Geo Engine: Workflow-backed Geo Data Portals

Christian Beilschmidt[1] Johannes Drönner[1] Michael Mattig[1] Philip Schweitzer[1] Bernhard Seeger[1 2]

Abstract: Geo data portals play a key role in the distribution and exploitation of domain-specific geo data. While such portals are highly specialized, they share a number of common requirements that span from data access and processing to UI components. Geo Engine is able to provide all the necessary parts for portal building. We demonstrate this on a real data portal we built for the dragonfly community. In addition, we show its general architecture and outline future improvements.

Keywords: Geo processing; Data portals

1 Introduction

In recent years, many tailor-made geo data portals have arisen to provide specific data and services to a domain-specific user group. The primary goal is to provide powerful geo-temporal insight as simply as possible for a few dedicated use cases. On the other hand, geographic information systems (GIS) have matured over the last two decades to address the needs of processing Big Data regarding volume, variety, and velocity. However, there is a high demand for systems providing powerful processing and the ability to create customized portals to empower non-technical users to take full advantage of open and FAIR data [Wi16]. So far, current portals are not capable of dealing with Big Data.

The foundation of generating customized geo data portals are building blocks for accessing large raster and vector data, geo-temporal workflow processing and analysis, and flexible data access by standard-compliant interfaces. These blocks are already integral components of Geo Engine, a novel service platform that has already served as the basis for various portals. Geo Engine is the successor of the VAT system (cf. Sect. 5), which already powered GFBio's [Di14] data portal until 2021. We took the lessons learned from our five-year development of VAT and designed Geo Engine as an entirely new system that overcomes previous limitations. Geo Engine offers new functionality to empower (data) scientists dealing with large and heterogeneous datasets via different semantically equivalent interfaces (e.g., a Python interface for programming enthusiasts and a no-code interface for less technically experienced users). Among the many novel features of Geo Engine is its abstraction for data access and its consistent temporal approach to treating every data item as a time series.

[1] Geo Engine GmbH, Am Kornacker 68, 35041 Marburg, Germany {firstname.lastname}@geoengine.de
[2] University of Marburg, Dept. of Mathematics and Computer Science, Hans-Meerwein-Str., 35032 Marburg, Germany seeger@mathematik.uni-marburg.de

doi:10.18420/BTW2023-55

This makes Geo Engine a unique system for geo-spatial time series processing. Geo Engine is also a progressive system [Be19, Ho20] supporting the early delivery of approximate results to users to support an interactive and exploratory way of working on Big Data. In addition, Geo Engine offers the building blocks for customized portals such that only a thin mashup layer is required to meet users' specific demands.

This paper is structured as follows: In Sect. 2, we present Geo Engine's architecture and describe its data and processing model, as well as abstractions and components. In Sect. 3, we outline the requirements of data portals and how Geo Engine tackles them by providing suitable building blocks. In Sect. 4, we showcase a demo portal that is built on top of Geo Engine. In Sect. 5, we present related work. Finally, in Sect. 6, we give a brief summary and point out future directions of Geo Engine.

2 Geo Engine: Architecture Overview

Geo Engine consists of a backend[3] and two frontends: *geoengine-ui*[4] for Web and *geoengine-python*[5] (Fig. 1). The backend handles data access, data management and query execution. It also provides APIs for the frontends and third-party applications. OGC[6]-compliant interfaces, e.g. Web Map Service (WMS), Web Coverage Service (WCS), and Web Feature Service (WFS), allow access to data layers and computed layers derived on-the-fly at runtime. This makes Geo Engine compatible with most other geo software. The remaining functionality is available through a custom RESTful Web API with an OpenAPI[7] specification. We deploy Geo Engine using OCI[8] containers (e.g. Docker) where the backend and the selected frontend run in separate containers.

The backend of Geo Engine is written in Rust, a system language that overcomes many of the deficiencies of C and C++. It consists of three modules: data types, operators, and services. *Data types* contains the primitives for vector and raster data as well as basic operations and spatial projections. *Operators* contains the spatio-temporal query execution engine and the implementation of operators. *Services* contains the data management, i.e. adding, updating and removing artifacts such as datasets, workflows and projects, and Web APIs on top of this functionality. Optionally, it also handles user management, authentication via OpenID Connect [Sa14] single sign-on (SSO) providers, and authorization, which allows restricting access to resources such as data and workflows to certain users and groups.

The main output of Geo Engine are *layers* of spatio-temporal data: either feature collections or raster images. *Workflows* specify the processing of the *layers* as a graph of operators. All

[3] `github.com/geo-engine/geoengine`

[4] `github.com/geo-engine/geoengine-ui`

[5] `github.com/geo-engine/geoengine-python`

[6] `www.opengeospatial.org`

[7] `www.openapis.org`

[8] `www.opencontainers.org`

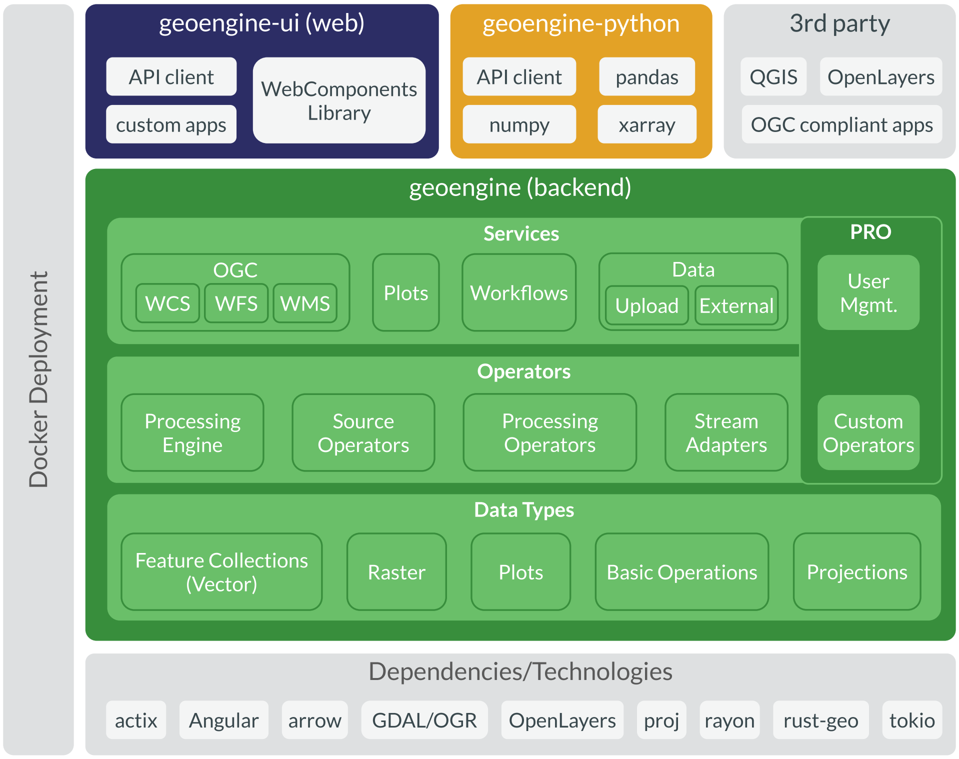


Fig. 1: A high-level overview of Geo Engine's architecture and its main components. The features in PRO are not open source but source-available. They are also freely available for non-commercial users and projects.

workflows consist of input operators and optionally processing operators. The two most important input operators are the *GdalSource* and the *OgrSource* that handle raster and vector data loading, respectively. They use the omnipresent GDAL library [Ro22] in order to support a variety of geospatial data formats as input. Processing operators either transform one piece of data into another, e.g. by filtering, or combine multiple inputs into a new output, e.g. attaching raster values to a point collection.

All data in Geo Engine is spatio-temporal and homogeneous. This means in a single raster all cells have the same size and the same data type. In contrast to data cubes, however, different rasters can still have other data types and sizes and will only be harmonized when it is necessary for a computation. In a feature collection, all features are of the same type, e.g. MultiPoint (cf. simple feature model [Op10]). This is necessary to define meaningful operations, e.g., filtering a collection's points based on them being contained in another collection's polygons.

Everything also has a temporal validity, defined as a half-open time interval $[\text{start}, \text{end})$. In a raster, all cells have the same temporal validity. In a feature collection, each feature has its own temporal validity. If a feature has multiple geometries, e.g. points in a MultiPoint, all geometries have the same temporal validity.

In order to enable the processing of datasets larger than the available main memory, all computations are performed on streams of chunks (vectors) and tiles (rasters) of a fixed size. A stream is a Rust data type that allows asynchronously producing and consuming data. Input operators produce chunks when they read the data and operators can *await* these chunks to be ready. In turn, other operators that use these outputs can again await them. This allows Geo Engine to interleave data loading and processing because operations do not block. The actual work is performed in a thread pool with a fixed number of workers, which is chosen with respect to the number of CPU cores.

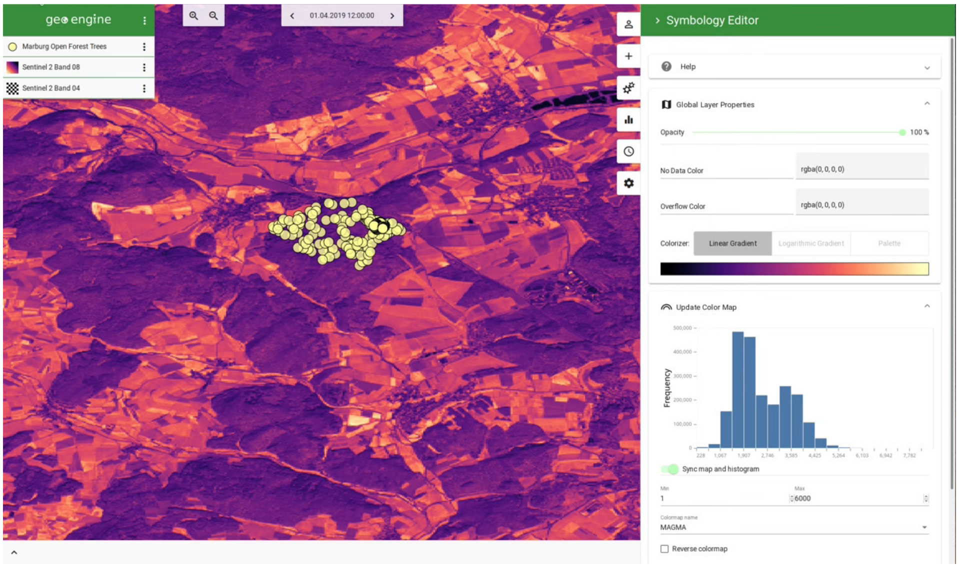
Raster tile streams are produced as *time first*, and *space second*. Starting with the first image in a raster time series, we produce the tiles starting from the top-left and increasing right and then down. Then, we continue with the image of the next time step in the same way. First of all, it is crucial to fix the tile order, such that consuming (parent) operators can rely on these order guarantees. Moreover, from our experience, this particular order is suitable for most common spatial operations on time series. For different requirements, e.g., temporal aggregations of single pixels, Geo Engine employs adapters that break the problem down into a set of subqueries that contain exactly one tile to generate a time-first stream.

Feature collections are split up by a fixed size limit such that chunks are of roughly the same size. Operators in turn have to produce new output chunks/tiles from input chunks. This sometimes requires reading the same piece of data multiple times, e.g. for a join or a convolution. This can be mitigated by employing an LRU cache to alleviate this problem of redundant computations.

For rasters, all processing is performed on a global grid with a fixed origin and a fixed tile size (e.g. 512×512 pixels). For a given query bounding box (BBox), we compute all the tiles that intersect this BBox. The major advantage of uniform tiling is that it allows us to easily combine multiple rasters, as the tiles are always aligned. It also allows easier re-use of cached results, because elements can be taken as they are and not be stitched together.

Geo Engine can access internal and external data. Our approach is to identify loadable data by an ID. An input operator obtains this ID as a parameter and resolves the necessary loading information using a metadata provider. This information is, for instance, the file name, the location, and the used spatial projection. Here, users can also specify regular time series by file names with date templates or a list of irregular time steps of a time series. Internal data are stored as *datasets* in a database. User can create their own datasets and share them with other users. External data is provided by *Data Providers*.

Data Providers allow (1) to browse and (2) to access data that is not managed by Geo Engine itself. In contrast to internal datasets, external data is referenced by an *external data ID* that



Opacity 100 %
No Data Color rgba(0, 0, 0, 0)
Overflow Color rgba(0, 0, 0, 0)
Colorizer: Linear Gradient | Logarithmic Gradient | Palette
Sync map and histogram
Min 1
Max 6000
Colormap name MAGMA
☐ Reverse colormap

Fig. 2: A screenshot of Geo Engine's GIS UI.

combines a *provider ID* with a *layer ID*. When an input operator gets an external data ID, it uses the data provider to resolve the necessary loading information for the layer ID. In contrast to local datasets, external data cannot be edited or deleted and the available data may change over time. Some examples of external data providers are generic WCS and STAC services, and more specialized ones like the GEO BON EBV Data Portal[9] and NFDI Core Storage[10]. To browse all available data, Geo Engine exposes internal and external data in a uniform layer collection API.

Our Web frontend *geoengine-ui* consists of three parts: a core library, a GIS application, and multiple apps and dashboards. The core provides a client implementation for the backend API services, e.g. for managing layers, and building block components like the map, plots, and operator dialogs. These components are all interconnected via application-wide states and services within a reactive application architecture. For instance, a data table and a map layer would relate to the same layer object that is managed by the core library. The GIS application (Fig. 2) offers the full functionality of Geo Engine, which is targeted at expert users. They can work with multiple layers, apply operators and review workflow graphs. All views (map, plots, data table) are synchronized and automatically adjust to the selected time and map extent. For instance, when a user pans to a different spatial area, a histogram plot would be recomputed to reflect the visible data on the screen. The dashboards are much simpler applications that focus on a concrete use case and only require access to a few

[9] `portal.geobon.org`
[10] `www.nfdi4biodiversity.org`

selected inputs. This allows for building easy-to-use domain-specific dashboards that still are able to leverage the full power of the Geo Engine data access, operator engine, and UI components.

3 Building Data Portals using Geo Engine

Data portals are highly specific to their target audience, but most of them share the same fundamental requirements. From our experience, the following list captures the most important properties of a modern geo data portal:

R1 Flexible data access to different data types and formats

R2 Combining local and remote data sources

R3 Basic layers and derived layers using GIS operations on available data

R4 A map as a central dashboard component

R5 A web-based user interface and reusable components

R6 User interactions, e.g. panning, zooming or selecting data subsets

R7 Time functionality for working with time series

R8 Multi-views, e.g. data tables and plots

R9 Access control, i.e. ensuring data privacy and having a multi-user system

R10 Administration tools for defining and managing the portal

R11 Response times that allow working interactively

The combination of rich data access (R1), flexible layer definitions that leverage workflows, basic GIS operators, and a UI component library makes it easy and efficient to build interactive geo data portals using Geo Engine. The data can either be added as internal datasets or accessed externally using a data provider, for instance, accessing a project database (R2). For displaying data as layers on the map, data is either used as it is or processed using advanced workflows (R3). In both cases, users can group the data as needed and uniformly browse them via our layer collection API. Then, the portal offers different display options, e.g., a simple list or cascaded dropdown lists for hierarchies (R6). Furthermore, the portal is able to show multiple datasets at once and, e.g., to enrich project data with other public data. As Geo Engine supports time as an integral dimension, it is easy to realize time sliders (R7). Moreover, in addition to the map, a portal can provide different kinds of plots and tabular views of the data (R8). If required, it can offer users to dynamically input their data, e.g., by drawing areas of interest, which act as new datasets that are read-to-use for processing in the portal.

Data is, by default, only processed with respect to the current map resolution, i.e. the number of pixels and the zoom level that is currently visible to the user. This allows Geo Engine to perform the calculations on overviews and avoids processing all the data at their finest level of detail. This leads to high interactivity (R11), even supporting a large number of operators and computing steps. If computations take too long for reasonable user interaction, administrators can also save the result of a workflow as a new dataset upfront and avoid recomputations.

Setting up a data portal currently consists of creating a new project that builds upon the core library of the geoengine-ui project. Then, developers can combine existing components into a domain-specific dashboard that can be enriched with custom texts and explanations. Additionally, they can define specific color schemes and styles, and add project-specific icons and logos. After setting up the application, one can set up datasets, layers, and other config items in the backend (R10). Datasets and layers are defined as JSON files and either loaded during the start of the backend or added later via REST. The layer IDs can be stored in the frontend's config for their retrieval at runtime. Finally, the dashboard is packaged as a container and deployed alongside a backend instance. For each user of the data portal, Geo Engine will create an anonymous user account on the fly and the user is logged into the system automatically. This allows users to get their own private session while avoiding a registration. However, it is also possible to add a global password or individual user accounts to an instance to allow access only to a specific target audience (R9).

Some of the main UI building blocks are the map, the layer list, the data table, plots, legends, and colorizers (R8). The map is typically the central point of attention in a geo data portal (R4). It overlays multiple layers of raster and vector data. The rasters are styled in the backend using a colorizer. The colorizer is predefined for each layer but can be changed by the user in an editor if it is integrated into the portal. Vectors are styled in the frontend itself and can also be customized, e.g., by varying the size of points depending on the value of an attribute. The plots are computed in the backend, but rendered in the frontend using the Vega plot grammar [Sa17]. This allows for a good visual quality and plot interactions but requires little computing resources. While in our GIS the plots are always linked to the map and data table with respect to time and space, portal creators are free to set time and space for maps and plots independently. Thus, it is, e.g., possible to show a plot for a whole year, while navigating through time month-by-month on the map.

Geo Engine's core UI library is based on Angular[11] components, which pose a form of Web Component[12] implementation (R5). Since Web Components are not yet fully standardized and thus have varying implementations, project dashboards currently need to be Angular

[11] www.angular.io
[12] www.webcomponents.org

projects as well. Since Angular and its Material Design[13] look-and-feel are widely used on the Web, they are familiar to users. In the future, it is likely that these components can more easily be used within different Web frameworks.

In addition to UI building blocks, for enthusiasts and advanced users, portal providers can use the geoengine-python library to allow users extended capabilities. Users can then get the portal data as geo data frames and `xarray`s [HH17] into their Jupyter Notebooks. There they can perform extended analyses and combinations with their own datasets.

4 A Use Case: Dragonfly Portal

As part of the NFDI4Biodiversity project, and in cooperation with the society of German-speaking odonatologists (GdO e.V)[14] we built a demo of a dragonfly geo portal for the GdOnline 2022 conference [GdOe22]. The portal allows visualizing occurrence data of dragonflies[15] for all of Germany. It offers some basic analysis functions, e.g., correlating occurrences with monthly temperature aggregates from remote sensing models (ECMWF ERA-5 Land [MS19]) or land cover based on Sentinel-2 data [Ri21]. It is meant as a collective hub of information about dragonflies (descriptions, pictures) and a contact point for people interested in dragonflies. Thus, it is a means to increase the visibility of the valuable work of volunteers who collect the data.

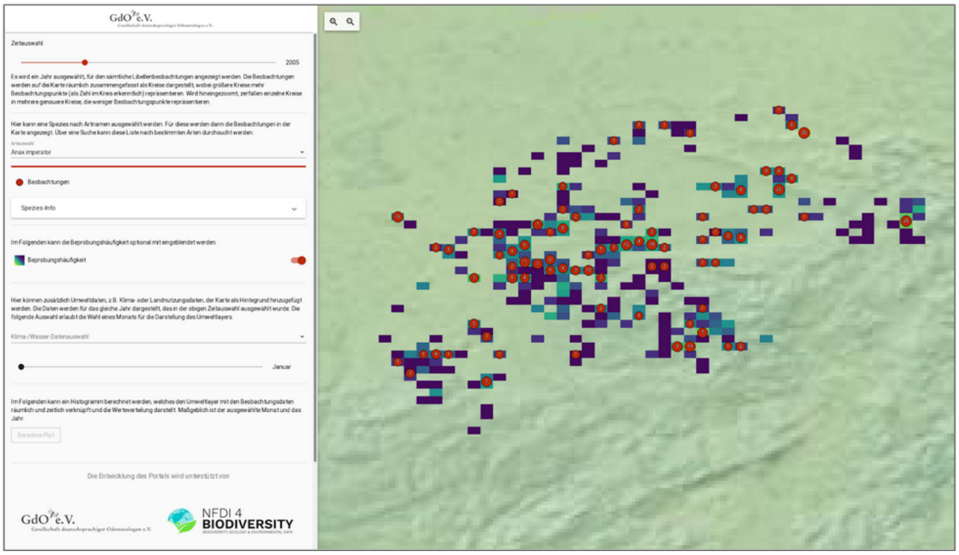
The portal is divided into two parts (Fig. 3). On the left-hand side, the data selection and plots are placed. On the right-hand side, we see the map. The user can select a dragonfly species by name and add an additional environmental layer, e.g. temperature or land use. Optionally, they can also activate the sampling frequency layer which gives some context to the absolute occurrence number shown in the observation points. This sampling frequency is pre-computed by a Geo Engine workflow of all dragonfly occurrences within a certain time period and the application of a grid-based rasterization that counts the number of observations per square.

The time selector on the top allows selecting the year. All observations within this year will be shown on the map as an aggregated number. This is internally done by applying a workflow that projects the temporal validity of the occurrences to full months. Doing this aligns the occurrences with the temporal selection as well as the temporal validity of the environmental layers. To avoid clutter and yield a better visualization, the points are clustered using the *VisualPointClustering* [Be17b] operator to present an overlap-free representation. The second time slider near the bottom allows selecting the month within the year. This slider is only relevant to the histogram plot that is optionally calculated when clicking a button at the bottom. The plot visualizes the correlation between the occurrences and the selected environmental variable within the chosen month.

[13] `www.material.io`

[14] `www.libellula.org`

[15] Demonstration data from AK Libellen NRW (2020), `www.ak-libellen-nrw.de`



Es wird ein Jahr ausgewählt, für den sämtliche Libellenbeobachtungen angezeigt werden. Die Beobachtungen werden auf dei Karte räumlich zusammengefasst als Kreise dargestellt, wobei größere Kreise mehr Beobachtungspunkte (als Zahl im Kreis erkenntlich) repräsentieren. Wird hineingezoomt, zerfallen einzelne Kreise in mehrere genauere Kreise, die weniger Beobachtungspunkte repräsentieren.

Hier kann eine Spezies nach Artnamen ausgewählt werden. Für diese werden dann die Beobachtungen in der Karte angezeigt. Über eine Suche kann diese Liste nach bestimmten Arten durchsucht werden.

Artauswahl: Anax imperator

Im Folgenden kann die Beprobungshäufigkeit optional mit eingeblendet werden.

Hier können zusätzlich Umweltdaten, z.B. Klima- oder Landnutzungsdaten, der Karte als Hintegrund hinzugefügt werden. Die Daten werden für das gleiche Jahr dargestellt, das in der obigen Zeitauswahl ausgewählt wurde. Die folgende Auswahl erlaubt die Wahl eines Monats für die Darstellung des Umweltlayers.

Im Folgenden kann ein Histogramm berechnet werden, welches den Umweltlayer mit den Beobachtungsdaten räumlich und zeitlich verknüpft und die Werteverteilung darstellt. Maßgeblich ist der ausgewählte Monat und das Jahr.

Berechne Plot
Die Entwicklung des Portals wird unterstützt von
GdO e.V.
Gesellschaft deutschsprachiger Odonatologen e.V.
NFDI 4 BIODIVERSITY
BIODIVERSITY, ECOLOGY & ENVIRONMENTAL DATA

Fig. 3: A screenshot of the GdO e.V. dragonfly portal demonstrator.

The dragonfly occurrence points use a number of Geo Engine features. The data is loaded from a GeoPackage database via Geo Engine's *OgrSource*. The filtering of points by species and aggregation over time is performed by the workflow processing engine. The selected time is used in the query rectangle of the query and allows reusing a fixed set of workflows for different points in time. In addition to the map view, the layer selection and legend displays are used from the core library. The plots use Geo Engine workflows to combine the occurrence points with the environmental raster data. This raster data is loaded from a set of Cloud-Optimized GeoTiffs (COG) via the *GdalSource*. The plots' data itself is calculated in the backend and displayed by the plot component from the core library.

The portal demonstrator poses an easy-to-use GIS subset within a domain-specific data portal. The portal providers can predefine a fixed set of data and derived data by registering workflows in the Geo Engine for future re-use. Note, that it is possible to use either Jupyter notebooks or the Geo Engine GIS application to formulate these workflows. In conclusion, the demonstrator is a successful example of leveraging Geo Engine's existing functionality instead of developing a tailor-fit data portal from scratch.

5 Related Work

Geo Engine is the successor of the VAT System [Au15b, Au15a, Be17a] that was developed as part of the GFBio project [Di14]. The original purpose of VAT was to make GFBio data more easily accessibly, but over time it developed into a more full-fledged geo data analysis platform. In comparison to VAT, Geo Engine offers a great number of improvements, of which we highlight the three most important ones. While VAT could only work on single rasters, Geo Engine is able to process arbitrary raster time series. Both for raster and vector data, Geo Engine is no longer limited by the amount of available main memory. While VAT required implementing special operators to access external data, Geo Engine introduces the data providers that only require creating the loading information and reusing the basic input operators.

A notable type of geoprocessing software that tackles similar problems as Geo Engine are data cubes. One representative is the Open Data Cube (ODC) [Ki18]. In contrast to Geo Engine, ODC harmonizes all data upfront while building the n-dimensional data cube. This makes it far more inflexible as the requirements, e.g., for the resolution and the included datasets must be known up-front, or the data cube has to be rebuilt. It is limited by the available main memory and does not support vector data in the processing in the same fashion as Geo Engine.

Google Earth Engine[16], as a representative of cloud GIS services, [Go17] is a popular tool for analyzing earth observation data. It provides a range of datasets that are ready to use and a variety of processing tools. In contrast to Geo Engine, Google Earth Engine is not Open Source and cannot be hosted on-premise. All data that is to be analyzed, has to be uploaded to Google. Also, the Google Earth Engine does not support temporal processing as a core feature of its language but rather has to be performed manually in its supported scripting language.

Carto[17] and Mapbox[18] are two providers that specialize in the creation of maps. Here, the focus is not on processing, but on designing good-looking and highly informative maps. In contrast to Geo Engine, there is no focus on processing pipelines and supplying GIS-like interfaces for generic geoprocessing.

GeoNode[19] is a data management platform that builds upon GeoServer [Ia17]. It allows non-expert users to create interactive maps and to publish data for external tools. As its focus is on data management and sharing, it lacks a flexible toolbox to process the data beyond simple filter mechanisms. In contrast to Geo Engine, analysis is done with external GIS tools, e.g., QGIS[20], rather than having workflows within GeoNode itself.

In the context of biodiversity data, GBIF hosted portals[21] are an easy way of creating custom geo portals. The portals make use of the existing GBIF infrastructure which alleviates the work and costs of hosting. However, the portals are limited to the data that is already

[16] earthengine.google.com
[17] www.carto.com
[18] www.mapbox.com
[19] www.geonode.org
[20] www.qgis.org
[21] www.gbif.org/hosted-portals

available on GBIF. Also, portals can only be created after a successful application and review by GBIF.

The Atlas of Living Australia[22] (ALA) offers open access to Australia's biodiversity data. It offers a rich set of feature modules for discovering and visualizing geo data, which in part uses existing geo software like GeoServer. The ALA software can be used for custom geo portals as well. In contrast to Geo Engine, it does not allow for custom interactive analysis. Also, it cannot be readily installed by anyone, but only on an individual per-request basis.

6 Summary and outlook

In this paper, we presented Geo Engine's architecture and its usage for powering geo data portals. For this, we outlined general requirements for modern data portals and how Geo Engine fulfills them. Moreover, we presented a use case of a Geo Engine data portal presenting dragonflies. Finally, we compared Geo Engine to related systems.

In the future, we will focus on improving the creation and administration of geo portals with Geo Engine. We are working on a portal builder that eliminates the need to write custom code in our geoengine-ui repository (which took the majority of the time in building the dragonfly portal). Instead, it will be possible to create dashboards declaratively in the Geo Engine UI. This will also make it easier to host multiple geo portals using a single Geo Engine instance. Currently, a single backend instance can already serve multiple frontends, but each portal frontend has to be deployed individually. We will also provide an administration UI for managing existing portals. This will reduce the required technical knowledge for operating portals. Furthermore, we are going to develop more geo portals ourselves as blueprints. This will help the adoption of our technology for portal building and facilitate community building.

For improving Geo Engine as a platform, we will create more functionality such as new operators to support a greater variety of use cases. In particular, we will develop means for creating machine learning models that are fed by Geo Engine's data pipeline. Then, experts can learn a model using Geo Engine's full-fledged capabilities. In data portals, one can use these models and apply them simply as another operator of a Geo Engine workflow.

Acknowledgments

This work was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under grant number O3EUPHE069.

[22] www.ala.org.au

Bibliography

[Au15a] Authmann, C.; Beilschmidt, C.; Drönner, J.; Mattig, M.; Seeger, B.: Rethinking spatial processing in data-intensive science. In: Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI). volume 242, 2015.

[Au15b] Authmann, Christian; Beilschmidt, Christian; Drönner, Johannes; Mattig, Michael; Seeger, Bernhard: VAT: A System for Visualizing, Analyzing and Transforming Spatial Data in Science. Datenbank-Spektrum, 15(3):175–184, 2015.

[Be17a] Beilschmidt, Christian; Drönner, Johannes; Mattig, Michael; Seeger, Bernhard: VAT: A System for Data-Driven Biodiversity Research. 20th International Conference on Extending Database Technology (EDBT), 2017.

[Be17b] Beilschmidt, Christian; Fober, Thomas; Mattig, Michael; Seeger, Bernhard: A Linear-Time Algorithm for the Aggregation and Visualization of Big Spatial Point Data. In: SIGSPATIAL '17: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, New York, NY, USA, pp. 73:1–73:4, 2017.

[Be19] Berg, Lukas; Ziegler, Tobias; Binnig, Carsten; Röhm, Uwe: ProgressiveDB - Progressive data analytics as a middleware. In: Proceedings of the VLDB Endowment. volume 12, 2019.

[Di14] Diepenbroek, Michael; Glöckner, Frank Oliver; Grobe, Peter; Güntsch, Anton; Huber, Robert; König-Ries, Birgitta; Kostadinov, Ivaylo; Nieschulze, Jens; Seeger, Bernhard; Tolksdorf, Robert; Triebel, Dagmar: Towards an Integrated Biodiversity and Ecological Research Data Management and Archiving Platform: The German Federation for the Curation of Biological Data (GFBio). In: GI-Jahrestagung. volume 232 of LNI, GI, Bonn, Germany, pp. 1711–1721, 2014.

[GdOe22] Gesellschaft deutschsprachiger Odonatologen e.V., Heidelberg: GdOnline 2022. In: 2. Digitalkonferenz der Gesellschaft deutschsprachiger Odonatologen (GdO e.V.), 18.-19. März 2022. 2022.

[Go17] Gorelick, Noel; Hancher, Matt; Dixon, Mike; Ilyushchenko, Simon; Thau, David; Moore, Rebecca: Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, 2017.

[HH17] Hoyer, Stephan; Hamman, Joe: xarray: N-D labeled Arrays and Datasets in Python. Journal of Open Research Software, 5(1), 2017.

[Ho20] Holanda, Pedro; Raasveldt, Mark; Manegold, Stefan; Mühleisen, Hannes: Progressive indexes: Indexing for interactive data analysis. In: Proceedings of the VLDB Endowment. volume 12, 2020.

[Ia17] Iacovella, Stefano: GeoServer Beginner's Guide: Share Geospatial Data using Open Source Standards. Packt Publishing Ltd, 2017.

[Ki18] Killough, Brian: Overview of the open data cube initiative. In: International Geoscience and Remote Sensing Symposium (IGARSS). volume 2018-July, 2018.

[MS19] Muñoz Sabater, J.: ERA5-Land Monthly Averaged Data From 1981 to Present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2019.

[Op10] Open Geospatial Consortium: OpenGIS Implementation Standard for Geographic Information - Simple Feature Access. OpenGIS Pro-ject Document, 2010.

[Ri21] Riembauer, Guido; Weinmann, Anika; Xu, Shaojuan; Eichfuss, Silas; Eberz, Charlotte; Neteler, Markus: Germany-wide Sentinel-2 based land cover classification and change detection for settlement and infrastructure monitoring. In: Proceedings of the 2021 conference on Big Data from Space. pp. 53–56, 2021.

[Ro22] Rouault, Even; Warmerdam, Frank; Schwehr, Kurt; Kiselev, Andrey; Butler, Howard; Łoskot, Mateusz; Szekeres, Tamas; Tourigny, Etienne; Landa, Martin; Miara, Idan; Elliston, Ben; Kumar, Chaitanya; Plesea, Lucian; Morissette, Daniel; Jolma, Ari; Dawson, Nyall: , GDAL, 7 2022.

[Sa14] Sakimura, Natsuhiko; Bradley, John; Jones, Mike; De Medeiros, Breno; Mortimore, Chuck: Openid Connect Core 1.0. The OpenID Foundation, p. S3, 2014.

[Sa17] Satyanarayan, Arvind; Moritz, Dominik; Wongsuphasawat, Kanit; Heer, Jeffrey: Vega-Lite: A Grammar of Interactive Graphics. IEEE Transactions on Visualization and Computer Graphics, 23(1), 2017.

[Wi16] Wilkinson, Mark D.; Dumontier, Michel; Aalbersberg, IJsbrand Jan; Appleton, Gabrielle; Axton, Myles; Baak, Arie; Blomberg, Niklas; Boiten, Jan-Willem; da Silva Santos, Luiz Bonino; Bourne, Philip E.; Bouwman, Jildau; Brookes, Anthony J.; Clark, Tim; Crosas, Mercè; Dillo, Ingrid; Dumon, Olivier; Edmunds, Scott; Evelo, Chris T.; Finkers, Richard; Gonzalez-Beltran, Alejandra; Gray, Alasdair J.G.; Groth, Paul; Goble, Carole; Grethe, Jeffrey S.; Heringa, Jaap; 't Hoen, Peter A.C; Hooft, Rob; Kuhn, Tobias; Kok, Ruben; Kok, Joost; Lusher, Scott J.; Martone, Maryann E.; Mons, Albert; Packer, Abel L.; Persson, Bengt; Rocca-Serra, Philippe; Roos, Marco; van Schaik, Rene; Sansone, Susanna-Assunta; Schultes, Erik; Sengstag, Thierry; Slater, Ted; Strawn, George; Swertz, Morris A.; Thompson, Mark; van der Lei, Johan; van Mulligen, Erik; Velterop, Jan; Waagmeester, Andra; Wittenburg, Peter; Wolstencroft, Katherine; Zhao, Jun; Mons, Barend: The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3(1):160018, 12 2016.