Earth Science Informatics
https://doi.org/10.1007/s12145-024-01249-y

SOFTWARE

OpenEOcubes: an open-source and lightweight R-based RESTful web service for analyzing earth observation data cubes

Brian Pondi[1] · Marius Appel[1,2] · Edzer Pebesma[1]

Received: 29 November 2023 / Accepted: 4 February 2024

Abstract

In recent decades, Earth Observation (EO) systems have seen remarkable technological advancements, leading to a surge in Earth-orbiting satellites capturing EO data. Cloud-based storage solutions have been adopted to manage the increasing data volume. Although numerous EO data management and analysis platforms have emerged to accommodate this growth, many suffer from limitations like closed-source software, leading to platform lock-in and restricted functionalities, restricting the scientific community from conducting open and reproducible research. To tackle these issues, we present OpenEOcubes, a lightweight EO data cubes analysis service that embraces open-source tools, standardized APIs, and containerized deployment, we demonstrate the service's capabilities in two user scenarios: performing vegetation analysis in Amazonia, Brazil for one year, and detecting changes in a forested area in Brandenburg, Germany based on five years of EO data.OpenEOcubes is an easy-to-deploy service that empowers the scientific community to reproduce small and medium-sized EO scientific analysis while aggregating over a potentially huge amount of data. It enables the extension of functionalities and validation of analysis carried out on different EO data processing platforms.

Keywords Earth observation data cubes · Reproducible research · OpenEO · User-defined functions

Introduction

The past years have witnessed a significant increase in Earth Observation (EO) data generation, driven by the deployment of numerous satellites orbiting the planet. For instance, in 2019 alone, satellites such as Moderate Resolution Imaging Spectroradiometer (MODIS), Landsat, and Sentinel produced a staggering 5 Petabytes of satellite images (Soille et al. 2017). Governments and space agencies have responded by implementing open data regulations, making these large EO datasets widely accessible to the public and scientific community. This availability has facilitated environmental monitoring and improved our understanding of global phenomena, encompassing aspects like urbanization, climate change, agricultural production, and risk assessment (Stromann et al. 2019; Kansakar and Hossain 2016). Consequently, EO data sets now align with the 5 V's of Big Data, which encompass Volume, Velocity, Variety, Veracity, and Value. To obtain value, the challenge is to manage the first four V's, and for scientists this needs to be done in an interoperable, reusable, and reproducible way.

Efficient organization, access, and processing of these vast EO data sets have challenged the scientific community (Giuliani et al. 2019). To address the issue of EO data search and discovery, the SpatioTemporal Asset Catalog (STAC) Specification [1] has emerged as a pivotal approach, offering clear specifications that EO data providers can implement to facilitate efficient discovery and search of massive EO datasets. Furthermore, the adoption of data cubes, a multidimensional array data structure, has gained momentum among

Communicated by: H. Babaie

✉ Brian Pondi
brian.pondi@uni-muenster.de

Marius Appel
marius.appel@hs-bochum.de

Edzer Pebesma
edzer.pebesma@uni-muenster.de

[1] Institute for Geoinformatics, University of Münster, Heisenbergstraße 2, Münster 48149, North-Rhine Westphalia, Germany

[2] Department of Geodesy, Bochum University of Applied Sciences, Am Hochschulcampus 1, Bochum 44801, North-Rhine Westphalia, Germany

[1] https://stacspec.org/en

Published online: 19 February 2024

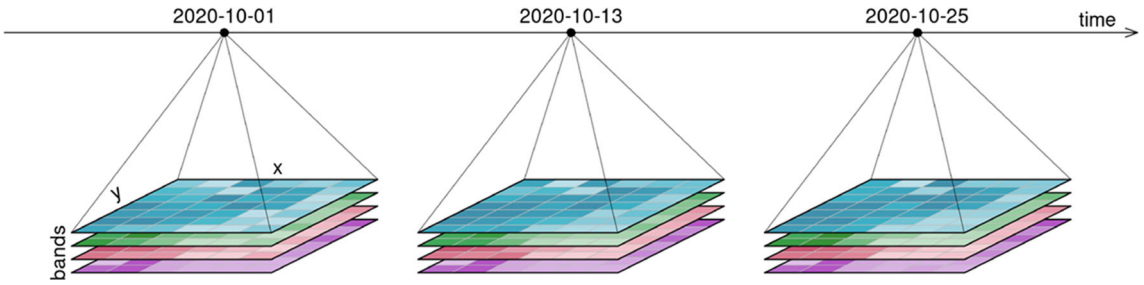

Fig. 1 A typical raster data cube comprises four dimensions: x, y, bands, and time. Source (Pebesma and Bivand 2023)

scientific communities dealing with EO data. Data cubes enable the handling, processing, and accessibility of Analysis Ready Data (ARD) for a broader audience interested in satellite imagery (Kopp et al. 2019). Several notable data cube software implementations include Earth System Data Cube (Mahecha et al. 2019), Open Data Cube (Lewis et al. 2017), Euro Data Cube [2], and gdalcubes (Appel and Pebesma 2019). Moreover, cloud computing environments have emerged as the solution for storing and managing datasets too large to download and/or handle locally (Zhang et al. 2017).

Numerous platforms have emerged to facilitate the management and analysis of EO data, encompassing large datasets. These platforms include Sentinel Hub (SH)[3], Open Data Cube, Google Earth Engine (GEE) (Gorelick et al. 2017), Microsoft Planetary Computer (MPC)[4], and the federated openEO platform (Jacob et al. 2022). Among them, GEE and SH have achieved broad popularity due to their user-friendly interfaces and comprehensive EO data analysis capabilities. However, these platforms exhibit certain limitations. For instance, the presence of closed-source software can impede research reproducibility, while the inability to easily replicate the infrastructure and extend functionalities restricts the existing platforms' flexibility.

This study leverages the STAC specification, the openEO API (Schramm et al. 2021), and gdalcubes as the fundamental components for developing a lightweight RESTful web service. This service provides Representational State Transfer (REST) Application Programming Interface (API) endpoints tailored for the analysis of Earth Observation (EO) data cubes within cloud environments. The decision to embrace the openEO API is driven by its successful implementation and widespread adoption across various EO cloud services. It furnishes standardized REST-like API functionalities that can be seamlessly incorporated into diverse programming languages.

The openEO standardized API serves as the underpinning framework for our RESTful web service dedicated to EO data cube analysis, ensuring an open governance model and delivering essential functionalities and processes to the scientific community. Notably, the openEO API are endorsed by several reference implementations, including EURAC[5], EODC[6], VITO[7], mundialis[8], Copernicus Data Space Ecosystem[9] and the federated openEO Platform (Jacob et al. 2022).

Conceptually, the openEO standardized API employs the notion of a JSON Process Graph[10], representing a symbolic expression tree stored as a directed acyclic graph in JSON format. This graph delineates processing tasks and operations, specifying input data, operations, and their parameters. Consequently, users can construct processing workflows of arbitrary complexity. When these graphs are transmitted to an openEO-compliant RESTful web service, it executes the specified computational steps and returns the desired output.

Appel and Pebesma (2019) defines image collections as "*a set of n images, where images contain m variables or spectral bands. Band data from one image share a common spatial footprint, acquisition date/time, and spatial reference system but may have different pixel sizes*". In contrast to data cubes, image collections may contain images that partially overlap, may have gaps and have irregular observation times (following the path of the satellite), and cover multiple coordinate reference systems. Earth observation (EO) data cubes, as illustrated in Fig. 1, refer to multidimensional arrays with regularized spatial and temporal dimensions, designed to simplify organization and management, enhance query performance, and simplify the analysis of EO data (Kopp et al. 2019). The gdalcubes R package is chosen for the data cube implementation due to its open-source nature and its offering of a broad range of data cube operations, enabling the creation and analysis of on-demand data cubes from satellite image collections. It utilizes the Geospatial Data Abstraction

[2] https://eurodatacube.com/

[3] https://www.sentinel-hub.com/

[4] https://planetarycomputer.microsoft.com/

[5] https://openeo.eurac.edu

[6] https://openeo.eodc.eu/openeo/1.1.0

[7] https://openeo.vito.be/openeo/1.2

[8] https://openeo.mundialis.de/api/v1.0

[9] https://openeo.dataspace.copernicus.eu/openeo/1.2

[10] https://api.openeo.org/v/0.4.2/processgraphs/

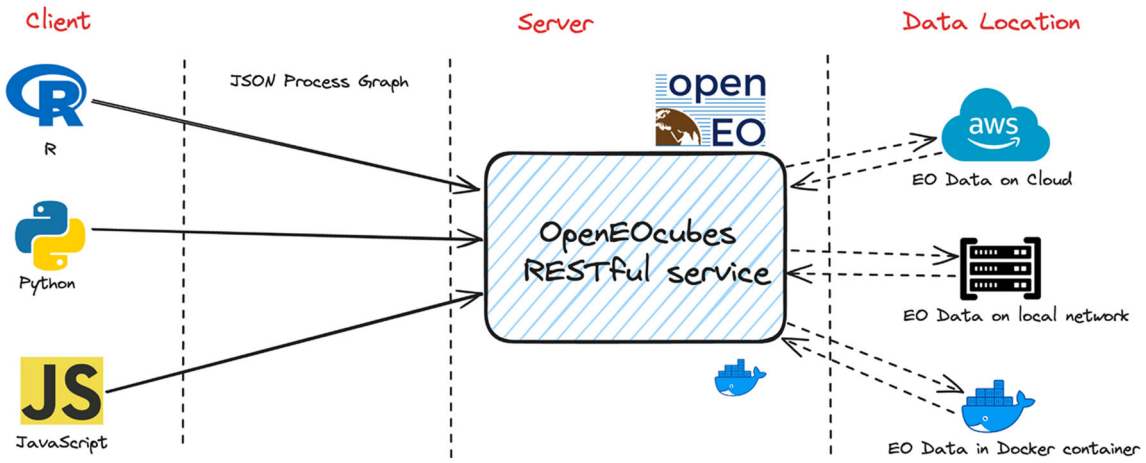

Fig. 2 Overview of the OpenEOcubes system architecture

Library (GDAL) to read and warp (reproject, rescale, crop, and resample) EO data, ensuring compatibility with various EO data collections and Cloud Optimized GeoTIFFs (Appel and Pebesma 2019).

The gdalcubes package is written in C++ and has an R interface, making it accessible to scientists and data users familiar with R. It allows for creating data cubes from image collections, with a high flexibility in spatial and temporal aggregation methods, and on-the-fly computation of band indices. The 4-dimensional EO data cube model incorporates band/variable, spatial, and temporal dimensions. Users can define the spatial resolution and target Coordinate Reference Systems during data cube creation. On-demand cube creation, lazy evaluation, and chunk-wise in-memory processing facilitate computational optimizations and reduced consumption of computing resources, while parallelization furthermore enhances processing speed. Moreover, users can employ custom R functions for customized analysis on EO data cubes.

This paper introduces an open-source and user-friendly software/service, called OpenEOcubes, which was designed for scientists and data users to deploy, explore, preprocess, and analyze EO datasets that may be too large to download. It empowers the R scientific community to execute custom functions written in the R programming language, thereby fostering research reproducibility. This service is lightweight, operating system-independent, extensible, follows open governance principles and is interoperable by adherence to the openEO standardized API.

The next section describes the design and implementation of the software. "Application" describes two use case scenarios that were chosen to illustrate the performance of the software. "Discussion" discusses the approach, and "Conclusion" lists conclusions. A final section describes the software availability and requirements.

Design and implementation

The RESTful web service is developed following a simple underlying architecture (see Fig. 2), which adheres to the openEO standardized API. As mentioned in Schramm et al. (2021), the openEO API serves as a communication interface, facilitating standardized access to EO data and processing capabilities across diverse cloud service providers, each with its unique low-level processing architecture. In the openEO ecosystem, there are clients in R, Python, JavaScript, and a Web editor, enabling straightforward interaction with REST services that have implemented the openEO standardized API.

The entire service implementation is carried out in the R[11] programming language, and gdalcubes R package as the primary EO data cube implementation.

To facilitate smooth interaction with the service, we utilize the R plumber package (refer to Schloerke and Allen (2023)). This package enables the creation of a RESTful API for an R script, making it possible to share it with a broader audience through the web. Additionally, all EO data handling processes are carefully designed to adhere to the openEO[12] standardized API.

For efficient search, discovery, and access to EO data, openEO uses the STAC specification. Utilizing the rstac (Simoes et al. 2021) package as an interface to STAC APIs, our service can make STAC API calls for instance to the AWS Earth Search STAC index[13], which offers comprehensive coverage of various data archives, including global Sentinel 2 level 2A, level 1C, and level 2A, and Landsat 8 L1 Collection

[11] https://www.r-project.org/about.html

[12] https://openeo.org/documentation/1.0/processes.html

[13] https://earth-search.aws.element84.com/v0

Algorithm 1 aggregate temporal period.

```
function(data, period, reducer, dimension = NULL, context = NULL){
  dt_period <- switch(period,
                "week" = "P7D",
                "dekad" = "P10D",
                "month" = "P1M",
                "year" = "P1Y",
                "decade" = "P10Y",
                stop("The specified period is not supported"))

  cube <- gdalcubes::aggregate_time(data, dt_period, reducer)
  return(cube)
}
```

1 data as Cloud Optimized GeoTIFFs.[14] Image collections, retrieved based on user-defined parameters (bounding box, time period, bands), are then processed using gdalcubes to create EO data cubes.

Aligning gdalcubes data cube operations with openEO processes[15] is achieved by encapsulating one or more gdalcubes functions within a wrapper function that adheres to the openEO standardized API. For example, the openEO process to generate a temporal aggregation on a data cube using calendar hierarchies like years, months, or weeks (*aggregate_temporal_period*) uses the *aggregate_time* gdalcubes function, as illustrated in algorithm code 1. Additionally, The openEO process to load image collections and return a data cube (*load_collection*) uses functionalities offered by gdalcubes, such as enabling EO data resampling, temporal aggregation, and coordinate reference system (CRS) definition.

The openEO standardized API has a concept of User-Defined Functions (UDFs) which address the common scenario where users require specialized operations or algorithms that are not part of openEO's predefined processes. UDFs enable users to define arbitrary algorithms or portions thereof using programming languages like Python and R. These defined scripts can then be executed on the RESTful web service, effectively expanding the openEO-compliant services' range of capabilities.

In the OpenEOcubes service, the *run_udf* function has been implemented by encapsulating two gdalcubes functions, namely, *reduce_time* and $apply\_pixel$. The appropriate function is applied after determining whether the UDF serves as an *apply-per-pixel function*(A function that computes new pixel values for every individual pixel in an image) or a *reducer function* (A function that computes a single, scalar outcome based on a set of input values such as a pixel time series). It is important to note that the service exclusively supports UDFs written in the R programming language. The list of implemented openEO processes is given in Table 1.

We defined a Dockerfile as suggested by Nüst et al. (2020) and packaged the RESTful web service into a Docker container image[16] and publicly hosted it on DockerHub. This approach reduces the deployment process into a single step, which can be as simple as

```
docker run -p 8000:8000 brianpondi/openeocubes
```

By adhering to the openEO standardized API and integrating the capabilities of gdalcubes, our service ensures interoperability, efficiency, and data cube functionalities, enabling EO data processing and analysis. Users typically interact with OpenEOcubes using openEO client libraries that convert the defined steps into a JavaScript Object Notation (JSON) Process Graph sent to the service.

Application

To demonstrate the functionality of the constructed RESTful web service, we deployed it to Amazon Web Services (AWS)[17] Elastic Compute Cloud (EC2) using Docker. Docker[18] encapsulates an application and its dependencies into a 'container image', a lightweight, standalone package that ensures consistent operation across various computing environments, a container image can be made available on Dockerhub[19] for public access. For our deployment, we used the Docker command:

[14] https://www.cogeo.org/

[15] https://processes.openeo.org/

[16] https://hub.docker.com/r/brianpondi/openeocubes

[17] https://aws.amazon.com/

[18] https://www.docker.com/

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

Table 1 Implemented openEO processes in the OpenEOcubes RESTful web service

openEO process	Description
add, subtract, divide, multiply	Mathematical operations to define Remote Sensing indices.
aggregate_temporal_period	Computes a temporal aggregation based on calendar hierarchies such as years, months, or weeks.
array_element	Get an element from an array.
filter_bands	Filters data cubes to retain the specified bands.
filter_bbox	Limits the data cube to the specified bounding box.
filter_temporal	Restricts the data cube to the specified date range.
filter_spatial	Constrains the spatial extent of the raster data cube to the specified geometries.
load_collection	Loads image collection from the current service, returns a processable data cube.
load_stac	Loads image collection from a STAC API, returns a processable data cube.
max	Computes the largest value of an array of numbers.
min	Computes the smallest value of an array of numbers.
median	Computes the statistical median of an array of numbers.
mean	Computes the arithmetic mean of an array of numbers.
merge_cubes	Combine two 'compatible' data cubes. Compatible implies that the data cubes to be combined should have a common subset of equal dimensions.
ndvi	Calculates the Normalized Difference Vegetation Index (NDVI).
rename_dimension	Renames a dimension in the data cube while preserving all other properties.
rename_labels	Renames the labels of the specified dimension in the data cube.
reduce_dimension	Applies a reducer function to a dimension within a data cube.
resample_spatial	Resamples the data cube's spatial dimensions (x, y) to a specified resolution.
run_udf	Runs a User Defined Function(UDF) written in R programming language.
save_result	Converts and stores the processed data in the specified file format.

`docker run -p 8000:8000 --env AWSHOST=x.x.x.x brianpondi/openeocubes` with `x.x.x.x` replaced by the IPv4 address of the server. This command retrieves the service container image from DockerHub, executes it, and binds it to port 8000, simultaneously configuring the AWS host environment variable to match the specified IPv4 address. With this single command, one can download, install, and deploy the service to an AWS EC2 instance, or any other instance that supports docker. The operation took 3 minutes.

For the AWS EC2 machine, we opted for one located in the us-west-2 region (Oregon).[20] The reason for this choice is that the EO datasets available in AWS STAC search are stored in the Oregon region. By opting for this region, we have optimized the processing of Earth Observation data in close proximity to their storage location, thus minimizing network latency between the service and the data storage server. This approach is not only efficient but also cost-effective.

The AWS EC2 machine used ran a Linux Operating System and had 32 Gigabytes (GB) of Random Access Memory(RAM), 4 virtual Central Processing Units (vCPUs), and a 192 GB Solid State Drive (SSD). To enable connectivity with the service, port 8000 was opened for external access. To change the default login username and password of the service, one needs to modify the respective variables in the 'openeocubes/R/SessionConfig-Class.R' file before deploying it using the command: `docker-compose up -d`.

For accessing and utilizing the deployed service, the openEO R client version 1.3.0 was employed with R version 4.3.0. Figure 3, shows an example of the R code connecting with the deployed service.

To validate the service's capabilities, we have devised two user scenarios:

1. Biodiversity Specialist (User A): User A is a 28-year-old Doctoral candidate specializing in forest biodiversity. Her goal is to conduct yearly NDVI (Normalized Difference Vegetation Index) analysis in specific areas within the Amazonia region. By comparing the NDVI output with tree species data, she aims to identify which tree species are most affected by deforestation and understand the implications for the Amazonia ecosystem. Using the service, User A can easily access the required EO (Earth Observation) data, apply relevant algorithms, and generate the necessary comparisons to carry out her research effectively.

[20] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

Fig. 3 R code snippet to connect to the deployed service, and get lists with available collections and processes

```
library(openeo)

# connect to the REST service
con = connect("http://<AWS-IPv4-ADDRESS>:8000")

# basic login with default params
login(user = "user", password = "password", login_type = "basic")

# get the collection list
collections = list_collections()

# to check description of a collection
collections$`sentinel-s2-l2a-cogs`$description

# get the process collection provided by the RESTful web service
p = processes()
```

2. Conservationist (User B): User B is a 32-year-old conservationist based in Brandenburg, Germany. His objective is to monitor the construction of the Tesla Gigafactory in Berlin-Brandenburg to ensure that the company has not cleared more trees than they were authorized to. User B is well-versed in the R programming language and prefers to utilize custom algorithms for change detection. With the service, User B can access up-to-date EO data, utilize his custom algorithms, and closely monitor the changes in the designated area to ensure compliance with environmental regulations.

These user scenarios demonstrate how the service can cater to the needs of different users, providing them with the necessary tools and data to conduct their respective research and conservation efforts efficiently and effectively.

User scenario 1: creating a composite NDVI image from satellite images spanning one year.

This case fulfills the user scenario example for the biodiversity specialist. We utilized the openEO R client and

Fig. 4 R code calculating NDVI for a timeframe of a year

```
# load the initial data collection and limit the amount of data loaded
datacube_init = p$load_collection(id = "sentinel-s2-l2a-cogs",
                                  spatial_extent = list(west = -7338335,
                                                        south = -1027138,
                                                        east = -7329987,
                                                        north = -1018790,
                                                        crs = 3857),
                                  temporal_extent = c("2022-01-01", "2022-12-31"))

# filter the data cube for the desired bands
datacube_filt = p$filter_bands(data = datacube_init, bands = c("B04", "B08"))

# aggregate data cube to yearly
datacube_agg = p$aggregate_temporal_period(data = datacube_filt,
                                period = "year", reducer = "median")

# ndvi calculation
datacube_ndvi = p$ndvi(data = datacube_agg, red = "B04", nir = "B08")

# supported formats
formats = list_file_formats()

# save as GeoTiff or NetCDF
result = p$save_result(data = datacube_ndvi, format = formats$output$GTiff)

# process and download data synchronously
compute_result(graph = result, output_file = "amazonia_ndvi.tif")
```

delineated the area of interest in specific sections of Amazonia, Brazil, bounded by (longitude, latitude): *-7338335, -1027138* and *-7329987, -1018790*, within EPSG *3857* CRS. The underlying default spatial resolution was 30 meters, and the time frame spanned 1 year, from 01.01.2022 to 31.12.2022. To meet these requirements, we employed the *load_collection* function, as demonstrated in Fig. 4.

The *filter_bands* function was employed to select the required bands, and subsequently, the *ndvi* function was used to calculate the NDVI. The process involved a total of 168 image scenes (approximately 100 Gigabytes). Upon completion, the final output was downloaded (2.1 Megabytes) in the GeoTIFF format and visualized in Fig. 5. The entire computation and download process was completed in approximately 25 seconds.

In Fig. 5, the NDVI visualization delineates regions of dense vegetation as areas of dark green, indicative of higher NDVI values, while yellow and orange hues represent regions with lower to moderate NDVI readings. These latter tones most likely indicate areas subjected to deforestation.

User scenario 2: a change detection approach customized by the user for analyzing satellite image time series

For the conservationist user scenario example, we utilized the openEO R client to define the area of interest in Brandenburg, Germany, bounded by (longitude, latitude): *416812.2, 5803577.5* and *422094.8, 5807036.1*, within EPSG *32633* CRS.

The underlying default spatial resolution was 30 meters, and the time frame covered 5 years, ranging from Jan 1, 2016 to Dec 31, 2020. These requirements were specified in the *load_collection* function, as illustrated in Fig. 6.

To carry out unsupervised land cover change detection, a user-defined R function was created and passed into the *run_udf* function as a string. This function employed the computationally intensive *bfastmonitor()* function from the *Breaks For Additive Season and Trend* (BFAST) R package (Verbesselt et al. 2010, 2011). A total of 457 image scenes (approximately 275 Gigabytes) were utilized for this task. The final output was downloaded as a NetCDF file (196 Kilobytes) and is visualized in Fig. 7. The entire process took approximately 4 minutes to complete.

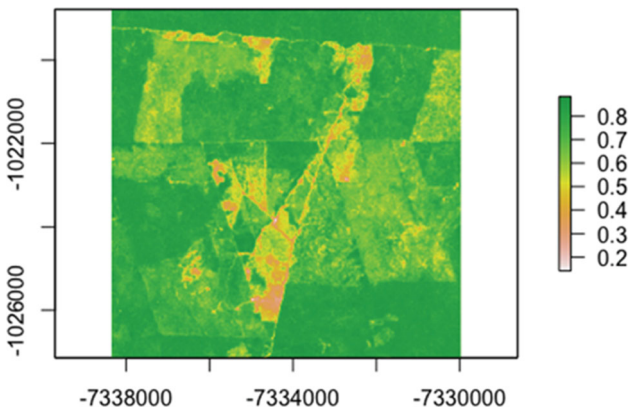

Fig. 5 A one-year median NDVI for a segment of the Amazonia region in Brazil

The visual analysis presented in Fig. 7 indicates substantial land changes in the western segment, corresponding with the construction activities of Tesla's Gigafactory Berlin-Brandenburg. Additionally, it uncovers instances of deforestation in regions not designated for such development. This observation calls for obtaining ground truth data to determine the authenticity of these deforestation signals and to rule out any false positives. The timing of these changes is shown in color, and depicted in the plot legend, which traces the evolution of the landscape from early 2020 through to the end of August 2020.

Discussion

The present rate and size of satellite imagery being collected and the limits to network bandwidth and local storage available to research groups has made it inevitable for researchers to use cloud resources to analyse image archives. Several mature cloud platforms for this are currently available, but typically lack several properties required by open science: ability to scrutinize the source code of the platform, ability to easily move from one platform to another, compare results between platforms or combine computations on platforms (platform lock-in), the ability to use own datasets without making these public, or the ability to extend the software with new algorithms. The openEO API is a relatively new, open, community-based standard that is not bound to one particular implementation of a back-end, that has several completely open source back-end implementations and that lets users compare these using a simple, modern and high-level interface.

We present a new open source openEO back-end implementation, called openEOcubes, which is written in R, interfacing the 'gdalcubes' package (Appel and Pebesma 2019). The functionality of this implementation is limited, but allows for creation of a regular data cube from an image collection, with flexibility in spatial and temporal aggregation methods, and ability to compute e.g. band indices or execute arbitraty R functions (user-defined functions) to create band indices or summarise pixel time series. It is also resource efficient, using lazy evaluation and multi-threading. Despite the limitations, it covers functionality that is in strong demand, as data size reduction through spatial and temporal resampling and/or aggregation is often the first step in analysis of a large data archive, and ultimate flexibility in how this

Fig. 6 R code running a UDF for change detection using BFAST R package

```
# load the initial data collection and limit the amount of data loaded
datacube_init = p$load_collection(id = "sentinel-s2-l2a-cogs",
                                  spatial_extent = list(west = 416812.2,
                                                        south = 5803577.5,
                                                        east = 422094.8,
                                                        north = 5807036.1,
                                                        crs = 32633),
                                  temporal_extent = c("2016-01-01", "2020-12-31"))

# filter the data cube for the desired bands
datacube_filt = p$filter_bands(data = datacube_init, bands = c("B04", "B08"))

# aggregate data cube to monthly
datacube_agg = p$aggregate_temporal_period(data = datacube_filt,
                           period = "month", reducer = "median")

# user-defined method, change detection using bfast
change_detection = 'function(x){
    library(bfast)
    knr <- exp(-((x["B08",]/10000)-(x["B04",]/10000))^2/(2))
    kndvi <- (1-knr) / (1+knr)
    if (all(is.na(kndvi))) {
        return(c(NA,NA))
    }
    kndvi_ts = ts(kndvi, start = c(2016, 1), frequency = 12)
    tryCatch({
        result = bfast::bfastmonitor(kndvi_ts, start = c(2020,1), level = 0.01)
        return(c(result$breakpoint, result$magnitude))
    },
    error = function(x) {
        return(c(NA,NA))})
    }'

# run udf
datacube_udf = p$run_udf(data = datacube_agg, udf = change_detection,
                context = c("change_date", "change_magnitude"), runtime = "R")

# supported formats
formats = list_file_formats()

# save as GeoTiff or NetCDF
result = p$save_result(data = datacube_udf, format = formats$output$NetCDF)

# process and download data synchronously
compute_result(graph = result, output_file = "detected_changes.nc")
```

is done is often important. Cloud removal and the creation of analysis-ready data (ARD) fall under this category.

The implementation we present is offered as an R package; it uses the R package ‘plumber‘ (Schloerke and Allen 2023) to expose R functions as RESTful API endpoints. The R package, along with upstream R packages and system libraries are all packaged in a docker container to make deployment as simple as executing a single ‘docker run‘ command. This allows users to either operate this service (i) from within R, if they are familiar with running R, (ii) on a local computer using datasets inside the docker container, (iii) on a local computer using local data or data in the network, provided through a STAC interface (iv) on an arbitrary cloud node provided that imagery archives are accessible and interfaced through a STAC interface. The user can interact with this service using the wide variety of openEO clients available (in Python, R, JavaScript, or through a visual browser-based editor).

We have demonstrated two fully reproducible use cases, both using a moderately large amount of Sentinel-2 data. Deployed on a single AWS node in the data center that serves the Sentinel-2 archive, the time-composite NDVI creation use case used 168 images (100 GB) and took 25 seconds, and the change detection use case based on the

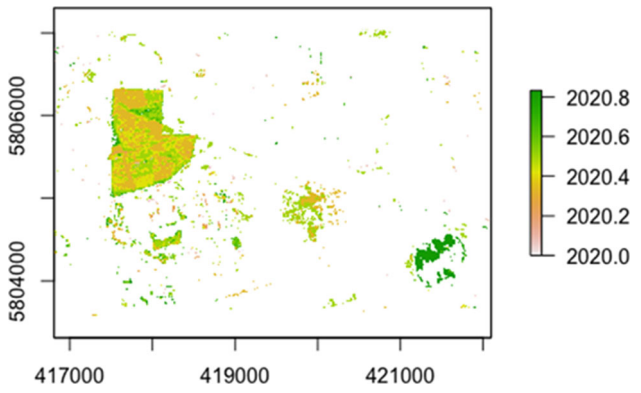

Fig. 7 Detected changes in a section of a forest in the state of Brandenburg, Germany during the year 2020

BFAST algorithm used 457 images (275 GB) that took 4 minutes. Moreover, the utilization of Cloud-Optimized GeoTIFFs (COGs)[21] Sentinel-2 data allowed us to selectively read and transfer only the necessary portions of the file.

Although the implementation we present offers image analysis functionality that is in high demand when applied to large satellite image archives, it does lack a number of features that are available in large-scale deployments of openEO like openEO Platform or the Copernicus Data Space Ecosystem. In particular, it lacks the ability to scale up over multiple compute nodes, only one user can access the service at a time, it has no ability to evaluate user credits or predict costs of computation, and it only runs synchronously. As such it is mostly targeting users who have full control over their own cloud resources. The capability to run processes locally before deploying them to the cloud however makes it easier to experiment with new algorithms. Executing them on low spatial or temporal resolution data cubes makes it easy to evaluate them before computations become lengthy and/or costly.

Running arbitrary R code on the cloud platform gives users the ultimate flexibility, but also poses challenges to the level of interoperability a generic openEO API can provide: a dependency of this code on further R packages used (such as 'bfast') becomes clear, and executing such user-defined functions on other openEO REST services, e.g. written in Python, becomes more complicated.

Conclusion

We present OpenEOcubes, a new open-source openEO service written in R mainly using the R packages gdalcubes and plumber. It is easily extensible, allows for running arbitrary user-defined R functions on pixel time series, and can be deployed locally or in the cloud as an R package, or as a docker image. This empowers users who have access to cloud resources and allows them to easily extend it with custom R functionality. Integrating this with the current large, multi-user and multi-node openEO deployments still poses challenges related to user authentication and authorisation.

Availability and requirements

OpenEOcubes Software: The OpenEOcubes code is open-source adopting Apache License Version 2.0 and available on GitHub at the following URL: https://github.com/PondiB/openeocubes. You can find the installation instructions on the GitHub repository.

Docker Image: For easy installation on a local personal computer or a cloud computing environment, a Docker image is provided. You can access the Docker image on DockerHub using the following link: https://hub.docker.com/r/brianpondi/openeocubes. The installation process requires a single docker command.

Operating System: Compatible with Windows, macOS, or Linux operating systems.

Acknowledgements We gratefully acknowledge the financial support provided to the development of this software through the European Union's Horizon Europe research and innovation programme. This support was granted under agreement No. 101059548 for the Open-Earth-Monitor Cyberinfrastructure(OEMC) project, and under agreement No. 101058386 for the interdisciplinary Digital Twin Engine for science(interTwin) project.

Author Contributions B.P. led the conceptualization and implementation of the software's core functionalities, with M.A. and E.P. conducting software testing and validation. B.P. drafted the primary manuscript text while E.P. and M.A. reviewed and edited the manuscript. The manuscript was reviewed by all authors.

Funding Open Access funding enabled and organized by Projekt DEAL. Funding for this work was provided by the European Commission under agreement No. 101059548 for the Open-Earth-Monitor Cyberinfrastructure project, and under agreement No.101058386 for the interdisciplinary Digital Twin Engine for science project.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

[21] https://www.cogeo.org/

References

Appel M, Pebesma E (2019) On-demand processing of data cubes from satellite image collections with the gdalcubes library. Data 4:92. https://doi.org/10.3390/data4030092

Giuliani G, Masó-Pau J, Mazzetti P, Nativi S, Zabala A (2019) Paving the way to increased interoperability of earth observations data cubes. Data 4:113

Gorelick N, Hancher M, Dixon M, Ilyushchenko S, Thau D, Moore R (2017) Google earth engine: Planetary-scale geospatial analysis for everyone. Remote Sens Environ 202. 10.1016/j.rse.2017.06.031

Jacob A, Dries J, Claus M, Rossi M, Briese C, Griffiths P, Mohr M, Lippens S, Ardizzone V, Thiex D et al (2022) Cloud platform federation through openeo: New concepts and implementations in the esa openeo platform. In: ESA Living Planet Symposium 2022

Kansakar P, Hossain F (2016) A review of applications of satellite earth observation data for global societal benefit and stewardship of planet earth. Space Policy 36:46–54

Kopp S, Becker P, Doshi A, Wright D, Zhang K, Xu H (2019) Achieving the full vision of earth observation data cubes. Data 4:94. https://doi.org/10.3390/data4030094

Lewis A, Oliver S, Lymburner L, Evans B, Wyborn LAI, Mueller N, Raevksi G, Hooke J, Woodcock R, Sixsmith J, Wu W, Tan P, Li F, Killough B, Minchin S, Roberts D, Ayers D, Bala B, Dwyer J, Wang L-W (2017) The australian geoscience data cube – foundations and lessons learned. Remote Sens Environ 202. https://doi.org/10.1016/j.rse.2017.03.015

Mahecha M, Gans F, Brandt G, Christiansen R, Cornell S, Fomferra N, Kraemer G, Peters J, Bodesheim P, Camps-Valls G, Donges J, Dorigo W, Estupinan-Suarez L, Gutierrez V, Gutwin M, Jung M, Londoño-Murcia M, Miralles D, Papastefanou P, Reichstein M (2019). Earth system data cubes unravel global multivariate dynamics. https://doi.org/10.5194/esd-2019-62

Nüst D, Sochat V, Marwick B, Eglen SJ, Head T, Hirst T, Evans BD (2020) Ten simple rules for writing Dockerfiles for reproducible data science. Public Library of Science San Francisco, CA USA

Pebesma E, Bivand R (2023) Spatial Data Science: with Applications in R. Chapman and Hall/CRC, Boca Raton. https://doi.org/10.1201/9780429459016

Schloerke B, Allen J (2023) Plumber: An API Generator for R. https://www.rplumber.io. https://github.com/rstudio/plumber

Schramm M, Pebesma EJ, Milenkovic M, Foresta L, Dries J, Jacob AW, Wagner W, Mohr M, Neteler M, Kadunc M, Miksa T, Kempeneers P, Verbesselt J, Gößwein B, Navacchi C, Lippens S, Reiche J (2021) The openeo api-harmonising the use of earth observation cloud services using virtual data cube functionalities. Remote Sens 13:1125

Simoes R, Souza FC, Zaglia M, Queiroz GR, Santos RDC, Ferreira KR (2021) Rstac: An r package to access spatiotemporal asset catalog satellite imagery. In: 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS, pp 7674–7677. https://doi.org/10.1109/IGARSS47720.2021.9553518

Soille P, Burger A, Marchi D, Kempeneers P, Rodriguez Aseretto D, Syrris V, Vasilev V (2017) A versatile data-intensive computing platform for information retrieval from big geospatial data. Futur Gener Comput Syst 81. https://doi.org/10.1016/j.future.2017.11.007

Stromann O, Nascetti A, Yousif O, Ban Y (2019) Dimensionality reduction and feature selection for object-based land cover classification based on sentinel-1 and sentinel-2 time series using google earth engine. Remote Sens 12:76. https://doi.org/10.3390/rs12010076

Verbesselt J, Hyndman R, Newnham G, Culvenor D (2010) Detecting trend and seasonal changes in satellite image time series. Remote Sens Environ 114(1):106–115. https://doi.org/10.1016/j.rse.2009.08.014

Verbesselt J, Zeileis A, Herold M (2011) Near real-time disturbance detection in terrestrial ecosystems using satellite image time series: drought detection in Somalia (2011-18)

Zhang C, Di L, Sun Z, Yu EG, Hu L, Lin L, Tang J, Rahman MS (2017) Integrating ogc web processing service with cloud computing environment for earth observation data. In: 2017 6th International Conference on Agro-Geoinformatics, pp 1–4. https://doi.org/10.1109/Agro-Geoinformatics.2017.8047065