
VSCode	<i>Visual Studio Code</i>
WCS	<i>Web Coverage Service</i>
WFS.....	<i>Web Feature Service</i>
WMS.....	<i>Web Map Service</i>
WMTS	<i>Web Map Tile Service</i>
WPS	<i>Web Processing Service</i>

ABBILDUNGSVERZEICHNIS

Abbildung 1: Cloud-Umgebung (Variante 1)	10
Abbildung 2: Cloud-Umgebung (Variante 2)	11
Abbildung 3: BBOX items bo_boflaeche_mv	19
Abbildung 4: Ldproxy Collection items bo_boflaeche_mv.....	24
Abbildung 5: Martin JSON-File bo_boflaeche_mv	26
Abbildung 6: pg_tileserv Kachelvorschau bo_boflaeche_mv	28
Abbildung 7: Tegola Kachelvorschau.....	31
Abbildung 8: TiPg Collection items bo_boflaeche_mv.....	33
Abbildung 9: Kachel des Test 1 vom Server Tegola.....	36
Abbildung 10: Kacheln des Test 2 vom Server Tegola.....	37
Abbildung 11: Kachel des Test 3 vom Server Tegola.....	37
Abbildung 12: Kachel des Test 4 vom Server Tegola.....	38
Abbildung 13: Kachel des Test 5 vom Server Tegola.....	38

Abbildung 14: Global Parameters	42
Abbildung 15: Server Parameters	42
Abbildung 16: Loop Controller	42
Abbildung 17: Test Plan BBOX.....	42
Abbildung 18: Thread Group bbox_1	42
Abbildung 19: HTTP Request Test_1	43
Abbildung 20: Durchschnittliche Antwortzeit pro Server (Übersicht)	46
Abbildung 21: Durchschnittliche Antwortzeit pro Server (Test 1)	47
Abbildung 22: Durchschnittliche Antwortzeit pro Server (Test 2)	47
Abbildung 23: Durchschnittliche Antwortzeit pro Server (Test 3)	48
Abbildung 24: Durchschnittliche Antwortzeit pro Server (Test 4)	49
Abbildung 25: Durchschnittliche Antwortzeit pro Server (Test 5)	50
Abbildung 26: Datenmenge pro Server.....	51
Abbildung 27: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 1, logarithmisch).....	52
Abbildung 28: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 1, linear)	52
Abbildung 29: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 2, logarithmisch).....	52
Abbildung 30: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 2, linear)	52
Abbildung 31: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 3, logarithmisch).....	53
Abbildung 32: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 3, linear)	53
Abbildung 33: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 4, logarithmisch).....	53
Abbildung 34: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 4, linear)	53
Abbildung 35: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 5, logarithmisch).....	53
Abbildung 36: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 5, linear)	53
Abbildung 37: Screenshot HTML mit den Vector Tiles Daten	57

1 EINLEITUNG

Mit der Digitalisierung werden auch immer mehr Geodaten statt als analoge Karten in Geodatenbanken gespeichert und ausgetauscht.

Im Jahr 1994 wurde das Open Geospatial Consortium (OGC) gegründet mit dem Ziel interoperable und offenen Standards zu definieren, um Geodaten auszutauschen (Reed 2011). Als im Jahr 2000 mit dem Web Map Service (WMS) der erste Standard für Geodienste durch OGC verabschiedet wurde, konnten Geodaten neu als georeferenzierte Kartenbilder über das Internet veröffentlicht werden (Morris 2006). Mit den neuen Standards, welche im Laufe der Jahre durch OGC verabschiedet wurden, wurde eine neue Ära in der Geoinformatik eingeläutet. Geodaten konnten nun einfach über das Internet genutzt und die Daten mussten nicht mehr heruntergeladen werden.

Im Verlauf der letzten Jahre haben sich die Trends in der Geoinformatik stark verändert. Dadurch, dass es immer einfacher und günstiger wird, grosse Datenmenge z.B. mittels Satelliten oder unmanned aerial vehicle (UAV) zu erfassen und speichern, wird die Speicherung und Analyse der enormen Datenmengen eine Herausforderung. Ein wichtiger Trend ist daher Big Data. Seit ca. Ende 2014 wird Big Data aber von dem neuen Trend Internet of Things (IoT) überholt (Kotsev et al. 2020). Mit dem IoT-Trend versucht man in verschiedenen Anwendungsbereiche mittels Sensoren den Zustand der physischen Welt zu messen und die Daten wenn möglich mit der Standortinformation auszuwerten (Kamilaris und Ostermann 2018).

Geodatenanbieter müssen sich überlegen, wie Sie ihre Geodateninfrastruktur aufbauen, um in Zukunft ihre Daten so effizient und einfach wie möglich zur Verfügung zu stellen. Traditionell werden Geodaten mittels der OGC Service Schnittstellen (z.B. WMS, WFS, WCS), welche auf dem Simple Object Access Protocol (SOAP) basieren, zur Verfügung gestellt. Mit der neuen OGC Standardsammlung namens OGC API, hat OGC eine neue Ära in der Geoinformatik eingeläutet. Die neuen Standards basieren auf den aktuellsten Webtechnologien, damit Geodaten in allen aktuellen Browser performant genutzt werden können. Moderne webbasierte APIs haben zahlreiche Vorteile, die für ihre Effizienz und Einfachheit sprechen. Sie bieten einen einfachen Ansatz für Datenverarbeitungs- und Managementfunktionen und können problemlos in verschiedenen Tools integriert werden. Zudem lassen sich die Daten durch gängige Suchmaschinen wie Google oder Bing einfacher entdecken (Simoes und Cerciello 2022).

Mit den klassischen OGC Standards WMS und WMTS werden Vektordaten durch einen Server gerendert und als Bildformate wie PNG oder JPEG an den Client übertragen. Dabei gehen verschiedene Informationen, die in den Vektordaten enthalten sind, verloren. Zudem wird die Darstellung der Daten vom Server vorgegeben und kann von den Nutzern nicht geändert werden. Der neue OGC Standard OGC API - Tiles ermöglicht es, Vektordaten in sogenannten Vector Tiles dem Client zur Verfügung zu stellen. Die Vector Tiles enthalten dabei keine Pixeldaten, sondern die entsprechenden Vektorgeometrien. Es ist ausserdem möglich, die Attributinformationen in den Geometrien mitzugeben. Es gibt bereits verschiedene Serverapplikationen, die den OGC API - Tiles Standard integriert haben und mit denen aus eigenen Vektordaten Vector Tiles erstellt werden können.

In dieser Masterarbeit wird untersucht, welche Open-Source-Vector-Tiles-Serverlösung die beste Performance bei der Bereitstellung von Geodaten aus einer PostGIS-Datenbank erzielt.

Dies wird erreicht, indem zunächst eine Einführung zu den neuen OGC API Standards und Vector Tiles gegeben wird. Anschliessend wird auf die Datengrundlage und Methodik eingegangen, wobei erklärt wird, welche Testdaten verwendet wurden und wie diese auf einem Linux Server in der Cloud mittels Docker bereitgestellt wurden. Des Weiteren werden sechs Open-Source-Serverlösungen für die

Erstellung und Bereitstellung von Vector Tiles miteinander verglichen. Die sechs Vector Tiles Server werden anschliessend mittels Docker-Containern implementiert und konfiguriert. Die Leistungsfähigkeit der Vector Tiles der verschiedenen Serverlösungen wird im Kapitel 5 mithilfe der Software Apache JMeter auf ihre Performance hin untersucht. Die Ergebnisse werden objektiv gemessen und verglichen. Das anschliessende Kapitel beinhaltet die Auswertung der Ergebnisse des Performancevergleichs sowie des visuellen Vergleichs der Serverlösungen. Die Forschungsfrage wird anhand der Ergebnisse beantwortet, und mögliche Einschränkungen und Limitationen der Arbeit werden diskutiert.

2 OGC API

Das Open Geospatial Consortium (OGC) ist eine Organisation, die offene Standards im GIS-Bereich entwickelt, um räumliche Informationen und Dienste Auffindbar, Zugänglich, Interoperable und Wiederverwendbar zu machen (Open Geospatial Consortium 2023).

Die neuen OGC API Standards bauen auf den bewährten OGC Webdienststandards (WMS, WFS, WCS usw.) auf und profitieren von modernen Webtechnologien. Sie sind darauf ausgerichtet, die Bereitstellung und Nutzung geografischer Daten im Web für jedermann einfach zu gestalten und eine nahtlose Integration mit anderen Datentypen zu ermöglichen. Durch ihre modulare Struktur können sie als „Bausteine“ verwendet werden, um massgeschneiderte APIs für verschiedene Anwendungen zu erstellen. Dies ermöglicht es, die benötigten OGC APIs flexibel und effizient in Anwendungen zu integrieren (OGC API 2024).

Folgende OGC APIs stehen zur Verfügung:

OGC API - Common

Diese API legt die Grundprinzipien und Funktionen für die Implementierung von OGC APIs fest. Sie zielt darauf ab, die Interoperabilität zwischen den verschiedenen OGC API-Implementierungen zu verbessern, indem sie sicherstellt, dass alle OGC APIs ähnliche Prinzipien und Funktionen verwenden und somit eine konsistente Struktur innerhalb der API Familie gewährleistet wird (OGC API - Common 2023).

OGC API – Features

Diese API definiert standardisierte Methoden zum Abrufen, Bearbeiten, Filtern und Abfragen von geografischen Informationen im Web. Sie nutzt moderne Webentwicklungsprinzipien und ist im Vergleich zu früheren Standards wie dem Web Feature Service (WFS) schlanker, einfacher zu implementieren und besser für die Integration in moderne Webanwendungen geeignet (OGC API - Features - OGC API workshop 2024).

OGC API - Features ist ein Standard, der eine standardisierte Methode zum Abfragen von geografischen Informationen im Web bietet. Diese API ermöglicht den Zugriff auf Sammlungen von geografischen Daten und definiert grundlegende API-Bausteine für die Interaktion mit Features, wobei ein „Feature“ ein Objekt in der realen Welt ist, das durch eine Geometrie und andere Eigenschaften beschrieben wird. Durch HTTP-Anfragen können Nutzer auf diese Daten zugreifen, sowohl für eine Liste aller verfügbaren Datensammlungen als auch für einzelne Features. Die Spezifikation unterstützt verschiedene Antwortformate wie HTML oder GeoJSON und ermöglicht die Filterung und Rückgabe von Daten. OGC API - Features fördert die Interoperabilität zwischen verschiedenen Anwendungen und Plattformen, indem es eine standardisierte Schnittstelle zum Austausch von geografischen Informationen bereitstellt (OGC API - Features: GitHub 2024).

OGC API – EDR (Environmental Data Retrieval)

Die OGC API - EDR bietet standardisierte, leichtgewichtige Schnittstellen zum Zugriff auf raumzeitliche Umweltdaten, die Abfragen an bestimmten Positionen, innerhalb von Bereichen, entlang von Trajektorien oder durch Korridore ermöglichen. Sie unterstützt die Ermittlung von API-Fähigkeiten und Metadaten sowie den Abruf von Vektorgeodaten und deren Eigenschaften. Der Standard fördert die Interoperabilität, indem er konsistente Datenabfragen und -zugriffe über verschiedene Implementierungen hinweg ermöglicht. (OGC API - EDR - OGC API workshop 2024).

OGC API – Processes

Der OGC API - Processes Standard ist ein moderner Ersatz für den älteren OGC Web Processing Service (WPS) Standard. Er ermöglicht die Ausführung von Berechnungsprozessen über Web-APIs und den Abruf von Metadaten, die den Zweck und die Funktionalität dieser Prozesse beschreiben. Typischerweise führen diese Prozesse klar definierte Algorithmen aus, um Vektor- und/oder Abdeckungsdaten zu verarbeiten und neue Datensätze zu erzeugen (OGC API - Processes 2023).

OGC API – Coverages

Die OGC API - Coverages ist eine Web-API, die den Zugriff auf räumliche Abdeckungen ermöglicht, wie z.B. Satellitenbilder. Sie ersetzt den älteren Standard Web Coverage Service (WCS). Abdeckungen sind Funktionen, die Werte aus ihrem Bereich für jede Position zurückgeben können. Diese API definiert standardisierte Bausteine für den Umgang mit Abdeckungen und ermöglicht eine konsistente Integration in Webanwendungen. Durch die Nutzung von OpenAPI können verschiedene Datenformate wie JSON und GeoTIFF unterstützt werden (OGC API - Coverages - Overview 2024).

OGC API – Records

OGC API - Records ist im Wesentlichen eine Weiterentwicklung des Catalogue Services for the Web (CSW). Er standardisiert immer noch den Zugriff auf Metadaten über geographische Daten im Web, aber auf eine modernere und flexiblere Weise. Er bietet ähnliche Funktionen wie CSW, wie das Auffinden und Abrufen von Metadaten über geographische Ressourcen, aber mit verbesserten Methoden zur Darstellung und Verwaltung dieser Daten (OGC API - Records 2022).

OGC API – Styles

Die OGC API - Styles definiert einen Standard zur Erstellung und Interpretation von Stilen, der von Style-Editoren für OGC API - Features, OGC API - Tiles und OGC API Coverages genutzt werden kann. Web-Bibliotheken wie Maplibre können diese standardisierten Stile interpretieren und die Daten entsprechend darstellen. Zusätzlich ermöglicht der Standard die Definition von Metadaten zu den Stilen. Er ersetzt den älteren SLD-Standard (OGC API - Styles 2024).

OGC API – Maps

Dieser Standard definiert, wie eine Clientanwendung eine Karte abrufen kann. Dabei wird festgelegt, welche Daten in welchem Stil und Ausschnitt abgerufen werden sollen. Zusätzlich können weitere Parameter wie Zeit, Transparenz, Massstab, Pixeldichte und das Koordinatenbezugssystem festgelegt werden. Dieser Standard stellt den Nachfolger des Web Map Service (WMS) dar (OGC API - Maps - Part 1: Core 2024).

OGC API – DGGS (Discret Global Grid System)

Die OGC API - DGGS ist eine Web-API-Spezifikation, die geografische Daten basierend auf einem Diskreten Globalen Gittersystem (DGGS) abrufen. DGGS unterteilen die Erde in Zellen unterschiedlicher Formen und Grössen, was eine effizientere Speicherung, Analyse und Abfrage von

Geodaten ermöglicht. Die API ermöglicht den Zugriff auf geografische Daten für spezifische Bereiche, Zeiten und Auflösungen. Zusätzlich ermöglicht sie die Abfrage der verfügbaren DGGS-Zonen und das Anpassen von Abfragen mit Filtern (OGC API - Discrete Global Grid Systems - Part 1: Core 2024).

OGC API – Routes

Die API bietet Anbietern eine einfache Möglichkeit, Routen als Ressourcen zu veröffentlichen und anderen Systemen zugänglich zu machen. Entwickler können vorhandene Routing-Engines und Algorithmen effizient nutzen, was Zeit und Kosten spart. Die gemeinsame Nutzung dieser API stellt einen bedeutenden Fortschritt in der geografischen Interoperabilität dar, da Routen unabhängig von den zugrunde liegenden Daten, Engines oder Algorithmen abgerufen werden können (OGC API - Routes - Part 1: Core 2024).

OGC API – Joins

Der OGC API - Joins Standard spezifiziert eine Web-API, die es ermöglicht, Daten entweder mit Feature-Sammlungen, die auf dem Server verfügbar sind, oder direkt mit anderen bereitgestellten Dateien zu verknüpfen. Sie bietet ausserdem Funktionen zur Anzeige von Metadaten und zum Verwalten von Verknüpfungen (OGC API - Joins - Overview 2023).

OGC API – Moving Features

Der OGC API – Moving Features Standard ist eine Erweiterung der OGC API - Common und der OGC API - Features Standards und definiert Methoden und Strukturen für den Zugriff auf sich bewegende geographische Entitäten, wie z.B. Fahrzeuge, Tiere oder andere bewegliche Objekte. Das Ziel besteht darin, eine einheitliche Schnittstelle bereitzustellen, um Daten von sich bewegenden Features zugänglich zu machen und den Austausch dieser Daten zu ermöglichen (OGC API - Moving Features 2024).

OGC API – 3D GeoVolumes

Die OGC API - 3D GeoVolumes ist eine Spezifikation, die eine einheitliche Programmierschnittstelle für den Zugriff auf 3D Inhalte bereitstellt. Die Spezifikation wird benötigt, da verschiedene Lösungen und Standards existieren, um auf 3D-geografische Inhalte zuzugreifen und sie zu übertragen (z. B. 3D-Tiles, I3S, glTF und andere). Die OGC API - 3D GeoVolumes-Spezifikation begegnet dieser Herausforderung, indem sie ein Ressourcenmodell und die entsprechende API bereitstellt, um verschiedene Ansätze zum Zugriff und zur Übertragung von 2D-/3D-geografischen Inhalten in eine einzige, auf offenen Standards basierende Lösung zu integrieren (OGC API - 3D GeoVolumes 2024).

OGC API – SensorThings

Die OGC SensorThings API ist ein offizieller OGC Standard, der einen offenen und einheitlichen Rahmen schafft, um Internet of Things (IoT)-Geräte, Daten und Anwendungen über das Web zu verbinden. Basierend auf den bewährten OGC Sensor Web Enablement (SWE) Standards ermöglicht er die einfache Bereitstellung von Sensor-Daten über das Internet. Beispielsweise könnte ein Netzwerk von Umweltsensoren, das Temperatur, Luftfeuchtigkeit, Luftqualität und andere Umweltparameter überwacht, über die SensorThings API verbunden werden. Anwendungen könnten dann auf diese Daten zugreifen, um Echtzeitinformationen über die Umweltbedingungen bereitzustellen (OGC SensorThings API - Overview 2023).

OGC API – Connected Systems

Diese API erweitert den OGC API - Features-Standard und ermöglicht die Verknüpfung mit anderen OGC API Standards wie z.B. 3D Geo Volumes, 3D Tiles, Coverages, SensorThings und anderen APIs. Dadurch können dynamische Sensordaten beispielsweise mit Features verbunden werden. Die OGC API - Connected Systems fungiert als Brücke zwischen statischen und dynamischen Daten, die von Sensoren gesammelt werden. Während die SensorThings API häufig in Anwendungen zum Einsatz kommt, die speziell auf IoT- und Sensorbereiche ausgerichtet sind, wie Smart Cities, Umweltüberwachungssysteme und Agrartechnologie, bietet die OGC API - Connected Systems eine breitere Anwendungspalette. Sie kann in verschiedenen Szenarien genutzt werden, die sowohl GIS- als auch IoT-Daten integrieren, wie z.B. Umweltüberwachung, Verkehrsmanagement, industrielle Automatisierung und Wasserressourcenmanagement (OGC API - Connected Systems GitHub 2024).

2.1 OGC API – TILES

Der OGC API - Tiles Standard definiert Bausteine, die in der Implementierung von Web-API-basierten Servern und kompatiblen Clients verwendet werden können, um die Abfrage von kachelbasierten Geodaten zu unterstützen. Der Standard übernimmt dabei die Strukturen des (OGC Two Dimensional Tile Matrix Set and Tile Set Metadata 2022) Standards. Dieser legt Regeln und Anforderungen für ein Tile Matrix Set fest, das dazu dient, den Raum anhand einer Gruppe regelmässiger Raster zu indizieren. Der OGC API — Tiles Standard ist eine Alternative zum OGC Web Map Tile Service (WMTS) Standard. Das grundlegende Konzept von TileMatrixSet hat sich im Vergleich zu WMTS nicht geändert. Daher können Kacheln, die über eine WMTS-Instanz bereitgestellt werden, und solche, die von einer OGC API — Tiles Implementierungsinstanz generiert werden und auf demselben TileMatrixSet basieren, problemlos zusammen visualisiert und verarbeitet werden. (OGC API - Tiles - Part 1: Core 2022)

Im Gegensatz zu den OGC Web Map Service (WMS) und WMTS-Standards, bei welchen die Daten in Ebenen (Layer) repräsentiert wurden, erlaubt es der neue OGC API Standard die Daten als Sammlung (collections) aus verschiedenen Datensätzen zu publizieren. So lassen sich nicht nur Rasterdaten, sondern auch Vektordaten als vorgerechnete Kacheln publizieren. (OGC API - Tiles - Part 1: Core 2022)

Die Autoren (Netek et al. 2020) haben acht Pilotanwendungen auf unterschiedliche Geschwindigkeiten und Ladezeiten zwischen Raster- und Vektorkacheln unter Verwendung verschiedener Formate und Backen-Technologien getestet. Die Resultate haben gezeigt, dass Vector Tiles nicht für alle Arten von Daten geeignet sind. Die Hauptherausforderung besteht in der Darstellung von unterschiedlichen Zoomstufen.

2.1.1 Raster Tiles

Um Geodaten als Rasterkacheln bereitzustellen, wandelt ein Server die Geodaten in Bilddateien um. Typischerweise erfolgt dies für ausgewählte Ausschnitte als quadratische Bilder in den Formaten PNG oder JPEG gemäss den definierten Stilen und wird dem Client übergeben. Beim klassischen WMS-Dienst werden diese Bilder dynamisch erstellt und für den benötigten Ausschnitt bereitgestellt. Zur Steigerung der Geschwindigkeit können diese Bilder auch pro Zoomstufe vorgerechnet und als WMTS-Kacheln auf dem Server gespeichert werden. Dadurch muss der Server bei Anfragen die Bilder nicht mehr generieren, sondern kann sie direkt ausliefern. Die Kacheln sind in einem definierten Kachelmatrix-Set (TMS) festgelegt und werden üblicherweise in Grössen von 256 x 256 Pixeln ausgeliefert. Auf Zoomstufe 0 wird normalerweise die gesamte Welt (ohne die Polarregionen) im Web-Mercator-Koordinatensystem (EPSG: 3857) in einer Kachel dargestellt. Mit jeder weiteren

Zoomstufe werden die Kacheln in vier neue Quadrate unterteilt. Dieser Ansatz erfordert zwar die Zwischenspeicherung grosser Datenmengen auf dem Server, bietet jedoch eine effiziente Methode zur Darstellung von Geodaten in Webkarten (Netek et al. 2020).

2.1.2 Vector Tiles

Vektorkacheln funktionieren ähnlich wie Rasterkacheln, indem sie Geodaten in kleinere Quadrate aufteilen und nur die relevanten Daten an den Client übermitteln. Im Gegensatz zu Rasterkacheln enthalten Vektorkacheln keine Bildinformationen, sondern Vektorgeometrien. Diese enthalten keine Pixel, sondern Geometrieobjekte, die auch um zusätzliche Attribute erweitert werden können. Das Tile Matrix Set (TMS) bestimmt, wie die Vektordaten in Quadrate geschnitten werden. Die Vector Tiles können zoomstufenabhängig in verschiedenen Auflösungen, auch als Levels of Detail (LOD) bekannt, bereitgestellt werden. Geometrien können in kleinerem Massstab generalisiert werden, indem bei den Geometrien bestimmte Stützpunkte gelöscht werden, wodurch die Dateigrösse der Kachel reduziert wird. Dadurch können die Vector Tiles schneller übertragen und clientseitig gerendert werden. Einige Vector Tiles Server unterstützen auch das Gruppieren von Layern, wodurch verschiedene Vektordaten in einer Gruppe zusammengefasst und als eine Kachel bereitgestellt werden können (Ingensand et al. 2016).

Die Vorteile von Vektorkacheln gegenüber Rasterkacheln sind vielfältig: Sie sind in der Regel kleiner und daher gut für den Download und die Offline-Nutzung geeignet. Ausserdem wird die Darstellung nicht vom Datenanbieter vorgegeben und in den Kacheln gespeichert, sondern kann vom Client selbst definiert und in Webkartenbibliotheken dargestellt werden. Die meisten Anbieter von Vector Tiles stellen bereits Style-Vorlagen in Form von JSON-Dateien bereit, um die Kacheln entsprechend zu gestalten. Diese Styles können mit Vector Tiles Editoren wie z.B. (Maputnik 2024) bearbeitet oder komplett neu definiert werden. Ein weiterer Vorteil sind die Attributinformationen, die in den Objekten gespeichert werden können, was eine differenzierte Darstellung ermöglicht, beispielsweise bei der unterschiedlichen Darstellung von Strassenachsen je nach Strassentyp.

2.1.2.1 Dateiformate

Protocolbuffer Binary Format (.pbf)

Das Protocolbuffer Binary Format (PBF) ist ein effizientes binäres Datenformat, das als Alternative zum XML-Format entwickelt wurde. Es ermöglicht die kompakte Speicherung und Übertragung von strukturierten Daten (PBF Format – OpenStreetMap Wiki 2023).

Beispielabfrage einer Kachel:

<https://vectortiles.geo.admin.ch/tiles/ch.swisstopo.leichte-basiskarte.vt/v3.0.0/7/67/44.pbf>

Mapbox Vector Tiles (.mvt)

Das Vector Tiles Format MVT mit dem Suffix .mvt basiert auf der Codierung des PBF-Formats. Es wurde von Mapbox als offener Standard für Vektorkacheln definiert (Mapbox 2023a).

Beide Dateiformate (.pbf und .mvt) verwenden die gleiche Codierung und sind für die Speicherung von Vector Tiles geeignet. Der einzige Unterschied besteht in den Dateieendungen.

MabBoxTiles (.mbtiles)

MBTiles ist ein Datenformat, das es ermöglicht, Kacheln in einer einzigen SQLite-Datenbankdatei zu speichern. Dabei werden Kacheln für verschiedene Zoomstufen in einer einzelnen Datei gespeichert. Dieses Format unterstützt sowohl Raster- als auch Vektorkacheln (Mapbox 2023b).

Beispielabfrage eines Kachelarchivs:

<https://vectortiles.geo.admin.ch/tiles/ch.swisstopo.leichte-basiskarte.vt/v3.0.0/ch.swisstopo.leichte-basiskarte.vt.mbtiles>

Protomaps (.pmtiles)

PMTiles ist ein neueres Datenformat zur Speicherung von gekachelten Daten in einer Datei. Es wurde vom Datenformat Cloud Optimized Geotiff (COG) inspiriert, unterscheidet sich jedoch darin, dass PMTiles nicht nur TIFF-Formate unterstützt. Je nach Anwendungsfall kann es sinnvoll sein, anstelle von PMTiles ein COG zu verwenden. Die Dokumentation von Protomaps erwähnt, dass PMTiles als COG für Vector Tiles betrachtet werden kann. PMTiles wurde so konzipiert, dass es nur einmal beschrieben werden kann. Daher ist es nicht möglich, einzelne Kacheln zu ersetzen; stattdessen muss die gesamte PMTiles-Datei bei einer Datennachführung neu erstellt werden. (Protomaps 2023).

2.1.2.2 ST_AsMVT

Mit der PostGIS Version 2.4.0 ist es mit den Funktionen ST_AsMVT und ST_AsMVTGeom nun möglich, aus den Geodaten Vector Tiles zu erstellen (ST_AsMVT 2023). Mit den Funktionen werden bei einer Kachelanfrage von einem Client die Vector Tiles im entsprechenden Ausschnitt auf der entsprechenden Zoomstufe dynamisch erstellt und dem Client übergeben. Dabei lassen sich die Geometrien mittels PostGIS Funktionen für verschiedene Zoomstufen in unterschiedlichen Detaillierungsgrade übergeben.

In kleinen Massstäben, beispielsweise bis Zoomstufe 10, können die Geometrien in den Kacheln mithilfe der PostGIS-Funktion ST_Simplify generalisiert werden. Der Algorithmus entfernt dabei nicht benötigte Knoten einer Geometrie, um die Datenmenge zu reduzieren. Für Zoomstufen über 10 könnten die Geometrien ohne Generalisierung als Vector Tiles übergeben werden, wodurch die Originalgeometrien für grössere Massstäbe angezeigt würden. Dies würde jedoch zu einer erhöhten Datenmenge aufgrund zusätzlicher Knoten führen, weshalb ein Kompromiss zwischen Performance und Genauigkeit gefunden werden muss. Wallner et al. (2022) weist jedoch darauf hin, dass mit der ST_Simplify Funktion die Topologie nicht mitberücksichtigt werden und somit bei Flächen Überlappungen und Löcher entstehen können. Eine Funktion, welche die Topologie mitberücksichtigt, wäre ST_SimplifyPreserveTopology.

Weil für die ST_AsMVT-Funktion (ST_AsMVT 2023) die Geometrie im Kachelkoordinatensystem vorhanden sein muss, müssen die Geometrien vorgängig mittels der ST_AsMVT-Funktion transformiert werden (ST_AsMVTGeom 2023).

3 METHODIK

Für die Performanceuntersuchung von Vector Tiles Serverlösungen werden die kantonalen Geodaten der Amtlichen Vermessung vom Amt für Geoinformation Thurgau als Testdaten verwendet. Diese umfassen diverse Tabellen, die in einer PostGIS-Datenbank gespeichert sind.

Um eine konsistente Performanceuntersuchung verschiedener Serverlösungen zu ermöglichen, wird für jede Server dieselbe PostGIS-Datenbank als Datenquelle genutzt. Zu diesem Zweck wird eine Public Cloud beim Schweizer Cloudanbieter Infomaniak eingerichtet, auf der eine Linuxinstanz bereitgestellt wird (Infomaniak 2023).

In Kapitel 3.3 Vector-Tiles-Server werden verschiedene Serverlösungen für die Erstellung und Bereitstellung von Vector Tiles untersucht. Dabei werden Funktionen, Vor- und Nachteile sowie Einsatzmöglichkeiten für unterschiedliche Datenmengen und Anforderungen beleuchtet.

3.1 GRUNDLAGEDATEN

Die verwendeten Grundlagedaten stammen von der Amtlichen Vermessung (AV) des Kantons Thurgau. Diese wurden kostenlos vom Amt für Geoinformation Thurgau (AGI) bereitgestellt. Aufgrund der Beschaffenheit der Originaldaten der Amtlichen Vermessung, die häufig Bögen enthalten, mussten vor dem Import in die PostGIS-Datenbank bestimmte Vorkehrungen getroffen werden.

Da die PostGIS-Datenbank zwar das Speichern von Bögen ermöglicht, jedoch Probleme bei weiteren Bearbeitungsschritten auftreten können, wurden die Daten segmentiert. Das AGI führte diese Segmentierung durch, wobei alle Stützpunkte auf einen Millimeter gerundet wurden. Der minimale Abstand zwischen den Stützpunkten beträgt somit 1 mm.

Die segmentierten Grundlagedaten sind im Schema „avprodukt“ in insgesamt 68 Tabellen gespeichert, darunter:

- bo_boflaeche: Bodenbedeckung (Polygon)
- gg_gemeindegrenze: Gemeindegrenze (Polygon)
- eo_linienelement: Einzelobjekte (LineString)
- li_grenzpunkt: Grenzpunkte (Point)
- li_liegenschaft: Liegenschaft (Polygon)

Alle Tabellen verwenden als eindeutigen Identifikator das Attribut „objectid“ und als Geometrie das Attribut „wkb_geometry“. Die Geometrien werden im Landeskoordinatensystem der Schweiz LV95 (EPSG: 2056) gespeichert. Zusätzlich enthalten die Tabellen weitere Attribute, die für die weiteren Berechnungsschritte der Vector Tiles nicht benötigt werden.

Um die Datenmenge der Vektordaten und somit auch die der einzelnen Vector Tiles zu reduzieren, wurden nur die für die Darstellung benötigten Attribute verwendet. Dies wurde durch die Erstellung separater Materialized Views erreicht, die jeweils nur die erforderlichen Attribute enthalten.

Das folgende SQL-Skript veranschaulicht die Definition der Tabelle „bo_boflaeche“ im Schema „avprodukt“.

```
-- Table: avprodukt.bo_boflaeche
-- DROP TABLE IF EXISTS avprodukt.bo_boflaeche;

CREATE TABLE IF NOT EXISTS avprodukt.bo_boflaeche
(
    objectid integer NOT NULL DEFAULT
nextval('avprodukt.bo_boflaeche_objectid_seq'::regclass),
    gid character varying COLLATE pg_catalog."default",
    entstehung character varying COLLATE pg_catalog."default",
    objektnummer_nummer character varying COLLATE pg_catalog."default",
    objektname_name character varying COLLATE pg_catalog."default",
    qualitaet character varying COLLATE pg_catalog."default",
    art character varying COLLATE pg_catalog."default",
    bbn_gueltigereintrag date,
    ist_waldgebiet character varying(200) COLLATE pg_catalog."default",
    wkb_geometry geometry(Polygon,2056),
    CONSTRAINT bo_boflaeche_pkey PRIMARY KEY (objectid)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS avprodukt.bo_boflaeche
    OWNER to postgres;
```

```
-- Index: bo_boflaeche_wkb_geometry_1458028526959
-- DROP INDEX IF EXISTS avprodukt.bo_boflaeche_wkb_geometry_1458028526959;

CREATE INDEX IF NOT EXISTS bo_boflaeche_wkb_geometry_1458028526959
  ON avprodukt.bo_boflaeche USING gist
  (wkb_geometry)
  TABLESPACE pg_default;
```

Ein Beispiel für eine solche Materialized View ist die Tabelle „avprodukt.bo_boflaeche_mv“, welche nur die Attribute „objectid“, „art“ und „wkb_geometry“ enthält.

```
-- View: avprodukt.bo_boflaeche_mv
-- DROP MATERIALIZED VIEW IF EXISTS avprodukt.bo_boflaeche_mv;

CREATE MATERIALIZED VIEW IF NOT EXISTS avprodukt.bo_boflaeche_mv
TABLESPACE pg_default
AS
  SELECT objectid,
         art,
         wkb_geometry
  FROM avprodukt.bo_boflaeche
WITH DATA;

ALTER TABLE IF EXISTS avprodukt.bo_boflaeche_mv
  OWNER TO postgres;

GRANT ALL ON TABLE avprodukt.bo_boflaeche_mv TO postgres;
GRANT SELECT ON TABLE avprodukt.bo_boflaeche_mv TO tileserver;

CREATE UNIQUE INDEX bo_boflaeche_mv_objectid_idx
  ON avprodukt.bo_boflaeche_mv USING btree
  (objectid)
  TABLESPACE pg_default;
```

Die SQL-Skripte, welche verwendet wurden, sind im [GitHub-Repository](#) zu finden.

3.2 PUBLIC CLOUD

Beim Erstellen eines neuen Kontos bei Infomaniak erhält man ein Startguthaben von 300 Franken, um die Public Cloud für einige Monate zu testen. Die Kosten für den Betrieb des eingerichteten Servers werden übersichtlich dargestellt und monatlich vom Guthaben abgezogen. Während der gesamten Masterthesis von Oktober bis Mai beliefen sich die Kosten auf insgesamt 160 Franken, die somit innerhalb des Startguthabens lagen.

In der Administrationsumgebung der Public Cloud können sogenannte Projekte erstellt werden, in denen die individuelle Cloud konfiguriert werden kann. Dabei wird das Cloud-Betriebssystem OpenStack genutzt (OpenStack Docs: 2024). Innerhalb des Projekts wurde eine neue Instanz mit Ubuntu 22.04 LTS Jammy Jellyfish, 4 VCPUs, 8GB RAM und 50GB Speicherplatz aufgesetzt. Es besteht die Möglichkeit, weitere Instanzen im Projekt zu erstellen, jedoch ist die Anzahl auf maximal 10 Instanzen, insgesamt 20 VCPUs und 64GB RAM begrenzt.

Mittels SSH-Keys, welche auf dem Server sowie auf dem Notebook gespeichert wurden, konnte mit Visual Studio Code (VSCoDe) eine direkte Verbindung in die Linuxumgebung des Servers aufgebaut werden.

3.2.1 Varianten Cloud-Umgebung

Beim Einrichten der Cloudumgebung wurden zwei Ansätze berücksichtigt. Bei der Variante 1 wird ein Linux-Server konfiguriert, auf dem verschiedene Anwendungen in separaten Docker-Containern gestartet werden. Die Grundlagedaten werden in einem eigenen Docker-Container namens „postgis“ gespeichert. Die Vector Tiles Server, die auf ihre Leistung hin untersucht werden sollen, werden jeweils in separaten Docker-Containern definiert, wobei jeder dieser Container über einen eigenen Port erreichbar ist. Zusätzlich zu den 6 Anwendungen wurden zwei weitere Container eingerichtet: „nginx“ dient als Webserver, um die Anwendungen und weitere HTML-Dateien extern zugänglich zu machen, und der Container „maputnik“ wird als Vector Tiles Style-Editor verwendet, um die Darstellung der Vector Tiles zu definieren.

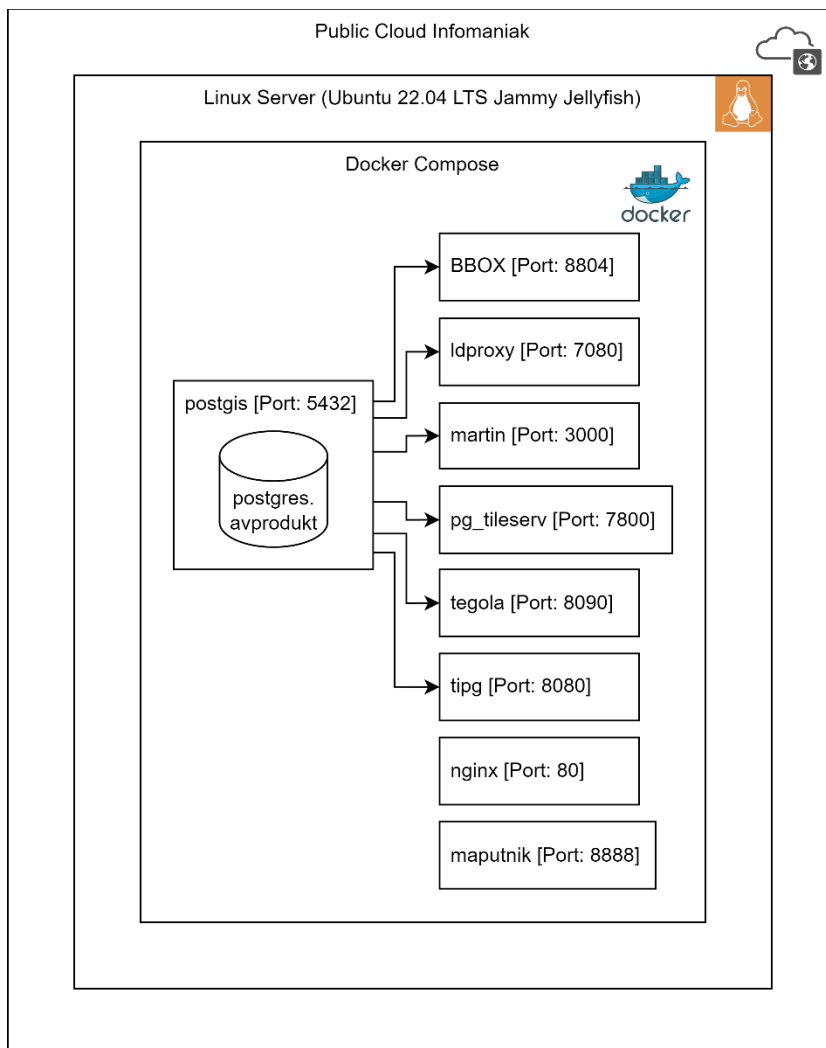


Abbildung 1: Cloud-Umgebung (Variante 1)

In Variante 2 wurde die Möglichkeit in Betracht gezogen, für jede Applikation eine separate Linux-Instanz zu erstellen. Auf allen 6 Instanzen wäre dann jeweils ein postgis- und ein Applikationscontainer erforderlich. Um sicherzustellen, dass die Datengrundlage für den Leistungstest konsistent ist, müssten die „postgis“-Container auf jeder Instanz identisch sein.



Abbildung 2: Cloud-Umgebung (Variante 2)

Für den Performancetest der sechs Vector Tiles Server wurde die Variante 1 in der Cloud-Umgebung umgesetzt. Hierbei können die verschiedenen Server leicht als separate Container mittels Docker Compose gestartet und gestoppt werden. Zusätzlich muss bei der Variante 1 die PostGIS-Datenbank nur einmal aufgesetzt werden, und alle Server verwenden die exakt gleichen Matviews als Datenquelle. Im Gegensatz dazu hätte bei der Variante 2 für jede Linuxinstanz der „postgis“-Container separat kopiert und Anpassungen an der Datenbank auf allen Instanzen vorgenommen werden müssen. Ein weiterer Vorteil der Variante 1 liegt in den geringeren Serverinstanzen und den damit verbundenen Kosten im Vergleich zur Variante 2, die sechsmal höher wären. Zudem ist die Testumgebung der Variante 1 einfach aufgebaut und lässt sich leicht auf andere Anwendungen anpassen.

Ein Nachteil der Variante 1 im Vergleich zur Variante 2 besteht darin, dass der Performancetest der Server nicht parallel ausgeführt werden kann. Dies liegt daran, dass die Server gleichzeitig auf die gleiche Datenbank zugreifen würden, wodurch sich die Anwendungen gegenseitig beeinflussen und

die Performance verfälschen könnten. Daher ist es bei Variante 1 zwingend erforderlich, dass der Performancetest nacheinander durchgeführt wird.

3.2.2 Docker Compose

Mittels Admin-Rechte wurde Docker (Version 24.0.6) mit der Installationsmethode „Install using the apt repository“ gemäss (Docker Documentation 2024a) installiert. Mit der Installation der neusten Docker-Version wurde automatisch auch Docker Compose V2 (Version 2.21.0) mitinstalliert. Bei älteren Versionen war das nicht der Fall, hier musste Docker-Compose V1 zusätzlich installiert werden (Docker Documentation 2024b).

Das [GitHub-Repository dieser Masterthesis](#) enthält eine env-Datei, in der wichtige Umgebungsvariablen definiert sind, darunter Benutzernamen und Passwörter für den Zugriff auf die PostGIS-Datenbank. Diese .env-Datei sollte geschützt und nicht in das Repository hochgeladen werden. Durch die Verwendung der zusätzlichen Datei gitignore kann festgelegt werden, welche Dateien und Verzeichnisse nicht mit dem GitHub-Repository synchronisiert werden sollen.

3.2.2.1 compose.yaml

Im GitHub Repository wurde das [compose.yaml](#) File angelegt, womit alle Container parametrisiert und konfiguriert werden. Mithilfe dieser Datei können Container sehr einfach gestartet, gestoppt, aktualisiert oder gelöscht werden.

Der folgende Codeausschnitt aus der compose.yaml Datei definiert die Parameter des „postgis“ Containers.

```
postgis:
  image: postgis/postgis:${POSTGRES_VERSION:-16-3.4-alpine}
  container_name: postgis
  profiles:
  ["postgis", "bbox", "pg_tileserv", "tipg", "ldproxy", "tegola", "martin", "all_server"]
  environment:
    #- POSTGRES_USER=${POSTGRES_ADMIN_USERNAME:-postgres} # default
    - POSTGRES_PASSWORD=${POSTGRES_ADMIN_PASSWORD:-admin_password}
    #- POSTGRES_DB=${POSTGRES_DB:-postgres} # default, if used, it
    creates a second DB with name postgres -> conflicts with sql imports.
  volumes:
    - ./postgis-data:/var/lib/postgresql/data
  ports:
    - 5432:5432
```

Für das Image wurde die neueste Version von PostGIS verwendet (Docker - postgis/postgis:16-3.4-alpine 2024). Die Versionsangabe kann entweder in der env-Datei oder direkt in der compose.yaml-Datei überschrieben werden. Im Parameter „profiles“ werden Profile definiert, die genutzt werden können, um mehrere Container gleichzeitig zu starten. Neben dem Profil „postgis“ sind auch die Namen der weiteren Container-Applikationen aufgelistet. Der environment-parameter definiert zusätzliche Parameter, die dem Container übergeben werden und von der entsprechenden Applikation interpretiert werden. Die Syntax und der Inhalt der Parameter variieren je nach Server, in den meisten Fällen wird jedoch nur der Zugriff auf die PostGIS-Datenbank definiert.

3.3 VECTOR-TILES-SERVER

Es gibt zahlreiche Serverlösungen zur Erstellung von Vector-Tiles. Eine Liste dieser Applikationen findet man im GitHub-Repository von Mapbox (<https://github.com/mapbox/awesome-vector-tiles>). Neben den Servern gibt es auch weitere kleinere Tools zur Bearbeitung von Vector Tiles sowie Clientlösungen mit welchen Vector Tiles eingebunden und visualisiert werden können.

Die Serverlösungen bieten unterschiedliche Funktionen und weisen jeweils ihre eigenen Vor- und Nachteile auf. Einige der leichteren Serverlösungen ermöglichen den direkten Zugriff auf die PostGIS-Funktion `ST_AsMVT`, womit PostGIS-Daten unmittelbar als Vector Tiles ausgegeben werden können. Dabei werden die Kacheln direkt in der Datenbank erstellt und dem Client dynamisch zur Verfügung gestellt. Im Gegensatz dazu nutzen andere Serverlösungen zwar die PostGIS-Datenbank als Quelle, berechnen jedoch die Kacheln serverseitig im Voraus, also statisch. Je nach Server wird auch die Möglichkeit unterstützt, dass bei Änderungen an den PostGIS-Daten die entsprechenden Vector Tiles automatisch aktualisiert werden.

Für umfangreiche Datenmengen wie Hintergrundkarten lohnt es sich, die Vector Tiles im Voraus zu berechnen, während für Geodaten mit wenigen Geometrien wie Gemeindegrenzen eine dynamische Bereitstellung ausreicht.

Applikationen vom Typ „Generator“ eignen sich zur Erzeugung von Vector-Tiles, jedoch nicht zur direkten Bereitstellung an den Client. Die meisten Server können jedoch die Vector Tiles generieren, zwischenspeichern und als Tileserver dem Client zur Verfügung stellen.

Vector-Tiles sind primär zur Visualisierung von Geodaten vorgesehen, können jedoch auch Attributdaten speichern. Um die Kacheln klein und performant zu halten, sollten nur die für die Visualisierung erforderlichen Attribute enthalten sein. Zusätzliche Attribute können über den Standard OGC API - Features bereitgestellt werden.

Einige Vector Tiles Server können auch direkt OGC API - Features bereitstellen, während andere zusätzliche Softwarelösungen dafür nutzen. Wieder andere Server sind ausschliesslich für die Erstellung und Bereitstellung von Vector Tiles konzipiert.

Einige Server sind speziell für die Bereitstellung von OpenStreetMap-Daten als Vector Tiles ausgelegt. Diese wurden in dieser Masterarbeit jedoch nicht weiter untersucht, da das Ziel die Bereitstellung eigener PostGIS-Daten war.

Für die Auswahl geeigneter Vector Tiles Server zur Performanceuntersuchung wurden folgende Kriterien definiert:

- Die Applikation steht kostenlos als Open Source zur Verfügung.
- Der Server unterstützt PostGIS als Datenquelle.
- Vector-Tiles lassen sich dynamisch generieren und bereitstellen.
- Der Server steht als Docker-Image zur Verfügung.

Aus der Liste des GitHub-Repository „awesome-vector-tiles“ wurden folgende Applikationen herausgefiltert, die diese Kriterien erfüllen:

- Baremaps
- BBOX
- Clusterbuster
- FastVector
- Geoserver
- Ldproxy

- Mapserver
- Martin
- Pg_tileserv
- T-rex
- Tegola
- TiPg
- Tippcanoe

Aufgrund des Umfangs dieser Masterarbeit ist es nicht möglich, alle diese Server zu vergleichen und auf ihre Leistung zu untersuchen. Daher wurden die folgenden sechs Server ausgewählt, um sie im Detail zu vergleichen und anschliessend in der Cloud als Docker-Container bereitzustellen, um ihre Performance zu analysieren:

- BBOX
- Ldproxy
- Martin
- Pg_tileserv
- Tegola
- TiPg

Weitere Vector Tiles Applikationen, die in dieser Arbeit nicht weiter untersucht wurden, sind:

- ArcGIS Online
- Cloud-Tileservers
- djangoestframework-mvt
- Hastile
- Kartotherian
- LOD
- Mbtileservers
- OSM Scout Server
- Portal for ArcGIS
- Postserver
- Tessella
- Tessera
- Tilenol
- TileServer
- TileServer GL
- Tilesplach
- Tilecache
- TileStache
- TileStrata
- Tyler
- SpatialServer
- SpatialDev
- Utilery
- ngx_http_mbtiles_module
- Vallaris Maps

3.3.1 Übersicht Vector Tiles Server

Name	BBOX	ldproxy	Martin	pg_tileserv	Tegola	TiPg
Entwickler	Sourcepole	interactive instruments	MapLibre	CrunchyData	Go Spatial	Development Seed
Programmiersprache	Rust	Java	Rust	Go	Go	Python
Quellformate	PostGIS, MBTiles, PMTiles	HTTP, MBTiles, PostGIS, GPKG, SQLite, WFS, (GraphQL)	PostGIS, MBTiles, PMTiles	PostGIS	PostGIS, GPKG, SAP HANA Spatial	PostGIS
Ausgabedatenformat	MVT, MBTiles, PMTiles	MVT, MBTiles	MVT, MBTiles	MVT	MVT	MVT
Erstellungsmethode	ST_AsMVT	Feature Provider	ST_AsMVT	ST_AsMVT	ST_AsMVT	ST_AsMVT
Unterstützte Kachelschemas	WebMercatorQuad (3857) Benutzerdefiniert	WebMercatorQuad (3857) Benutzerdefiniert	WebMercatorQuad (3857)	WebMercatorQuad (3857)	WebMercatorQuad (3857)	WebMercatorQuad (3857) Benutzerdefiniert
Filterfunktionen	✓	✓	✓	✓	✓	✓
Multi-Layer-Tiles	✓	✗	✓	✓	✓	✗
Caching	✓	✓	✓	✗	✓	✗
OGC API - Features	✓	✓	✗	✗	✗	✓
GitHub	(BBOX - GitHub 2024)	(ldproxy - GitHub 2024)	(Martin - GitHub 2024)	(pg_tileserv - GitHub 2024)	(Tegola - GitHub 2024)	(TiPg - GitHub 2024)
Docker-Image	(sourcepole/bbox-server-qgis - Docker Image 2024)	(iide/ldproxy - Docker Image 2024)	(martin - Docker Image 2024)	(pramsey/pg_tilese rv - Docker Image 2024)	(gospatial/tegola - Docker Image 2024)	(tipg - Docker Image 2024)
Weitere Dokumentationen	(BBOX - Documentation 2024)	(ldproxy - Documentation 2024)	(Martin - Documentation 2024) (Martin - Webpage 2023)	(pg_tileserv - Documentation 2023)	(Tegola - Documentation 2017) (Tegola - Webpage 2017)	(TiPg - Documentation 2024)

Tabelle 1: Übersicht Vector Tiles Server

4 IMPLEMENTIERUNG DER VECTOR TILES SERVER

In diesem Kapitel werden die Vector Tiles Server-Applikationen aus der Tabelle 1: Übersicht Vector Tiles Server beschrieben und erklärt, wie sie mittels Docker gemäss dem Schema in Abbildung 1: Cloud-Umgebung (Variante 1) aufgesetzt und konfiguriert wurden.

Auf der aufgesetzten Linuxumgebung wurde die folgende Ordnerstruktur erstellt, die im GitHub Repository <https://github.com/FabianRechsteiner/vector-tiles-benchmark> bereitgestellt wurde:

vector-tiles-benchmark

- bbox
 - bbox.toml
- ldproxy
 - cfg.yml
- martin
 - config.yaml
- nginx
 - html
 - nginx.conf
- pg_tileserv
 - pg_tileserv.toml
- sql-scripts
- tegola
 - config.toml
- tipg
 - .env
- vector-tiles-benchmark
 - test_plans
 - test_results
- .env
- .gitignore
- README.md
- compose.yaml

Die Datei „compose.yaml“ enthält die Konfigurationen aller Docker-Container. Die Konfigurationen der einzelnen Container werden in den folgenden Unterkapiteln erläutert. Für die Container-Images wurden jeweils die aktuellen Versionen zum Stand vom 23.02.2024 verwendet. Alternativ kann auch das neueste Image `latest` anstelle der Versionsnummer verwendet werden.

Jedem Container wurde ein Profil zugewiesen, mit dem der Container zusammen mit der PostGIS-Datenbank gestartet und gestoppt werden kann. Mit dem folgenden Befehl können die entsprechenden Container gestartet werden: `docker compose --profile profilname up -d`. Der Parameter `-d` oder `--detach` startet die Container im Hintergrund (detached mode), was bedeutet, dass keine Log-Ausgaben in der Kommandozeile angezeigt werden. Ohne diesen Parameter werden die Log-Ausgaben in der Kommandozeile angezeigt, aber der Container wird automatisch gestoppt, sobald die Kommandozeile geschlossen wird.

4.1 BBOX

[BBOX Server](#) ist eine Open-Source-Webserveranwendung zur Bereitstellung von Geodaten. Sie wurde von der Schweizer Firma [Sourcepole AG](#) entwickelt und unterstützt verschiedene OGC API Dienste wie OGC API - Features, Maps und Tiles, sowie Prozesse. Der Server kann als Mikroservice oder als integriertes System genutzt werden und bietet eine gemeinsame Kernfunktionalität für alle Dienste (BBOX - Documentation 2024).

Die Implementierung in Rust ermöglicht hohe Leistung und Zuverlässigkeit. BBOX nutzt standardisierte Endpunkte und unterstützt neben den neuen OGC API Standards auch klassische wie OGC WMS 1.3 und WMTS. Dank seiner modularen Architektur und der Unterstützung von verschiedenen Backends ist BBOX flexibel und erweiterbar. Zukünftige Entwicklungen könnten zusätzliche Funktionen wie Metadaten Dienste, Volltextsuche und Unterstützung für Story Maps umfassen (Kalberer 2023).

In seinem aktuellen Blog „Sabbatical - The Plan“ erwähnt (Kalberer), dass BBOX bereits einen Grossteil der Funktionalitäten seines alten Rust-Tile-Servers (t-rex - GitHub 2024) implementiert hat, jedoch noch weiterentwickelt werden muss, um eine umfassende All-in-One-Lösung anzubieten.

compose.yaml

```
bbox:
  image: sourcepole/bbox-server-qgis:v0.5.0-alpha5 #latest
  container_name: bbox
  environment:
    - PGPASSWORD=${POSTGRES_USER_PASSWORD:-user_password}
    - PGUSER=${POSTGRES_USER_USERNAME:-user_name}
    - PGDATABASE=${POSTGRES_DB:-postgres_db}
    - PGHOST=${POSTGRES_HOST:-postgres_host}
    - BBOX_WEBSERVER_CORS_ALLOW_ALL_ORIGINS=true
  profiles: ["bbox", "all_server"]
  ports:
    - 8804:8080
  volumes:
    - ./bbox:/var
    - ./bbox/assets:/assets
  depends_on:
    - postgres
```

Codeausschnitt „bbox“ aus dem File [compose.yaml](#).

Konfigurationsfile

Das Konfigurationsfile ist im Verzeichnis [bbox/bbox.toml](#) gespeichert.

Im vorliegenden Konfigurationsfile sind alle Parameter für die modularen Server (Feature-Server, Map-Server, Tile-Server, Asset-Server, Processes-Server, Routing-Server) definiert. Aktuell sind nur der Feature-Server und Tile-Server für den Performancetest eingerichtet.

Zunächst werden die Webserver-Parameter festgelegt. Da der Standard-Port 8080 des BBOX-Docker-Images bereits belegt ist, musste dieser angepasst werden. Daher wurde der Parameter „server_addr“ entsprechend geändert.

```
# -- webserver settings --
[webserver]
# Environment variable prefix: BBOX_WEBSERVER__
server_addr = "127.0.0.1:8804" # Default: 127.0.0.1:8080
# worker_threads = 4 # Default: number of CPU cores
```

Für den Feature-Server ist es erforderlich, die Verbindung zur PostGIS-Datenbank anzugeben, um alle Tabellen automatisch als Feature-Collections bereitzustellen. Dazu muss auch die entsprechende Datenquelle für den Parameter „collections.postgis“ definiert werden.

```
# -- Feature-Server --
# -- datasources --
[[datasource]]
name = "postgres"
[datasource.postgis]
url = "postgres://@postgis/"
#postgres://user_name:user_password@postgis/postgres_db

# -- collections with auto discovery --
[[collections.postgis]]
url = "postgres://@postgis/"
#postgres://user_name:user_password@postgis/postgres_db
[[collections.directory]]
dir = "../assets"
```

Eine automatische Bereitstellung der Tabellen als Vector Tiles ist für den Tile-Server derzeit nicht möglich. Daher müssen die entsprechenden Datenquellen mit Angaben wie Tabellename, Geometriefeld, eindeutigem Identifikator und SRID der Daten angegeben werden. Standardmässig werden die Geodaten automatisch nach EPSG:3857 transformiert und im Standard TileMatrixSet (TMS) WebMercatorQuad bereitgestellt. BBOX unterstützt jedoch auch benutzerdefinierte TMS, die als JSON-Datei im Verzeichnis „assets“ gespeichert werden können. Um die Kacheln im benutzerdefinierten TMS bereitzustellen, muss der Parameter „tms“ im Tileset definiert werden, wobei der Name der ID im TMS-JSON-File entspricht.

```
# -- Tile-Server --
# -- custom tile grids --
[[grid]]
json = "../assets/SwissLV95CellSizes.json"

# -- vector tiles from PostGIS table --
[[tileset]]
name = "bo_boflaeche_mv"
# tms = "SwissLV95CellSizes"
[tileset.postgis]
datasource = "postgres"
attribution = "Bodenbedeckung Amtliche Vermessung Thurgau"

[[tileset.postgis.layer]]
name = "bo_boflaeche_mv"
table_name = "avprodukt.bo_boflaeche_mv"
geometry_type = "POLYGON"
geometry_field = "wkb_geometry"
fid_field = "objectid"
srid = 2056
```

Container starten

Server starten: `docker compose --profile bbox up -d`

Server beenden: `docker compose --profile bbox down`

Landing Page BBOX: http://localhost:8804/collections/bo_boflaeche_mv/items



BBOX		
bo_boflaeche_mv		
JSON		
id	art	objectid
	befestigt.Strasse_Weg	66417534
	Gebaeude	66417535
	befestigt.Strasse_Weg	66417536
	Gebaeude	66233660
	Gebaeude	66233661
	humusiert.Gartenanlage	66355933
	Gebaeude	66355934
	befestigt.Strasse_Weg	66355935
	humusiert.uebrige_humusierete	66355936
	befestigt.uebrige_befestigte	66355937
	bestockt.uebrige_bestockte	66355938
	Gebaeude	66472674
	Gebaeude	66417537

Abbildung 3: BBOX items bo_boflaeche_mv

Vector-Tiles-Anfrage (Beispiel: bo_boflaeche_mv):

`http://localhost:8804/xyz/bo_boflaeche_mv/{z}/{x}/{y}.pbf`

4.2 LDPROXY

[Ldproxy](#) ist eine Webanwendung, die von der Firma [Interactive Instruments](#) entwickelt wurde. Diese Firma hat ihren Sitz in Bonn und wurde im Jahr 1985 gegründet. Die Entwicklung von Ldproxy begann im Jahr 2015, seither wurde die Anwendung kontinuierlich weiterentwickelt und es wurden verschiedene OGC API Standards integriert, um Geodaten einfach über eine moderne Web-API bereitzustellen (Ldproxy - GitHub 2024).

Die Software Ldproxy hat das Ziel, Geodaten über moderne Web-APIs zugänglich, interoperabel und wiederverwendbar zu machen. Sie zeichnet sich durch eine Vielzahl von Merkmalen aus, die ihre Funktionalität und Benutzerfreundlichkeit verbessern. Die Benutzeroberfläche von Ldproxy ist so gestaltet, dass sie einfach zu bedienen ist. Die unterstützten APIs bieten verschiedene Formate wie JSON und HTML, wodurch Entwickler die Daten leicht abrufen und nutzen können. Insbesondere wird darauf geachtet, dass die HTML-Ausgaben intuitiv verständlich sind, sodass Benutzer die APIs direkt im Webbrowser verwenden können. Die Plattform folgt den geltenden Standards und ist insbesondere eine umfassende Implementierung der OGC API Standards. Dies bedeutet, dass Ldproxy von Clients direkt genutzt werden kann, wobei eine Vielzahl von Formaten wie GeoJSON, Mapbox Vector Tiles und TileJSON unterstützt werden (Ldproxy - Documentation 2024).

Ldproxy unterstützt drei Arten zur Bereitstellung von Vector-Tiles, wobei es drei Arten von Feature-Providern gibt. Für die Bereitstellung der PostGIS-Daten als Vector Tiles wird die Variante Features/SQL verwendet.

- **Features:** Die Kacheln werden aus einem Feature-Provider abgeleitet.
 - **SQL:** Die Features sind in einer SQL-Datenbank gespeichert (PostgreSQL/PostGIS, GeoPackage, SQLite/Spatialite).
 - **WFS:** Die Features werden von einem OGC WFS bezogen.
 - **GraphQL:** Die Features werden von einer GraphQL API bezogen.
- **MbTiles:** Die Kacheln liegen in einer oder mehreren MBTiles-Dateien vor.
- **HTTP:** Die Kacheln werden via HTTP abgerufen, z.B. von einer TileServer GL Instanz.

Neben dem Standard-Kachelschema (TMS) WebMercatorQuad können Vector Tiles auch in einem benutzerdefinierten Kachelschema bereitgestellt werden. Das TMS-File muss dafür im Verzeichnis „tile-matrix-sets“ im JSON-Format abgelegt werden (Ldproxy - Tiles 2024).

Ldproxy unterstützt auch das Caching, um die Vector Tiles entweder vorab oder bei einem ersten Aufruf zwischenspeichern. Dafür wird automatisch im Verzeichnis „cache“ ein neues Verzeichnis pro Tabelle angelegt und die Kacheln als mbtiles abgelegt.

compose.yaml

```
Ldproxy:
  image: iide/Ldproxy:3.6.3
  container_name: Ldproxy
  environment:
    - POSTGRES_USER=${POSTGRES_USER_USERNAME:-user_name}
    - POSTGRES_USER_PASSWORD_BASE64=${POSTGRES_USER_PASSWORD_BASE64}
    - POSTGRES_DB=${POSTGRES_DB:-postgres_db}
    - POSTGRES_HOST=${POSTGRES_HOST:-postgres_host}
  volumes:
    - ./Ldproxy:/Ldproxy/data
  depends_on:
    - postgis
  profiles: ["Ldproxy", "all_server"]
  ports:
    - 7080:7080
```

Codeausschnitt „Ldproxy“ aus dem File [compose.yaml](#).

Im Gegensatz zu den anderen Vector Tiles Server benötigt Ldproxy ein base64 codiertes Passwort. Daher muss der Parameter POSTGRES_USER_PASSWORD_BASE64 anstelle von POSTGRES_USER_PASSWORD definiert werden.

Konfigurationsfile

Bei Ldproxy werden die Serverparameter mithilfe mehrerer Dateien definiert. Dies ermöglicht die Festlegung übergeordneter Parameter, die für alle weiteren Provider und Dienste gelten. Diese Parameter können dann auf den unteren Ebenen erneut definiert werden, um die Standardparameter zu überschreiben.

Im Verzeichnis [Ldproxy/store](#) wurden die folgenden Konfigurationsdateien erstellt:

- Store
 - cfg.yml
 - defaults
 - services
 - ogc_api.yml
 - entities
 - providers
 - avprodukt.yml
 - geodienste.yml
 - Services
 - avprodukt.yml

cfg.yml: In dieser Datei werden die globalen Servereinstellungen wie beispielsweise Port oder Log-Datei definiert.

defaults/services/ogc_api.yml: In dieser Datei werden allgemeine Informationen zur API sowie Parameter, die standardmässig für alle Dienste gelten sollen, festgelegt.

entities/providers/avprodukt.yml: Im Verzeichnis „providers“ werden die verschiedenen Datenquellen jeweils in einer YAML-Datei konfiguriert. In diesem File wird das Schema "avprodukt" aus der PostGIS-Datenbank als Datenquelle verwendet. Dazu müssen der providerType „FEATURE“ und providerSubType „SQL“ definiert werden. Darüber hinaus werden in dieser Datei weitere Informationen zu den Tabellen festgelegt.

```
---
id: avprodukt
entityStorageVersion: 2
createdAt: 1708101679119
lastModified: 1708101679119
providerType: FEATURE
providerSubType: SQL
nativeCrs:
  code: 2056
  forceAxisOrder: NONE
connectionInfo:
  connectorType: SLICK
  database: ${POSTGRES_DB}
  host: ${POSTGRES_HOST}
  user: ${POSTGRES_USER}
  password: ${POSTGRES_USER_PASSWORD_BASE64}
  schemas:
```

```
- avprodukt
pool:
  maxConnections: -1
  minConnections: 1
  initFailFast: true
  initFailTimeout: 1
  idleTimeout: 10m
  shared: false
sourcePathDefaults:
  primaryKey: objectid
  sortKey: objectid
types:
  bo_boflaeche_mv:
    label: Bodenbedeckung
    description: 'Bodenbedeckung der Amtlichen Vermessung.'
    sourcePath: /bo_boflaeche_mv
    properties:
      objectid:
        sourcePath: objectid
        type: INTEGER
        role: ID
      art:
        sourcePath: art
        type: STRING
      wkb_geometry:
        sourcePath: wkb_geometry
        type: GEOMETRY
        role: PRIMARY_GEOMETRY
        geometryType: POLYGON
...
```

entities/providers/geodienste.yml: In diesem File wird ein WFS-Dienst als Featurequelle verwendet, um Geodaten als OGC API - Features oder OGC API - Tiles bereitzustellen.

entities/services/avprodukt.yml: In diesem File werden die in den Providern definierten Tabellen als OGC API - Features und OGC API - Tiles bereitgestellt. Mittels des Parameters „buildingBlock“ werden die entsprechenden Bausteine definiert.

```
---
id: avprodukt
createdAt: 1708101679809
lastModified: 1708101679809
entityStorageVersion: 2
label: avprodukt
enabled: true
metadata:
  attribution: 'Amtliche Vermessung Thurgau'
  keywords:
    - Amtliche Vermessung
serviceType: OGC_API
apiValidation: NONE

api:
- buildingBlock: COLLECTIONS
  additionalLinks:
    - rel: describedby
      type: application/pdf
      title: Amtliche Vermessung
...
```

Anschließend müssen die Tabellen, die bereitgestellt werden sollen, als „collections“ aufgelistet werden.

```
collections:
  bo_boflaeche_mv:
    id: bo_boflaeche_mv
    label: bo_boflaeche_mv
    enabled: true
  bo_projgebaeude_mv:
    id: bo_projgebaeude_mv
    label: bo_projgebaeude_mv
    enabled: true
...
```

```
- buildingBlock: TILES
  tileProvider:
    type: FEATURES
    tileEncodings:
      - MVT
    zoomLevels:
      WebMercatorQuad:
        min: 0
        max: 23
    # seeding:
    #   WebMercatorQuad:
    #     min: 11
    #     max: 13
...
```

Container starten

Server starten: `docker compose --profile ldproxy up -d`

Server beenden: `docker compose --profile ldproxy down`

Landing Page Ldproxy: <http://localhost:7080/rest/services>

Collection Items bo_boflaeche_mv:

http://localhost:7080/rest/services/avprodukt/collections/bo_boflaeche_mv/items?f=html

Home / avprodukt / Daten / Bodenbedeckung / Objekte GeoJSON

Bodenbedeckung

Bodenbedeckung der Amtlichen Vermessung.

Filter

« < 8 9 10 11 12 > »

60937675	
objectid	60937675
art	Gebaeude
60937676	
objectid	60937676
art	Gebaeude
60937677	
objectid	60937677
art	Gewaesser.stehendes
60937678	
objectid	60937678
art	humusiert.Acker_Wiese_Weide
60937679	
objectid	60937679
art	Gebaeude

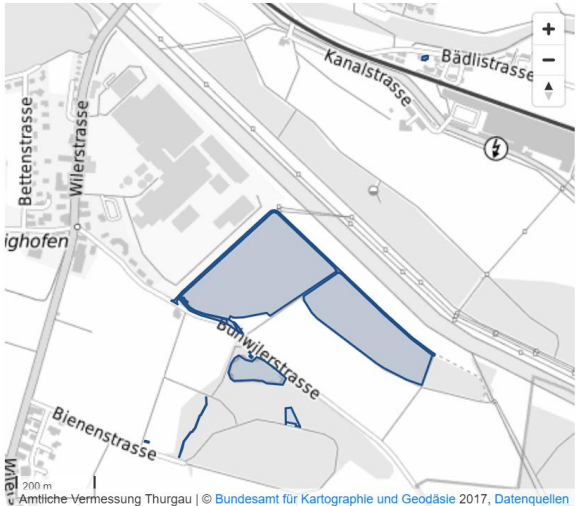


Abbildung 4: Ldproxy Collection items bo_boflaeche_mv

Vector-Tiles-Anfrage (Beispiel: bo_boflaeche_mv):

`http://localhost:7080/rest/services/avprodukt/collections/bo_boflaeche_mv/tiles/WebMercatorQuad/{z}/{y}/{x}`

4.3 MARTIN

[Martin](#) ist ein Open-Source-Vector-Tile-Server, der es ermöglicht, MVT-Vector-Tiles aus PostGIS-Tabellen, Views sowie aus PMTiles- oder MBTiles zu generieren und bereitzustellen. Dabei können mehrere Datenquellen dynamisch in einem einzelnen Vector-Tile kombiniert werden. Martin ist plattformunabhängig und kann auf Linux, macOS, Windows oder als Docker-Image genutzt werden. Zur effizienten Zwischenspeicherung von häufig abgerufenen Kacheln kann Martin hinter einem NGINX-Proxy eingesetzt werden. Die Vector Tiles in Martin werden direkt in der PostGIS-Datenbank mithilfe der PostGIS-Funktion ST_AsMVT erzeugt. Dabei erkennt und veröffentlicht der Server automatisch alle Tabellen auf welcher er Lesezugriff hat. Der Zugriff auf Tabellen kann mithilfe der standardmässigen Datenbankzugriffssteuerung eingeschränkt werden (Martin - Documentation 2024).

Im Gegensatz zu BBOX oder Ldproxy ist Martin ein reiner Vector-Tile-Server und kann keine weiteren OGC API - Dienste wie OGC API - Features bereitstellen. Diese Einschränkung macht den Server jedoch sehr leicht und effizient .

compose.yaml

```
martin:
  image: ghcr.io/maplibre/martin:v0.13.0
  container_name: martin
  environment:
    - DATABASE_URL=postgresql://${POSTGRES_USER_USERNAME:-
user_name}:${POSTGRES_USER_PASSWORD:-user_password}@postgis/${POSTGRES_DB:-
postgres_db}
  volumes:
    - ./martin:/opt/martin_config
  depends_on:
    - postgis
  profiles: ["martin", "all_server"]
  ports:
    - 3000:3000
```

Codeausschnitt „Martin“ aus dem File [compose.yaml](#).

Konfigurationsfile

Das Konfigurationsfile ist im Verzeichnis [martin/config.yaml](#) gespeichert.

Hier können die Standardparameter übersteuert werden. Für den Performancetest wurden die Standardparameter übernommen und nur der Parameter „default_srid“ von 4326 auf 2056 geändert, weil die Grundlagedaten im Landeskoordinatensystem der Schweiz EPSG:2056 gespeichert sind.

Container starten

Server starten: `docker compose --profile martin up -d`

Server beenden: `docker compose --profile martin down`

Liste der verfügbaren Layer: <http://localhost:3000/catalog>

TileJSON von bo_boflaeche_mv: http://localhost:3000/bo_boflaeche_mv

```
{
  "tilejson": "3.0.0",
  "tiles": [
    "http://localhost:3000/bo_boflaeche_mv/{z}/{x}/{y}"
  ],
  "vector_layers": [
    {
      "id": "bo_boflaeche_mv",
      "fields": {
        "art": "varchar",
        "objectid": "int4"
      }
    }
  ],
  "bounds": [
    8.661115396174294,
    47.37924970210111,
    9.485901423365242,
    47.68411227743354
  ],
  "description": "avprodukt.bo_boflaeche_mv.wkb_geometry",
  "name": "bo_boflaeche_mv"
}
```

Abbildung 5: Martin JSON-File bo_boflaeche_mv

Vector-Tiles-Anfrage (Beispiel: bo_boflaeche_mv):

```
http://localhost:3000/bo_boflaeche_mv/{z}/{x}/{y}
```

```
http://localhost:3000/bo_boflaeche_mv,bo_projgebaeude_mv/{z}/{x}/{y}
```

4.4 PG_TILESERV

[Pg_tileserv](#) ist ein leistungsstarker Vector-Tile-Server, der von [CrunchyData](#) in der Programmiersprache Go entwickelt wurde. Er arbeitet ausschliesslich mit PostGIS und nimmt HTTP-Anfragen für Kacheln entgegen, die er durch Ausführung von SQL-Befehlen generiert.

Für die dynamische Generierung der Kacheln nutzt `pg_tileserv` die PostGIS-Funktion `ST_AsMVT`. Dabei erkennt und veröffentlicht der Server automatisch alle Tabellen, auf denen er Lesezugriff hat. Der Zugriff auf Tabellen und Funktionen kann mithilfe der standardmässigen Datenbankzugriffssteuerung eingeschränkt werden. Durch die Verwendung von Funktionslayern kann der Server beliebige SQL-Befehle ausführen, um Kacheln zu generieren. Jegliche Datenverarbeitung, Feature-Filterung oder Datensatzaggregation, die in SQL ausgedrückt werden kann, kann als parametrisierte Kachelquellen freigegeben werden ([pg_tileserv - Documentation 2023](#)).

Im Gegensatz zu anderen Servern ermöglicht es `pg_tileserv` nicht, Geodaten auch gemäss anderer OGC API Standards wie beispielsweise OGC API - Features bereitzustellen. CrunchyData bietet dafür jedoch einen weiteren Server mit dem Namen `pg_featureserv` an ([pg_featureserv 2023](#)).

compose.yaml

```
pg_tileserv:
  image: pramsey/pg_tileserv:20240131
  container_name: pg_tileserv
  environment:
    - DATABASE_URL=postgresql://${POSTGRES_USER_USERNAME:-
user_name}:${POSTGRES_USER_PASSWORD:-user_password}@postgis/${POSTGRES_DB:-
postgres_db}
  volumes:
    - ./pg_tileserv:/config
  depends_on:
    - postgis
  profiles: ["pg_tileserv", "all_server"]
  ports:
    - 7800:7800
```

Codeausschnitt „`pg_tileserv`“ aus dem File [compose.yaml](#).

Konfigurationsfile

Das Konfigurationsfile ist im Verzeichnis [pg_tileserv/pg_tileserv.toml](#) gespeichert. Hier können die Standardparameter übersteuert werden. Für den Performancetest wurden die Standardparameter übernommen.

Container starten

Server starten: `docker compose --profile pg_tileserv up -d`

Server beenden: `docker compose --profile pg_tileserv down`

Liste der verfügbaren Layer: <http://localhost:7800>

Kachelvorschau `bo_boflaeche_mv`:

http://localhost:7800/avprodukt.bo_boflaeche_mv.html

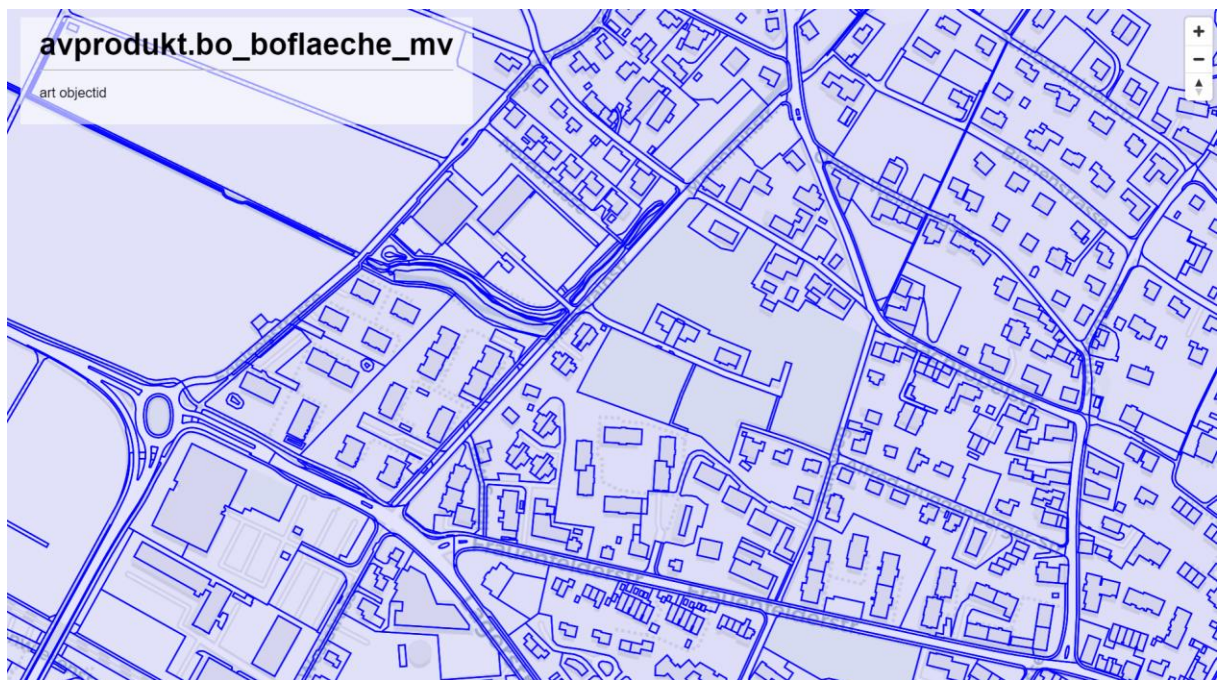


Abbildung 6: `pg_tileserv` Kachelvorschau `bo_boflaeche_mv`

Vector-Tiles-Anfrage (Beispiel: `bo_boflaeche_mv`):

`http://localhost:7800/avprodukt.bo_boflaeche_mv/{z}/{x}/{y}.pbf`

`http://localhost:7800/avprodukt.bo_boflaeche_mv,avprodukt.bo_projgebaeude_mv/{z}/{x}/{y}.pbf`

Im Gegensatz zu Martin muss bei `pg_tileserv` der Schemaname vor dem Tabellennamen hinzugefügt werden. Ausserdem muss das Format „.pbf“ angegeben werden, damit die Kacheln korrekt interpretiert werden können.

4.5 TEGOLA

[Tegola](#) ist ein Open-Source-Vector-Tile-Server, der in Go geschrieben wurde. Der Server unterstützt PostGIS, GeoPackage und SAP HANA Spatial als Datenquellen, die als Vector Tiles bereitgestellt werden können. Er ermöglicht das Cachen von Tiles für ausgewählte Ausschnitte und Zoomstufen. Tegola unterstützt als Datengrundlage standardmässig die Projektionen WebMercator (EPSG:3857) und WGS84 (EPSG:4326). Mittels benutzerdefinierter SQL-Abfragen können jedoch auch Daten in anderen Projektionen wie z.B. EPSG:2056 als Vector Tiles bereitgestellt werden. Dabei werden die Daten beim Erstellen der Kacheln von EPSG:2056 nach EPSG:3857 transformiert und die Kacheln als WebMercatorQuad bereitgestellt. Tegola nutzt die PostGIS-Funktion `St_AsMVT` zur Erstellung der Kacheln aus der PostGIS-Datenbank (Tegola - Documentation 2017).

Im Gegensatz zu anderen Servern ist es mit Tegola nicht möglich, Geodaten gemäss anderer OGC API Standards wie beispielsweise OGC API - Features bereitzustellen.

Die zu veröffentlichenden Tabellen werden von Tegola nicht automatisch erkannt und bereitgestellt, wenn benutzerdefinierte SQL-Abfragen definiert wurden. Stattdessen müssen sie einzeln in der Konfigurationsdatei definiert werden, bevor sie veröffentlicht werden können. Tegola unterstützt Multilayer, womit mehrere Datenquellen jeweils als Layer in einer Vector-Kachel bereitgestellt werden können.

compose.yaml

```
tegola:
  image: gospatial/tegola:latest
  container_name: tegola
  environment:
    - POSTGRES_USER=${POSTGRES_USER_USERNAME:-user_name}
    - POSTGRES_PASSWORD=${POSTGRES_USER_PASSWORD:-user_password}
    - POSTGRES_DB=${POSTGRES_DB:-postgres_db}
    - POSTGRES_HOST=${POSTGRES_HOST:-postgres_host}
    - TEGOLA_SQL_DEBUG=LAYER_SQL # LAYER_SQL or EXECUTE_SQL
  volumes:
    - ./tegola:/opt/tegola_config
  entrypoint: /opt/tegola
  command: "serve --config /opt/tegola_config/config.toml serve --log-
level DEBUG"
  depends_on:
    - postgres
  profiles: ["tegola", "all_server"]
  ports:
    - 8090:8090
```

Codeausschnitt „tegola“ aus dem File [compose.yaml](#).

Konfigurationsfile

Das Konfigurationsfile ist im Verzeichnis [tegola/config.toml](#) gespeichert.

Weil die PostGIS Grundlagedaten nicht in der WebMercator-Projektion vorliegen, muss der Provider-Typ „mvt_postgis“ statt „postgis“ verwendet werden. Als Provider wurde in diesem Beispiel das Schema avprodukt verwendet.

```
# --- register data providers ---
[[providers]]
name = "avprodukt" # provider name is referenced from map layers
                    (required)
type = "mvt_postgis"
uri =
"postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${POSTGRES_HOST}:5432/${POSTGRES_DB}"
```

Zusätzlich zu den Providern müssen die entsprechenden Layer konfiguriert werden. Im SQL-Parameter wird die entsprechende Tabelle mit den benötigten Attributen abgefragt und die Geometrie mittels der PostGIS-Funktion ST_Transform nach EPSG:3857 transformiert. Dabei ist zu beachten, dass die Boundingbox „!BBOX!“ bei der Bedingung in die Projektion der Originaldaten, in diesem Fall EPSG:2056, transformiert wird.

```
# --- provider-layers ---

[[providers.layers]]
name = "bo_boflaeche_mv"
id_fieldname = "objectid"
geometry_fieldname = "wkb_geometry"
geometry_type = "polygon"
sql = "SELECT ST_AsMVTGeom(ST_Transform(wkb_geometry,
3857),ST_Transform(!BBOX!,3857)) AS wkb_geometry, objectid, art FROM
avprodukt.bo_boflaeche_mv WHERE wkb_geometry && ST_Transform(!BBOX!,2056) "
```

Anschliessend muss die Karte definiert und die Layer hinzugefügt werden.

```
[[maps]]
name = "avprodukt"
attribution = "Amtliche Vermessung"
center = [9.08545538325693, 47.57304195845273, 10.0] # set the center
of the map so the user is auto navigated to Canton Thurgau

[[maps.layers]]
provider_layer = "avprodukt.bo_boflaeche_mv"
min_zoom = 10
max_zoom = 22
```

Container starten

Server starten: `docker compose --profile tegola up -d`

Server beenden: `docker compose --profile tegola down`

Kachelvorschau: <http://localhost:8090>

Capabilities: <http://localhost:8090/capabilities>

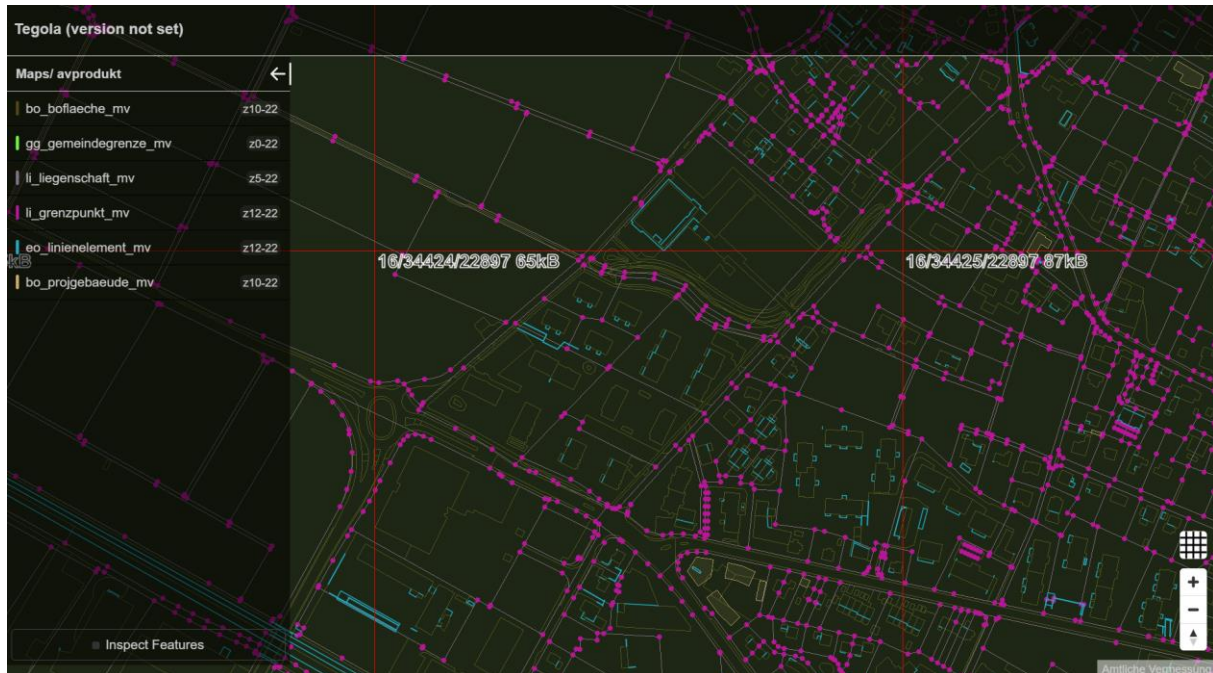


Abbildung 7: Tegola Kachelvorschau

Vector-Tiles anfrage (Beispiel: `bo_boflaeche_mv`):

`http://localhost:8090/maps/avprodukt/bo_boflaeche_mv/{z}/{x}/{y}.pbf`

`http://localhost:8090/maps/avprodukt/bo_boflaeche_mv,bo_projgebaeude_mv/{z}/{x}/{y}.pbf`

4.6 TiPg

TiPg ist eine Open-Source-Applikation in Python, die OGC API - Features und OGC API - Tiles aus einer PostGIS-Datenbank bereitstellt. Es vereint die Funktionen zweier separater Projekte: „[tifeatures](#)“ und „[timvt](#)“. Mit TiPg können Vector Tiles dynamisch mithilfe der PostGIS-Funktion ST_AsMVT generiert werden. Es unterstützt mehrere Tile Matrix Sets (TMS) für die Bereitstellung von Vector-Tiles. Standardmässig wird das TMS WebMercatotQuad verwendet. Benutzerdefinierte TMS können zusätzlich ergänzt werden, um beispielsweise Kacheln auch im lokalen Koordinatensystem bereitzustellen. Ein Beispiel eines Benutzerdefinierten TMS EPSG_2056.json wurde im Verzeichnis `tipg/tms` abgelegt. Damit der Server dieses File automatisch interpretieren kann, muss im `compose.yaml` File der Pfad, in welchem sich das JSON befindet, als Parameter `TILEMATRIXSET_DIRECTORY` angegeben werden (TiPg - Documentation 2024).

compose.yaml

```
tipg:
  image: ghcr.io/developmentseed/tipg:latest
  container_name: tipg
  environment:
    - POSTGRES_USER=${POSTGRES_USER_USERNAME:-user_name}
    - POSTGRES_PASS=${POSTGRES_USER_PASSWORD:-user_password}
    - POSTGRES_DBNAME=${POSTGRES_DB:-postgres}
    - POSTGRES_HOST=${POSTGRES_HOST:-postgis}
    - POSTGRES_PORT=5432
    - PORT=8080
    - DEBUG=TRUE
    - TILEMATRIXSET_DIRECTORY=tms
    ### This Paramaters can also be saved in a seperate .env-File
    ##- TIPG_NAME="Welcome to my OGC Features and Tiles API"
    ##- TIPG_DB_SCHEMAS=["avprodukt", "public"]
  volumes:
    - ./tipg:/tmp
  depends_on:
    - postgis
  profiles: ["tipg", "all_server"]
  ports:
    - 8080:8080
```

Codeausschnitt „tipg“ aus dem File [compose.yaml](#).

Konfigurationsfile

Die Konfigurationsdatei befindet sich im Verzeichnis `tipg/.env`. Die entsprechenden Parameter können entweder in dieser Datei oder als Parameter in der „`compose.yaml`“ definiert werden. Die Anleitung zur Konfiguration ist hier verfügbar:

https://developmentseed.org/tipg/user_guide/configuration

Für den Performancetest wurden die Standardparameter übernommen und nur die folgenden Parameter geändert:

```
TIPG_DB_SCHEMAS='["avprodukt"]'
TIPG_FALLBACK_KEY_NAMES=["ogc_fid", "id", "pkey", "gid", "objectid"]
TIPG_SET_MVT_LAYERNAME=true
TIPG_DEBUG=true
```

Container starten

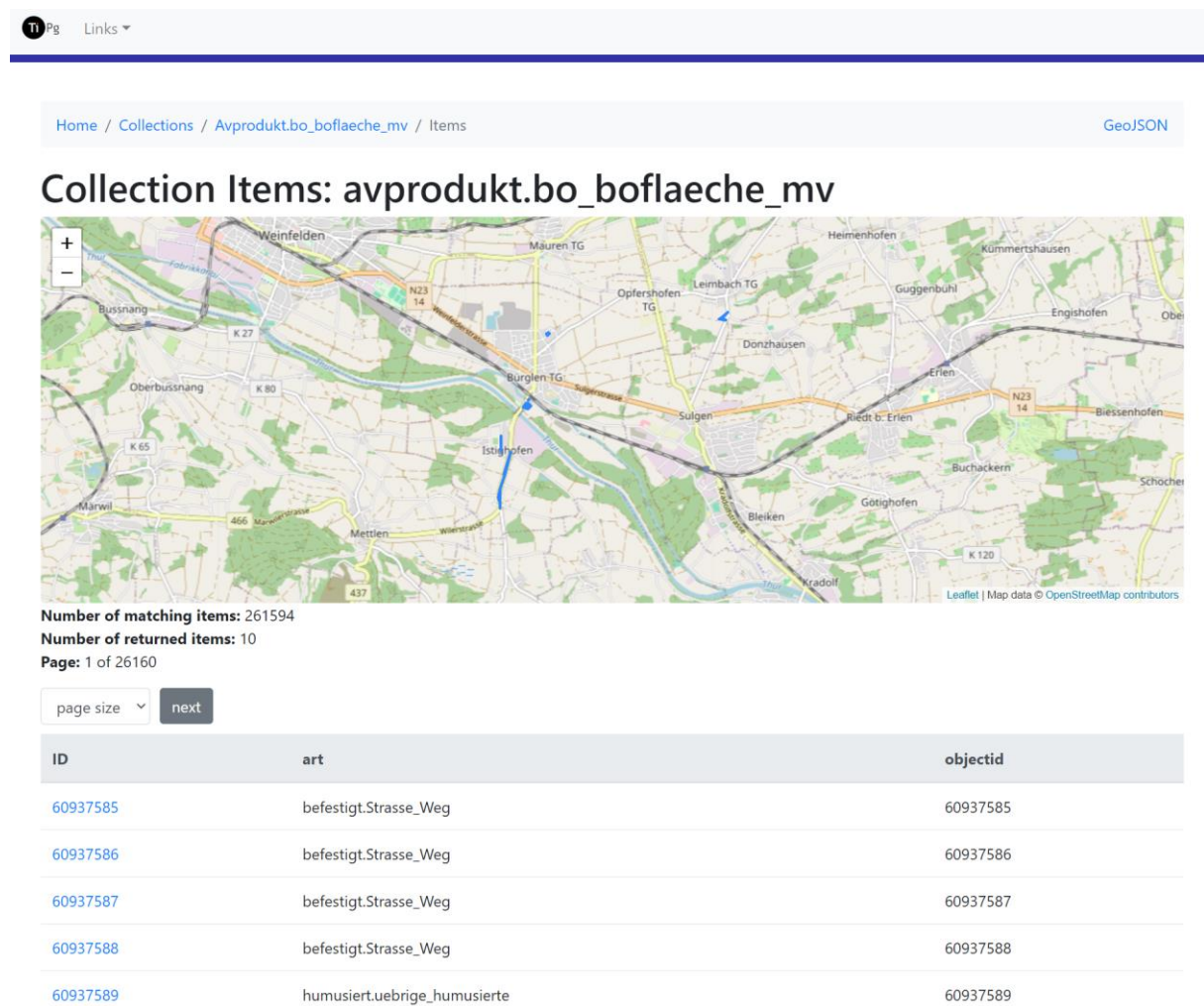
Server starten: `docker compose --profile tipg up -d`

Server beenden: `docker compose --profile tipg down`

Landing Page TiPg: <http://localhost:8080>

Feature Collection Items `bo_boflaeche_mv`:

http://localhost:8080/collections/avprodukt.bo_boflaeche_mv/items



The screenshot shows the TiPg web interface. At the top, there is a navigation bar with "Home / Collections / Avprodukt.bo_boflaeche_mv / Items" and a "GeoJSON" link. The main heading is "Collection Items: avprodukt.bo_boflaeche_mv". Below this is a map of a region in Germany, showing various locations like Weinfelden, Mäuren TG, Leimbach TG, and others. Below the map, there is a table of collection items.

Number of matching items: 261594
Number of returned items: 10
Page: 1 of 26160

page size

ID	art	objectid
60937585	befestigt.Strasse_Weg	60937585
60937586	befestigt.Strasse_Weg	60937586
60937587	befestigt.Strasse_Weg	60937587
60937588	befestigt.Strasse_Weg	60937588
60937589	humusiert.uebrige_humusierte	60937589

Abbildung 8: TiPg Collection items `bo_boflaeche_mv`

Vector-Tiles-Anfrage (Beispiel: `bo_boflaeche_mv`):

`http://localhost:8080/collections/avprodukt.bo_boflaeche_mv/tiles/WebMercatorQuad/{z}/{x}/{y}`

5 PERFORMANCEUNTERSUCHUNG

Im Kapitel 4 wurden verschiedene Applikationen zur Erstellung von Vector Tiles konfiguriert. Dieser Abschnitt widmet sich nun der eingehenden Untersuchung der Performance dieser Serveranwendungen.

Ein Performancetest ist ein wichtiger Schritt, um sicherzustellen, dass eine Software-Anwendung nicht nur funktional ist, sondern auch die erforderliche Leistung erbringt, insbesondere bei gleichzeitiger Interaktion mit mehreren Benutzern oder eingeschränkten Ressourcen. Die wichtigsten Arten von Performancetests umfassen laut (Lenka et al.) Last-, Stress- und Performancetests. Während Last- und Stress-Tests dazu dienen, die Fähigkeit einer Anwendung zu testen, unter Last oder schwierigen Bedingungen zu arbeiten, um Engpässe oder Fehler zu identifizieren, dient der Performancetest dazu, die Leistungsfähigkeit des Systems hinsichtlich seiner Funktionalität und Reaktionszeit zu validieren.

Um einen Performancetest durchzuführen, sind verschiedene Schritte erforderlich, darunter die Identifizierung des Testumfelds, die Auswahl und Planung der Tests sowie die Durchführung, Analyse und Berichterstattung über die Ergebnisse. Es ist auch wichtig, klare Testziele zu definieren, um die Effektivität und Genauigkeit des Tests sicherzustellen.

Es gibt verschiedene Tools, die für Performancetests verwendet werden können. Einige beliebte Werkzeuge sind beispielsweise Apache JMeter, Gatling oder LoadRunner. Diese Tools können dazu beitragen, die Tests zu automatisieren und genaue Ergebnisse zu liefern.

Im Gegensatz zu (Lenka et al.) unterscheiden (Wang und Wu) vier Arten von Performancetests.

Bei einem allgemeinen Performancetest wird das System in einer normalen Hardware- und Softwareumgebung ohne Druck getestet. Es wird also nur die Zeit gemessen, die eine Anfrage an den Server benötigt.

Beim Stabilitätstest werden hingegen kontinuierlich Anfragen an den Server gesendet, um die Stabilität des Systems zu überprüfen.

Der Belastungstest überprüft wie der Stabilitätstest kontinuierlich das System, jedoch unter starker Last bei kritischem Zustand des Servers.

Beim Stresstest wird der Druck auf das System schrittweise erhöht, bis das System zusammenbricht, um den maximalen Druck zu testen, den das System aushalten kann.

5.1 TEST-DESIGN

Das Test-Design bezieht sich auf den Prozess, bei dem die Struktur und der Plan für die Durchführung von Tests festgelegt werden. Es ist ein wesentlicher Bestandteil des gesamten Software-Testprozesses und beinhaltet die Definition von Testzielen, Testscenarien, Testdaten und anderen Elementen, die für eine umfassende Prüfung einer Software-Anwendung erforderlich sind.

5.1.1 Testziele

Das Hauptziel dieser Tests ist es, die Reaktionszeiten verschiedener Applikationen beim Bereitstellen exakt gleicher Vektorkacheln zu vergleichen. Dabei sollen unterschiedliche Geometrietypen und Zoomstufen berücksichtigt werden. Ein weiteres Ziel ist es, das Verhalten der Server zu analysieren, wenn mehrere Benutzer gleichzeitig auf die gleichen Kacheln zugreifen. Es wird untersucht, wie sich die Applikationen verhalten, wenn nicht nur ein einzelner Benutzer, sondern 10 oder 100 Benutzer gleichzeitig auf die Kacheln zugreifen.

Nicht im Fokus dieser Untersuchung liegt die Ermittlung der Anzahl von Anfragen, die ein Server verarbeiten kann, bevor er abstürzt (Stresstest).

5.1.2 Testscenarien

Damit verschiedene Geometrien und Zoomstufen getestet werden können und zusätzlich das Verhalten der Server bei unterschiedlicher Anzahl von Benutzern getestet werden kann, werden fünf Testscenarien erstellt, die jeweils mit nur einem Benutzer, 10 und 100 Benutzern getestet werden. Um eine statistische Genauigkeit der Messungen zu erhalten, werden die Anfragen jeweils mehrmals wiederholt und die Ergebnisse gemittelt.

Bei einem Initialtest mit den Servern BBOX und Martin wurde festgestellt, dass bei nur einer oder 10 Wiederholungen signifikante Abweichungen der Messwerte im Vergleich zu einer Messung mit 1000 Wiederholungen auftreten. Bei 100 Wiederholungen wurden jedoch keine signifikanten Änderungen gegenüber den 1000 Wiederholungen festgestellt. Aufgrund dieser Erkenntnisse und der Tatsache, dass sich bei 1000 Wiederholungen die Testdauer erheblich verlängert hat, wurde entschieden, dass für die Tests mit nur einem Benutzer die Anfragen 100 Mal wiederholt werden. Bei Tests mit 10 Benutzern wurde die Anfrage nur 10-mal, bei Tests mit 100 Benutzern nur einmal an den Server gesendet. Wenn die Anfragen mit mehreren Benutzern ebenfalls 100-mal ausgeführt worden wären, hätte der Server lange gebraucht, um die Anfragen zu beantworten, und der Test hätte jeweils über 30 Minuten gedauert.

Übersicht Testscenarien

Testnummer	Anzahl Kacheln	Zoomstufe/Kacheln	Datensatz	Tabelle
Test_1	1	14/8604/5723	Bodenbedeckung	bo_boflaeche_mv
Test_2	4	15/17208/11446 15/17209/11446 15/17208/11447 15/17209/11447	Bodenbedeckung	bo_boflaeche_mv
Test_3	1	8/134/89	Gemeindegrenzen	gg_gemeindegrenze_mv
Test_4	1	14/8604/5723	Grenzpunkte	li_grenzpunkt_mv
Test_5	1	14/8604/5723	Einzelobjekte	eo_linienelement_mv

Tabelle 2: Übersicht Testscenarien

In allen Testscenarien ausser Test_2 und Test_3 werden die gleichen Kacheln verwendet. Im Test_2 wird zwar der gleiche Ausschnitt wie in den anderen Tests verwendet, jedoch eine Zoomstufe tiefer.

Dadurch werden im Test_2 jeweils gleichzeitig 4 Kacheln angefragt. Je nach den Standardeinstellungen der verschiedenen Server für die Generalisierung der Vektordaten kann es sein, dass die Geometrien auf Zoomstufe 15 mehr Stützpunkte enthalten als auf Zoomstufe 14. Dies könnte wiederum die Dateigrösse der Kacheln erhöhen. Mit diesem Test sollen die Dateigrössen zwischen Test_1 und Test_2 verglichen werden können. Test_3 verwendet die Zoomstufe 8, um den kompletten Datensatz abzurufen.

Test 1 – Bodenbedeckung Zoomstufe 14



Abbildung 9: Kachel des Test 1 vom Server Tegola

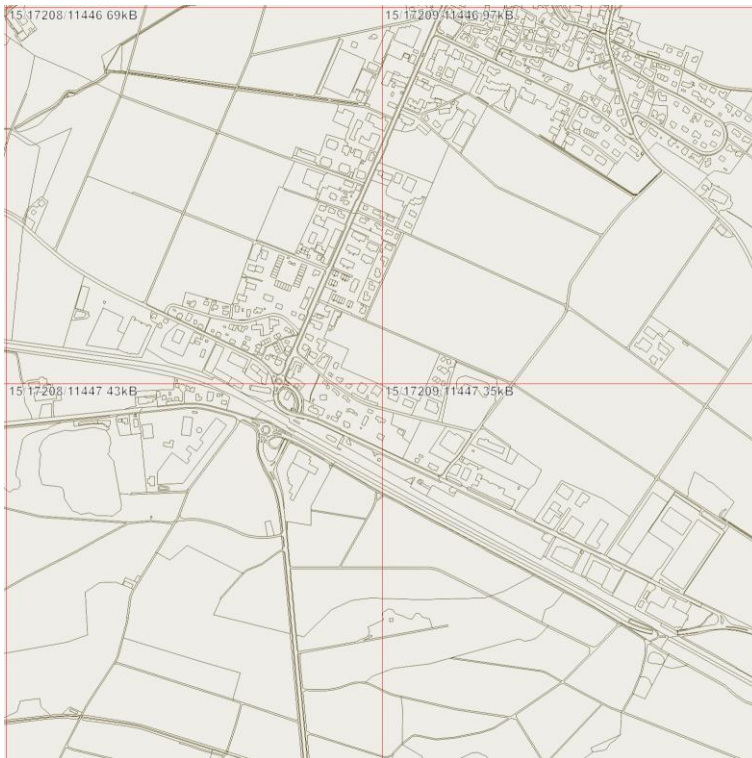
Test 2 – Bodenbedeckung Zoomstufe 15 (4 Kacheln)

Abbildung 10: Kacheln des Test 2 vom Server Tegola

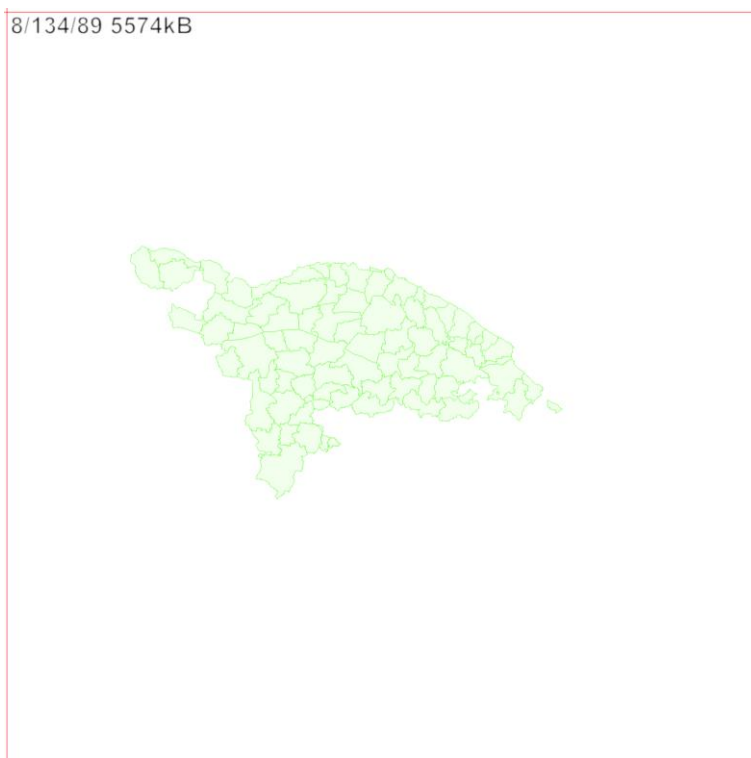
Test 3 – Gemeindegrenze Zoomstufe 8

Abbildung 11: Kachel des Test 3 vom Server Tegola

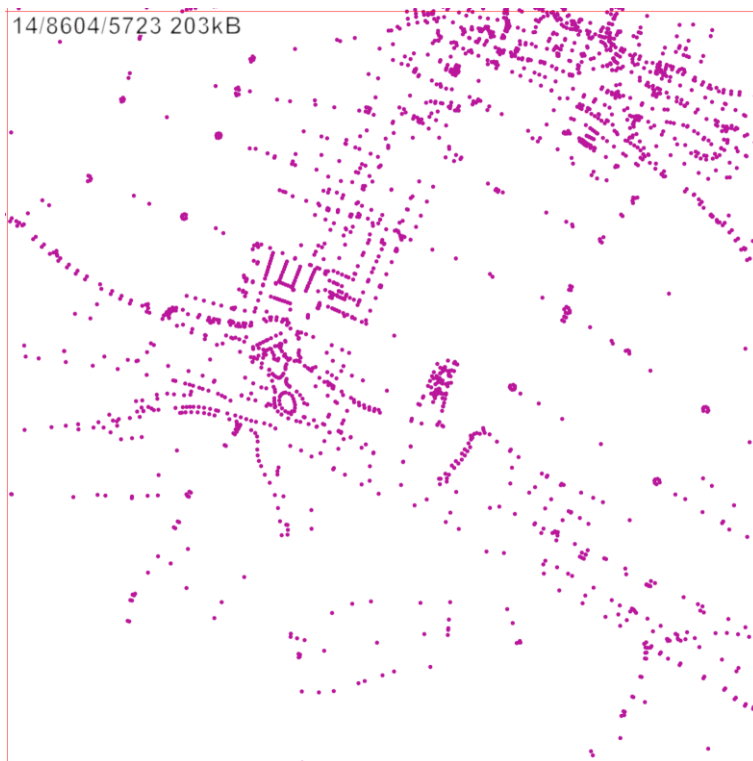
Test 4 – Grenzpunkte Zoomstufe 14

Abbildung 12: Kachel des Test 4 vom Server Tegola

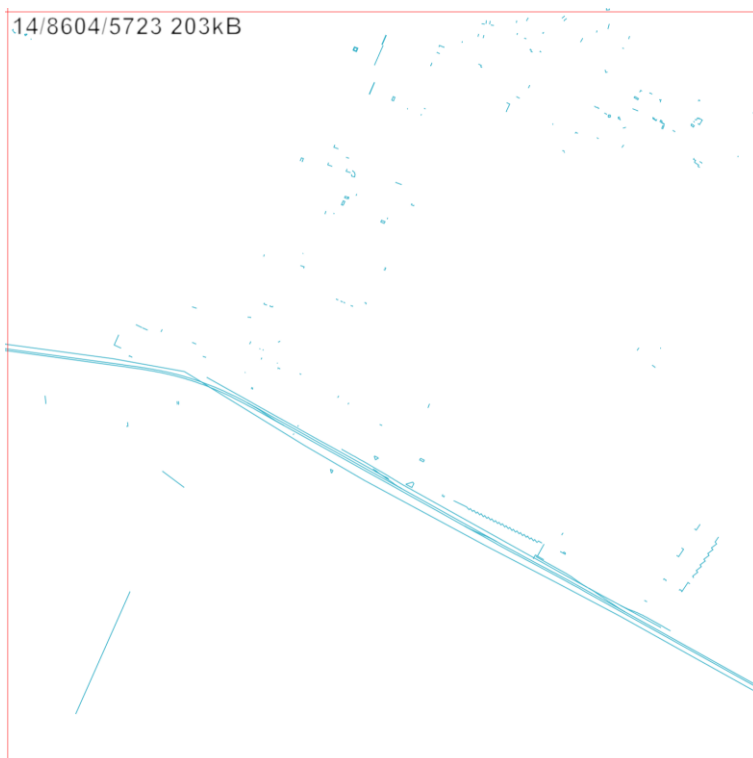
Test 5 – Einzelobjekte (Linienelemente) Zoomstufe 14

Abbildung 13: Kachel des Test 5 vom Server Tegola

5.1.3 Testdaten

Für die Performancetests werden ausgewählte Geodaten aus der PostGIS-Datenbank verwendet, wie bereits in Kapitel 3.1 Grundlagedaten erwähnt. Um die Performance zu verbessern, wurden die Originaldaten aus den Tabellen auf wenige Attribute reduziert. Die verwendeten Datensätze umfassen:

Datensatz	Tabelle	Attribute	Geometrietyp	Koordinatensystem
Bodenbedeckung	bo_boflaeche_mv	objectid, art, wkb_geometry	Polygon	EPSG:2056
Gemeindegrenze	gg_gemeindegrenze_mv	objectid, gemeinde_name, gemeinde_bfsnr, wkb_geometry	Polygon	EPSG:2056
Grenzpunkte	li_grenzpunkt_mv	objectid, punktzeichen, wkb_geometry	Point	EPSG:2056
Einzelobjekte	eo_linienelement_mv	objectid, art, wkb_geometry	LineString	EPSG:2056

Tabelle 3: Testdaten aus der PostGIS Datenbank

Verwendete Server

Die Server-APIs sind teilweise unterschiedlich definiert, was dazu führt, dass einige der Server standardmässig den Schemanamen in der Collection integrieren und andere nur die Tabelle selbst verwenden. Zudem benötigen einige der Server bei der Anfrage der Vektorkachel die Dateiendung „.pbf“, während andere dies nicht erfordern. Die Anfragen-APIs werden folgend zusammengefasst:

Server	MVT-Request Layer: bo_boflaeche_mv
BBOX	http://localhost:8804/xyz/bo_boflaeche_mv/{z}/{x}/{y}.pbf
Ldproxy	http://localhost:7080/rest/services/avprodukt/collections/bo_boflaeche_mv/tiles/WebMercatorQuad/{z}/{y}/{x}
Martin	http://localhost:3000/bo_boflaeche_mv/{z}/{x}/{y}
pg_tileserv	http://localhost:7800/avprodukt.bo_boflaeche_mv/{z}/{x}/{y}.pbf
Tegola	http://localhost:8090/maps/avprodukt/bo_boflaeche_mv/{z}/{x}/{y}.pbf
TiPg	http://localhost:8080/collections/avprodukt.bo_boflaeche_mv/tiles/WebMercatorQuad/{z}/{x}/{y}

Beim Server Ldproxy ist zu berücksichtigen, dass die Koordinaten x und y im Vergleich zu den anderen Servern vertauscht sind. Dies musste bei der Erstellung des Performancetests in JMETER mitberücksichtigt werden.

5.1.4 Testumgebung

Als Testumgebung wurde clientseitig ein Lenovo Notebook Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz mit 16 GB RAM und Windows 10 verwendet. Darauf wurde Apache JMeter Version 5.6.2 ausgeführt (Apache JMeter - Download Apache JMeter 2024). Während der Testphase wurden alle Programme bis auf Excel, JMeter und Visual Studio Code (VS Code) geschlossen, um allfällige Performancebeeinflussungen zu eliminieren. VS Code wurde benutzt, um eine localhost Verbindung zum Linuxserver in der Public Cloud aufzubauen, und um die Docker Container zu starten und zu stoppen.

Das Notebook ist mit einem LAN-Kabel mit dem Router verbunden. Als Internetabo wird ein Glasfaseranschluss mit einer maximalen Down- und Upload Geschwindigkeit von 1000 mbit/s verwendet. Kurz vor dem Performancetest wurde ein Internetgeschwindigkeitstest mit (Speedtest.net) ausgeführt und eine Downloadgeschwindigkeit von 902 mbit/s (Ping: 3ms) und Upload Geschwindigkeit von 937 mbit/s (Ping: 9ms) erreicht.

Serverseitig wurde in der Public Cloud von Infomaniak ein Ubuntu-Server (Ubuntu 22.04 LTS Jammy Jellyfish) mit 4 VCPUs, 8GB RAM und 50GB Speicherplatz aufgesetzt. Auf diesem Server wurde Docker mit integriertem Docker Compose installiert. Damit wurden die entsprechenden Applikationen als Docker Image heruntergeladen, konfiguriert und gestartet.

Um gegenseitige Beeinflussungen beim Testen der verschiedenen Applikationen zu vermeiden, wurden die Server nicht parallel gestartet. Stattdessen wurden die Applikationen nacheinander in separaten Containern auf demselben Server ausgeführt. Als Datenquelle wurde jeweils der postgres Container gestartet, um die exakt gleichen Grundlagedaten zu verwenden. Dabei ist zu beachten, dass die Applikationen nicht gleichzeitig getestet werden können, da sonst die Serverressourcen zwischen den Applikationen aufgeteilt würden und gleichzeitig auf die gleichen PostGIS-Daten zugegriffen würde.

5.2 PERFORMANCETEST MIT APACHE JMETER

Apache JMeter ist ein leistungsfähiges Open-Source-Tool, das für das Testen der Leistung von Webanwendungen, APIs und anderen Diensten verwendet wird. Es ermöglicht Entwicklern und Testern, verschiedene Arten von Tests durchzuführen, um die Leistung, Skalierbarkeit und Stabilität ihrer Anwendungen zu überprüfen. Mit Apache JMeter können Benutzer Lasttests, Stresstests, Durchsatztests, Verhaltens- und Funktionstests und weitere durchführen. Das Tool kann eine grosse Anzahl von Benutzern simulieren, die gleichzeitig auf die Anwendung zugreifen, und zeichnet Metriken wie Antwortzeiten, Durchsatz, CPU-Auslastung und Fehler auf, um die Leistung der Anwendung unter verschiedenen Bedingungen zu bewerten. Zusätzlich bietet JMeter eine benutzerfreundliche grafische Benutzeroberfläche für die Konfiguration von Testszenarien und zeichnet sich durch Skalierbarkeit und Flexibilität aus (Apache JMeter - Apache JMeter™ 2024).

Um Apache JMeter im grafischen Modus zu nutzen, wird die Software von ihrer [offiziellen Webseite](#) heruntergeladen und anschliessend die Datei 'jmeter.bat' gestartet. In diesem Modus können verschiedene Testszenarien definiert und ausgeführt werden. Es wird jedoch dringend empfohlen, den Performancetest nicht im GUI-Modus, sondern im CLI-Modus (auch Non-UI-Modus genannt) auszuführen (Apache JMeter - User's Manual: Getting Started 2024).

Um alle Server unter möglichst gleichen Bedingungen testen zu können, wird für jeden Server ein eigener Testplan erstellt. Dieser enthält sämtliche Konfigurationen und Schritte, die während des Tests ausgeführt werden. Ein solcher Testplan besteht aus verschiedenen Elementen, darunter Thread-Group, Logic-Controller, Sampler, Listener, Timer und Konfigurationen.

Die Thread-Group bildet dabei das oberste Element in der Baumstruktur eines jeden Testplans. Hier können diverse Parameter festgelegt werden, die für die nachfolgenden Elemente innerhalb dieser Gruppe gelten. Zum Beispiel kann der Parameter „Number of Threads (users)“ definiert werden, um anzugeben, wie viele virtuelle Benutzer gleichzeitig Anfragen an den Server senden.

Unterhalb der Thread-Group werden Logic-Controller-Elemente definiert, mit denen gesteuert wird, wie die Anfragen an den Server gesendet werden. Innerhalb dieser Logic-Controller-Elemente können Sampler erstellt werden, die die entsprechenden Serveranfragen wie zum Beispiel HTTP-Requests repräsentieren.

Listener können an beliebigen Stellen in der Baumstruktur eingesetzt werden, um die während des Tests gesammelten Statistikdaten zu visualisieren. Dies kann in Form von Tabellen, Diagrammen oder zusammenfassenden Berichten erfolgen (Apache JMeter - User's Manual: Elements of a Test Plan 2024).

Test Plan BBOX

Im Anschluss wird als Beispiel der Testplan „vector-tiles-benchmark_bbox“ im Detail erläutert. Dabei verwenden alle Server die gleiche Struktur, jedoch mit auf den jeweiligen Server angepassten Parametern. Die Testpläne sind unter folgendem Link abgelegt:

https://github.com/FabianRechsteiner/vector-tiles-benchmark/vector-tiles-benchmark/test_plans

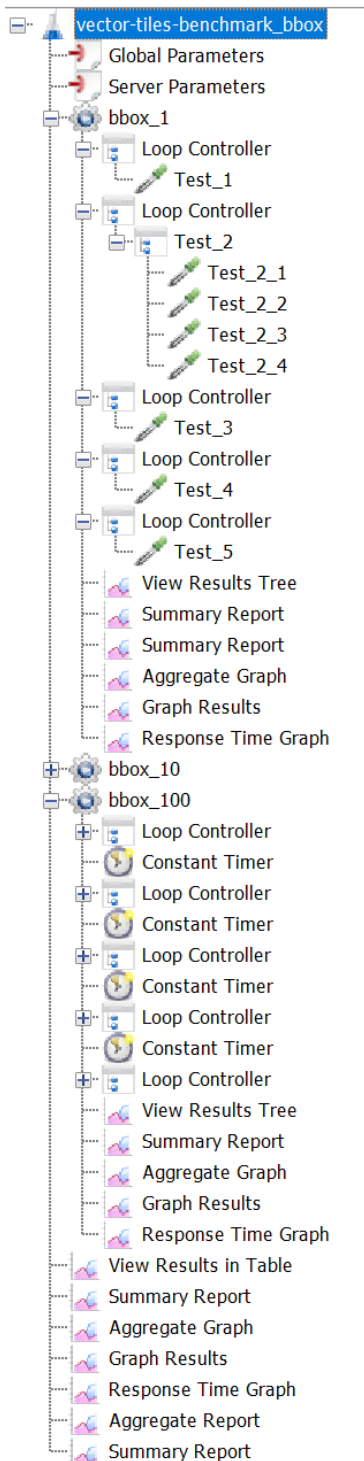


Abbildung 17: Test Plan BBOX

Parameters	
Name:	User_1
loop_count	100
host	localhost
test_1_zoom_tile	14/8604/5723
test_3_zoom_tile	8/134/89
test_4_zoom_tile	14/8604/5723
test_5_zoom_tile	14/8604/5723
test_1_layer	bo_boflaeche_mv
test_2_layer	bo_boflaeche_mv
test_3_layer	gg_gemeindegrenze_mv
test_4_layer	li_grenzpunkt_mv
test_5_layer	eo_linienelement_mv
wait_timer	10000

Abbildung 14: Global Parameters

Parameters	
Name:	User_1
port	8804
format	.pbf
layer_prefix	xyz/
layer_suffix	
server	bbox

Abbildung 15: Server Parameters

Loop Controller

Name:

Comments:

Loop Count: Infinite

Abbildung 16: Loop Controller

Thread Group

Name:

Comments:

Action to be taken after a Sampler error

Continue Start Next Thread Loop Stop Thread Stop Test

Thread Properties

Number of Threads (users):

Ramp-up period (seconds):

Loop Count: Infinite

Abbildung 18: Thread Group bbox_1

HTTP Request

Name: Test_1

Comments: One Tile with Layer bo_boflaechе_mv on zoom 14

Basic Advanced

Web Server

Protocol [http]: http Server Name or IP: \${host} Port Number: \${port}

HTTP Request

Method: GET Path: \${layer_prefix}\${test_1_layer}\${layer_suffix}/\${test_1_zoom_tile}\${format} Content

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data Browser-compatible headers

Abbildung 19: HTTP Request Test_1

Die Abbildung 17: Test Plan BBOX zeigt den vollständigen Testplan für den Server BBOX. Zuerst werden die globalen Parameter definiert, die für alle Server identisch sind, sowie die server-spezifischen Parameter. Dadurch wird die Konfiguration der Testpläne für jeden Server vereinfacht. Unterhalb der Parameterdefinitionen sind die drei Thread Groups aufgeführt. Diese sind bis auf den Parameter „Number of Threads (user)“ und den zusätzlichen Constant Timer in der Thread Group `bbox_100` exakt gleich. Mithilfe von Threads wird gesteuert, wie viele virtuelle Benutzer gleichzeitig Anfragen an den Server senden. Für die Thread Groups wurde dieser Parameter jeweils auf 1, 10 und 100 Benutzer angepasst:

- 1 User: In der ersten Thread-Group werden die 5 Testszenarien mit nur einem Benutzer ausgeführt. Dabei wird jede Anfrage 100-mal wiederholt, und die Antwortzeiten werden gemittelt.
- 10 User: In der zweiten Thread-Group senden 10 virtuelle Benutzer gleichzeitig Anfragen, wobei jede Anfrage 10-mal wiederholt und die Werte gemittelt werden.
- 100 User: In der dritten Thread-Group senden 100 virtuelle Benutzer gleichzeitig Anfragen, wobei die Anfragen nicht wiederholt werden, sondern nur die Werte der 100 Benutzer gemittelt werden.

Jede der drei Thread Groups erzeugt 100 Messungen pro Test. Um sicherzustellen, dass die Server während der dritten Thread Group (`bbox_100`) mit den Anfragen zurechtkommen und der nächste Test erst startet, nachdem alle Anfragen des vorherigen Tests beantwortet wurden, wurde ein Timer von 10 Sekunden zwischen den verschiedenen Testszenarien eingerichtet.

Wie in Abbildung 18: Thread Group `bbox_1` zu sehen ist, kann hier ein Loop-Count definiert werden, um zu steuern, wie oft die entsprechende Request-Anfrage gesendet wird. Da jedoch beim Test 2 gleichzeitig 4 Kacheln abgefragt werden sollen, wurde dafür ein separater Loop Controller wie in Abbildung 16 verwendet. Unterhalb dieses Loop-Controllers wurde ein Parallel-Controller erstellt, der sicherstellt, dass die 4 Kachelanfragen gleichzeitig an den Server gesendet werden und die Gesamtzeit gemessen wird, um alle 4 Kacheln zurückzugeben. Die Abbildung 19: HTTP Request Test_1 zeigt die HTTP-Request-Anfrage des Test 1 mit den entsprechenden Parametern, die verwendet wurden.

Verschiedene Listeners wurden sowohl innerhalb als auch ausserhalb der Thread-Groups implementiert, um die Messdaten der Tests beispielsweise in CSV-Dateien zu erfassen oder grafisch darzustellen. Diese Listeners wurden mehrfach konfiguriert, um eine detaillierte Analyse der Ergebnisse sowohl pro Thread-Group als auch pro Test zu ermöglichen. Am Ende wird ein zweiter Summary Report erstellt, der für alle Server dieselbe CSV-Datei verwendet. Auf diese Weise wurden sämtliche Messwerte für jeden Server in der Datei „`all_server_total_summary_report.csv`“ zusammengeführt. Obwohl ausschliesslich der Summary Report die Daten im CSV-Format speichert, können sie nach dem Performancetest von verschiedenen Listeners interpretiert und visualisiert

werden. Alle Listeners speichern die Daten in identischer Form, präsentieren sie jedoch jeweils auf unterschiedliche Weise.

Informationen zum Performancetest

Der Performancetest wurde am 5. April 2024 um 09:00 Uhr gestartet und dauerte etwa 2,5 Stunden.

Im Testplan „[testplan_vector-tiles-benchmark](#)“ sind die durchgeführten Schritte mit Zeitstempel aufgezeichnet.

Dazu gehörte das sukzessive Starten des Docker-Containers und des Testplans im CLI-Modus für jeden Server. Nachfolgend ist die Gesamtdauer aufgeführt, die die Server für den Performancetest benötigt haben.

- BBOX 9 min
- Ldproxy 54 min
- Martin 5 min
- Pg_tileserv 11 min
- Tegola 9 min
- TiPg 23 min

6 ERGEBNISSE

Die folgende Tabelle zeigt einen Ausschnitt der Ergebnisse für den Server „BBOX“ aus der Datei „diagrams“, in der alle Ergebnisse der 6 Server zusammengefasst wurden. Die vollständige Liste sowie ein HTML-Report pro Server sind im [GitHub-Repository](#) abgelegt. Darüber hinaus wurden aus dieser Liste die verschiedenen Diagramme generiert, die in den folgenden Unterkapiteln dargestellt sind.

Die Ergebnisse des Performancetests wurden in der Tabelle diagrams zusammengefasst und ist im GitHub Repository abgelegt.

Server:Tests	Samles	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes	Avg. Kilobytes
BBOX											
bbox_1:Test_1	100	365	0	637	35.59	0.00%	2.73052	274.63	0.39	102990	103
bbox_1:Test_2	100	521	0	610	33.28	0.00%	1.91516	270.99	1.11	144893	145
bbox_1:Test_3	100	394	0	490	42.8	0.00%	2.53049	141.29	0.37	57175	57
bbox_1:Test_4	100	228	0	294	29.7	0.00%	4.38001	195.03	0.63	45595	46
bbox_1:Test_5	100	157	0	211	14.65	0.00%	6.35486	37.67	0.94	6070	6
bbox_10:Test_1	100	1335	0	2544	405.76	0.00%	6.19655	623.23	0.89	102990	103
bbox_10:Test_2	100	2609	0	3251	451.16	0.00%	3.27783	463.8	1.91	144893	145
bbox_10:Test_3	100	741	0	2697	416.15	0.00%	8.54774	477.26	1.24	57175	57
bbox_10:Test_4	100	499	0	710	100.38	0.00%	14.1623	630.6	2.05	45595	46
bbox_10:Test_5	100	193	0	339	43.13	0.00%	25.6476	152.03	3.78	6070	6
bbox_100:Test_1	100	6860	0	12557	3456.04	0.00%	7.91452	796.01	1.14	102990	103
bbox_100:Test_2	100	49967	0	57982	4885.56	0.00%	1.43466	203	0.84	144893	145
bbox_100:Test_3	100	381	0	481	40.81	0.00%	3.47066	193.78	0.5	57175	57
bbox_100:Test_4	100	234	0	340	34.45	0.00%	3.48918	155.36	0.5	45595	46
bbox_100:Test_5	100	130	0	189	20.54	0.00%	3.49858	20.74	0.52	6070	6

Tabelle 4: Testresultate BBOX

Die Daten zeigen, dass bei den Tests mit jeweils einem und zehn Benutzern alle Anfragen erfolgreich von den Servern beantwortet wurden, ohne dass Fehler auftraten. Bei den Tests mit 100 Benutzern traten jedoch Fehlermeldungen auf den Servern Ldproxy und pg_tileserv auf. Beim Server Ldproxy dauerte der Test 1 mit 100 Benutzern durchschnittlich mehr als zwei Minuten und wurde daher serverseitig abgebrochen. Daher konnten auch die folgenden Tests nicht durchgeführt werden. Bei pg_tileserv wurden im Test 1 mit 100 Benutzern zwar alle Anfragen beantwortet, jedoch die meisten mit Fehlermeldungen. Bei den folgenden Tests wurden zwar jeweils vier Anfragen beantwortet, jedoch wurden die Tests anschliessend serverseitig abgebrochen. Diese Fehler traten auf, da die Server die grosse Anzahl an Anfragen nicht an die PostGIS-Datenbank weiterleiten konnten und dadurch die Verbindung blockiert wurde. Diese Anfragen erhielten daher keine Antwort. Mit Ausnahme der Server Ldproxy und pg_tileserv konnten alle anderen Server, wenn auch teilweise mit erheblicher Verzögerung, alle Kacheln übergeben.

6.1 RESULTATE MIT EINEM BENUTZER

Antwortzeiten

Die Abbildung 20 zeigt eine Übersicht der Antwortzeiten aller Server und Tests. In den weiteren Diagrammen sind die Resultate inklusiv der Standardabweichung pro Test dargestellt.

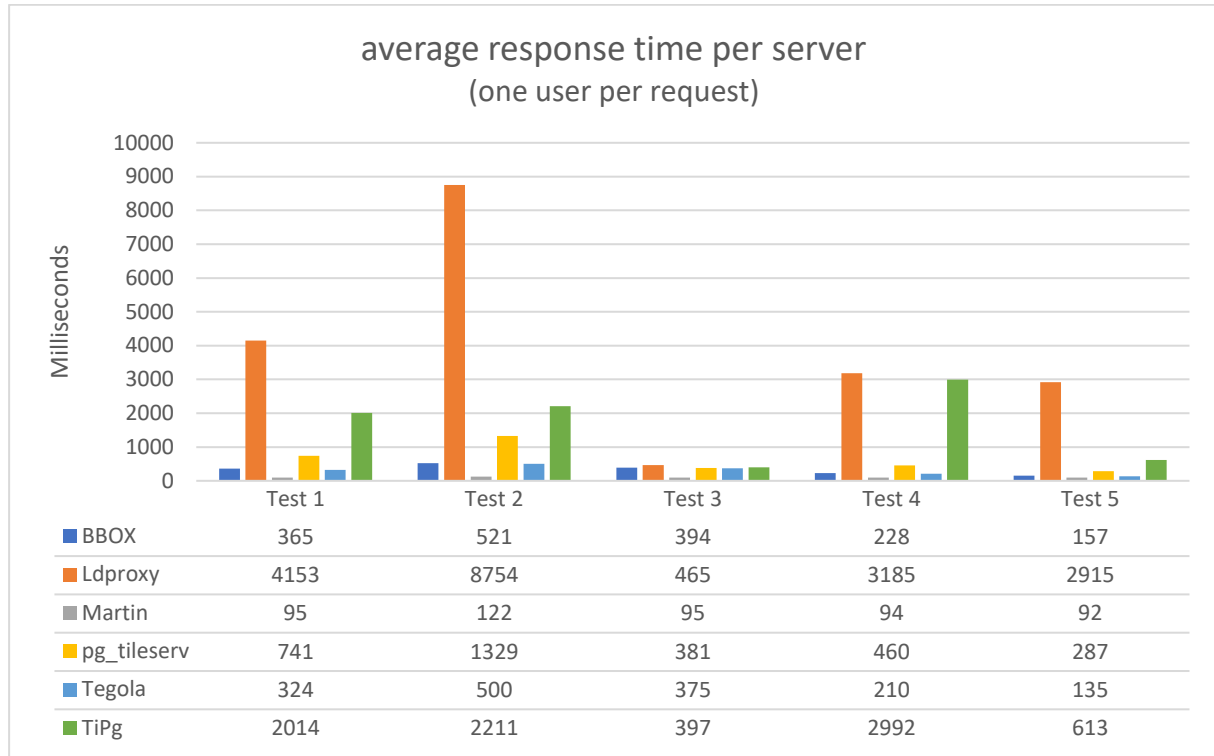


Abbildung 20: Durchschnittliche Antwortzeit pro Server (Übersicht)

Das Balkendiagramm verdeutlicht die breite Spannweite der Antwortzeiten pro Server. Über sämtliche Testszenarien hinweg zeigte der Server Martin die schnellste Leistung bei der Bereitstellung der Kacheln. Bei allen Servern dauerte die Kachelanfragen beim Testszenario 2 am längsten. Es fällt auf, dass die Antwortzeiten der Server in den meisten Testszenarien, abgesehen vom Test 3, teilweise stark variieren. Beispielsweise lagen die Werte im Test 1 zwischen 95 und 4153 Millisekunden, während die Unterschiede im Test 2 mit Werten zwischen 122 und 8754 Millisekunden noch grösser waren.

Über alle Tests hinweg wiesen die beiden Server BBOX und Tegola eine maximale Antwortzeitdifferenz von 41 Millisekunden auf und hatten somit sehr ähnliche Antwortzeiten. Der Server Martin lieferte die Kacheln zwei- bis dreimal schneller aus als der zweitschnellste Server. Ldproxy benötigte für die Auslieferung der Kacheln 4 bis 70 mal so lange wie Martin.

Mit Ausnahme von Test 3 lässt sich für alle Testszenarien folgende Rangliste der Server festlegen:

1. Martin
2. Tegola
3. BBOX
4. Pg_tileserv
5. TiPg
6. Ldproxy

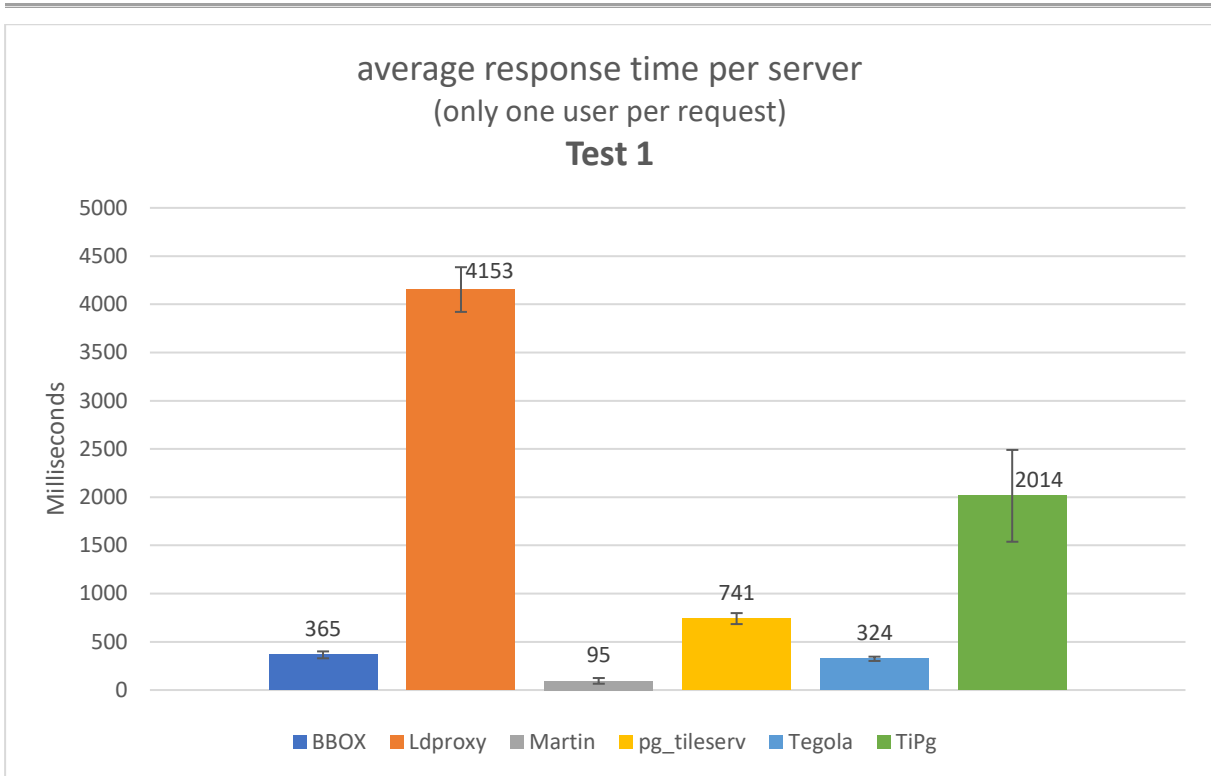


Abbildung 21: Durchschnittliche Antwortzeit pro Server (Test 1)

Im Testszenario 1 wurde jeweils eine Kachel der Zoomstufe 14 mit der Bodenbedeckung als Polygondatensatz angefordert. Martin erzielte mit durchschnittlich 95 Millisekunden die schnellsten Antwortzeiten. Pg_tileserv benötigte mit 741 Millisekunden mehr als doppelt so lange wie die Server BBOX und Tegola oder fast achtmal so lange wie Martin für die Auslieferung der Kachel. TiPg und Ldproxy benötigten jeweils durchschnittlich 2 bzw. 4 Sekunden. Allerdings wies TiPg eine hohe Standardabweichung von 476 und einen Maximalwert von 2767 Millisekunden auf.

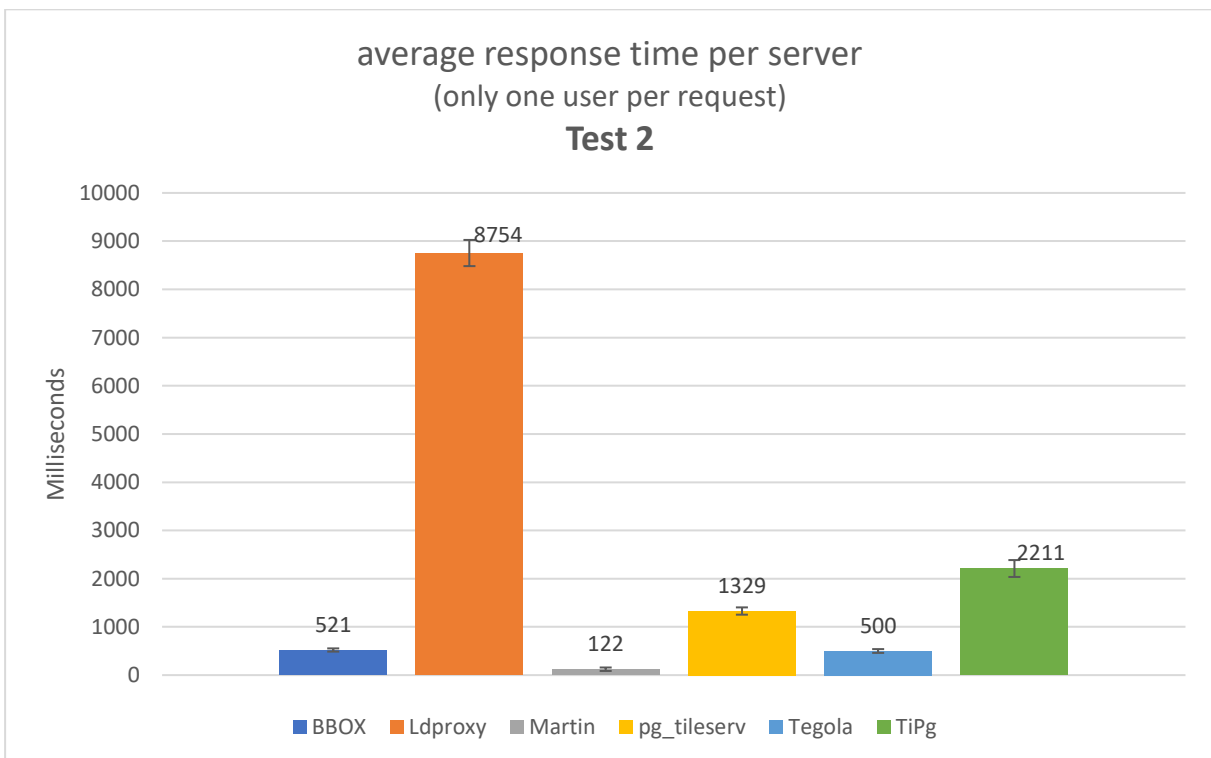


Abbildung 22: Durchschnittliche Antwortzeit pro Server (Test 2)

Im Testszenario 2 wurden die gleichen Daten wie im Test 1 angefordert, jedoch in Zoomstufe 15. Daher wurden jeweils 4 Kacheln gleichzeitig angefragt. Ldproxy benötigte mit fast 9 Sekunden für die Ausgabe der vier Kacheln über 70-mal so lange wie Martin mit 122 Millisekunden. Wie im Test 1, brauchten die Server BBOX und Tegola mit einer halben Sekunde, ungefähr gleiche lange.

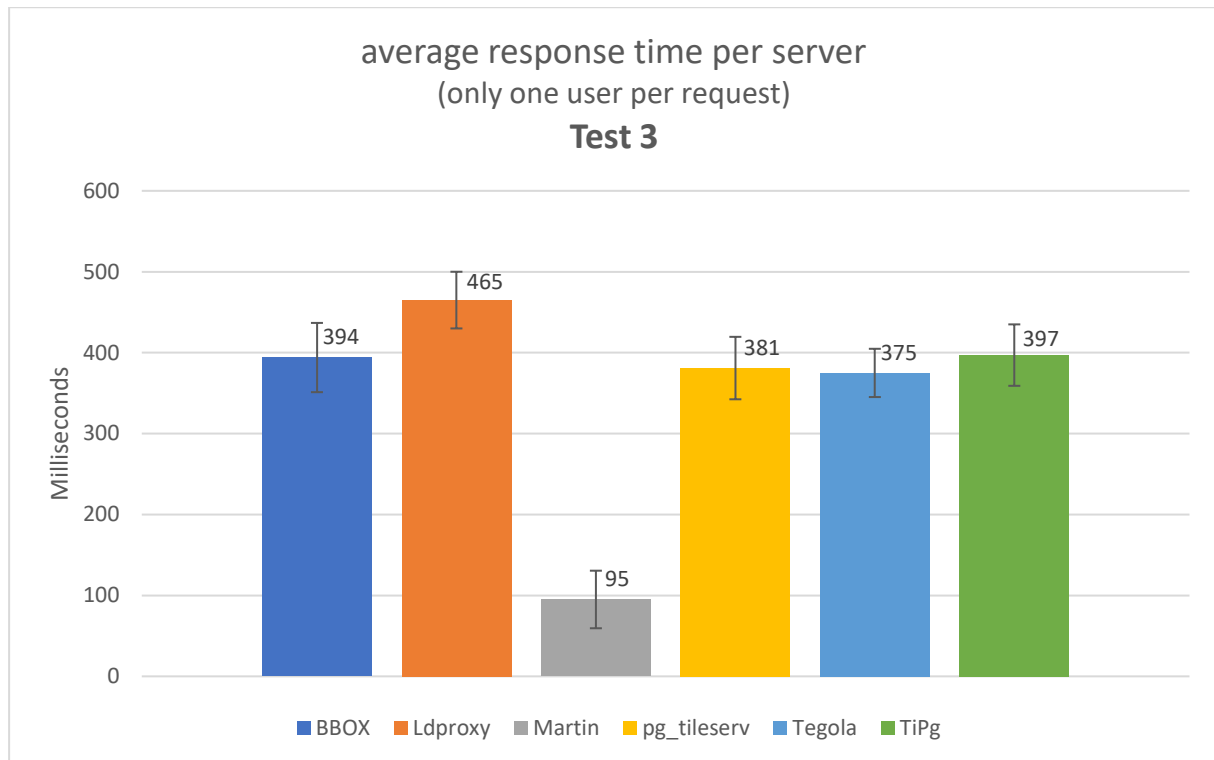


Abbildung 23: Durchschnittliche Antwortzeit pro Server (Test 3)

Im Test 3 wurde der komplette Datensatz der Gemeindegrenzen in Zoomstufe 8 in einer Kachel angefordert. Abgesehen vom Server Martin haben die Server sehr ähnliche Antwortzeiten. Auch Ldproxy konnte die Kacheln mit 465 Millisekunden fast 10-mal schneller ausliefern als beim Test 1. Martin benötigte mit durchschnittlich 95 Millisekunden jedoch genau so lange für die Auslieferung der Kachel wie beim Test 1, ist damit aber trotzdem noch fast 4-mal schneller als der zweitschnellste Server Tegola. Test 3 ist der einzige Test, bei dem Pg_tileserv die Kacheln schneller ausliefern konnte als BBOX, weshalb sich hier die Plätze in der Rangliste vertauschen.

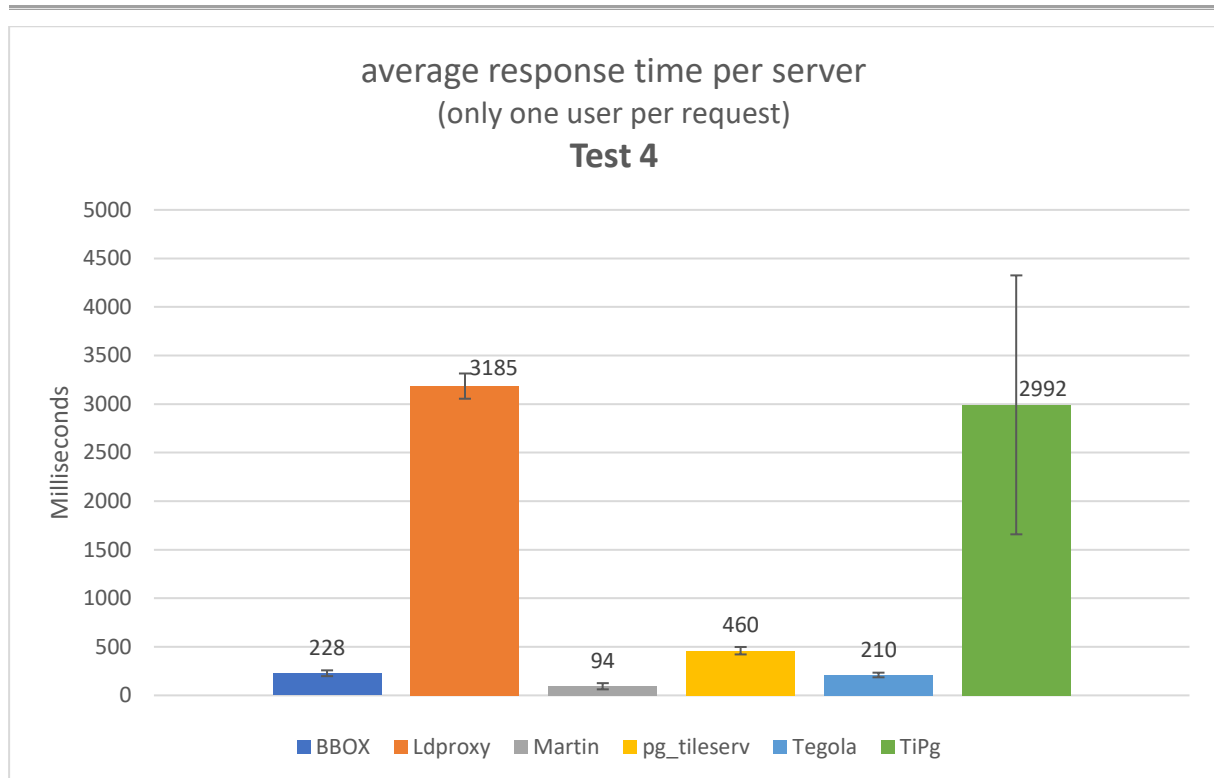


Abbildung 24: Durchschnittliche Antwortzeit pro Server (Test 4)

Beim Test 4 wurden die gleichen Kacheln wie im Test 1, jedoch mit den Grenzpunkten als Datensatz, angefragt. Auffällig in diesem Diagramm ist die im Vergleich zu den anderen Tests hohe Antwortzeit des Servers TiPg. Ausserdem weisen die Ergebnisse eine sehr hohe Standardabweichung von 1333 Millisekunden und einen Maximalwert von 4683 Millisekunden auf. Während der Analyse fiel auf, dass einige der 100 Anfragen nur etwa 400 Millisekunden und andere etwa 4000 Millisekunden benötigten. Der Grund für die starken Schwankungen konnte aus den Daten nicht ermittelt werden. Die durchschnittliche Antwortzeit des Servers TiPg im Test 4 sollte daher nicht als zuverlässiger Indikator für seine Leistungsfähigkeit betrachtet werden.

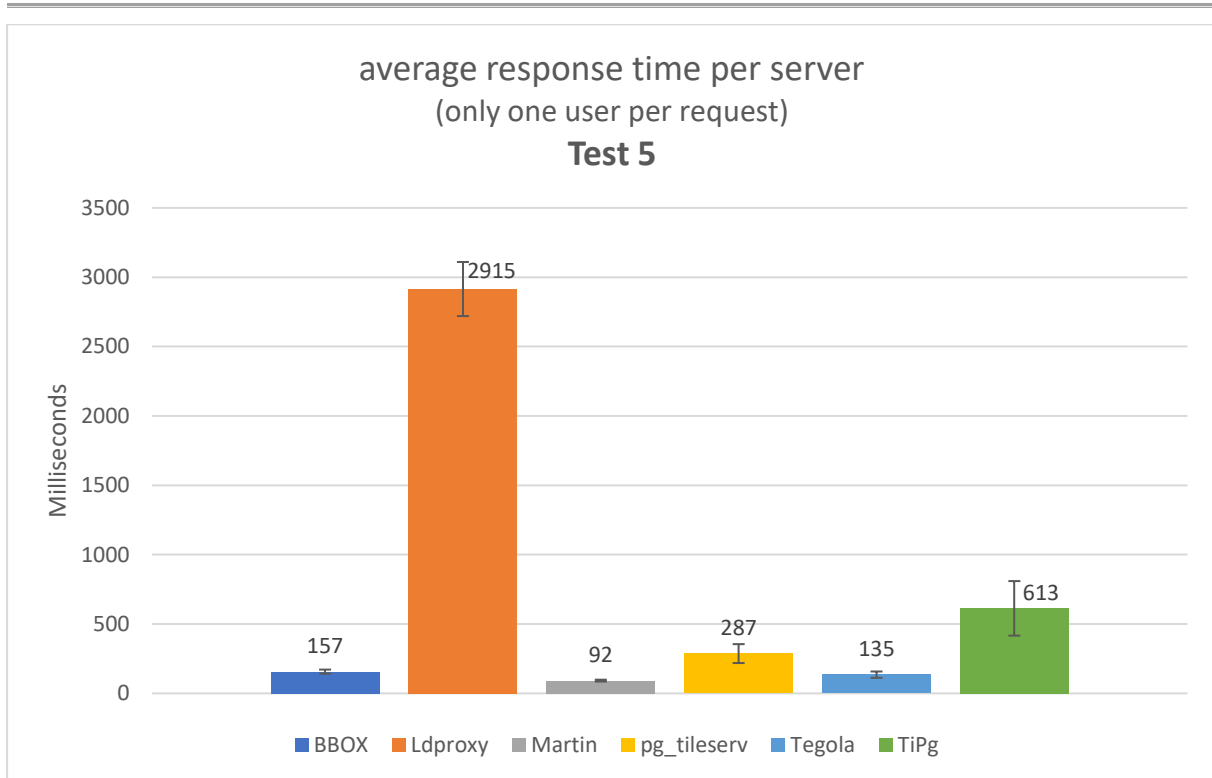


Abbildung 25: Durchschnittliche Antwortzeit pro Server (Test 5)

Im Test 5 wurden Liniengeometrien der Einzelobjekte angefragt. Das Balkendiagramm ähnelt stark dem aus Test 1, jedoch mit kürzeren Antwortzeiten für alle Server. Mit 92 Millisekunden lieferte Martin die Kachel im Test 5 am schnellsten zurück. Im Vergleich zu den 95 Millisekunden im Test 1 war die Anfrage jedoch nicht signifikant schneller. Andere Server benötigten hingegen nur halb oder sogar ein Drittel der Zeit für die Auslieferung der Kacheln im Vergleich zum Test 1.

Datenmenge

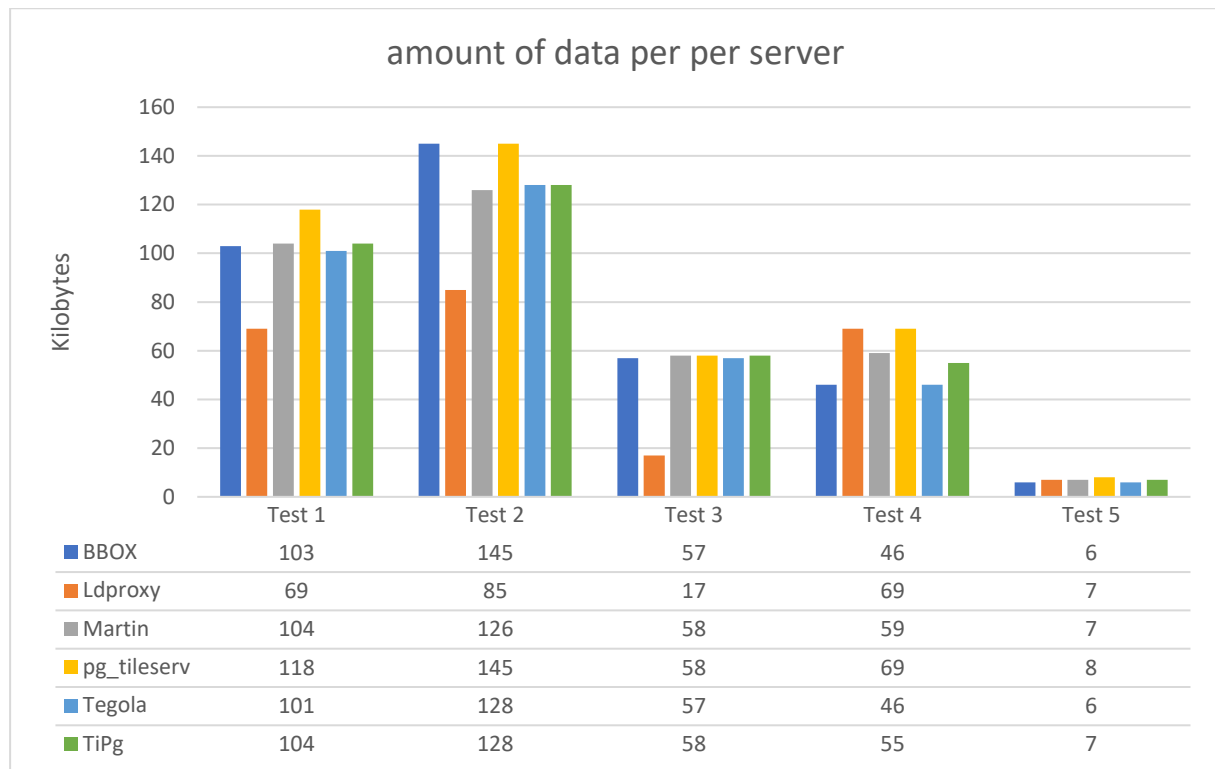


Abbildung 26: Datenmenge pro Server

Die Abbildung 26 zeigt die Datenmengen in Kilobyte, die bei jedem Test pro Server heruntergeladen wurden. Über alle Tests hinweg sind die Dateigrößen bei pg_tileserv am höchsten. Beim Test 2 wurden bei allen Servern mehr Daten heruntergeladen als beim Test 1, obwohl dieselbe Datengrundlage und derselbe Ausschnitt verwendet wurden. Es sei nochmals erwähnt, dass im Test 2 eine tiefere Zoomstufe und 4 Kacheln angefragt wurden. Abgesehen vom Test 2 sind die Datenmengen der Server BBOX, Martin, Tegola und TiPg bei allen Tests ungefähr gleich. Der Server Ldproxy bezog bei den Tests 1 bis 3 mit den Polygoneometrien jeweils 30 bis 70 % weniger Daten als die anderen Server. Beim Test 4 mit den Grenzpunkten war die Datenmenge bei Ldproxy mit 69 Kilobyte jedoch am höchsten. Die grösste Kachel wurde im Test 2 von den Servern BBOX und pg_tileserv mit je 145 Kilobyte erzeugt. Beim Test 5 mit den Liniengeometrien wurden mit Datenmengen von 6 bis 8 Kilobyte die kleinsten Kacheln generiert.

6.2 RESULTATE MIT MEHREREN BENUTZER

Die folgenden Abbildungen zeigen die jeweiligen Server und deren Antwortzeiten für 1, 10 und 100 Benutzer pro Test an. Einige Server konnten nicht alle Tests mit 100 Benutzern erfolgreich durchführen, weshalb für diese keine Daten angezeigt werden können. Zur besseren Visualisierung und Interpretation der Daten wurden in allen fünf Diagrammen sowohl eine logarithmische als auch eine lineare Skalierung definiert. Im Test 1 wird die Skalierung von 1 bis 1'000'000 angezeigt, anstelle von nur bis 100'000.

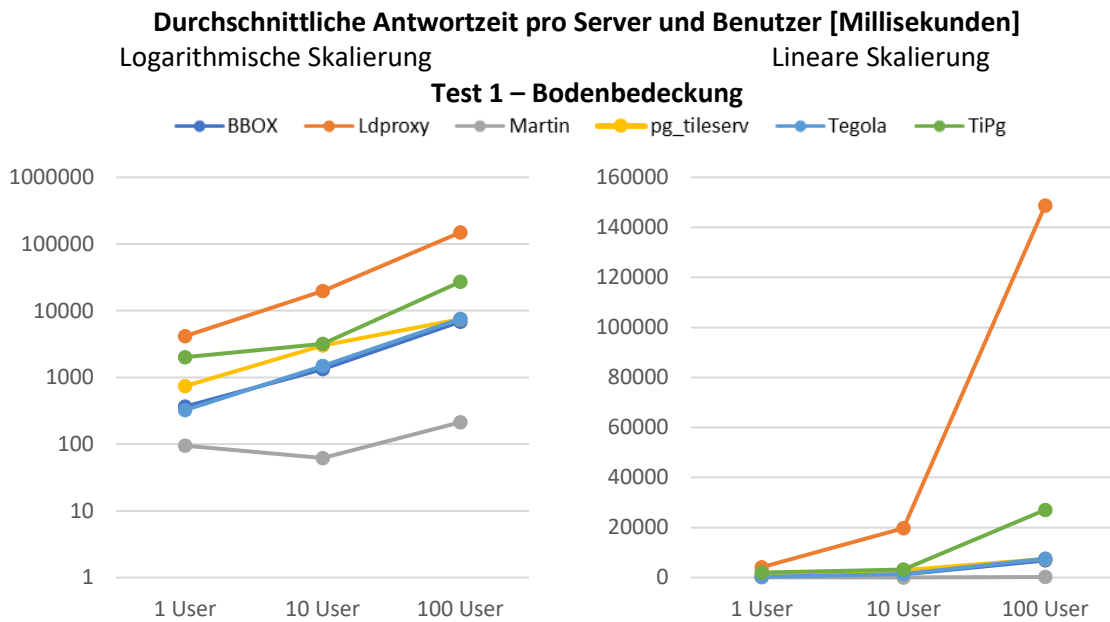


Abbildung 27: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 1, logarithmisch)

Abbildung 28: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 1, linear)

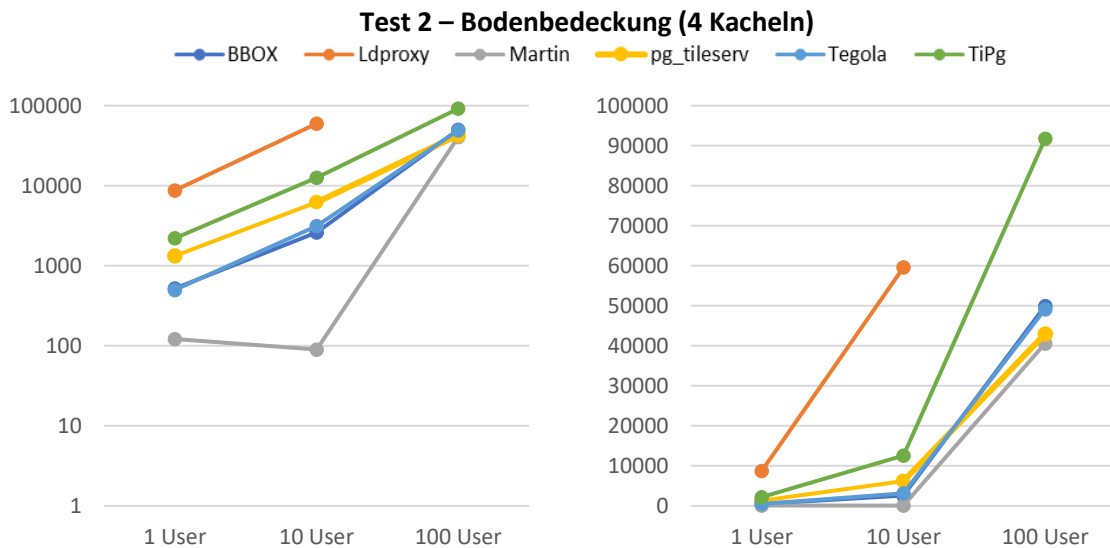


Abbildung 29: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 2, logarithmisch)

Abbildung 30: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 2, linear)

Test 3 – Gemeindegrenze

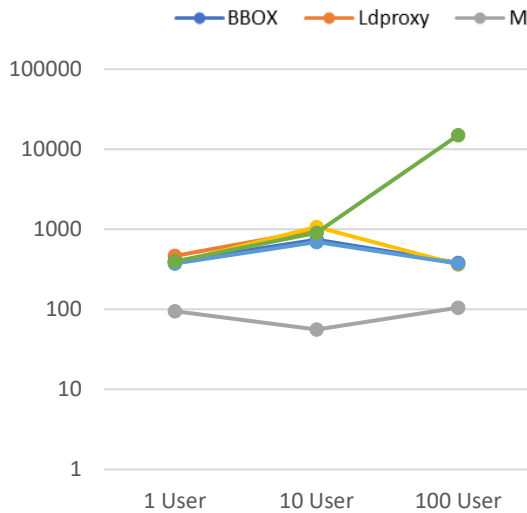


Abbildung 31: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 3, logarithmisch)

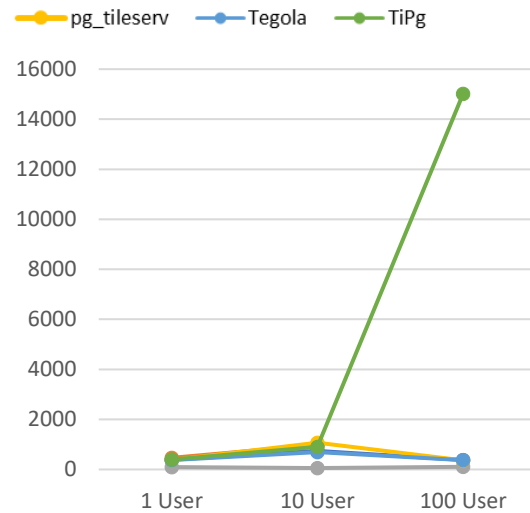


Abbildung 32: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 3, linear)

Test 4 – Grenzpunkte

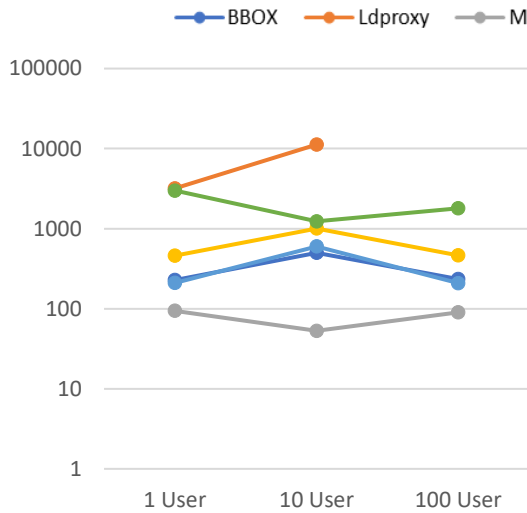


Abbildung 33: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 4, logarithmisch)

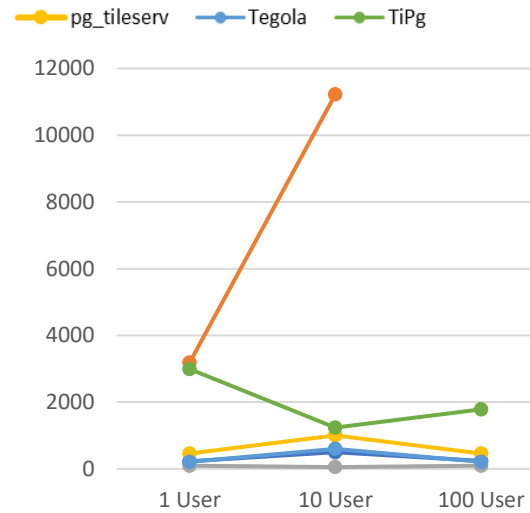


Abbildung 34: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 4, linear)

Test 5 – Einzelobjekte (Linienelemente)

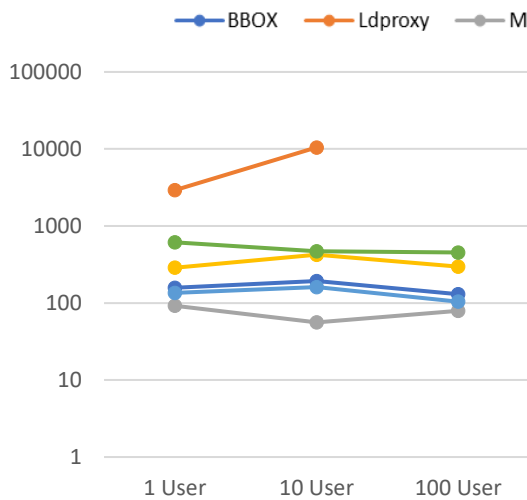


Abbildung 35: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 5, logarithmisch)

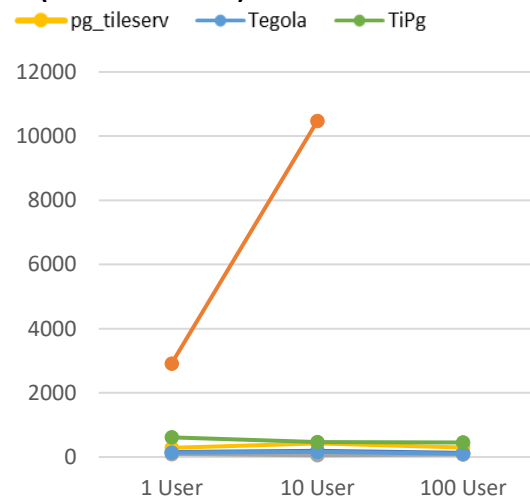


Abbildung 36: Durchschnittliche Antwortzeit pro Server und Benutzer (Test 5, linear)

Der Server Martin ist der einzige Server, bei dem die durchschnittlichen Antwortzeiten mit 10 Benutzern über alle Tests hinweg kleiner waren als die Anfragen mit nur einem Benutzer. Bei TiPg waren bei der Anfrage mit 10 oder 100 Benutzern die Antwortzeiten in den Tests 4 und 5 ebenfalls kleiner. Bei den anderen Servern haben sich die Antwortzeiten von einem Benutzer zu 10 Benutzern verdoppelt bis sogar versechsfacht.

Im Test 1 dauerte es bei allen Servern ausser bei Martin ungefähr viermal so lange, um auf Anfragen von 10 Benutzern zu reagieren. So stieg beispielsweise die Antwortzeit beim Server BBOX von 365 auf 1335 und 6860 Millisekunden. Beim Server Ldproxy stiegen die Antwortzeiten von 4153 auf 19779 und 148674 Millisekunden. Ldproxy benötigte somit über 2 Minuten, um die gleichzeitige Kachelanfrage von 100 Benutzern zu verarbeiten. Danach konnte der Server Ldproxy die weiteren Tests nicht mehr durchführen.

Alle Server hatten besonders bei Test 2 mit 100 gleichzeitigen Anfragen Schwierigkeiten. BBOX mit fast 7 Sekunden und Tegola mit 7.5 Sekunden brauchten über 95-mal länger als bei der einzelnen Anfrage. Ldproxy und TiPg benötigten über 35-mal länger.

Auffällig ist, dass Martin bei allen Tests bis auf Test 2 ähnliche Antwortzeiten aufweist. Im Test 2 stiegen die Antwortzeiten von 90 ms bei 10 Benutzer auf knapp 40 Sekunden bei der Anfrage mit 100 Benutzern, was eine Steigerung um das über 400-fache bedeutet. Die Analyse des Log-Files hat gezeigt, dass beim Test 2 mit 100 Benutzern die 4 einzelnen Kachelanfragen jeweils ungefähr 600 ms benötigten, der Parallel-Controller jedoch 40'000 ms brauchte. Es scheint, als hätte der Controller die vier Einzelanfragen nicht korrekt aufsummiert. Die Antwortzeit mit 100 Benutzern des Servers Martin darf deshalb nicht als zuverlässiger Indikator für die Leistungsfähigkeit dieses Servers betrachtet werden.

Beim Test 3 mit 100 Benutzern ist auffallend, dass TiPg ungefähr 40-mal länger für die Antworten brauchte als die anderen Server. Bei genauerer Analyse des Log-Files ist aufgefallen, dass TiPg beim Start des Tests 3 mit 100 Benutzern noch Anfragen vom Test 2 am Verarbeiten war, obwohl zwischen den Tests jeweils 10 Sekunden gewartet wurde. Beim Start des Tests 3 hatte der Server Antwortzeiten von 60 Sekunden. Parallel wurden aber Anfragen vom Test 2 und sogar bereits von Test 4 ausgeführt. Als alle Anfragen vom Test 2 beantwortet wurden, erreichte der Test 3 Antwortzeiten unter 400 ms. Aufgrund der parallelen Verarbeitung der verschiedenen Testanfragen ergab sich eine durchschnittliche Antwortzeit von 16 Sekunden.

Über alle Tests hinweg kann die Rangliste von den Tests mit einem Benutzer übernommen werden. Der Server Martin, der bei allen Tests mit einem Benutzer die Kacheln in kürzester Zeit ausliefert, liefert auch bei grösserer Last von 10 und 100 Benutzern die Kacheln am schnellsten aus. Die Server BBOX und Tegola haben wie bei den Tests mit einem Benutzer auch bei den Tests mit mehreren Benutzern sehr ähnliche Werte.

6.3 VISUELLER PERFORMANCEVERGLEICH

Um einen visuellen Geschwindigkeitsvergleich der Server durchzuführen, wurden für jeden Server individuelle Karten auf einer Webseite erstellt und die entsprechenden Vector Tiles integriert. Ein HTML-File wurde erstellt, das die JavaScript-Bibliothek Maplibre GL JS verwendet, um die Karten zu generieren. Die sechs Karten wurden synchronisiert, so dass Änderungen wie Zoomen und Verschieben auf einer Karte automatisch auf die anderen übertragen werden. Eine übergeordnete Layer-Steuerung ermöglicht das Ein- und Ausblenden folgender Layer für alle 6 Karten:

- Bodenbedeckung (bo_boflaeche_mv)
- Gemeindegrenze (gg_gemeindegrenze_mv)
- Liegenschaften (li_liegenschaft_mv)
- Grenzpunkte (li_grenzpunkt_mv)
- Einzelobjekte Linien (eo_linienelement_mv)
- Projektierte Gebäude (bo_projgebaeude_mv)

Im JavaScript wurden Funktionen genutzt, um den Import der gleichen Layer für alle Server zu vereinfachen. Alle Layer wurden in einer Variable „layers“ definiert, einschliesslich ihrer Datentypen, Zoomstufen und Stylings, und dann allen Karten hinzugefügt. Die URL für die Vector Tiles wurde parametrisiert, um Duplikationen zu vermeiden. Hier ist ein Codeausschnitt aus der Datei main.js:

```
Object.keys(layers).forEach(layer => {
  map.addSource(layers[layer].source, {
    'type': 'vector',
    'tiles': [
      `${prefix}${layers[layer].layer}${suffix}`
    ],
    'attribution': attribution,
    'minzoom': layers[layer].minzoom
  });

  map.addLayer({
    'id': layer,
    'type': layers[layer].type,
    'source': layers[layer].source,
    'source-layer': layers[layer].layer,
    'paint': layers[layer].paint
  });
});
```

Das HTML-File mit den zusätzlichen Dateien ist im GitHub-Repository unter folgendem Link abgelegt: <https://github.com/FabianRechsteiner/vector-tiles-benchmark/nginx>

Um die Vector Tiles für alle Server gleichzeitig bereitzustellen, wurde im Docker Compose-File ein zusätzliches Profil erstellt. Mit folgendem Befehl lassen sich alle sechs Servercontainer sowie der PostGIS-Container starten:

```
docker compose --profile all_server up -d
```

Es ist zu beachten, dass alle sechs Server dieselbe PostGIS-Datenbank verwenden, wodurch alle Anfragen gleichzeitig von PostGIS verarbeitet werden müssen, was sich auf die Performance auswirken kann.

Subjektive Beurteilung

Die Abbildung 37 zeigt einen Screenshot der erstellten Webseite mit den sechs Karten und den entsprechenden Vector Tiles Daten. In dem kurzen Video, das im GitHub-Repository bereitgestellt wurde, ist der Ladevorgang der Webseite und der Vector Tiles erkennbar. Die Ladezeiten der Vector Tiles wurden nicht weiter untersucht. Es erfolgte lediglich eine subjektive Beurteilung, in welcher Reihenfolge die Server ihre Daten darstellen konnten. Die Rangliste der Server in Bezug auf die Darstellungsgeschwindigkeit lautet wie folgt:

1. Martin
2. BBOX
3. Tegola
4. pg_tileserv /TiPg
5. Ldproxy

Ldproxy benötigte doppelt so lange wie die Server mit der zweitgeringsten Geschwindigkeit, pg_tileserv und TiPg, welche die Daten ungefähr gleichzeitig dargestellt haben.

Da die Anfragen für diesen visuellen Performancevergleich gleichzeitig an den PostGIS-Server gesendet werden und keine Kontrolle darüber besteht, wie PostGIS die Reihenfolge der Anfragen verarbeitet, sind keine aussagekräftigen Schlüsse über die Serverperformance möglich. Dennoch kann diese Methode dazu dienen, die Rangliste des Performancevergleichs mit JMeter zu bestätigen.

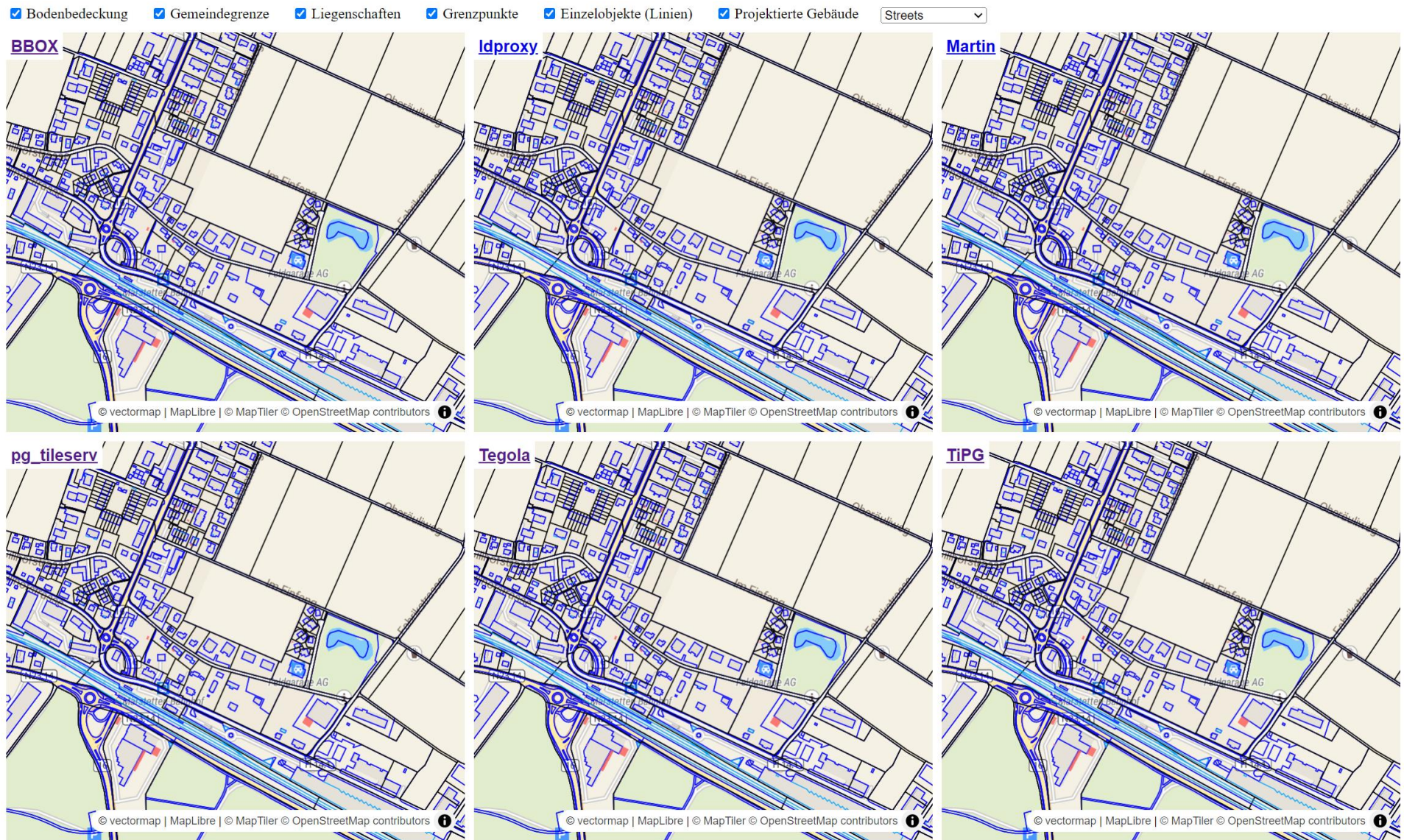


Abbildung 37: Screenshot HTML mit den Vector Tiles Daten

7 DISKUSSION

Zunächst einmal ist anzumerken, dass alle getesteten Serverlösungen bei Tests mit einem und zehn Benutzern erfolgreich waren, während bei Tests mit 100 Benutzern Fehler auftraten, insbesondere bei den Servern Ldproxy und pg_tileserv. Diese Fehler deuten darauf hin, dass die Server möglicherweise nicht in der Lage waren, die grosse Anzahl von Anfragen an die PostGIS-Datenbank weiterzuleiten, was zu Verbindungsproblemen und Fehlermeldungen führte.

Der Performancetest mit JMeter hat deutliche Unterschiede in den Geschwindigkeiten der Server aufgezeigt. Der Server Martin zeigte insgesamt die schnellste Leistung bei der Bereitstellung der Kacheln, während der Server Ldproxy auffällig langsamer war als die anderen Server. Dies könnte auf die unterschiedlichen Methoden zur Erstellung der Kacheln zurückzuführen sein, wie in Tabelle 1: Übersicht Vector Tiles Server aufgeführt. Ldproxy ist der einzige Server, der zur Generierung der Vector Tiles nicht die PostGIS-Funktion ST_AsMVT verwendet, sondern die Kacheln aus den Features generiert. Ldproxy kann die Daten sowohl als OGC API - Tiles als auch als OGC API - Features bereitstellen. Für die Generierung der Kacheln werden die Features generalisiert und auf die Kacheln zugeschnitten. Es scheint, dass dieser Prozess wesentlich langsamer ist als die direkte Generierung und Bereitstellung der Kacheln auf der Datenbank.

Die Antwortzeiten bei Tests mit einem Benutzer waren im Allgemeinen konsistenter als bei Tests mit mehreren Benutzern. Insbesondere der Server Martin zeigte eine bemerkenswerte Konsistenz seiner Antwortzeiten, während andere Server, wie Ldproxy und pg_tileserv, deutlich an Leistung einbüssten, wenn die Benutzerlast zunahm. Dies legt nahe, dass die Skalierbarkeit ein wichtiger Aspekt bei der Auswahl eines geeigneten Vector Tiles Servers ist, insbesondere wenn eine hohe Benutzerzahl erwartet wird.

Der visuelle Vergleich bestätigt grösstenteils die Rangliste der Server basierend auf den Antwortzeiten der Performancetests. Es ist jedoch wichtig zu beachten, dass aufgrund der parallelen Verarbeitung der Anfragen auf der PostGIS-Datenbank keine abschliessenden Schlussfolgerungen zur Serverperformance gezogen werden können.

In Abbildung 26: Datenmenge pro Server sind deutliche Unterschiede in den Dateigrössen der Polygondaten des Servers Ldproxy im Vergleich zu den anderen Servern zu erkennen. Es scheint, dass Ldproxy bei der Umwandlung der Features in Vector Tiles eine stärkere Generalisierung vornimmt und mehr Stützpunkte löscht als die PostGIS-Funktion ST_AsMVT. Ein visueller Vergleich der Kacheln mit QGIS bestätigt diese Annahme, insbesondere bei Kurven, wo Ldproxy deutlich mehr Stützpunkte entfernt als die anderen Server.

8 SCHLUSSFOLGERUNG UND AUSBLICK

Diese Arbeit gibt einen Überblick über die neuen OGC API Standards, wobei besonderes Augenmerk auf die OGC API - Tiles gelegt wird. Sechs Open-Source-Vector-Tiles-Server wurden in einer Public Cloud mittels Docker-Containern aufgesetzt und konfiguriert. Anschliessend wurde die Performance dieser Server bei der Bereitstellung von Vektordaten aus einer PostGIS-Datenbank untersucht. Der Server Martin erwies sich dabei als der eindeutig schnellste Server über alle Testszenarien hinweg. Die Vector Tiles aller sechs Server konnten erfolgreich in einer Webapplikation integriert und die Vektordaten visualisiert werden.

Obwohl gezeigt werden konnte, welche Server im relativen Vergleich am schnellsten waren, gestaltet sich eine absolute Beurteilung der Werte schwierig, da die Geschwindigkeiten stark von äusseren Einflüssen abhängen.

Grosses Potenzial sehe ich beim Server BBOX, dessen Entwicklung erst kürzlich begonnen hat und weitere Optimierungen zu erwarten sind. Dieser Server strebt neben OGC API - Tiles auch weitere OGC APIs an, um eine All-in-one-Lösung zur effizienten und modernen Bereitstellung von Geodaten anzubieten. Zudem unterstützt der Server durch externe Applikationen die Bereitstellung der alten OGC Standards WMS, WMTS und WFS.

Zur dynamischen Bereitstellung von Vector Tiles ohne Caching ist Ldproxy derzeit nicht geeignet. Die Nutzung von Ldproxy als Vector Tiles Server könnte optimiert werden, wenn die Kacheln mithilfe der PostGIS-Funktion `St_AsMVT` anstelle von Feature Collections erzeugt werden könnten. Interessant ist auch, dass Ldproxy WFS-Dienste als Datenquelle zur Erstellung der Vector Tiles nutzen kann. Es besteht somit die Möglichkeit, bestehende WFS-Dienste neu auch als OGC API - Features und OGC API - Tiles bereitzustellen. Jedoch ist anzunehmen, dass die Performance im Vergleich zum Original-WFS spürbar schlechter wäre. Falls die Kacheln jedoch sowieso vorgerechnet werden, wäre dies eine mögliche Option.

Es wäre ebenso interessant zu untersuchen, wie sich die Performance der Server verbessern würde, wenn die Kacheln nicht nur dynamisch vom Server berechnet, sondern auch serverseitig vorgerechnet und gecached würden. Es könnte vorteilhaft sein, bei bestimmten Datensätzen und Zoomstufen die Vector Tiles auch vorgerechnet und als MbTiles oder PMTiles filebasiert bereitzustellen, um die Performance weiter zu optimieren.

Ich bin überzeugt, dass die Stärken der Vector Tiles im Vergleich zu WMTS dazu führen werden, dass wir zukünftig schweizweit Vector Tiles zur Visualisierung von Vektordaten im Web einsetzen werden. Mein persönliches Ziel ist es, zu testen, wie die bestehende Base Map von swisstopo, die als Vector Tiles Service bereitgestellt wird, mit den Daten der Amtlichen Vermessung ergänzt werden kann. Dadurch könnten bei grossem Massstab statt der Bundesdaten die Vermessungsdaten dargestellt werden. Die Bereitstellung amtlicher Daten als Vector Tiles würde es ermöglichen, verschiedene Darstellungen von Hintergrundkarten für verschiedene Anwendungsbedürfnisse anzubieten.

Abschliessend möchte ich alle Geodatenanbieter ermutigen, ihre Daten gemäss den neuen OGC API Standards bereitzustellen. Durch die Nutzung dieser Standards können Geodaten besser ausgetauscht, genutzt und miteinander verknüpft werden und tragen so zu einer umfassenderen und effizienteren Nutzung der Geodaten bei.

9 LITERATURVERZEICHNIS

Apache JMeter - Apache JMeter™ (2024). Online verfügbar unter <https://jmeter.apache.org/>, zuletzt aktualisiert am 07.01.2024, zuletzt geprüft am 21.05.2024.

Apache JMeter - Download Apache JMeter (2024). Online verfügbar unter https://jmeter.apache.org/download_jmeter.cgi, zuletzt aktualisiert am 26.01.2024, zuletzt geprüft am 21.05.2024.

Apache JMeter - User's Manual: Elements of a Test Plan (2024). Online verfügbar unter https://jmeter.apache.org/usermanual/test_plan.html, zuletzt aktualisiert am 07.01.2024, zuletzt geprüft am 21.05.2024.

Apache JMeter - User's Manual: Getting Started (2024). Online verfügbar unter <https://jmeter.apache.org/usermanual/get-started.html#running>, zuletzt aktualisiert am 07.01.2024, zuletzt geprüft am 21.05.2024.

BBOX - Documentation (2024). Online verfügbar unter <https://www.bbox.earth/index.html>, zuletzt aktualisiert am 09.02.2024, zuletzt geprüft am 21.05.2024.

BBOX - GitHub. `bbox-services/bbox`: BBOX services (2024). Online verfügbar unter <https://github.com/bbox-services/bbox>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Docker - `postgis/postgis:16-3.4-alpine` (2024). Online verfügbar unter <https://hub.docker.com/layers/postgis/postgis/16-3.4-alpine/images/sha256-6c3cab2248825704ab57de84b65033a8c2df532d2f8cfc3a1a2909e17b93e991?context=explore>, zuletzt aktualisiert am 09.02.2024, zuletzt geprüft am 21.05.2024.

Docker Documentation (2024a): Install Docker Engine on Ubuntu. Online verfügbar unter <https://docs.docker.com/engine/install/ubuntu/>, zuletzt aktualisiert am 09.02.2024, zuletzt geprüft am 21.05.2024.

Docker Documentation (2024b): Migrate to Compose V2. Online verfügbar unter <https://docs.docker.com/compose/migrate/#what-are-the-differences-between-compose-v1-and-compose-v2>, zuletzt aktualisiert am 09.02.2024, zuletzt geprüft am 21.05.2024.

`gospatial/tegola` - Docker Image (2024). Online verfügbar unter <https://hub.docker.com/r/gospatial/tegola>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

`iide/ldproxy` - Docker Image (2024). Online verfügbar unter <https://hub.docker.com/r/iide/ldproxy>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Infomaniak (2023): Public Cloud – Cloud-Infrastruktur für Ihr Unternehmen. Online verfügbar unter <https://www.infomaniak.com/de/hosting/public-cloud>, zuletzt aktualisiert am 09.10.2023, zuletzt geprüft am 21.05.2024.

Ingensand, Jens; Nappez, Marion; Moullet, Cédric; Gasser, Loïc; Ertz, Olivier; Composto, Sara (2016): Implementation of tiled vector services : a case study. In: *Proceedings of the Workshop on Spatial Data on the Web (SDW 2016), 27-30 September 2016, Montreal, Canada*. Online verfügbar unter <https://arodes.hes-so.ch/record/4669>, zuletzt geprüft am 21.05.2024.

Kalberer, Pirmin (2023): BBOX – a modular OGC API server. Online verfügbar unter <https://www.youtube.com/watch?v=FnDj84ECgv0>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Kalberer, Pirmin (2024): Pirmin's Blog - Sabbatical - the plan. Online verfügbar unter <https://kalberer.org/pirmin/blog/sabbatical-plans/>, zuletzt aktualisiert am 09.02.2024, zuletzt geprüft am 21.05.2024.

Kamilaris, Andreas; Ostermann, Frank (2018): Geospatial Analysis and the Internet of Things. In: *IJGI* 7 (7), S. 269. DOI: 10.3390/ijgi7070269.

Kotsev, Alexander; Minghini, Marco; Tomas, Robert; Cetl, Vlado; Lutz, Michael (2020): From Spatial Data Infrastructures to Data Spaces—A Technological Perspective on the Evolution of European SDIs. In: *IJGI* 9 (3), S. 176. DOI: 10.3390/ijgi9030176.

`ldproxy` - Documentation (2024). Online verfügbar unter <https://docs.ldproxy.net/>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Idproxy - GitHub. interactive-instruments/Idproxy: Share geospatial data via modern Web APIs (2024). Online verfügbar unter <https://github.com/interactive-instruments/Idproxy>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Idproxy - Tiles (2024). Online verfügbar unter <https://docs.idproxy.net/de/providers/tile/>, zuletzt aktualisiert am 15.03.2024, zuletzt geprüft am 21.05.2024.

Lenka, Rakesh Kumar; Rani Dey, Meenu; Bhanse, Pranali; Barik, Rabindra Kumar (2018): Performance and Load Testing: Tools and Challenges. In: 2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE): IEEE.

Mapbox (2023a): Vector tiles standards. Online verfügbar unter <https://docs.mapbox.com/data/tilesets/guides/vector-tiles-standards/>, zuletzt aktualisiert am 29.08.2023, zuletzt geprüft am 21.05.2024.

Mapbox (2023b): MBTiles. Online verfügbar unter <https://docs.mapbox.com/help/glossary/mbtiles/>, zuletzt aktualisiert am 30.09.2023, zuletzt geprüft am 21.05.2024.

Maputnik (2024). Online verfügbar unter <https://maputnik.github.io/>, zuletzt aktualisiert am 14.02.2024, zuletzt geprüft am 21.05.2024.

martin - Docker Image (2024). Online verfügbar unter <https://github.com/maplibre/martin/pkgs/container/martin>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Martin - Documentation (2024). Online verfügbar unter <https://maplibre.org/martin/>, zuletzt aktualisiert am 08.02.2024, zuletzt geprüft am 21.05.2024.

Martin - GitHub. maplibre/martin: Blazing fast and lightweight PostGIS, MBtiles and PMtiles tile server, tile generation, and mbtiles tooling (2024). Online verfügbar unter <https://github.com/maplibre/martin>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Martin - Webpage (2023). Online verfügbar unter <https://martin.maplibre.org/>, zuletzt aktualisiert am 10.11.2023, zuletzt geprüft am 21.05.2024.

Morris, Steven P. (2006): Geospatial Web Services and Geoarchiving: New Opportunities and Challenges in Geographic Information Service. In: *Library Trends* 55 (2), S. 285–303. DOI: 10.1353/lib.2006.0059.

Netek, Rostislav; Masopust, Jan; Pavlicek, Frantisek; Pechanec, Vilem (2020): Performance Testing on Vector vs. Raster Map Tiles—Comparative Study on Load Metrics. In: *IJGI* 9 (2), S. 101. DOI: 10.3390/ijgi9020101.

OGC API (2024). Online verfügbar unter <https://ogcapi.ogc.org/>, zuletzt aktualisiert am 16.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - 3D GeoVolumes (2024). Online verfügbar unter <https://docs.ogc.org/DRAFTS/22-029.html>, zuletzt aktualisiert am 27.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Common (2023). Online verfügbar unter <https://ogcapi.ogc.org/common/>, zuletzt aktualisiert am 11.04.2023, zuletzt geprüft am 21.05.2024.

OGC API - Connected Systems GitHub (2024). Online verfügbar unter <https://github.com/opengeospatial/ogcapi-connected-systems>, zuletzt aktualisiert am 28.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Coverages - Overview (2024). Online verfügbar unter <https://ogcapi.ogc.org/coverages/overview.html>, zuletzt aktualisiert am 13.02.2024, zuletzt geprüft am 21.05.2024.

OGC API - Discrete Global Grid Systems - Part 1: Core (2024). Online verfügbar unter <https://opengeospatial.github.io/ogcna-auto-review/21-038.html>, zuletzt aktualisiert am 23.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - EDR - OGC API workshop (2024). Online verfügbar unter <https://ogcapi-workshop.ogc.org/api-deep-dive/environmental-data-retrieval/>, zuletzt aktualisiert am 11.03.2024, zuletzt geprüft am 21.05.2024.

OGC API - Features - OGC API workshop (2024). Online verfügbar unter <https://ogcapi-workshop.ogc.org/api-deep-dive/features/>, zuletzt aktualisiert am 11.03.2024, zuletzt geprüft am 21.05.2024.

OGC API - Features: GitHub (2024). Online verfügbar unter <https://github.com/opengeospatial/ogcapi-features>, zuletzt aktualisiert am 20.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Joins - Overview (2023). Online verfügbar unter <https://ogcapi.ogc.org/joins/overview.html>, zuletzt aktualisiert am 14.11.2023, zuletzt geprüft am 21.05.2024.

OGC API - Maps - Part 1: Core (2024). Online verfügbar unter <https://docs.ogc.org/DRAFTS/20-058.html>, zuletzt aktualisiert am 23.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Moving Features (2024). Online verfügbar unter <https://ogcapi.ogc.org/movingfeatures/>, zuletzt aktualisiert am 04.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Processes (2023). Online verfügbar unter <https://ogcapi.ogc.org/processes/>, zuletzt aktualisiert am 04.04.2023, zuletzt geprüft am 21.05.2024.

OGC API - Records (2022). Online verfügbar unter <https://ogcapi.ogc.org/records/>, zuletzt aktualisiert am 09.05.2022, zuletzt geprüft am 21.05.2024.

OGC API - Routes - Part 1: Core (2024). Online verfügbar unter <https://docs.ogc.org/DRAFTS/21-000.html>, zuletzt aktualisiert am 23.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Styles (2024). Online verfügbar unter <https://docs.ogc.org/DRAFTS/20-009.html>, zuletzt aktualisiert am 23.04.2024, zuletzt geprüft am 21.05.2024.

OGC API - Tiles - Part 1: Core (2022). Online verfügbar unter <http://www.opengis.net/doc/IS/ogcapi-tiles-1/1.0>, zuletzt aktualisiert am 07.12.2022, zuletzt geprüft am 21.05.2024.

OGC SensorThings API - Overview (2023). Online verfügbar unter <https://ogcapi.ogc.org/sensorthings/overview.html>, zuletzt aktualisiert am 05.12.2023, zuletzt geprüft am 21.05.2024.

OGC Two Dimensional Tile Matrix Set and Tile Set Metadata (2022). Online verfügbar unter <http://www.opengis.net/doc/IS/tms/2.0>, zuletzt aktualisiert am 16.09.2022, zuletzt geprüft am 21.05.2024.

Open Geospatial Consortium (2023): About - Open Geospatial Consortium. Online verfügbar unter <https://www.ogc.org/about-ogc/>, zuletzt aktualisiert am 04.04.2023, zuletzt geprüft am 21.05.2024.

OpenStack Docs: (2024). Online verfügbar unter <https://docs.openstack.org/>, zuletzt aktualisiert am 04.01.2024, zuletzt geprüft am 21.05.2024.

PBF Format – OpenStreetMap Wiki (2023). Online verfügbar unter https://wiki.openstreetmap.org/wiki/PBF_Format, zuletzt aktualisiert am 06.08.2023, zuletzt geprüft am 21.05.2024.

pg_featureserv (2023). Online verfügbar unter https://access.crunchydata.com/documentation/pg_featureserv/latest/, zuletzt aktualisiert am 11.10.2023, zuletzt geprüft am 21.05.2024.

pg_tileserv - Documentation (2023). Online verfügbar unter https://access.crunchydata.com/documentation/pg_tileserv/latest/, zuletzt aktualisiert am 11.10.2023, zuletzt geprüft am 21.05.2024.

pg_tileserv - GitHub. CrunchyData/pg_tileserv: A very thin PostGIS-only tile server in Go. Takes in HTTP tile requests, executes SQL, returns MVT tiles (2024). Online verfügbar unter https://github.com/CrunchyData/pg_tileserv, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

pramsey/pg_tileserv - Docker Image (2024). Online verfügbar unter https://hub.docker.com/r/pramsey/pg_tileserv, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Protomaps. Protomaps is a serverless system for planet-scale maps (2023). Online verfügbar unter <https://protomaps.com/>, zuletzt aktualisiert am 03.10.2023, zuletzt geprüft am 21.05.2024.

Reed, Carl N. (2011): The Open Geospatial Consortium and Web Services Standards. In: Liping Di und Peisheng Zhao (Hg.): Geospatial web services. Advances in information interoperability. Hershey PA: Information Science Reference (Advances in Geospatial Technologies), S. 1–16.

Simoes, J.; Cerciello, A. (2022): SERVING GEOSPATIAL DATA USING MODERN AND LEGACY STANDARDS: A CASE STUDY FROM THE URBAN HEALTH DOMAIN. In: *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.* XLVIII-4/W1-2022, S. 419–425. DOI: 10.5194/isprs-archives-XLVIII-4-W1-2022-419-2022.

sourcepole/bbox-server-qgis - Docker Image (2024). Online verfügbar unter <https://hub.docker.com/r/sourcepole/bbox-server-qgis>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Speedtest.net (2024): Speedtest by Ookla - The Global Broadband Speed Test. Online verfügbar unter <https://www.speedtest.net/>, zuletzt aktualisiert am 26.01.2024, zuletzt geprüft am 21.05.2024.

ST_AsMVT (2023). Online verfügbar unter https://postgis.net/docs/ST_AsMVT.html, zuletzt aktualisiert am 25.09.2023, zuletzt geprüft am 21.05.2024.

ST_AsMVTGeom (2023). Online verfügbar unter https://postgis.net/docs/ST_AsMVTGeom.html, zuletzt aktualisiert am 25.09.2023, zuletzt geprüft am 21.05.2024.

Tegola - Documentation (2017). Online verfügbar unter <https://tegola.io/documentation/>, zuletzt aktualisiert am 29.11.2017, zuletzt geprüft am 21.05.2024.

Tegola - GitHub. go-spatial/tegola: Tegola is a Mapbox Vector Tile server written in Go (2024). Online verfügbar unter <https://github.com/go-spatial/tegola>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Tegola - Webpage (2017). Online verfügbar unter <https://tegola.io/>, zuletzt aktualisiert am 29.11.2017, zuletzt geprüft am 21.05.2024.

tipg - Docker Image (2024). Online verfügbar unter <https://github.com/developmentseed/tipg/pkgs/container/tipg>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

TiPg - Documentation (2024). Online verfügbar unter <https://developmentseed.org/tipg/>, zuletzt aktualisiert am 02.02.2024, zuletzt geprüft am 21.05.2024.

TiPg - GitHub. developmentseed/tipg: Simple and Fast Geospatial OGC Features and Tiles API for PostGIS (2024). Online verfügbar unter <https://github.com/developmentseed/tipg>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

t-rex - GitHub. t-rex-tileservers/t-rex: t-rex is a vector tile server specialized on publishing MVT tiles from your own data (2024). Online verfügbar unter <https://github.com/t-rex-tileservers/t-rex>, zuletzt aktualisiert am 15.02.2024, zuletzt geprüft am 21.05.2024.

Wallner, Andreas Georg; Piechl, Thomas; Paulus, Gernot; Anders, Karl-Heinrich (2022): Open source vector tile creation for spatial data infrastructure applications. In: *AGILE GIScience Ser. 3*, S. 1–7. DOI: 10.5194/agile-giss-3-67-2022.

Wang, Junmei; Wu, Jihong (2019): Research on Performance Automation Testing Technology Based on JMeter. In: 2019 International Conference on Robots & Intelligent System (ICRIS): IEEE.