OGC® DOCUMENT: 24-035

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t20-D020

Open Geospatial Consortium

OGC TESTBED 20 GEODATACUBE (GDC) API PROFILE REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2025-02-13
Approval Date: 2025-06-12
Publication Date: 2025-06-26
Editor: Jonas Eberle

Note

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogc, testbed, geodatacube, api, web service, OpenEO, Processes, Coverage, earth observation, weather, profile

II CONTRIBUTORS

All questions regarding this document should be directed to the editor or the contributors:

Table — Table Contributors

NAME	ORGANIZATION	ROLE
Jonas Eberle	German Aerospace Center	Editor
Matthias Mohr	Matthias Mohr — Softwareentwicklung	Contributor
Jérôme Jacovella-St-Louis	Ecere Corporation	Contributor
Patrick Dion	Ecere Corporation	Contributor
Michele Claus	Eurac Research	Contributor
Francis Charette Migneault	CRIM	Contributor

III OVERVIEW

The geospatial community has invested significant resources in defining and developing Geospatial Data Cubes (GDCs), demonstrating a commitment to creating an infrastructure that enables the organized storage and use of multidimensional geospatial data. While progress has been made, the state of the art falls short of fully interoperable GDCs capable of meeting specific organizational requirements. Establishing reference implementations to ensure GDCs are both interoperable and exploitable—particularly in Earth Observation (EO) contexts—remains a priority.

Based on the collective efforts of the OGC Testbed-19 GDC Activity and the GeoDataCube Standards Working Group (SWG), the definition of the draft OGC API — GeoDataCube Standard strives to unify the disparate threads of geospatial data cube technology. By integrating elements from the openEO API, OGC API — Processes, and the SpatioTemporal Asset Catalog (STAC), the GDC draft API presents a holistic approach to accessing, managing, and processing Earth Observation data.

A GeoDataCube API combines data access and analysis functionality. A GDC API implementation would support access to datasets in the form of data cubes with the basic capabilities of filtering, subsetting, and aggregating. Further advanced processing is enabled through the integration of existing processing API endpoints (e.g., OGC API – Processes Standard and the openEO API Specification).

This Testbed 20 Report focuses on profiling existing capabilities defined by other OGC-approved and draft standards, such as the OGC API — Processes Standard and OGC API — Coverages Standard, as well as community-driven standards such as the openEO API.

NOTE: Hereafter in this document, a profile of a GeoDataCube API integrating support for one or more approved or candidate OGC Standards or Community Standards will be referred to as a GDC API.

FUTURE OUTLOOK

Vector Data Cubes and OGC API EDR Conformance While the current focus is on geospatial raster data as part of GDC, the integration of geospatial vector data into GDC remains an open area of investigation. Additionally, aligning a GDC API Standard with the OGC API — Environmental Data Retrieval (EDR) will require deeper exploration to ensure seamless compatibility.

Expanding STAC API Utilization Further evaluation of the SpatioTemporal Asset Catalog (STAC) API as a /collections endpoint is necessary. This will enable the handling of non-coverage data, such as geospatial data cubes containing vector data, broadening the applicability of the API for diverse geospatial datasets.

Standardized Processing and Workflow Execution The establishment of well-known processes to support structured processing languages is crucial for interoperability. Identifying and standardizing the representation of GDCs as process inputs will facilitate more efficient and consistent data processing across different platforms.

Harmonization of Workflow Execution A standardized API endpoint for submitting and executing workflows (e.g., OpenEO process graphs, Common Workflow Language [CWL]) is needed to unify execution across various workflow languages. Ongoing discussions in Testbed-20 lay the groundwork for achieving a harmonized approach to workflow orchestration.

By addressing these areas, GDC API profiles—such as those implemented in Testbed-20—can evolve into more robust and flexible solutions, supporting a wider range of geospatial data

types and processing capabilities. Future efforts will focus on refining these aspects to enhance usability, scalability, and interoperability within the geospatial community.

V VALUE PROPOSITION

Unified Data Access and Transformation

A GDC API is designed to enable seamless integration of **data access** and **analysis** by allowing users to interact with data in the form of **data cubes**, which can be filtered, subsetted, and reshaped to fit specific analytical needs. This combination optimizes workflows by eliminating redundant steps and ensuring efficient data handling.

The following key features show the value of a GDC API:

1. **Chaining Data Access and Reshaping:**
    - **Without a GDC API:**
        - Collections retrieved from an implementation of the OGC API — Coverage Standard must be processed independently.
        - A separate download step is required between reshaping and analysis, increasing workflow complexity.
    - **With a GDC API:**
        - Collections from an implementation of the OGC API — Coverage Standard can be reshaped and analyzed in a **single uninterrupted workflow**, demonstrating significant efficiency improvements.
2. **Accessing Data from Multiple Infrastructures:**
    - Utilize a **common API definition** to access data sets hosted on different infrastructures/platforms, enabling integration across diverse sources.
    - Example workflows:
        - Combine data from **Copernicus DataSpace Ecosystem** and **Climate Data Store** in a single pipeline (though the OGC API — Coverage Standard and STAC collections are not yet interoperable).
        - Define workflows using collections from different backends that support the implementation of a GDC API, demonstrating full compatibility and streamlined data processing.

Enabling FAIR Data Principles for Derived Data Sets

Data products generated through a GDC API enabled workflows are not only reusable but also adhere to **FAIR principles** (Findable, Accessible, Interoperable, Reusable):

- **Chained Workflows:**
  - Data sets produced from chained processes (e.g., reshaping, subsetting, filtering) can be accessed as new collections through an implementation instance of a GDC API.
  - These derived collections can be further integrated into new workflows, enabling continuous data transformation and analysis.

By leveraging a GDC API, users can achieve an efficient, traceable, and FAIR-compliant approach to data access, transformation, and analysis, fostering reproducibility and scalability across various domains and infrastructures.

1 INTRODUCTION

1 INTRODUCTION

Datacubes are multi-dimensional arrays that include additional information about their dimensionality. They offer a clean and organized interface for spatiotemporal data, as well as for the operations that may be performed on them. While arrays are similar to raster data, datacubes can also contain vector data.

In Testbed 20, the focus was on datacubes containing raster data. GeoDataCubes (GDCs) are a specific type of datacube that includes one or more spatial dimensions, such as x and y. Unlike other OGC standards like the OGC API – Processes, working with datacubes involves the description, discovery, access, and processing of multi-dimensional data.

In Testbed 19, the emphasis was on comparing existing standards for the discovery (e.g., OGC API — Records and STAC) and processing (e.g., openEO and OGC API — Processes) of GDCs. In contrast, Testbed 20 aimed to define the functionality provided by a GDC and explore how existing standards can meet those requirements.

This work included the following standardization tasks:

- Define functionality of a GeoDataCube;
- Discuss applicability of existing standards to provide these functions;
- Define profiles of a GDC API to provide a different set of functionality being provided via a GDC API endpoint.
- Discuss harmonization of different processing standards with the aim of a standardized API endpoint to execute both a process and an openEO process graph

Although the definition of GeoDataCubes is an important step, it is not the goal of Testbed 20 to provide consensus agreement of each specific aspect and component of the term `GeoDataCube`. As there can be many variants of a GeoDataCube, such agreement was out of scope.

1.1. Aims

The integration of GeoDataCubes into geospatial workflows is essential for advancing data-driven solutions across various domains. The Testbed 20 GDC Task focused on establishing a standardized approach to the use of GeoDataCubes by:

- Define key terminology and concepts relevant for the execution of Testbed 20.
- Create interoperability between openEO, CWL, and OGC API — Processes through a language-independent framework for GeoDataCube utilization.

- Provide GDC API endpoints that offer data and functionalities aligned with specific use cases.
- Conduct Technology Integration Experiments (TIE) to assess and validate the interoperability of GDC API endpoints.

These efforts will foster seamless data access, processing, and integration across heterogeneous platforms, enhancing the scalability and efficiency of geospatial workflows.

1.2. Objectives

The primary objective of the GDC task was to develop a language-independent extension for the GeoDataCube (GDC) draft API, as defined in Testbed 19. This API combines components from approved and candidate OGC standards, enabling the creation of workflows that remain consistent across different workflow description languages. The task involved participants from various platforms integrating this extension using technologies such as openEO, CWL, and OGC API-Processes. The extension was designed to promote interoperability among different implementations and streamline the workflow creation process, drawing on existing frameworks established in both approved and candidate OGC standards, as well as community standards for GeoDataCubes.

2 TOPICS

2 TOPICS

2.1. Definition of GeoDataCube

GeoDataCubes (GDC) are a special case of datacubes in that they have one or multiple spatial dimensions, e.g. `x` and `y`. GeoDataCubes for raster data often consist of the dimensions x, y, time and bands. Sometimes they also have multiple temporal dimensions. GeodataCubes for vector data often consist of geometries, time, and a variable (e.g., temperature). Generally, datacubes can consist of any combination of dimensions — the dimensions are unrestricted. The spatial dimension of GeoDataCubes may get removed during processing.

The following additional information is usually available for datacubes:

- the dimensions (see below)
- a grid cell type / sampling method (area or point)
- a unit for the values

This additional information could be provided upfront via metadata.

2.1.1. Dimensions

A dimension refers to a certain axis of a datacube. This includes all variables (e.g. bands), which are represented as dimensions. An example raster datacube could have the spatial dimensions `x` and `y`, and the temporal dimension `t`. Furthermore, it could have a `bands` dimension, extending into the realm of **what kind of information** is contained in the cube.

The following properties are usually available for dimensions:

- A name.
- A type. Potential types include spatial (raster or vector data), temporal, and other variables such as bands.
- Labels, usually exposed through textual or numerical representations, in the metadata as nominal values and/or extents.
- A reference system and/or unit of measure for the labels. A unit may implicitly be defined through the reference system.
- A resolution / step size.

- Other information specific to the dimension type such as the geometry types for a dimension containing geometries.

Specific implementations of datacubes may prescribe details such as sorting orders or representations of labels. For example, some implementations may always sort temporal labels in their inherent order and encode them in an ISO8601 compliant way.

Datacubes contain scalar values (e.g. strings, numbers or Boolean values), with all other associated attributes stored in dimensions (e.g. coordinates or timestamps). Attributes such as the Coordinate Reference System (CRS) or the sensor can also be turned into dimensions. Be advised that in such a case, the uniqueness of pixel coordinates may be affected. When usually, `(x, y)` refers to a unique location, that changes to `(x, y, CRS)` when `(x, y)` values are reused in other coordinate reference systems (e.g. two neighboring UTM zones).

2.1.2. Common operations

Below are some operations that are commonly applied to datacubes:

- **subset** — Restrict the extent of dimensions, such as remove all temporal information not in the year 2021
- **apply (map)** — Compute values from operations on single values, e.g. multiply all values by 10
- **reduce** — Reduce a dimension by computing a single value for all values along a dimension, such as compute the maximum value along the temporal dimension
- **resample** — The **layout** of a certain dimension is changed into another **layout**, most likely also changing the resolution of that dimension, such as downscaling from daily to monthly values

Every operation that returns a subset of the datacube or the complete datacube is **datacube access**.

Every operation that is computing new values is **datacube processing**.

2.2. Capabilities

The main objective of implementing APIs for GeoDataCubes is to make the handling of multi-dimensional data easier. This includes the following capabilities:

- **Description** of data cubes (e.g., available dimensions).
- **Discovery** of data within a data cube (e.g., filter by specific parameters).

- **Accessing** raw data contained in a data cube (e.g., static or dynamic provision of data, server-side subsetting).
- **Processing** of data cubes (e.g., changing values of data within a data cube).
- **Visualization** of data cubes.

For all those capabilities there are existing standards and specifications, which might be reused by an implementation of a GDC API.

2.3. Applicable OGC standards

There are multiple approved or draft OGC Standards, including Community Standards, that define such standardized interfaces.

Each standard or specification covers different aspects of GeoDataCube capabilities as described above:

- Description and discovery of GeoDataCubes (collections / coverages).
- Discovery of components of a single GeoDataCube.
- Facilitating access to data with server-side computation (custom client requests, where the server merges different pieces and return data at requested resolution).
- Performing server-side computations (beyond accessing part / resampling of the data).
- Encoding of data.

There is some overlap in terms of addressing these capabilities between multiple standards.

Data description and discovery

Implementation of the following API standards provide data description and data discovery capabilities:

- OGC API — Common — Part 2: Geospatial Data (Collections and Uniform Multidimensional Collections)
- OGC API — Records
- SpatioTemporal Asset Catalog (STAC)
- OGC API — Coverages — Part 3

Note: OGC API — Coverages and OGC API — EDR also provides this functionality by depending on OGC API — Common — Part 2.

Implementations of OGC API — Common — Part 2, the STAC API and OGC API — Records "Searchable Catalog Deployment" all use the `/collections` endpoint to provide and describe a list of available collections.

Implementations of OGC API — Coverages — Part 3: Scenes describe individual scenes within a single collection at `/collections/{collectionId}/scenes`.

Data access (partial pieces based on server-side computations)

Implementations of the following API standards provide server-side data access capabilities:

- OGC API — Coverages (e.g., `/collections/{collectionId}/coverage`)
- OGC API — EDR (e.g., `/collections/{collectionId}/position`)

Implementations of both standards provide endpoints that support subsetting query parameters and merge data as needed when the backing data store consists of separate components. While the OGC API — Coverages — Part 1 standard defines a requirement class for requesting data at a specific resolution (resampling), the OGC API — EDR — Part 1 standard does not offer this functionality.

Although data access can also be implemented as processes within the openEO or OGC API — Processes definitions, these implementations do not provide a dedicated endpoint for data access. Additionally, data access can be achieved by referencing static files over HTTPS (e.g., an HTTPS URL pointing to a file hosted in a cloud-optimized data format).

Data processing (server-side computations)

Implementations of the following API standards provide server-side data processing capabilities:

- OGC API — Processes — Part 1: Core (+ upcoming parts) (e.g., `/processes/{processId}`)
- OGC Web Coverage Processing Service (WCPS) (e.g., `request=ProcessCoverages`)
- openEO (e.g., `/result` or `/jobs`)
- OGC API — Coverages — Part 2: Filtering, Deriving and Aggregating fields (e.g., `/collections/{collectionId}/coverage` with query parameters specifying OGC Common Query Language (CQL2) expressions

Implementations of these standards provide endpoints to start a previously defined process. With WCPS and openEO, the processing can be defined by the user as part of the request. With OGC API — Processes, only pre-defined processes (e.g., calculation of a specific algorithm) can be executed by users with their indivdidual input parameters.

Data visualization

Implementations of the following API standards provide data visualization capabilities:

- OGC API — Maps
- OGC API — Tiles (using map tiles)

Data visualization was not part of the Testbed 20 activity, thus no discussion took place. However, some example requests and responses can be seen in the implementation annexes for server-side visualization of the Vegetation Health Index (VHI) workflow.

Data formats

The following formats were used in this initiative.

- OGC GeoTIFF (including Cloud Optimized GeoTIFF)
- OGC netCDF
- Zarr Storage Specification (Community Standard)

The analysis of data formats was not a focus of the Testbed 20 activity, but some basic capabilities and limitations of these encodings are described below to paint a more complete picture of the GDC ecosystem.

An advantage of Cloud Optimized GeoTIFF is support for overviews and tiles, allowing to perform HTTP range requests to retrieve only portions of interest. A disadvantage of GeoTIFF is that as of version 1.0, it is primarily limited to two-dimensional data, and does not define metadata for describing fields (bands). While Zarr has chunks providing functionality similar to tiles, extensions for overviews were not yet finalized at the time of writing this report (e.g., https://github.com/zarr-developers/zarr-specs/issues/125).

2.3.1. Standards defining capabilities which can be integrated within a GDC API

As demonstrated in the previous section, there is some overlap and interoperability among the various standards when it comes to addressing these capabilities. The table below aims to illustrate and clarify this overlap by highlighting the specific capabilities defined within each standard, as well as how these standards can be integrated within a GDC API implementation.

- A cell value with **bold** text means that this capability is a key feature of this standard defined within the standard itself (see note below),
- A value with N/A means that this capability is not provided by this particular standard, but may be achieved in combination with other Standards as indicated

> ***IMPORTANT***
>
> *This table is not intended as a comparison of the value of different standards, but as a quick overview of the GDC standards landscape, allowing to easily look up which capabilities are provided by which standards, as well as how all of these standards can be integrated together within a GDC API implementation. The fact that a particular standard itself specifies a particular capability (indicated in bold) is not to be interpreted as an*

advantage of that particular standard compared to leveraging that capability from another standard.

Table 1 — Comparison of existing standards for GDC APIs

STANDARD	GEODATACUBE DISCOVERY & DESCRIPTION	DATA PIECES DISCOVERY	SINGLE REQUEST TO RETRIEVE DATA FOR SPECIFIC AREA AND RESOLUTION	SERVER-SIDE COMPUTATIONS	COMBINATIONS
OGC WCS with scaling and range subsetting extensions	Yes (discovery limited to listing all coverages)	No	**Yes**	**OGC WCPS**	
OGC API — Common — Part 2: Geospatial Data, Part 3?: Schemas, Part 4?: Discovery in numerous collections	**Yes**	N/A	N/A	N/A	Basis for most OGC APIs, Records — Local (Collection) Resource Catalog
OGC API — Records	**Yes** (description not datacube-specific)	**Yes**	N/A	N/A	Links to files and services
STAC	**Datacube extension**	**Yes**	N/A	N/A	Links to files and services
OGC API — Coverages	Common	**Part 3: Scenes**	**Yes**	**Part 2** w/ **CQL2** [1], Processes	CQL2, Processes, STAC items (for scenes)
OGC API — EDR	Common	N/A	Subsetting (no resizing in Part 1)	N/A	Processes
OGC API — Tiles	Common	**2DTMS**	N/A	CQL2 extension, w/ Processes	Processes, Coverages, CQL2
OGC API — DGGS	Common	**DGGRS**	resampling with `zone-depth` for single DGGRS zone	Zone Queries, CQL2 extension, w/ Processes	CQL2, Processes
OGC API — Processes —	N/A	N/A	using custom processes	**Yes** [2]	CQL2, CWL, WCPS, STAC, Coverages,

STANDARD	GEODATACUBE DISCOVERY & DESCRIPTION	DATA PIECES DISCOVERY	SINGLE REQUEST TO RETRIEVE DATA FOR SPECIFIC AREA AND RESOLUTION	SERVER-SIDE COMPUTATIONS	COMBINATIONS
Part 1: Core, Part 2: Deploy, Replace, Undeploy, Part 3: Workflows					EDR, Tiles, DGGS, Maps
openEO	STAC	STAC	using pre-defined processes	**Yes**	STAC, Tiles, Maps, Processes, CWL, Coverages

[1] CQL2 defines arithmetic and relational operators/functions, while OGC API — Coverages — Part 2 will define additional functions for aggregation.

[2] There is a need for defining well-known processes (similar to openEO processes), and declaring that a specific OGC API — Processes process implements a particular well-known process. An OGC Naming Authority register could be used for this purpose, with a URI property of the process description which could point to a well-known process. See also related discussion for well-known (CQL2) functions.

2.4. Discussion on profiles of a GDC API

Based on the outcomes of the Testbed 19 GeoDataCube task, profiles for a GDC API needed to be discussed and defined in Testbed 20. In this context, a profile is a defined set of minimal capabilities that a system must implement to conform to specific functionality. A profile defines a structured way to integrate OGC (and other) standards without redefining data retrieval methods.

The following profiles were discussed for the main GDC capabilities described in the previous section:

- Core
- Partial Access
- Resample Access
- Data Processing with sub-profiles conforming to different workflow languages (e.g., openEO API, OGC API — Processes)

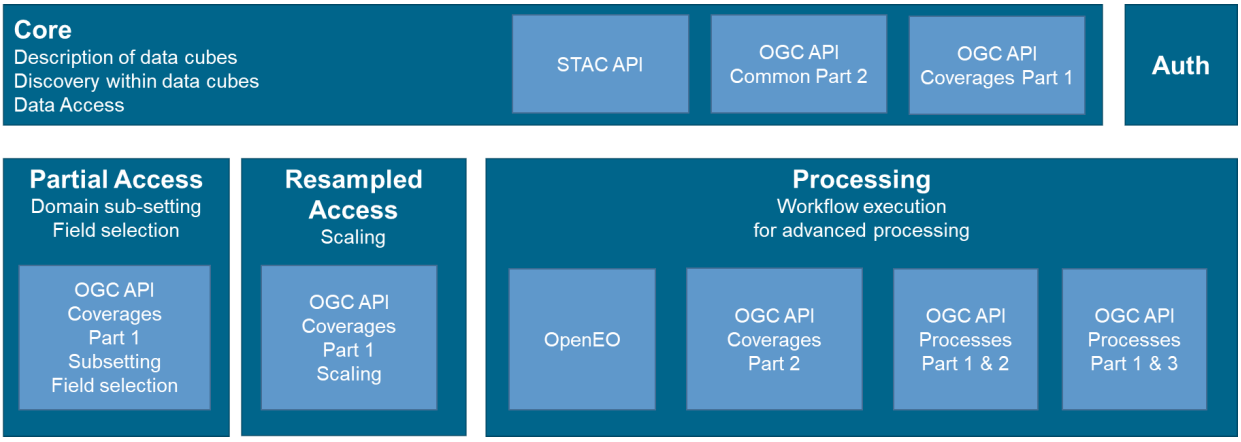

Figure 1

2.4.1. GDC Core (Full Data Access)

Defines how GeoDataCubes are described through metadata and how they can be downloaded "as-is" (e.g. a netCDF formatted package downloaded from object storage).

The initial discussion by the Testbed GDC task participants was to follow OGC API — Coverages endpoints:

- `GET /`
- `GET /conformance`
- `GET /collections`
- `GET /collections/{collectionId}`
- `GET /collections/{collectionId}/schema`
- `GET /collections/{collectionId}/coverage`

However, the GDC API implementations in Testbed 20 differ between the provision of STAC API for the `/collections` endpoint as opposed to OGC API — Coverages.

Note that the endpoints/paths in a static context may have any path in the URL, but the responses must follow the same schema (comparable with STAC and STAC API). This allows the geospatial data to be hosted on any cloud storage infrastructure without the need for an API (comparable to static STAC catalogs).

So a GDC instance is described through the collection metadata. The data contained in a GDC can be found through the link relation type `http://www.opengis.net/def/rel/ogc/1.0/coverage` (comparable with STAC assets).

An API implementation of the OGC API — Coverages Standard is always compliant with the requirements for this GDC Core profile.

2.4.2. GDC Partial Access

- Requires: GDC Core

This profile is a full implementation of the OGC API — Coverages — Part 1 Standard. Additional parts may be implemented (i.e. the landing page). A partial access capability also requires:

- Domain subsetting: This is filtering for a given time/area of interest, or additional dimensions not considered part of the "range". In other words the output of the coverage function for a given direct position in its domain — `subset=`, `datetime=`, `bbox=`.
- Field Selection: This is filtering on values returned for a given position, such as requesting specific band(s) or climate variable(s) — `properties=`.

2.4.3. GDC Resampled Access

- Requires: GDC Partial Access

This profile adds scaling capabilities to the GDC (Down/Up sampling — `scale-axes=`, `scale-factor=`, `scale-size=`, `width=`, `height=`).

2.4.4. Authentication

- Requires: GDC Core

The authentication profile will be adopted as defined in openEO, so implementations should at least implement one of the following endpoints and send tokens afterwards to the actual endpoints.

- `GET /credentials/oidc`
- `GET /credentials/basic`

2.4.5. GDC Data Processing

- Requires: GDC Core

The initial Testbed 20 discussion was to submit a workflow definition to one endpoint for advanced data processing, such as `POST /tasks`. However, the implementations conducted by the Testbed 20 participants differ related to the processing standards used.

An implementation of this profile may process data in different modes: Synchronously, asynchronously, or on-demand (i.e. create a new GDC API deployment or virtual collection). Which mode to choose is specified through a `mode` query parameter, which can be either `sync`, `async`, `collection`, or `api`.

A workflow definition could be anything, for example:

- OGC API — Processes — Part 3 ("MOAW") — conformance class: `http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/moaw`
- CWL — conformance class: `http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/cwl`
- openEO process — conformance class: `http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/openeo`
- WCPS — conformance class: `http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/wcps`
- ...

The body of the request is the workflow document.

2.4.5.1. GDC Data Processing via openEO

An implementation of the openEO API conforming to API profile L1 (minimal) and Processes profiles L1 (minimal). The implementation of an openEO API can be integrated within a GDC API implementation or be separate (see note A). Additional extensions may be implemented.

A link with relation type `openeo` points to the openEO instance (path of the well-known document without `/.well-known/openeo`, e.g. `https://openeo.cloud` or `https://earthengine.openeo.org`).

See https://openeo.org, https://api.openeo.org and https://processes.openeo.org for details

2.4.5.2. GDC Data Processing via OGC API — Processes

This profile is an implementation of the OGC API — Processes — Part 1 Standard. An implementation of the OGC API — Processes Standard can be integrated within a GDC API implementation or be separate (see note A). Additional parts may be implemented (i.e. the landing page).

A link with relation type `service` points to an OGC API — Processes instance.

See https://docs.ogc.org/is/18-062r2/18-062r2.html for details.

Note A: At the time of the Testbed 20 initiative, there were still some potential conflicts combining OGC API — Processes — Part 1 and openEO under the same API tree within a single GDC API deployment. One opinion was that if implementors want to offer both openEO and OGC API — Processes, one of the APIs needs to be implemented separately.

The potential conflicts identified relate to the different responses to `/processes`, as well as the response for retrieving the list of running jobs at `/jobs`.

The `/processes` conflict could be addressed by more specific JSON content profile negotiation being considered for OGC API — Processes version 2.0 (issue https://github.com/opengeospatial/ogcapi-processes/issues/481). The `/jobs` conflict is mitigated by each job in OGC API — Processes (but not openEO) including a `type` property identifying a particular type of asynchronous jobs for different APIs (such as OGC API — Coverages, OGC API — Processes or openEO), as well as the fact that clients could be restricted to only see the job they created themselves. Another participant's opinion is that it is desirable for interoperability and it should be possible going forward to develop implementations able to deploy both capabilities at the same end-point, making such end-points compatible with both types of clients.

2.4.5.3. GDC Data Processing via OGC API — Coverages — Part 2

An implementation of the proposed OGC API — Coverages — Part 2: Filtering, deriving and aggregating fields extends the OGC API — Coverages — Part 1: Core capabilities with server-side processing using expressions defined in the OGC Common Query Language (CQL2).

2.5. Use Case: Vegetation Health Index

For the Testbed 20 GDC implementation and testing work. the Vegetation Health Index (VHI) was chosen as a use case. This datacube use case combines vegetation and temperature data.

> The Vegetation Health Index that is implemented in the Alpine Drought Observatory (ADO) is used for detecting vegetation stress conditions, which arise when there is limited availability of soil moisture to plants. The VHI allows to identify drought impacts on vegetation which correspond to a combination of thermal stress, which is detected as an increase in Land Surface Temperature (LST), and a decrease in vegetation greenness, which is identified by lower-than-average values of the Normalized Difference Vegetation Index (NDVI). In the ADO project, VHI is computed on an 8-day basis and considers the reference period 2000–2020 for calculating extreme values of NDVI and LST.

Source: https://raw.githubusercontent.com/Eurac-Research/ado-data/main/factsheets/VHI_4.pdf

2.5.1. Algorithm

The **Vegetation Health Index (VHI)** is a composite index used to detect vegetation stress, particularly agricultural drought. The VHI combines two sub-indices:

Vegetation Condition Index (VCI)

$$VCI = 100\frac{NDVI - NDVI_{min}}{NDVI_{max} - NDVI_{min}} \quad (1)$$

where:

- **NDVI** = Normalized Difference Vegetation Index (smoothed over a certain period)
- **NDVI_min** and **NDVI_max** = Minimum and maximum NDVI values over the reference period.

2. Thermal Condition Index (TCI)

$$TCI = 100\frac{LST_{max} - LST}{LST_{max} - LST_{min}} \quad (2)$$

where:

- **LST** = Land Surface Temperature
- **LST_min** and **LST_max** = Minimum and maximum LST values over the reference period.

VHI Formula

$$VHI = \alpha VCI + (1 - \alpha)TCI \quad (3)$$

where **α** is the weight assigned to VCI, typically set to 0.5 for equal contribution from the TCI.

2.5.2. Interpretation of VHI Values

Table 2

VHI [%]	Drought Intensity
0–25	Extreme drought
25–35	Severe drought
35–42	Mild drought
>42	No drought

2.5.3. Data

When using Sentinel-2 for NDVI in the Vegetation Health Index (VHI), the data comes from high-resolution multispectral imagery, offering detailed vegetation greenness indicators.

Sentinel-2 provides NDVI at a 10–20 m spatial resolution, ideal for monitoring agricultural and forested areas.

For Land Surface Temperature (LST), datasets like ECMWF ERA5-Land or CMIP6 provide modeled temperature estimates. ERA5-Land offers hourly global data at approximately 9 km resolution, while CMIP6 provides climate projections with coarser spatial resolution, useful for long-term trend analysis and drought assessment. These sources complement VHI by offering thermal stress data.

2.5.4. Area and time of interest

Geospatial extent

The specified geospatial extent defines a rectangular bounding box covering the majority of Slovenia:

- Southwest Corner:
  - Latitude: 45.4236367
  - Longitude: 13.3652612
- Northeast Corner:
  - Latitude: 46.8639623
  - Longitude: 16.5153015

This region encompasses Slovenia's diverse landscapes, from the Julian Alps in the northwest to the Pannonian Plain in the east, including key urban areas like Ljubljana, Maribor, and Koper.

Time extent

The defined time period used spans September 1, 2018, to September 1, 2021, covering three full years. This time period allows for temporal analysis of geospatial phenomena, such as:

- Seasonal patterns in vegetation and land surface temperatures.
- Climate variability and trends, particularly related to drought or extreme weather.
- Impact assessment of significant environmental events during this timeframe.

This setup is ideal for studying environmental dynamics and changes within Slovenia using geospatial data from sources like Sentinel-2 for vegetation and ECMWF ERA5 for climate data.

2.6. Overview of implementations

The GDC API Profiles were implemented using existing standards:

Table 3 — Overview of GDC API Profile implementations

PROFILE/STANDARD	CRIM	ECERE	ELLIPSIS DRIVE	EURAC	MMS
Core					
OGC API — Common — Part 2*	-	Yes	Yes	-	partially
OGC API — Coverages — Part 1*	-	Yes	Yes	Yes	Yes
STAC API	Yes	-	Yes	Yes	Yes
Partial Access					
Coverages — Part 1* Subsetting, Field Selection	-	Yes	No	Yes	partially
Resampled Access					
OGC API — Coverages — Part 1* Scaling	-	Yes	Yes	-	-
Processing					
OpenEO API	-	-	-	Yes	Yes
OGC API — Coverages — Part 2*	-	Yes	-	-	-
OGC API — Processes — Part 1	Yes	Yes	Yes	-	-
OGC API — Processes — Part 2*	Yes	-	?	-	-
OGC API — Processes — Part 3*	Yes	Yes	-	-	-

*Draft standards

The Vegetation Health use case described above has been described with the following profiles and standards in this testbed:

2.6.1. Processing: OGC API — Coverages — Part 2 (using CQL2) implemented by Ecere

At the time of the Testbed 20 initiative, Ecere's implementation of the proposed Part 2: Filtering, Deriving and Aggregating fields profile supported requests such as the following enabling computing OGC Common Query Language (CQL2) expressions referencing coverage fields.

```
https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:NDVI_
ref_and_monthly_2020/coverage?
   subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015),time("2020-07")&
   width=1024&
   f=image/tiff;application=geotiff&
   properties=100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)
```

Listing 1 — HTTPS GET request to `/coverage` with embedded processing computing Vegation Condition Index (VCI) as implemented by Ecere at the time of the initiative

Suggested changes to the proposed Part 2 using an `alias` query parameter (not yet implemented) would support referencing newly derived fields and leave the `properties` query parameter strictly for field selection. Using this approach, the above request would become the following.

```
/collections/T20-VHI:MonthlyRef_2017_2020:NDVI_ref_and_monthly_2020/coverage?
   subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015),time("2020-07")&
   width=1024&
   f=image/tiff;application=geotiff&
   alias[VCI]=100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)&
   properties=VCI
```

Listing 2 — Request to `/coverage` with embedded processing computing Vegation Condition Index (VCI) with suggested changes

With a proposed `joinCollections` parameter, cross-collection queries such as the full VHI workflow could be implemented as follows.

```
/collections/sentinel2-l2a/coverage?
   subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015),time=2020-07&
   joinCollections=https://planetarycomputer.microsoft.com/api/stac/v1/
collections/nasa-nex-gddp-cmip6&
   filter=SCL = 4 and cmip6:model = 'GFDL-ESM4' and cmip6:scenario = 'ssp585'&
   alias[NDVI_monthly]=AggregateMulti((B08 - B04)/(B08 + B04), Max, ('time'),
('P1M'))&
   alias[NDVI_max]=AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'), ('R4/
2017-01-01/P1Y'), ('2020-01-01'))&
   alias[NDVI_min]=AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'), ('R4/
2017-01-01/P1Y'), ('2020-01-01'))&
   alias[VCI]=100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)&
   alias[LST_monthly]=AggregateMulti(tas, Avg, ('time'), ('P1M'))&
   alias[LST_max]=AggregateMulti(LST_monthly, Max, ('time'), ('P1M'), ('R4/2017-
01-01/P1Y'), ('2020-01-01'))&
```

```
   alias[LST_min]=AggregateMulti(LST_monthly, Min, ('time'), ('P1M'), ('R4/2017-
01-01/P1Y'), ('2020-01-01'))&
   alias[TCI]=100 * (LST_max - LST_monthly) / (LST_max - LST_min)&
   alias[VHI]=0.5 * VCI + 0.5 * TCI&
   properties=VHI
```

Listing 3 — Request to `/coverage` with embedded processing computing Vegation Health Index (VHI) with cross-collection queries

2.6.2. Processing: OpenEO API implemented by Eurac and Matthias Mohr (openEO for Google Earth Engine)

Processing on a data cube can be performed using the OpenEO API. An OpenEO process graph (see below) can be created with client libraries and applications, such as the OpenEO Python Client or the OpenEO WebEditor.

```
spatial_extent = {
          "west": 13.3652612,
          "east": 16.5153015,
          "south": 45.4236367,
          "north": 46.8639623
        }

temporal_extent = ["2020-06-01","2020-07-01"]
bands = ["red","nir","scl"]
s2_cube = conn.load_stac(url="https://earth-search.aws.element84.com/v1/
collections/sentinel-2-l2a",
                             spatial_extent=spatial_extent,
                             temporal_extent=temporal_extent,
                             bands=bands).resample_spatial(resolution=0.25,
projection="EPSG:4326")
s2_cube.metadata = s2_cube.metadata.add_dimension("time", label=None, type=
"temporal")
s2_cube.metadata = s2_cube.metadata.add_dimension("latitude", label=None, type=
"spatial")
s2_cube.metadata = s2_cube.metadata.add_dimension("longitude", label=None, type=
"spatial")
s2_cube = s2_cube.rename_dimension(source="latitude",target="lat").rename_
dimension(source="longitude",target="lon")
scl = s2_cube.band("scl")
vegetation_mask = (scl == 4)
s2_cube_masked = s2_cube.filter_bands(["red","nir"]).mask(vegetation_mask)

B04 = s2_cube_masked.band("red")
B08 = s2_cube_masked.band("nir")
ndvi = (B08 - B04) / (B08 + B04)

ndvi_max = ndvi.reduce_dimension(dimension="time",reducer="max").add_
dimension(name="bands",type="bands",label="NDVI_MAX")
ndvi_min = ndvi.reduce_dimension(dimension="time",reducer="min").add_
dimension(name="bands",type="bands",label="NDVI_MIN")
ndvi_min_max = ndvi_max.merge_cubes(ndvi_min)

aggregated_ndvi = ndvi.aggregate_temporal_period("month",reducer="max")
ndvi_min = ndvi_min_max.band("NDVI_MIN")
ndvi_max = ndvi_min_max.band("NDVI_MAX")

diff = ndvi_max - ndvi_min
```

```
VCI = aggregated_ndvi.merge_cubes(ndvi_min,overlap_resolver="subtract").merge_
cubes(diff,overlap_resolver="divide").add_dimension(name="bands",type="bands",
label="value")
```

Listing 4 — Building an OpenEO process graph with Python

```
{
  "process_graph": {
    "loadcollection1": {
      "process_id": "load_collection",
      "arguments": {
        "bands": [
          "red",
          "nir",
          "scl"
        ],
        "id": "SENTINEL2_L2A",
        "spatial_extent": {
          "west": 13.3652612,
          "east": 16.5153015,
          "south": 45.4236367,
          "north": 46.8639623
        },
        "temporal_extent": [
          "2020-01-01",
          "2020-12-31"
        ]
      }
    },
    "resamplespatial1": {
      "process_id": "resample_spatial",
      "arguments": {
        "align": "upper-left",
        "data": {
          "from_node": "loadcollection1"
        },
        "method": "near",
        "projection": "EPSG:4326",
        "resolution": 0.125
      }
    },
    "filterbands1": {
      "process_id": "filter_bands",
      "arguments": {
        "bands": [
          "red",
          "nir"
        ],
        "data": {
          "from_node": "resamplespatial1"
        }
      }
    },
    "reducedimension1": {
      "process_id": "reduce_dimension",
      "arguments": {
        "data": {
          "from_node": "resamplespatial1"
        },
        "dimension": "bands",
        "reducer": {
          "process_graph": {
            "arrayelement1": {
```

```
            "process_id": "array_element",
            "arguments": {
              "data": {
                "from_parameter": "data"
              },
              "index": 2
            }
          },
          "eq1": {
            "process_id": "eq",
            "arguments": {
              "x": {
                "from_node": "arrayelement1"
              },
              "y": 4
            },
            "result": true
          }
        }
      }
    }
  },
  "mask1": {
    "process_id": "mask",
    "arguments": {
      "data": {
        "from_node": "filterbands1"
      },
      "mask": {
        "from_node": "reducedimension1"
      }
    }
  },
  "reducedimension2": {
    "process_id": "reduce_dimension",
    "arguments": {
      "data": {
        "from_node": "mask1"
      },
      "dimension": "bands",
      "reducer": {
        "process_graph": {
          "arrayelement2": {
            "process_id": "array_element",
            "arguments": {
              "data": {
                "from_parameter": "data"
              },
              "index": 1
            }
          },
          "arrayelement3": {
            "process_id": "array_element",
            "arguments": {
              "data": {
                "from_parameter": "data"
              },
              "index": 0
            }
          },
          "subtract1": {
            "process_id": "subtract",
            "arguments": {
```

```
            "x": {
              "from_node": "arrayelement2"
            },
            "y": {
              "from_node": "arrayelement3"
            }
          }
        },
        "add1": {
          "process_id": "add",
          "arguments": {
            "x": {
              "from_node": "arrayelement2"
            },
            "y": {
              "from_node": "arrayelement3"
            }
          }
        },
        "divide1": {
          "process_id": "divide",
          "arguments": {
            "x": {
              "from_node": "subtract1"
            },
            "y": {
              "from_node": "add1"
            }
          },
          "result": true
        }
      }
    }
  }
},
"aggregatetemporalperiod1": {
  "process_id": "aggregate_temporal_period",
  "arguments": {
    "data": {
      "from_node": "reducedimension2"
    },
    "period": "month",
    "reducer": {
      "process_graph": {
        "median1": {
          "process_id": "median",
          "arguments": {
            "data": {
              "from_parameter": "data"
            }
          },
          "result": true
        }
      }
    }
  }
},
"reducedimension3": {
  "process_id": "reduce_dimension",
  "arguments": {
    "data": {
      "from_node": "aggregatetemporalperiod1"
    },
```

```
      "dimension": "time",
      "reducer": {
        "process_graph": {
          "max1": {
            "process_id": "max",
            "arguments": {
              "data": {
                "from_parameter": "data"
              }
            },
            "result": true
          }
        }
      }
    }
  },
  "adddimension1": {
    "process_id": "add_dimension",
    "arguments": {
      "data": {
        "from_node": "reducedimension3"
      },
      "label": "NDVI_MAX",
      "name": "bands",
      "type": "bands"
    }
  },
  "reducedimension4": {
    "process_id": "reduce_dimension",
    "arguments": {
      "data": {
        "from_node": "aggregatetemporalperiod1"
      },
      "dimension": "time",
      "reducer": {
        "process_graph": {
          "min1": {
            "process_id": "min",
            "arguments": {
              "data": {
                "from_parameter": "data"
              }
            },
            "result": true
          }
        }
      }
    }
  },
  "adddimension2": {
    "process_id": "add_dimension",
    "arguments": {
      "data": {
        "from_node": "reducedimension4"
      },
      "label": "NDVI_MIN",
      "name": "bands",
      "type": "bands"
    }
  },
  "mergecubes1": {
    "process_id": "merge_cubes",
    "arguments": {
```

```
        "cube1": {
          "from_node": "adddimension1"
        },
        "cube2": {
          "from_node": "adddimension2"
        }
      }
    },
    "renamedimension1": {
      "process_id": "rename_dimension",
      "arguments": {
        "data": {
          "from_node": "mergecubes1"
        },
        "source": "latitude",
        "target": "y"
      }
    },
    "renamedimension2": {
      "process_id": "rename_dimension",
      "arguments": {
        "data": {
          "from_node": "renamedimension1"
        },
        "source": "longitude",
        "target": "x"
      }
    },
    "saveresult1": {
      "process_id": "save_result",
      "arguments": {
        "data": {
          "from_node": "renamedimension2"
        },
        "format": "GTiff",
        "options": {}
      },
      "result": true
    }
  }
}
```

Listing 5 — OpenEO process graph decoded in JSON

2.6.3. Processing: Draft OGC API — Processes — Part 1: Core and Part 2: Deploy, Replace, Undeploy implemented by CRIM

Processes can be deployed to provide functions necessary for data cube operations for data discovery, filtering, math operations, and so forth. Processes can merge multiple operations together, which can be deployed using EO Application Packages. This approach can be combined with OGC API — Processes — Part 3: Workflows & Chaining.

The following example is an execution body for the `ndvi-batch` process previously registered as CWL.

```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/ndvi-batch",
  "inputs": {
    "stac_items": {
```

```
      "collection": "https://earth-search.aws.element84.com/v1/collections/
sentinel-2-l2a",
      "datetime": "2020-01-01T00:00:00Z/2021-01-01T00:00:00Z",
      "bbox": [13.3652612, 45.4236367, 16.5153015, 46.8639623],
      "format": "stac-items",
      "type": "application/geo+json"
    }
  }
}
```

Listing 6 — Chaining of Processes for NDVI batch from STAC Collection

2.6.4. Processing: Draft OGC API — Processes — Part 3: Workflows implemented by Ecere and CRIM

As implemented by Ecere during Testbed 20, the following JSON workflow definition uses "Nested Processes", and "Input Field Modifiers". Based on the JSON workflow definition, the final step of the VHI can be performed using OGC Common Query Language (CQL2). Within this expression input collections ("Collection Input") are referenced. Finally this expression can be posted to the `/processes/{processId}/execution?response=collection` endpoint to create a new virtual collection ("Collection Output"). Afterwards, OGC API — Coverages, OGC API — Tiles or OGC API — DGGS requests can be used to retrieve data from this resulting virtual collection, triggering on-demand processing for a specific area, time and resolution of interest. Server-side visualization can similarly be performed using implementations of the OGC API — Maps or OGC API — Tiles (for map tiles) Standards.

```
{
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
            "inputs": {
               "data": [
                  { "collection": "https://maps.gnosis.earth/ogcapi/collections/
T20-VHI:MonthlyRef_2017_2020:VCI_2020" },
                  { "collection": "https://maps.gnosis.earth/ogcapi/collections/
T20-VHI:MonthlyRef_2017_2020:TCI_2020" }
               ]
            },
            "properties": { "VHI": "0.8 * VCI + 0.2 * TCI" }
         }
      ]
   }
}
```

Listing 7 — Example workflow definition from OGC API - Processes - Part 3: Workflows as implemented by Ecere during the initiative

The execution endpoint for virtual "Collection Output" was proposed to be changed to `/collections` instead. However, this change was not yet implemented by the end of the initiative. Also not yet working by the end of the initiative, a complete VHI workflow directly referencing the source STAC metadata and making use of an *AggregateMulti()* function, an "Input Field Modifiers" `filter` as well as "Output Field Modifiers" could be as follows.

```
{
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://earth-search.aws.element84.com/v1/collections/
sentinel-2-l2a",
            "filter": "SCL = 4",
            "properties": {
               "NDVI_monthly": "AggregateMulti((B08 - B04)/(B08 + B04), Max,
('time'), ('P1M'))",
               "NDVI_max": "AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "NDVI_min": "AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "VCI": "100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)"
            }
         },
         {
            "collection": "https://planetarycomputer.microsoft.com/api/stac/v1/
collections/nasa-nex-gddp-cmip6",
            "filter": "cmip6:model = 'GFDL-ESM4' and cmip6:scenario = 'ssp585'",
            "properties": {
               "LST_monthly": "AggregateMulti(tas, Avg, ('time'), ('P1M'))" },
               "LST_max": "AggregateMulti(LST_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "LST_min": "AggregateMulti(LST_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "TCI": "100 * (LST_max - LST_monthly) / (LST_max - LST_min)"
            }
         }
      ]
   },
   "properties": { "VHI": "0.5 * VCI + 0.5 * TCI" }
}
```

Listing 8 — Workflow for generating a Vegetation Health Index (VHI) using the *PassThrough* process, CQL2 and *AggregateMulti()* function, defined as a single execution request and directly referencing the source STAC metadata

CRIM implemented support for the “Nested Processes”, “Collection Input” and “Remote Collections” requirement classes of Processes — Part 3: Workflows.

3

CONCLUSION & OUTLOOK

3 CONCLUSION & OUTLOOK

Testbed 20 work demonstrated that chained processing can be used with requirements classes from various standards, such as OGC API — Coverages (derived fields), OGC API — Processes (workflows & chaining), and OpenEO. However, some of the standards are still in draft stages. Especially the additional parts such as OGC API – Processes: Part 3 Workflows & Chaining and OGC API – Coverages Part 2 Derived Fields do not have multiple implementations or sufficient experience/feedback from users. As shown in the implementation overview, this leads to three different ways users can define processing steps and execute those steps. Also, from a data provider's point of view, there are three different standards they can choose from to make their data available with this kind of Geo Data Cube API. Therefore, Testbed 20 concluded with differing implementations that users could not easily interact with in an interoperable way.

However, Testbed 20 work did demonstrate which standards exist to conduct processing on geospatial (raster) data cubes and led to stronger collaboration between the developers of those standards. Although GDC API profiles were defined, how to proceed and whether an API definition is the best way forward is still being discussed.

There are some topics for further alignment between the above-mentioned standards that have started and can be further discussed after Testbed 20 completion:

- Common process names between OpenEO, CQL2, and OGC API Processes. OpenEO already has a pre-defined list of process names, which can be further adapted in other standards (e.g., in CQL2).
- Common API endpoint to post workflows (e.g., /tasks or /jobs).

4

SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

4 SECURITY, PRIVACY AND ETHICAL CONSIDERATIONS

During the course of this project, a thorough review was conducted to identify any potential security, privacy, and ethical concerns. After careful evaluation, it was determined that none of these considerations were relevant to the scope and nature of this project. Therefore, no specific measures or actions were required in these areas.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Pedro Gonçalves, editor (2021) 'OGC Best Practice for Earth Observation Application Package', http://www.opengis.net/doc/BP/eoap/1.0.

[2] Benjamin Pross, Panagiotis A. Vretanos, editors (2021). 'OGC 18-062r2: OGC API — Processes — Part 1: Core', http://www.opengis.net/doc/IS/ogcapi-processes-1/1.0.

[3] Matthias Schramm, Edzer Pebesma, Milutin Milenković, Luca Foresta, Jeroen Dries, Alexander Jacob, Wolfgang Wagner, Matthias Mohr, Markus Neteler, Miha Kadunc, and et al. 2021. "The openEO API–Harmonising the Use of Earth Observation Cloud Services Using Virtual Data Cube Functionalities" Remote Sensing 13, no. 6: 1125. https://doi.org/10.3390/rs13061125

[4] Eurac Research (2023). Vegetation Health Index Factsheet of the Interreg Alpine Space Alpine Drought Observatory. Retrieved from https://raw.githubusercontent.com/Eurac-Research/ado-data/main/factsheets/VHI_4.pdf.

[5] Alexander Jacob, editor (2024). 'OGC 23-047: OGC Testbed-19 GeoDataCubes Engineering Report'. http://www.opengis.net/doc/PER/T19-D011.

[6] Gérald Fenoy, editors (2025). 'DRAFT OGC API — Processes — Part 4: Job Management', https://docs.ogc.org/DRAFTS/24-051.html.

[7] Charles Heazel, Jérôme Jacovella-St-Louis, editors (2025). 'DRAFT: OGC API — Coverages — Part 1: Core', https://docs.ogc.org/DRAFTS/19-087.html.

[8] Jérôme Jacovella-St-Louis, Panagiotis A. Vretanos, editors (2025). 'DRAFT OGC API — Processes — Part 3: Workflows and Chaining', https://docs.ogc.org/DRAFTS/21-009.html.

A

ANNEX A (INFORMATIVE) ABBREVIATIONS/ACRONYMS

A ANNEX A (INFORMATIVE) ABBREVIATIONS/ACRONYMS

API	Application Programming Interface
CQL	Common Query Language
CRS	Coordinate Reference System
CWL	Common Workflow Language
ECMWF	European Centre for Medium-Range Weather Forecasts
EDR	OGC Environmental Data Retrieval
ERA5	ECMWF Reanalysis v5
GDC	Geo Data Cube
LST	Land Surface Temperature
NDVI	Normalized Difference Vegetation Index
STAC	Spatio Temporal Asset Catalog
VHI	Vegetation Health Index

B

ANNEX B (INFORMATIVE) EURAC RESEARCH GDC API

B

ANNEX B (INFORMATIVE) EURAC RESEARCH GDC API

B.1. GDC API Components

The Eurac Research GDC API implementation was further developed starting with the work done during the 2023 OGC Testbed 19. The Java openEO API implementation project called openeo-spring-driver https://github.com/Open-EO/openeo-spring-driver was the starting point This is the user-facing component handling the API calls and redirecting them to the other backend services. The public openEO API endpoint address is https://dev.openeo.eurac.edu/.

The second essential architecture component is the openeo-odc-driver https://github.com/Open-EO/openeo_odc_driver . This software is a Python processing engine that converts the openEO process graphs to executable Python code, computes the result, and returns the results back to the openeo-spring-driver. This component is based on two main open-source projects:

1. openeo-processes-dask (https://github.com/Open-EO/openeo-processes-dask): A Python implementation of most openEO processes written using the Xarray and Dask libraries.
2. openeo-pg-parser-networkx (https://github.com/Open-EO/openeo-pg-parser-networkx/): A library to parse the JSON process graphs to executable functions based on openeo-processes-dask.

The third essential component of the GDC API implementation developed by Eurac Research is the STAC API. The deployment is based on the stac-fastapi project (https://github.com/stac-utils/stac-fastapi) with a PostgreSQL back-end (https://github.com/stac-utils/stac-fastapi-pgstac). This component contains a STAC Catalog, which indexes all the openEO batch job results as STAC Collections. The public STAC API stac openeo is available at https://stac.openeo.eurac.edu/api/v1/pgstac/ .

B.2. OGC API – Coverages

A basic implementation of a `/coverage` endpoint is available in the GDC API instance of Eurac Research, which internally translates the request to an openEO process graph. The endpoint

is supported only by the specific collections created for Testbed 20 (`SENTINEL2_L2A`, `ERA5_FORECAST`, `ERA5_REANALYSIS`) and requires a basic authentication token to be passed in the headers.

The following request parameters are available at the endpoint:

- **`bbox`**`=west,south,east,north` for spatial filters
- **`datetime`**`=from[,to]` parameter for time trimming (or slicing in case a single timestamp is provided); input timestamp shall be in the format *YYYY-HH-MMThh:mm:ssTZ* (e.g., 2017-10-31T10:00:00Z); '..' wildcards are supported for open-ended intervals
- **`properties`**`=p1[,p2]*` for bands sub-selection, whose names are available in the `cube:dimensions/bands` section of in the STAC collection document
- **`f`**`=mime` for specifying the coverage output format (eg. *application/x-netcdf*, *image/tiff*)

> ***IMPORTANT***
>
> *The following **exceptions** apply to the current implementation.*

- The **`subset`** (and `subset-crs`) parameter is not accepted, hence subsetting needs to be requested through the `bbox` and `datetime` parameters.
- **`bbox-crs`** is not accepted, and lat/lon WGS84 decimal-degrees coordinates are assumed in the `bbox` parameter.
- **`scale`** is not accepted, only the original resolution of the underlying datacube can be used.

Further Notes:

- At least a `bbox` or a `datetime` filter are required to inhibit download of huge amounts of data (when requested in GeoTiff or NetCDF formats);
- Authentication tokens are required in the HTTP request for retrieve a coverage;

B.2.1. Data cubes (collections) of interest

Three data cubes were made available via the GDC API deployment, covering the needs of the sample workflow (VHI) and Provenance Demonstration.

- **Sentinel-2 L2A**: Global Sentinel-2 Level-2A imagery, based on the STAC Collection available from Element84 https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a .
- **ERA5 Reanalysis**: Global ERA5 Reanalysis collection based on the STAC Collection available from Microsoft Planetary Computer https://planetarycomputer.microsoft.com/api/stac/v1/collections/era5-pds .

- **ERA5 Forecast**: Global ERA5 Forecast collection based on the STAC Collection available from the Microsoft Planetary Computer https://planetarycomputer.microsoft.com/api/stac/v1/collections/era5-pds .

Unfortunately, the ERA5 data offered by Microsoft has not been recently updated and the latest samples are for late 2020. Recently, an official ERA5 STAC Collection, made available within the Destination Earth project was released (https://earthdatahub.destine.eu/api/stac/v1/collections/era5). The Testbed participants did not have time to replace the Microsoft dataset with this data source, but would definitely be the new reference ERA5 collection going forward.

B.2.2. Example use of implementation

A sample GET `/coverage` request to the Eurac Research server instance is the following:

https://dev.openeo.eurac.edu/collections/SENTINEL2_L2A/coverage?properties=blue,green,red&datetime=2023-01-08T00:00:00Z/2023-01-08T23:59:00Z&bbox=11.38,46.45,11.40,46.47&f=geotiff

The request specifies an area of interest above the city of Bolzano, where Eurac Research is located. The request selects only the blue, green, and red bands using the properties filtering and stores the result as a GeoTIFF.

B.2.3. Workflow Demonstration of Vegetation Health Index (VHI) Use Case

The workflow common to all the GDC API implementations (early 2025) involves using an index that combines satellite optical data and atmospheric data. The Copernicus Sentinel-2 mission provides the red (B04) and near-infrared (B08) spectral bands necessary for computing the Normalized Difference Vegetation Index (NDVI) values. To calculate the Vegetation Health Index (VHI), determining the Vegetation Condition Index (VCI) is required. The minimum and maximum NDVI values over an extended period must be computed, which will then be used to normalize the current NDVI value being analyzed.

VCI workflow using the openEO API of Eurac Research

```
import openeo

conn = openeo.connect('link:++https://dev.openeo.eurac.edu').authenticate_
basic(username="",password=""++[])
spatial_extent = {
          "west": 13.3652612,
          "east": 16.5153015,
          "south": 45.4236367,
          "north": 46.8639623
        }

temporal_extent = ["2020-01-01","2020-12-31"]
bands = ["red","nir","scl"]
s2_cube = conn.load_collection("SENTINEL2_L2A",
                               spatial_extent=spatial_extent,
                               temporal_extent=temporal_extent,
```

```
                                bands=bands).resample_spatial(resolution=0.125,
projection="EPSG:4326")

## the next metadata steps are necessary due to missing auto extraction of
metadata when using load_stac
s2_cube.metadata = s2_cube.metadata.rename_dimension("y","latitude")
s2_cube.metadata = s2_cube.metadata.rename_dimension("x","longitude")
scl = s2_cube.band("scl")
vegetation_mask = (scl == 4)
s2_cube_masked = s2_cube.filter_bands(["red","nir"]).mask(vegetation_mask)

B04 = s2_cube_masked.band("red")
B08 = s2_cube_masked.band("nir")
ndvi = (B08 - B04) / (B08 + B04)

ndvi_monthly_median = ndvi.aggregate_temporal_period("month",reducer="median")
ndvi_max = ndvi_monthly_median.max_time().add_dimension(name="bands",type=
"bands",label="NDVI_MAX")
ndvi_min = ndvi_monthly_median.min_time().add_dimension(name="bands",type=
"bands",label="NDVI_MIN")
ndvi_min_max = ndvi_max.merge_cubes(ndvi_min)
ndvi_min_max = ndvi_min_max.rename_dimension(source="latitude",target=
"y").rename_dimension(source="longitude",target="x")

temporal_extent = ["2020-07-01","2020-08-01"]

s2_cube = conn.load_stac(url="link:++https://earth-search.aws.element84.com/v1/
collections/sentinel-2-l2a++[]",
                                spatial_extent=spatial_extent,
                                temporal_extent=temporal_extent,
                                bands=bands).resample_spatial(resolution=0.125,
projection="EPSG:4326")

s2_cube.metadata = s2_cube.metadata.add_dimension("time", label=None, type=
"temporal")
scl = s2_cube.band("scl")
vegetation_mask = (scl == 4)
s2_cube_masked = s2_cube.filter_bands(["red","nir"]).mask(vegetation_mask)

B04 = s2_cube_masked.band("red")
B08 = s2_cube_masked.band("nir")
ndvi = (B08 - B04) / (B08 + B04)
ndvi_monthly_median = ndvi.aggregate_temporal_period("month",reducer="median")

ndvi_min = ndvi_min_max.band("NDVI_MIN")
ndvi_max = ndvi_min_max.band("NDVI_MAX")

diff = ndvi_max - ndvi_min

VCI = ndvi_monthly_median.merge_cubes(ndvi_min,overlap_resolver=
"subtract").merge_cubes(diff,overlap_resolver="divide")
VCI = VCI * 100

VCI = VCI.add_dimension(type="bands",name="bands",label="VCI")
```

Listing B.1

The next step is to calculate the second component of the VHI. This step takes into account the atmospheric conditions of the area being studied, specifically the Thermal Condition Index (TCI). The temperature data for this calculation is sourced from ERA5 Reanalysis.

TCI workflow using the openEO API of Eurac Research

```
spatial_extent = {
          "west": 13.3652612,
          "east": 16.5153015,
          "south": 45.4236367,
          "north": 46.8639623
        }

temporal_extent = ["2020-01-01","2020-12-31"]
bands = ["air_temperature_at_2_metres"]
era5 = conn.load_collection("ERA5_REANALYSIS",
                            spatial_extent=spatial_extent,
                            temporal_extent=temporal_extent,
                            bands=bands)

t_max = era5.max_time().rename_labels(dimension="bands",target=["T_MAX"])
t_min = era5.min_time().rename_labels(dimension="bands",target=["T_MIN"])
t_min_max = t_max.merge_cubes(t_min)

temporal_extent = ["2020-07-01","2020-08-01"]

era5 = conn.load_collection("ERA5_REANALYSIS",
                            spatial_extent=spatial_extent,
                            temporal_extent=temporal_extent,
                            bands=bands).drop_dimension("bands").aggregate_
temporal_period("month",reducer="mean")

t_min = t_min_max.band("T_MIN")
t_max = t_min_max.band("T_MAX")

TCI = (((era5 * -1) + t_max) / (t_max - t_min)) * 100
```

Listing B.2

Finally, the two indexes can be combined into the VHI. The VCI and TCI data cubes have different resolutions and therefore projection, resolution and pixel centers need to be aligned, which is possible using the `resample_cubes_spatial` openEO process.

VHI workflow using the openEO API of Eurac Research

```
VCI_aligned = VCI.resample_cube_spatial(target=TCI,method="average")

alpha = 0.5

VHI = alpha * VCI_aligned + (1 - alpha) * TCI
VHI = VHI.add_dimension(type="bands",name="bands",label="VHI")
```

Listing B.3

The complete example is available as an interactive Python notebook (`.ipynb`) on GitHub at https://github.com/clausmichele/OGC-T20-D144/blob/main/VHI_openeo_eurac_research.ipynb.

C

ANNEX C (INFORMATIVE) ELLIPSIS GDC API

ANNEX C (INFORMATIVE) ELLIPSIS GDC API

This annex was an optional deliverable for the participants. For more details, please contact Ellipsis Drive directly.

D

ANNEX D (INFORMATIVE) ECERE GDC API – GNOSIS MAP SERVER

ANNEX D (INFORMATIVE) ECERE GDC API — GNOSIS MAP SERVER

For the OGC Testbed 20 D140 GeoDataCube API Profile deliverable, Ecere provided an implementation of various OGC API standards as part of the Ecere GNOSIS Map Server. During Testbed 20, several enhancements were made to the GNOSIS Map Server and the underlying GNOSIS Software Development Kit (SDK). These enhancements supported temporal aggregations, as well as calculations and integration across disparate collections of raster data. Support for the OGC Common Query Language (CQL2) was also completed, with conformance to all requirements classes except for the array functions.

This included:

- Completing the support for the Well-Known Text (WKT) encoding for geometry.
- Completing the development of an in-house spatial query engine built around the Dimensionally Extended-9 Intersection Model (DE-9IM) to support spatial queries.
- Support for temporal queries and support for CQL2-JSON.
- Support for accessing data and results of processing through the candidate OGC API — Discrete Global Grid Systems (DGGS) Standard was also improved during the Testbed. This include improved support for the ISEA3H Discrete Global Grid Reference System, such as correctly converting between geodetic and authalic latitudes, and an implementation of the DGGS-optimized DGGS-JSON format.

The DGGS-JSON format relies on a canonical deterministic ordering of sub-zones (zones of a finer refinement level at least partially contained within a parent zone of a coarser refinement level) defined by the DGGRS.

D.1. Workflow Demonstration of Vegetation Health Index (VHI) Use Case

Ecere provided an implementation of the Vegetation Health Index (VHI) workflow combining a Vegetation Condition Index (VCI) computed from satellite imagery with Temperature Condition Index (TCI) from near-surface air temperature. The workflow is defined using OGC API — Processes execution requests extended with capabilities defined in the candidate OGC API Processes — Part 3: Workflows Standard. This included collection inputs, as well as input field

modifiers specifying OGC Common Query Language (CQL2) expressions deriving new values from the fields of the input(s). These execution requests can be submitted as a payload of POST requests to create virtual collections. For the purpose of the VHI workflow demonstration, the submission of execution requests for creating virtual collections was still at `/processes/{processId}/execution?response=collection` as defined by OGC API — Processes — Part 1: Core and the latest draft of Part 3: Workflows "Collection Output". However, during the initiative it was discussed that virtual collections might best be set up by posting workflow definitions to `/collections` instead, since the new virtual collections are created as new resources at `/collections/{collectionId}`. This would also allow for instantiating virtual collections using workflow definition languages other than OGC API — Processes execution requests.

Once virtual collections are set up, clients can request data from them using implementations of OGC API Standards that support data access mechanisms such as OGC API — Coverages, OGC API — Discrete Global Grid Systems (DGGS) and OGC API — Tiles (tiled coverage data output). As requests are made for a given area, time and resolution of interest, the necessary computations are performed on-the-fly to fulfill the request. In addition to the final workflow, the intermediate steps in the workflow are also provided as virtual collections. Whereas reference monthly minimum and maximum based on years 2017-2020 were used, a separate workflow was set up using a yearly maximum for year 2020 instead for the purpose of comparing with workflows implemented this way by other Testbed participants. The workflow was tested and pre-processed for the entirety of the Slovenia area of interest for the year 2020, at the native 10 meters resolution of the source sentinel-2 data.

D.1.1. First input: sentinel-2 Level-2A satellite imagery

The sentinel-2 Level 2-A collection, used in previous OGC Testbeds and Pilots, was used again in this initiative as the first input for computing the vegetation aspects of the VHI index. The data is sourced from the Cloud Optimized GeoTIFF (COG) hosted on Amazon Web Services (AWS) and managed by Element 84. The data cube is built around a local relational database of scene metadata initialized from around twenty million STAC items downloaded using the AWS S3 tool. The sentinel-2 catalog has about doubled in size to almost 40 million granules since then, which have not been fully indexed on the GNOSIS demonstration server. Initial attempts to use the STAC API deployment (https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a) in 2022 during the previous Climate Resilience Pilot initiative ran into issues relating to paging items which are described further in a dedicated section below. The numbers of STAC items matched (8952) from the latest STAC API endpoint for the Slovenia area of interest and 2017-2020:

```
https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a/items?
   bbox=13.3652612,45.4236367,16.5153015,46.8639623&
   datetime=2017-01-01T00:00:00Z/2020-12-31T23:59:59Z
```

Listing

differs significantly from the number of scenes for this same area and time of interest returned by the local STAC item database query (5174). This may be explained by additional scenes having been added for this time interval since items were fetched in 2022 for past initiatives. This may result in significant discrepancy when comparing the output of Ecere's implementation

of the VHI workflow with implementations from other participants who are using the STAC API deployment directly to identify granules of interest.

Data is requested from the COGs as needed using HTTP range requests for the overviews and tiles corresponding to the area, time, resolution and bands (fields) of interest to satisfy requests to the datacube.

This collection is available at https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a using all of the implementations of OGC API Standards access mechanisms supported by the GNOSIS Map Server, including OGC API — Coverages, Tiles (Coverage Tiles, supporting several registered 2D Tile Matrix Sets) and OGC API — DGGS (supporting three DGGRSs: GNOSIS Global Grid, ISEA3H and ISEA9R). In addition, data can be visualized directly using implementations of the OGC API — Maps or OGC API — Tiles (map tiles) Standards, or the collections can be referenced as input to processing workflows using the candidate OGC API — Processes — Part 3: Workflows Standard.

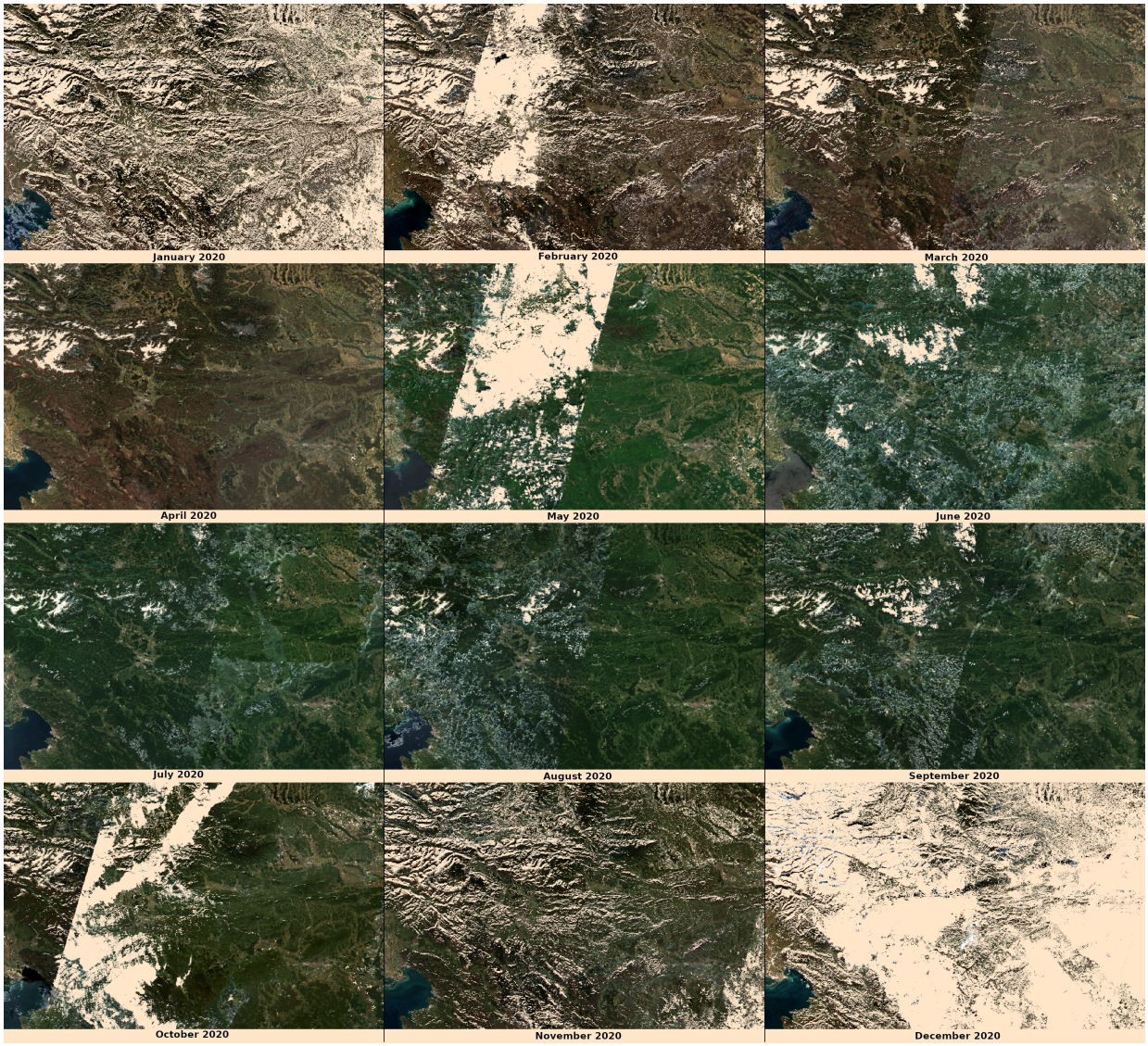

Figure D.1 — Visualization of monthly natural color imagery (using bands B04, B03 and B02) for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) throughout 2020, with `filter=SCL in (4,5,6)` (vegetation, bare soils and water)

Figure D.2 — Visualization of monthly natural color imagery (using bands B04, B03 and B02) for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) throughout 2020, with `filter=SCL in (4,5,6)` (vegetation, bare soils and water)

D.1.2. Monthly Normalized Difference Vegetation Index (NDVI) Time Series

The first step of the workflow involves computing a monthly time series of Normalized Difference Vegetation Index (NDVI) from the sentinel-2 data. Since satellite imagery is not available for the entire area of interest for every day, aggregating the data at a coarser monthly resolution provides an opportunity to eliminate most gaps and cloudy areas. A filter is applied using the Scene Classification Layer (SCL) selecting only area with vegetation (`SCL=4`), while the NDVI itself is computed using the B04 and B08 bands.

Like all other steps of this VHI computation, a single process named `PassThrough` is used for defining the workflow. This process is defined as simply passing through the input data as outputs, combining multiple data sources if multiple inputs are provided by assembling fields from all inputs. Although the process definition is agnostic of whether the data is of a vector or raster nature, the process is only used with raster sources for the purpose of the VHI workflow. The input to the process for this first step is the sentinel-2 data, passed using the Collection Input requirements class defined in the candidate OGC API — Processes — Part 3: Workflows.

The input field modifiers capability defined in the candidate OGC API — Processes — Part 3: Workflows draft Standard is used to express the aggregation and NDVI calculation as a CQL2 expression in the `properties` JSON property of the input.

An extension to CQL2 defining a ternary conditional operator is used in this first step for the purpose of applying the vegetation filter using the SCL field. This functionality could also be achieved with the `filter` JSON property also defined in input field modifiers instead, but due to challenges encountered with the implementation and time constraints, this approach ended being used to successfully demonstrate the workflow.

In addition to the Arithmetic Expressions requirements class defined by CQL2, a new function named `AggregateMulti` is introduced with the capability to aggregate across one or more dimensions, either to a single value across each dimension or to a specific coarser resolution. The function also supports more complex seasonal aggregation capabilities as additional parameters, which are used for the later steps of the workflow. The prototype of this *AggregateMulti()* function is as follows:

AggregateMulti (
input_expression: any,
reducing_operation: function,
dimensions: array{string}
[, *coarser_resolution*: array{string | number}
[, *repeated_interval*: array{string}
[, *repeat_shift*: array{string | number}]]]
)

The parameters used in this first step of the workflow are as follows:

- ***input_expression***: The field or expression to aggregate.

- ***reducing_operation***: The reducing operation function such as *Max*, *Min* or *Avg*. This could alternatively be an array, allowing a different reducing operation to be used for each dimension, but the definition the *AggregateMulti()* for the initiative only allowed for a single reducing operation.
- ***dimensions***: The names of the dimensions to aggregate on, such as `time`.
- ***coarser_resolution*** (optional): The coarser resolution at which to aggregate for each dimension. For a temporal dimension, the ISO8601 duration notation is used e.g., `P1M` for monthly. When omitted, the entire dimension is aggregated to a single value.

The entire workflow for the monthly NDVI time series, using the maximum NDVI value for each month and filtering on vegetation, is shown below.

```
{
   "id": "NDVI_monthly",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/
sentinel2-l2a",
            "properties": {
               "NDVI_monthly": "AggregateMulti(SCL = 4 ? (B08 - B04)/(B08 + B04)
: null, Max, ('time'), ('P1M'))"
            }
         }
      ]
   }
}
```

Listing D.1 — Workflow to generate Monthly NDVI Time Series

A predefined virtual collection for this step of the workflow is available from https://maps.gnosis.earth/ogcapi/collections/T20-VHI:NDVI_monthly. A request to an OGC API — Coverages endpoint for July 2020 for the whole of Slovenia would be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:NDVI_monthly/coverage?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

The `bbox` and `datetime` convenience parameters, which are more familiar to users of implementations of the OGC Web Map Service (WMS) Standard, can also be used as an alternative to the `subset` parameter. The `scale-factor`, `scale-axes` or `scale-size` parameters, or the convenience `width` and `height` parameters, can also be used to request a coverage at a particular resolution.

By default, the response is returned in GeoTIFF, while alternate formats can be negotiated using the `Accept:` HTTP request header. Support for netCDF output, which would support returning a time series as a single request, is planned as future work.

The following figures illustrate the time series as rendered for each month of 2020 using a request to an OGC API — Maps endpoint for the whole of Slovenia as well as for Ljubljana, which for July 2020 for the whole of Slovenia would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:NDVI_monthly/map?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

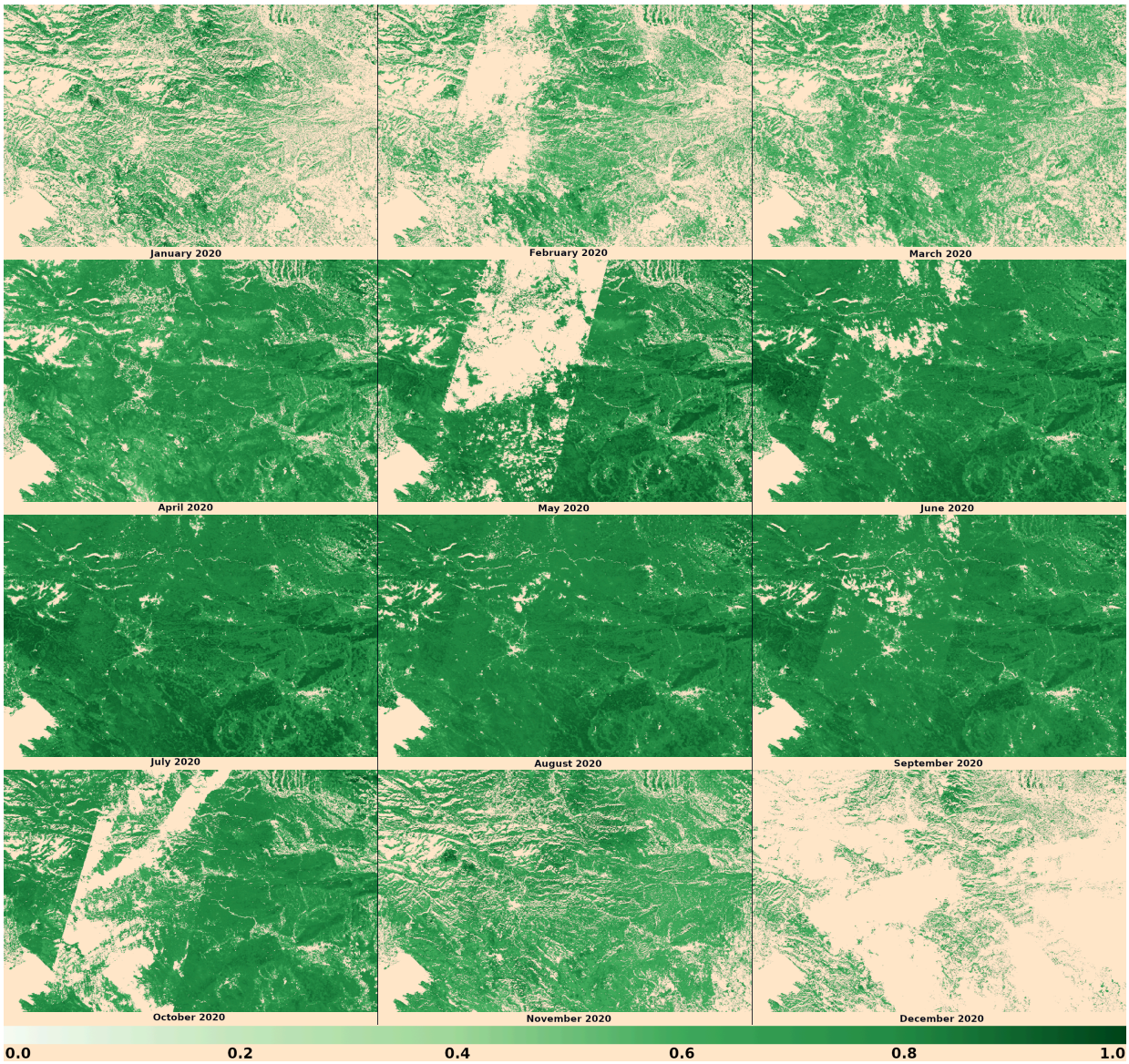

Figure D.3 — Visualization of monthly NDVI (maximum for the month) computed using bands B04 and B08, filtering for only vegetation (`SCL=4`), for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) throughout 2020

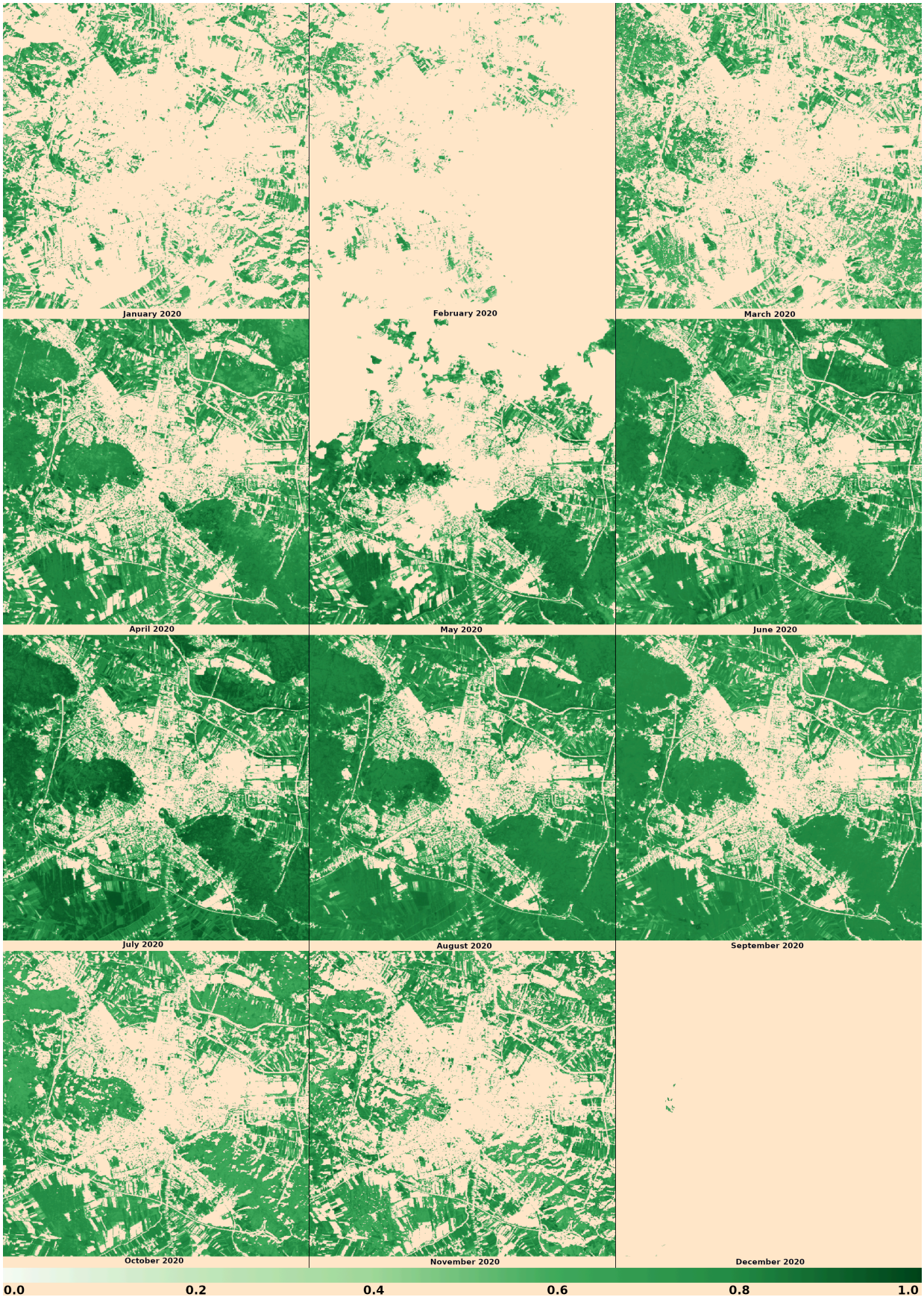

Figure D.4 — Visualization of monthly NDVI (maximum for the month) computed using bands B04 and B08, filtering for only vegetation (`SCL=4`), for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) throughout 2020

D.1.3. Reference NDVI Minimum and Maximum

The next step of the VHI workflow involves computation of reference minimum and maximum NDVI values which are used in the computation of the VCI. In the primary VHI workflow implemented by Ecere, the monthly minimum and maximum are calculated for each month for years 2017 through 2020. This calculation is computationally intensive and relies on a large quantity of data as the NDVI monthly time series from the previous step must first be produced for all four years.

The same *PassThrough* process and *AggregateMulti* function are used again, this time making use of the two additional parameters:

- ***repeated_interval***: Intervals for each dimension being aggregated, which combined with the *coarser_resolution* parameter, allows performing repeated aggregation over separate periods, such as calculating seasonal normals. For time, the ISO8601 interval notation is used, specifying how many times to repeat, when to start, and the time elapsed between repetition. For example, `R4/2017-01-01/P1Y` with a `P1M` *coarser_resolution* will aggregate together values of each month across four years starting in January 2017.
- ***repeat_shift***: Positions for each dimension being aggregated at which to shift the output of the repeated aggregation, so as to assign it a position different than the starting point of the repeated aggregation. This is set to the year of interest (2020 in the workflow demonstration) so that the time of the reference values corresponds to the time of the monthly values against which they are to be compared, which simplifies the computation of the VCI and TCI indices.

The resulting NDVI_ref_and_monthly_2020 collection includes the reference maximum (`NDVI_max`), minimum (`NDVI_min`), while also carrying over the current year monthly NDVI value (`NDVI_monthly`) for the year 2020.

```
{
   "id": "NDVI_ref_and_monthly_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
NDVI_monthly",
            "properties": {
               "NDVI_max": "AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "NDVI_min": "AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "NDVI_monthly": "NDVI_monthly"
            }
         }
      ]
   }
}
```

Listing D.2 — Workflow to generate reference NDVI minimum and maximum for each month based on years 2017-2020

A request to an OGC API — Coverages endpoint for July 2020 including only the maximum and minimum for the whole of Slovenia would be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:NDVI_ref_and_monthly_2020/coverage?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07&properties=NDVI_max,NDVI_min

The following figures illustrate the time series for the maximum and minimum as rendered for each month of 2020 using a request to an OGC API — Maps endpoint for the whole of Slovenia as well as for Ljubljana. For the maximum in July 2020 for the whole of Slovenia, a request to an OGC API — Maps endpoint would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:NDVI_ref_and_monthly_2020/map?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

whereas for visualizing the minimum reference value, a different style named `min` was set up, thus making the request:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:NDVI_ref_and_monthly_2020/styles/min/map?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

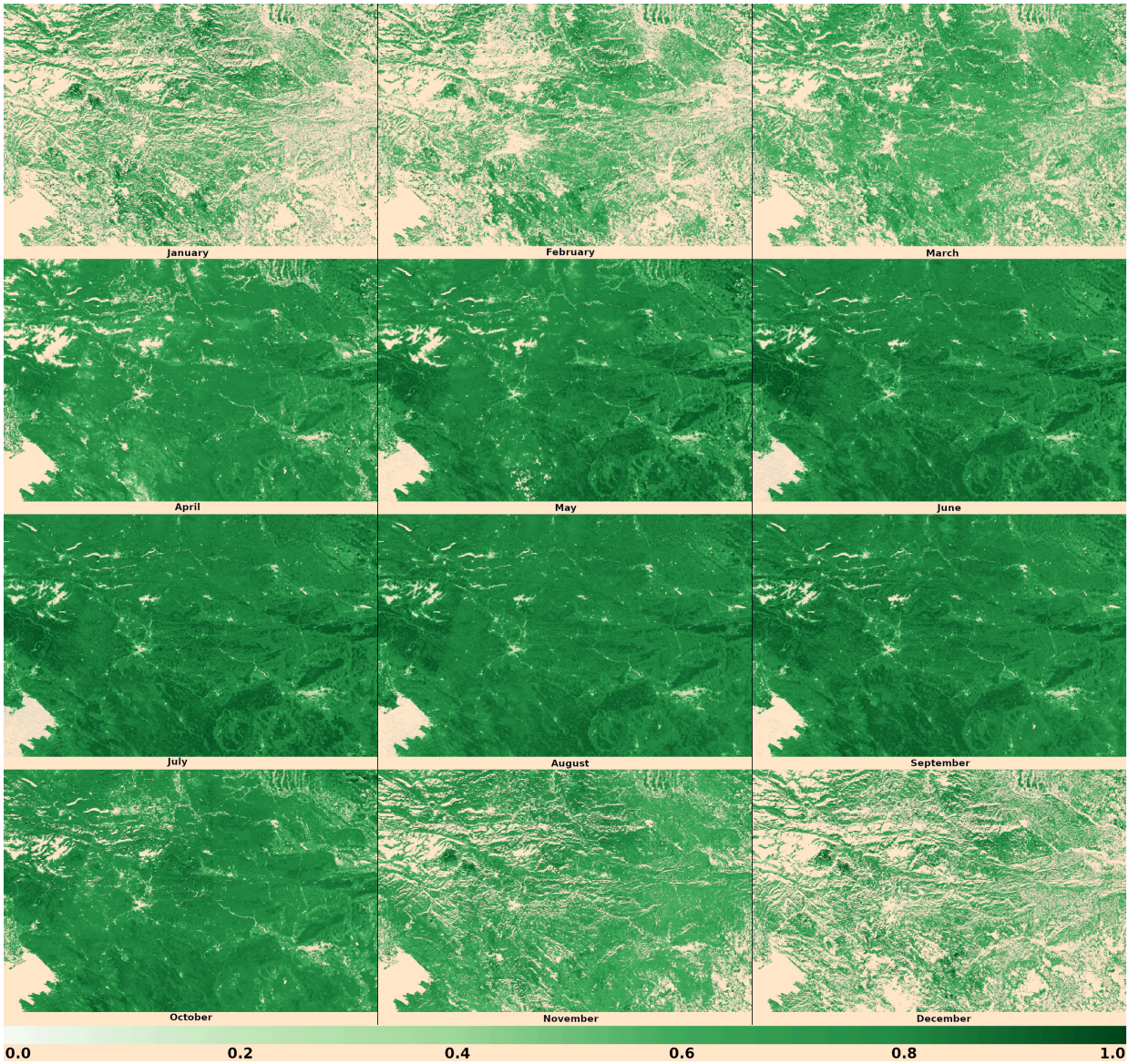

Figure D.5 — Visualization of reference monthly NDVI Maximum for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) based on years 2017-2020

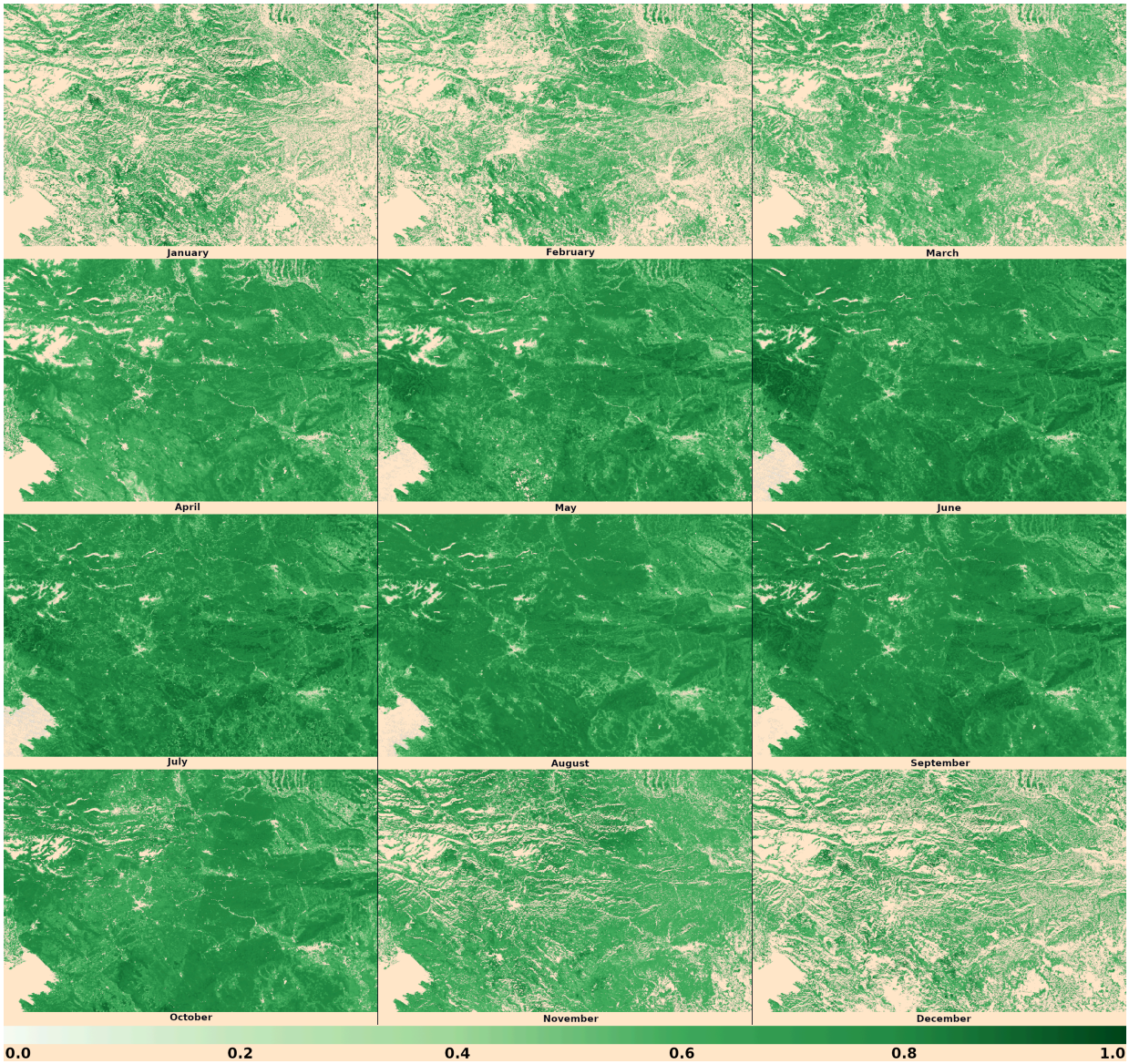

Figure D.6 — Visualization of reference monthly NDVI Minimum for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) based on years 2017-2020

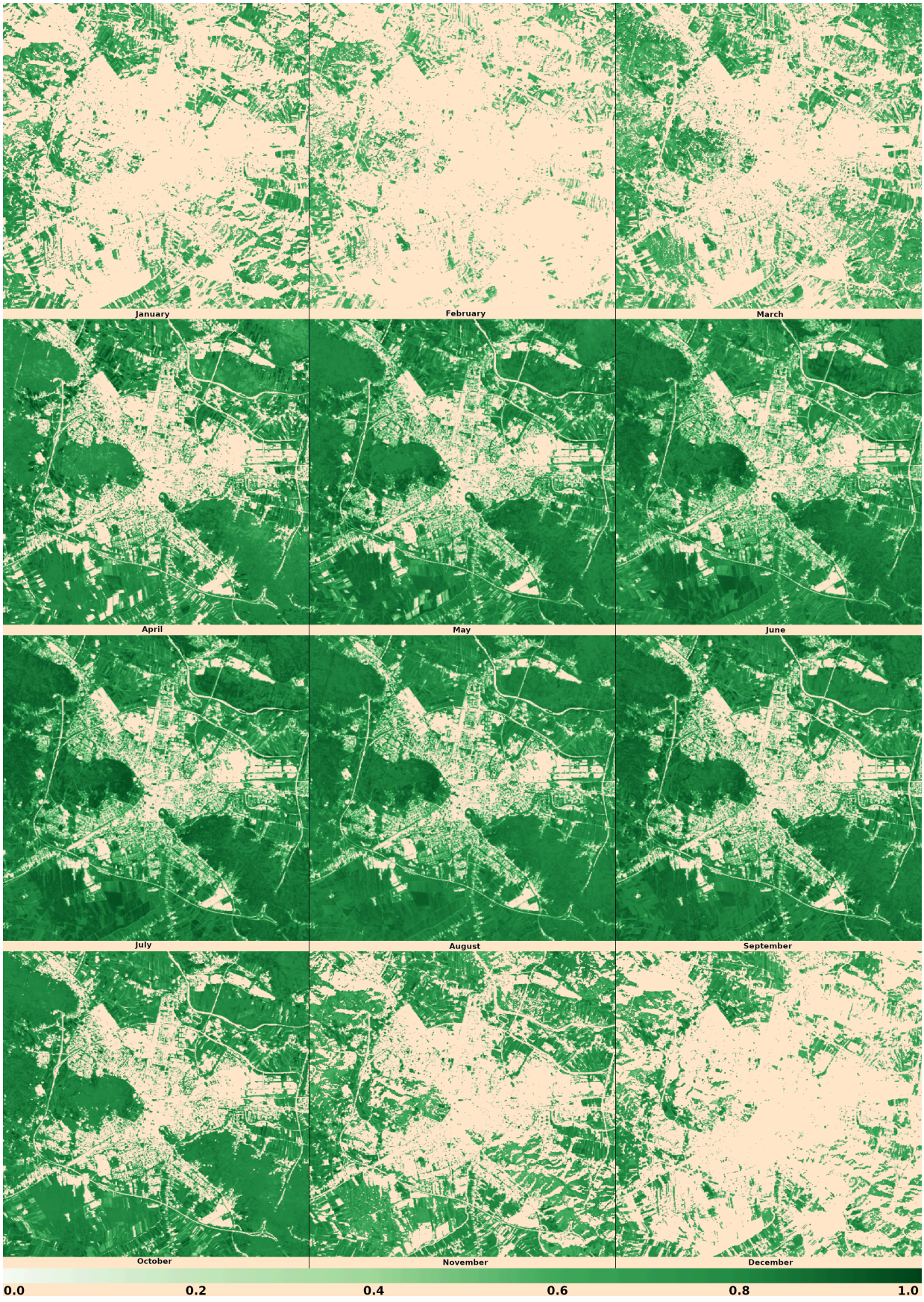

Figure D.7 — Visualization of reference monthly NDVI Maximum for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) based on years 2017-2020

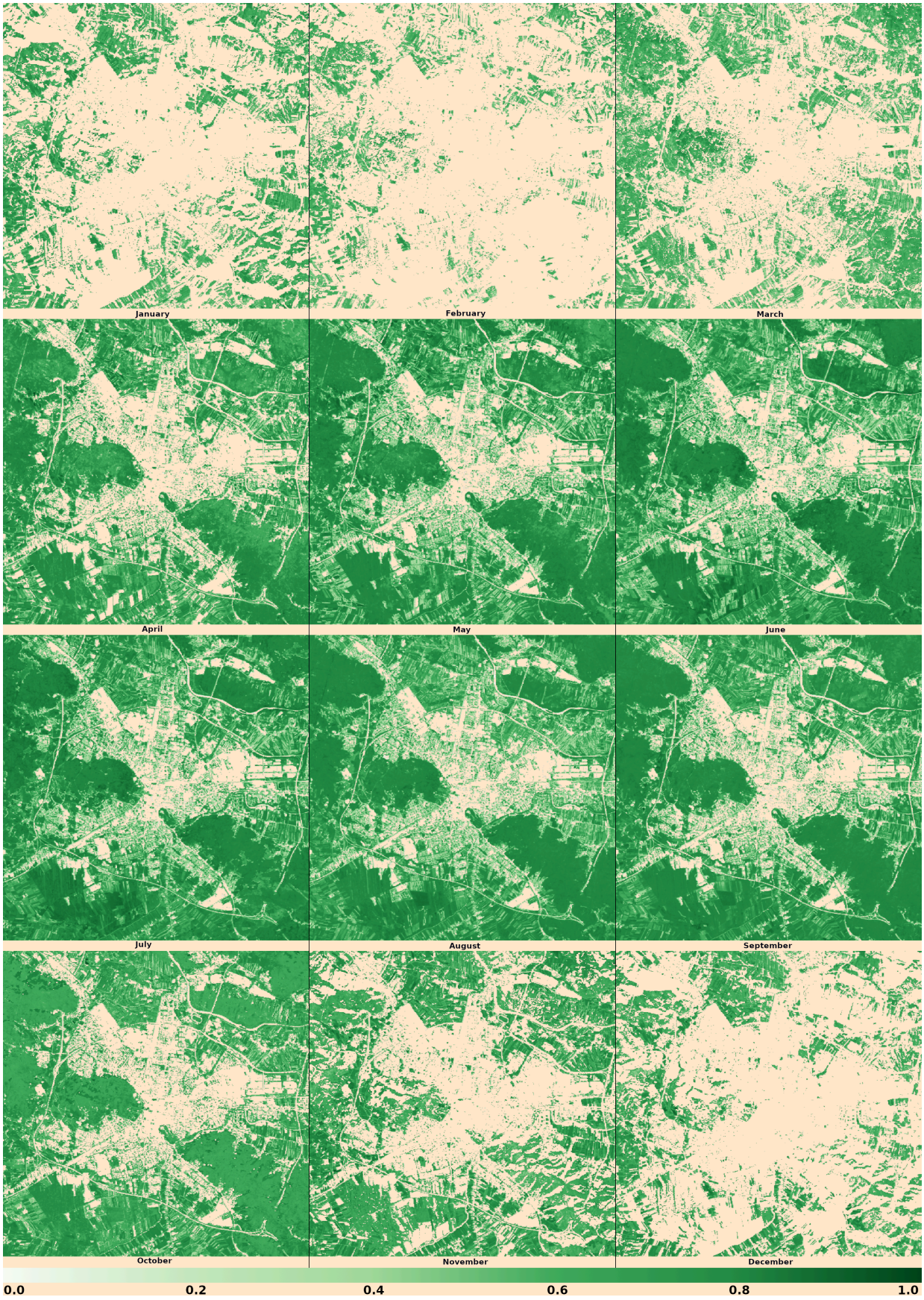

Figure D.8 — Visualization of reference monthly NDVI Minimum for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) based on years 2017-2020

Partly because of the huge computational and data requirements of computing monthly reference values across multiple years, other participants demonstrated workflows using reference minimum and maximum values for the whole year 2020 instead. These workflows yield different results less akin to the proper indices, since seasonal variation is not accounted for. Ecere also implemented this approach as a separate set of workflow definitions and virtual collections for the purpose of comparing the workflow output with other participants. The workflow for computing minimum and maximum NDVI values for the whole year 2020 is shown below. The *repeated_interval* parameter for *AggregateMulti* starts in 2020, repeating 12 times, with 1 month between repetitions.

```
{
   "id": "NDVI_ref_and_monthly_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
NDVI_monthly",
            "properties": {
               "NDVI_max": "AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'),
('R12/2020-01-01/P1M'), ('2020-01-01'))",
               "NDVI_min": "AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'),
('R12/2020-01-01/P1M'), ('2020-01-01'))",
               "NDVI_monthly": "NDVI_monthly"
            }
         }
      ]
   }
}
```

Listing D.3 — Workflow to generate reference NDVI minimum and maximum based on entire year 2020 (for comparison with other participants workflows)

These reference values were also made available as virtual collections at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:WholeYearRef_2020:NDVI_ref_and_monthly_2020, and illustrated below as rendered by the OGC API — Maps implementation.

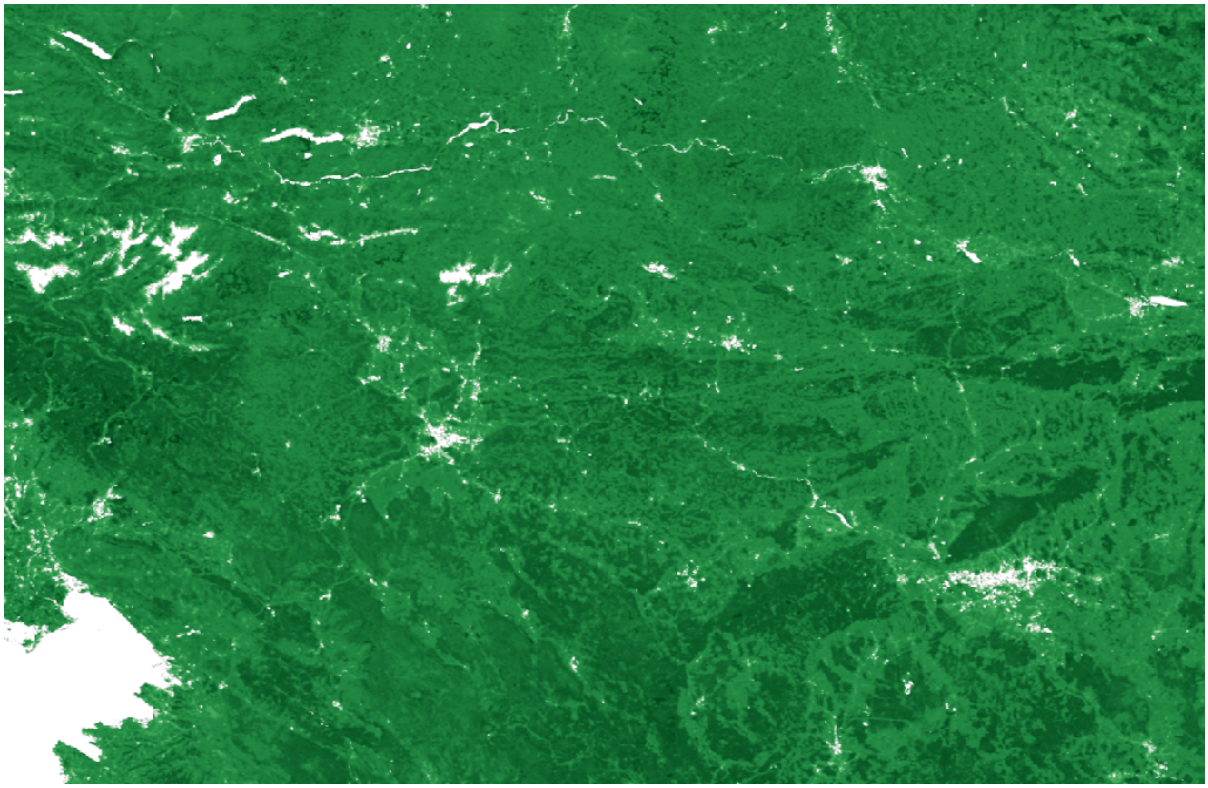

Figure D.9 — Visualization of reference NDVI Maximum for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) based on whole year 2020

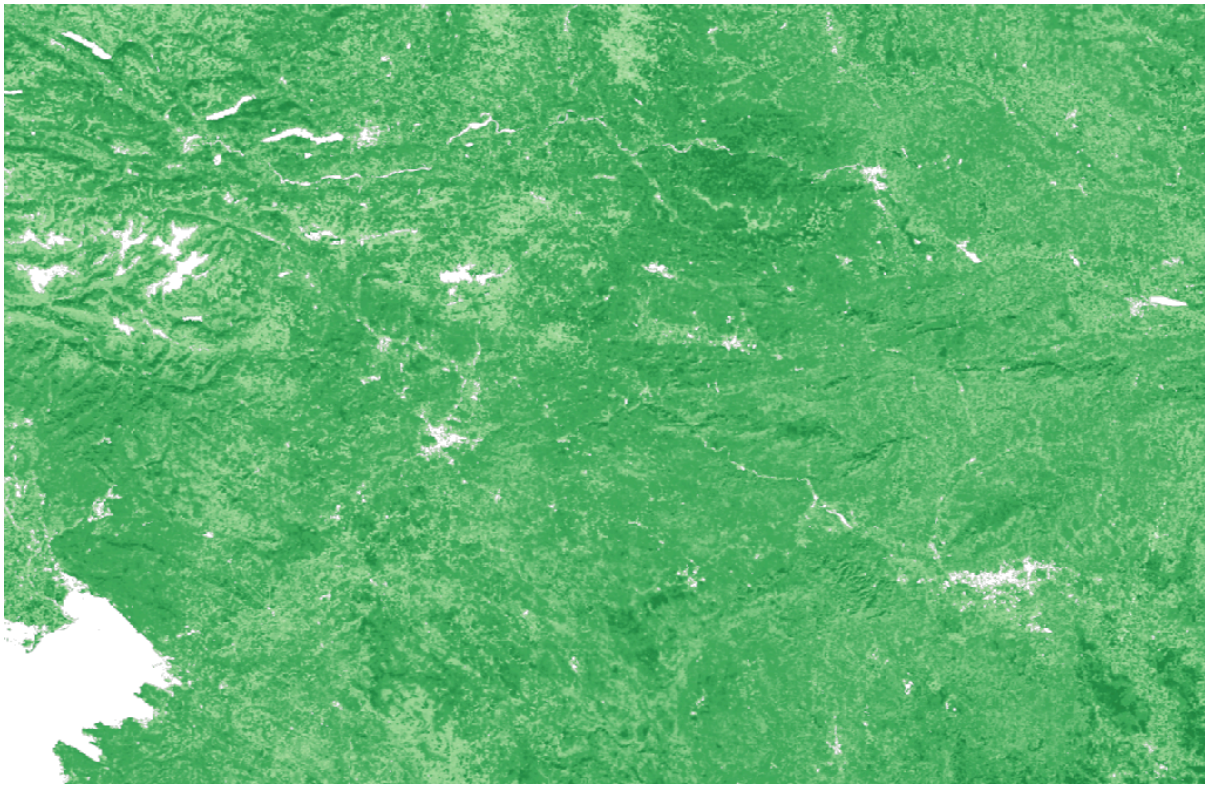

Figure D.10 — Visualization of reference NDVI Minimum for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) based on whole year 2020

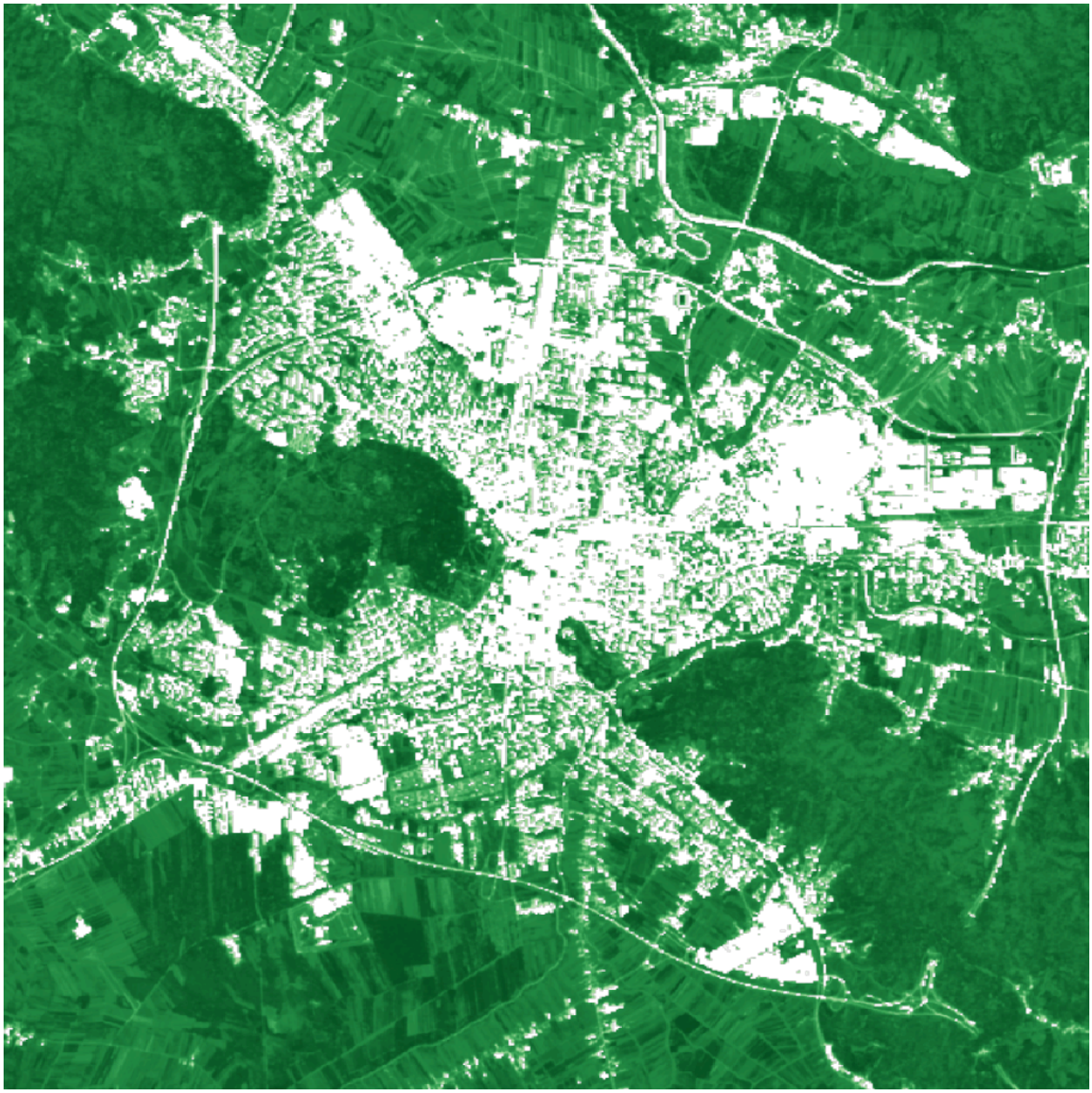

Figure D.11 — Visualization of reference NDVI Maximum for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) based on whole year 2020

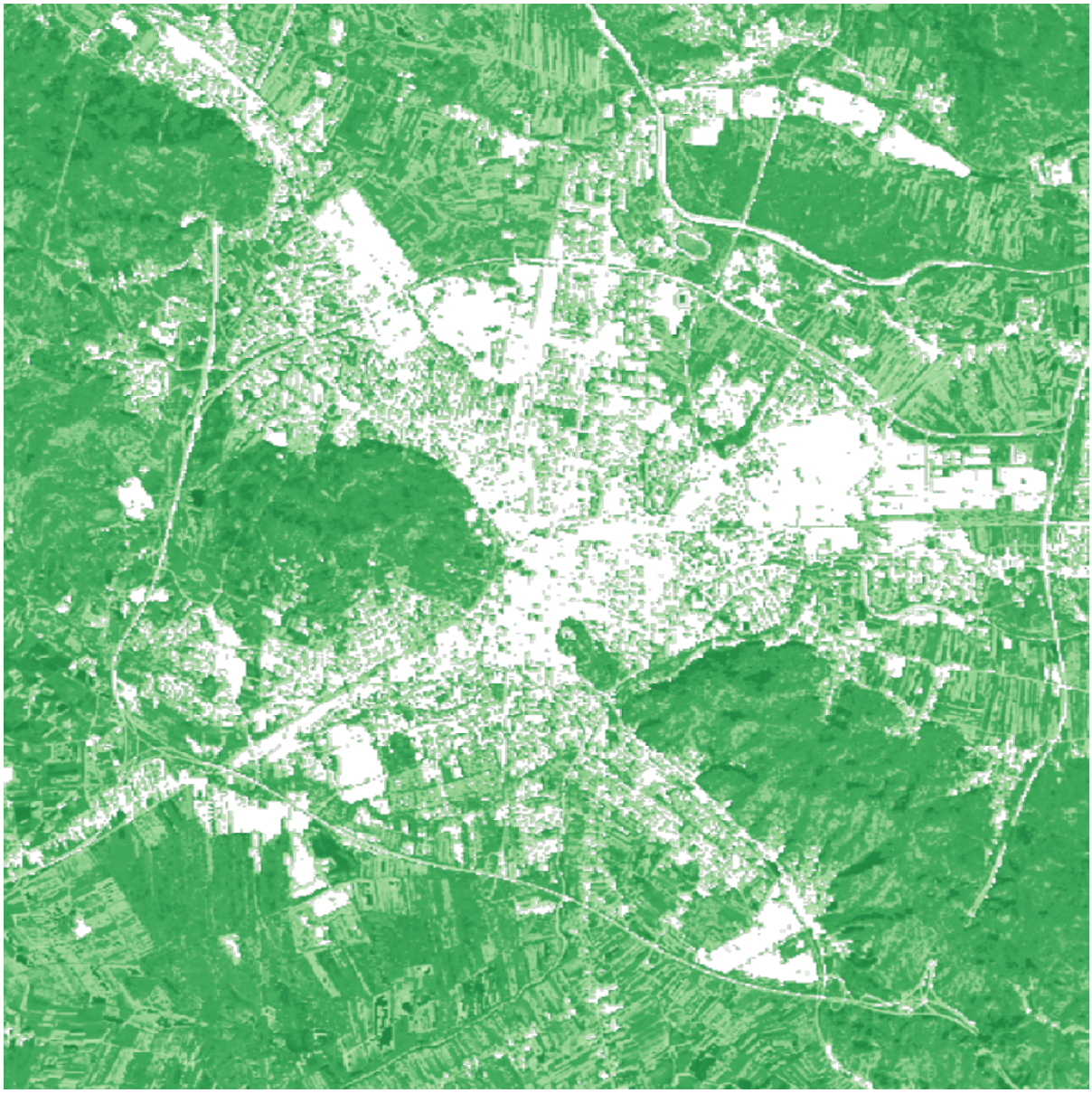

Figure D.12 — Visualization of reference NDVI Minimum for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) based on whole year 2020

D.1.4. Vegetation Condition Index (VCI)

The third step of the workflow produces the Vegetation Condition Index (VCI) by comparing the monthly NDVI values with the reference maximum and minimum, which were assembled together in the previous step of the workflow. The same *PassThrough* process is used again, and this step is performed using a simple CQL2 arithmetic expression directly reflecting the formula:

$$VCI = 100\frac{NDVI_{monthly} - NDVI_{min}}{NDVI_{max} - NDVI_{min}} \tag{D.1}$$

without the need for aggregation or other special functions.

An NDVI value corresponding to the reference maximum will yield a score of 100 contributing positively to the VHI, whereas an NDVI value corresponding to the reference minimum will yield a score of 0 indicative of poor vegetation health.

```
{
   "id": "VCI_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
MonthlyRef_2017_2020:NDVI_ref_and_monthly_2020",
            "properties": {
               "VCI": "100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)"
            }
         }
      ]
   }
}
```

Listing D.4 — Workflow to generate reference Vegetation Condition Index (VCI) (here using 2017-2020 monthly reference minimum and maximum)

A resulting virtual collection was made available at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VCI_2020.

A request to an OGC API — Coverages endpoint for July 2020 for the whole of Slovenia could be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VCI_2020/coverage?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

The following figures illustrate the VCI time series as rendered for each month of 2020 using a request to an OGC API — Maps endpoint for the whole of Slovenia as well as for Ljubljana. For July 2020 for the whole of Slovenia, a corresponding request would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VCI_2020/map?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

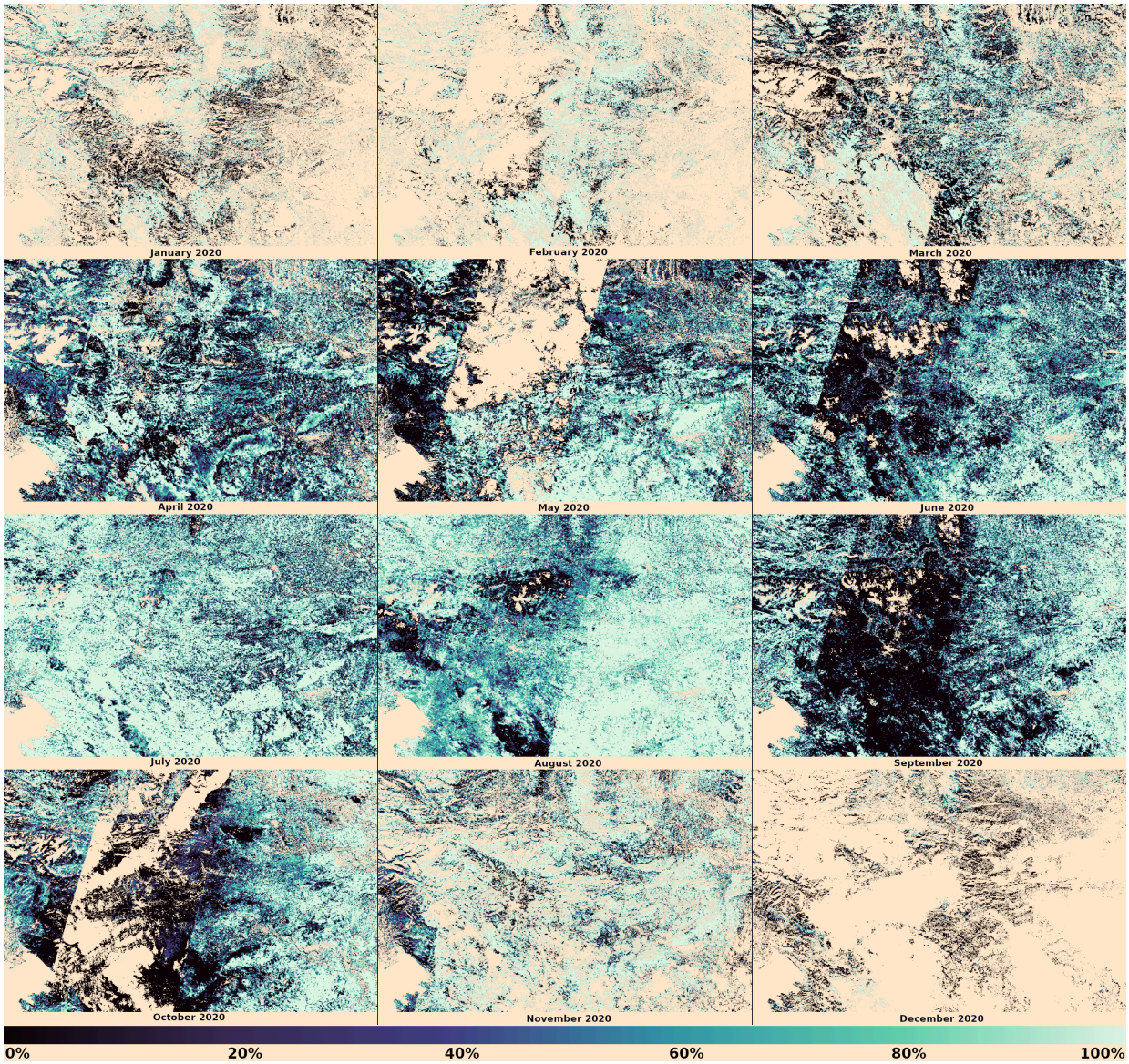

Figure D.13 — Visualization of Vegetation Condition Index (VCI) for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) throughout 2020 using the corresponding months of 2017-2020 for reference minimum and maximum

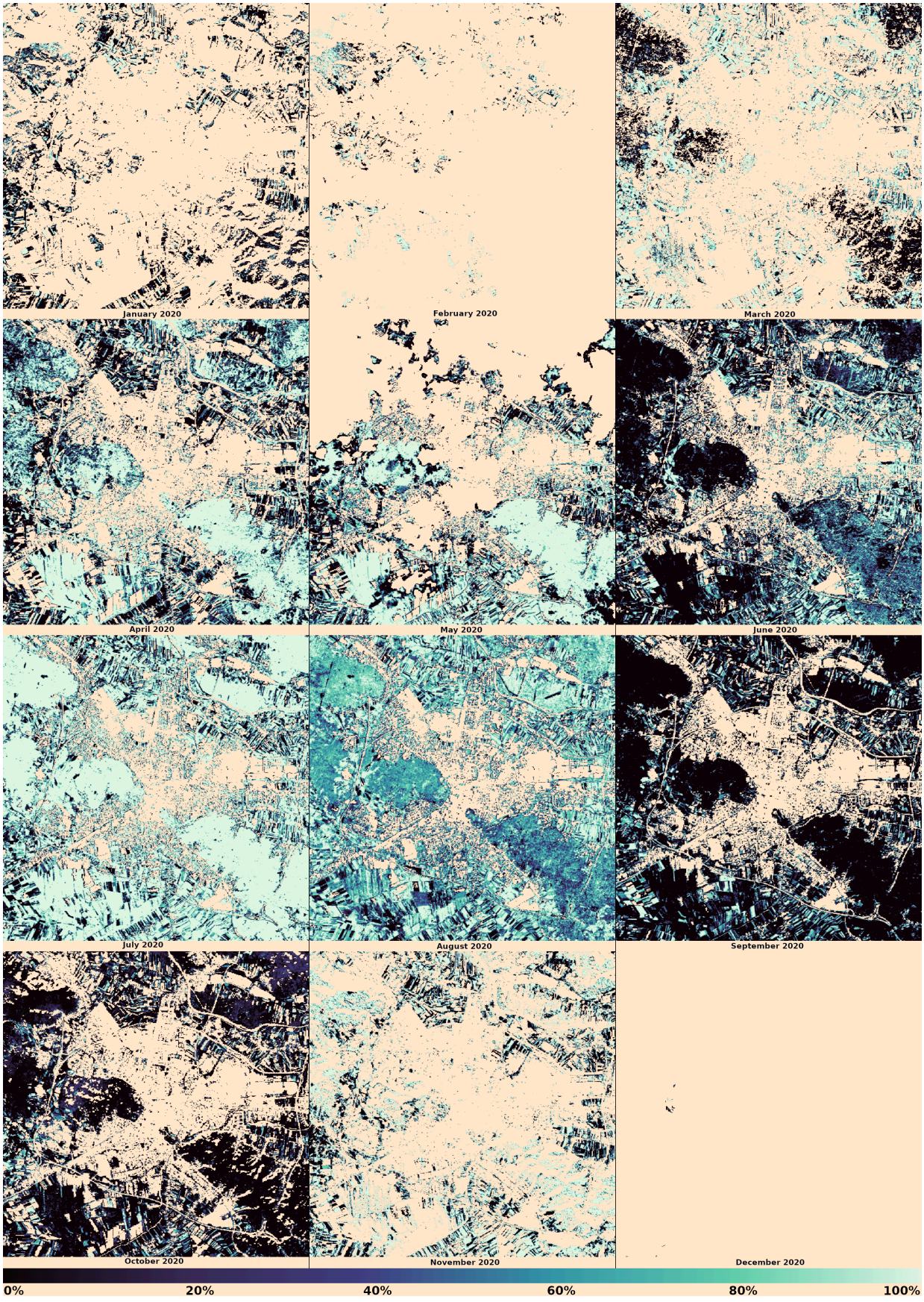

Figure D.14 — Visualization of Vegetation Condition Index (VCI) for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) throughout 2020 using the corresponding months of 2017-2020 for reference minimum and maximum

D.1.4.1. Second input: CMIP6 Surface Temperature

The surface temperature data from global downscaled CMIP6 climate projection was used for the climate-related component of the VHI workflow. The data from the NASA Center for Climate Simulation was retrieved as netCDF files from the Microsoft Planetary Computer by accessing links inside assets of the STAC items provided. These items were filtered specifically for the GFDL-ESM4 model, variant r1i1p1f1 (r: realization, i: initialization, p: physics, f: forcing), Shared Socio Economic scenario ssp585 (an update of the CMIP5 scenario RCP8.5 combined with socioeconomic reasons).

The collection with global coverage (excluding Antarctica) for daily values between November 21, 2016 and November 19, 2021, including the Daily Near-Surface Air Temperature (`tas`), Daily Maximum Near-Surface Air Temperature (`tasmax`) and Daily Minimum Near-Surface Air Temperature (`tasmin`) was loaded onto the demonstration GNOSIS Map Server as https://maps.gnosis.earth/ogcapi/collections/T20-VHI:cmip6-tb20, which allows access via implementations of all all OGC API Standards supported by the GNOSIS Map Server. Issues encountered with the STAC API deployment for this dataset are described below.

D.1.5. Monthly Land Surface Temperature (LST) Time Series

The first step for the temperature aspect of the VHI workflow consists of creating a monthly time series from the CMIP6 surface temperature dataset. This step is very similar to the step used to compute a monthly NDVI series, except that the CMIP6 dataset has full spatial coverage on a daily basis. The data is also available at a much coarser spatial resolution (15 minutes of latitude and longitude) compared to the sentinel-2 data's 10 meters resolution. The same *PassThrough* process and `AggregateMulti()` function are used, but in this case the average of each daily temperature (itself an average) is used, as shown in the workflow definition shown below.

```
{
   "id" : "LST_monthly",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
cmip6-tb20",
            "properties":
            {
               "LST_monthly": "AggregateMulti(tas, Avg, ('time'), ('P1M'))"
            }
         }
      ]
   }
}
```

Listing D.5 — Workflow to generate monthly Land Surface Temperature (LST) time series

A resulting virtual collection was made available at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:LST_monthly.

A request to an OGC API — Coverages endpoint for July 2020 for the whole of Slovenia could be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:LST_monthly/coverage?datetime=2020-07

The following figures illustrate the monthly LST time series as rendered for each month of 2020 using a request to an OGC API — Maps endpoint for the whole world. For July 2020, a corresponding request would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:LST_monthly/map?datetime=2020-07

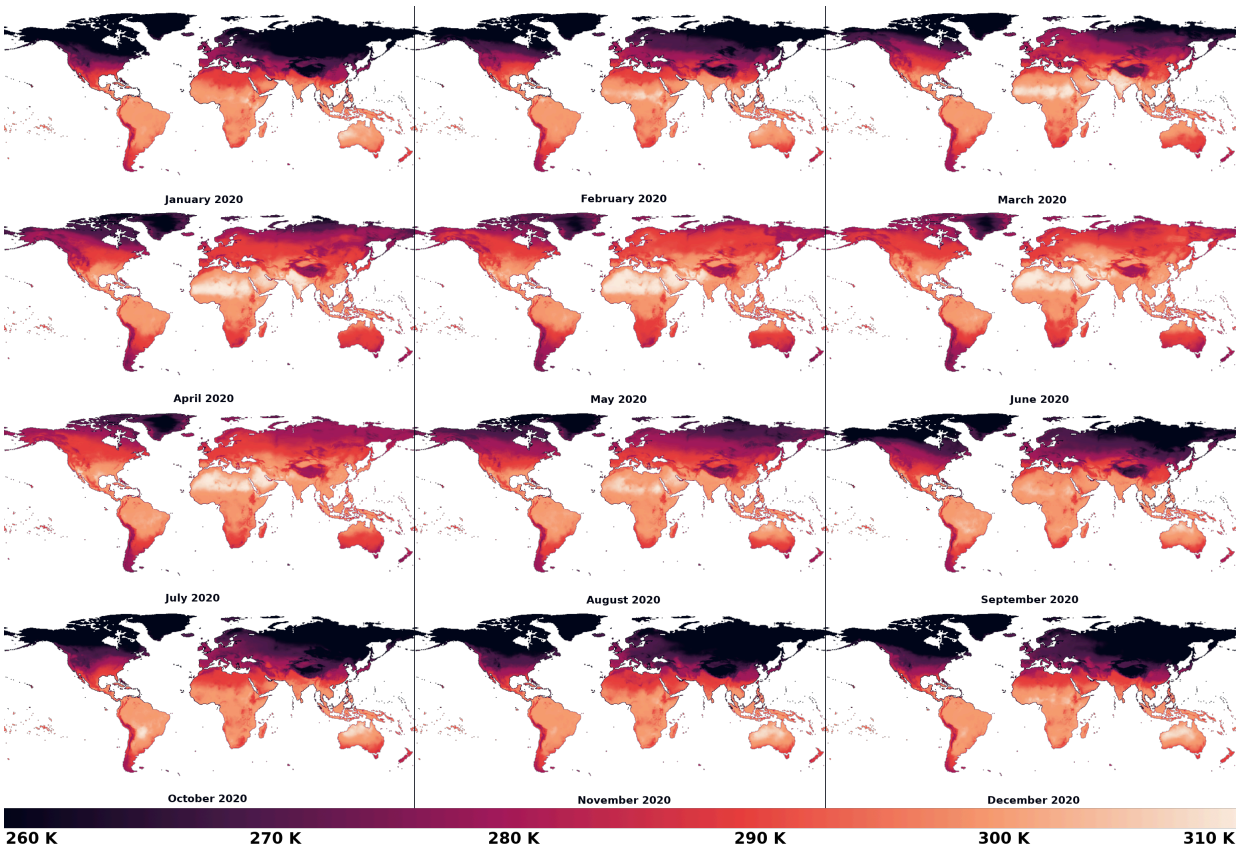

Figure D.15 — Visualization of monthly Land Surface Temperature (average for the month of daily average) for the world (excluding Antarctica) throughout 2020

D.1.6. Reference LST Minimum and Maximum

The following step of the workflow involves computation of reference minimum and maximum LST values which are used in the computation of the TCI from the monthly time series ovtained from the previous step. This step is almost identical to how reference values are computed for the NDVI. The same *PassThrough* process and *AggregateMulti* function are used again, using exactly the same parameters as for the NDVI reference values.

The resulting LST_ref_and_monthly_2020 collection includes the reference maximum (`LST_max`), minimum (`LST_min`), while also carrying over the current year monthly LST value (`LST_monthly`) for the year 2020.

```
{
   "id": "LST_ref_and_monthly_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
LST_monthly",
            "properties":
            {
               "LST_max": "AggregateMulti(LST_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "LST_min": "AggregateMulti(LST_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "LST_monthly": "LST_monthly"
            }
         }
      ]
   }
}
```

Listing D.6 — Workflow to generate reference LST minimum and maximum for each month based on years 2017-2020

A request to an OGC API — Coverages endpoint for July 2020 including only the maximum and minimum would be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:LST_ref_and_monthly_2020/coverage?datetime=2020-07&properties=LST_max,LST_min

The following figures illustrate the time series for the maximum and minimum as rendered for each month of 2020 using a request to an OGC API — Maps endpoint. For the maximum in July 2020, a request would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:LST_ref_and_monthly_2020/map?datetime=2020-07

whereas for visualizing the minimum reference value, a different style named `min` was set up, thus making the request:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:LST_ref_and_monthly_2020/styles/min/map?datetime=2020-07

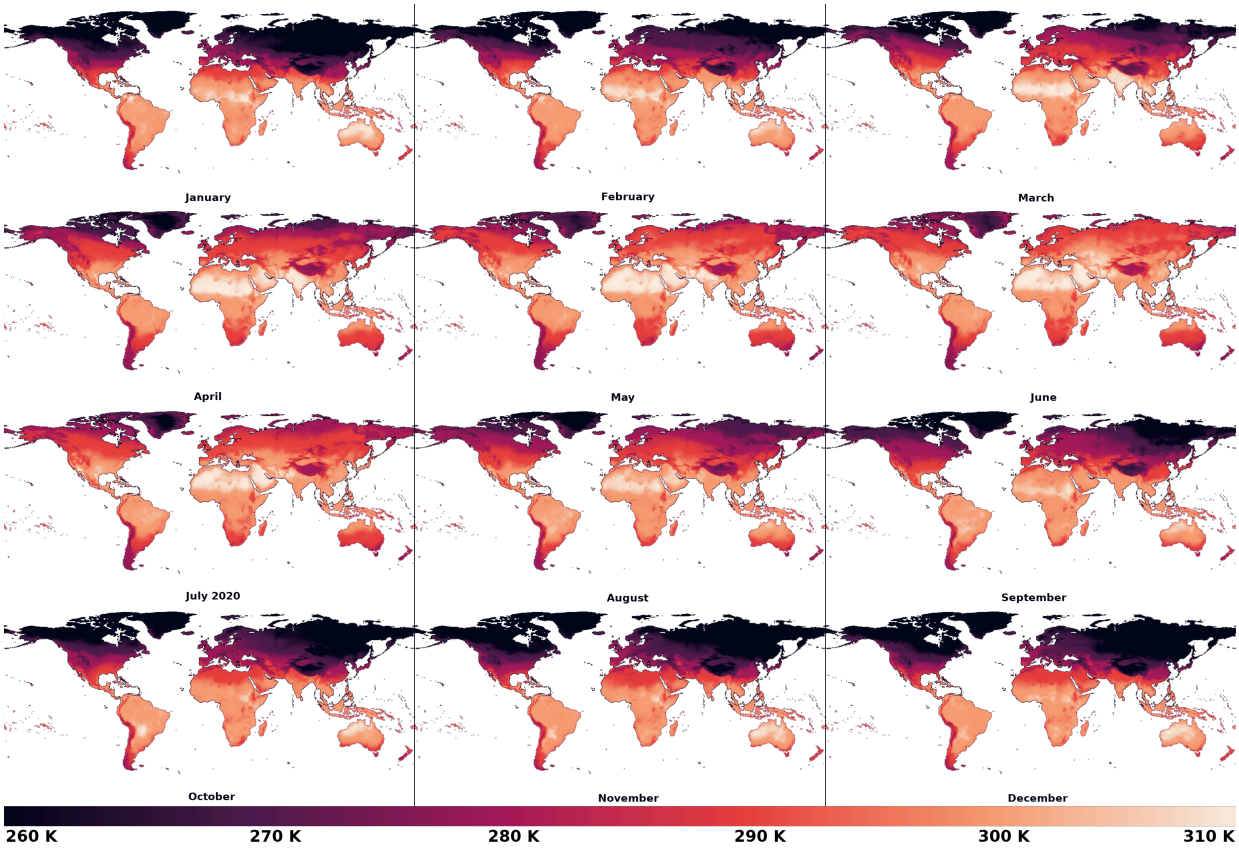

Figure D.16 — Visualization of reference monthly LST Maximum for the world (excluding Antarctica) based on years 2017-2020

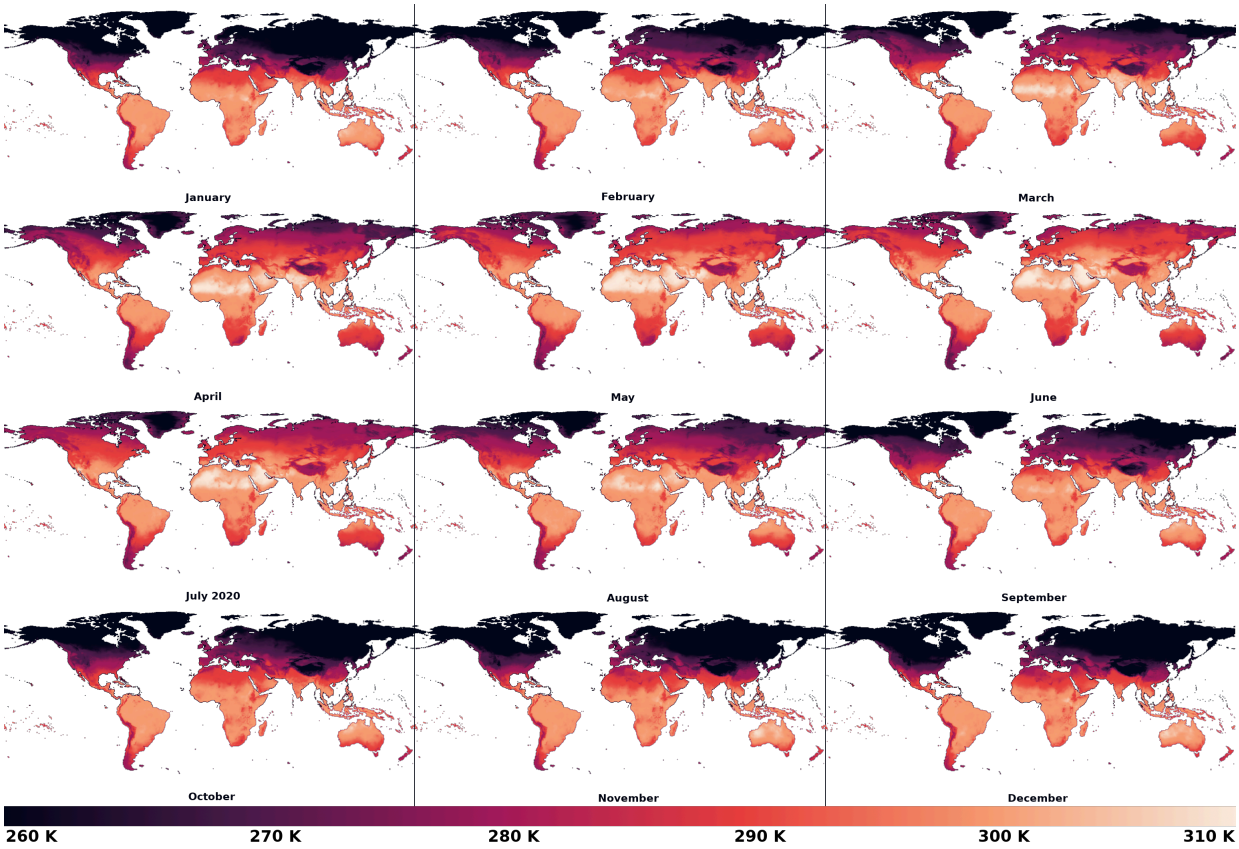

Figure D.17 – Visualization of reference monthly LST Minimum for the world (excluding Antarctica) based on years 2017-2020

Other Testbed 20 participants demonstrated workflows using reference minimum and maximum values for the whole year 2020 for the TCI computations as well. Therefore, Ecere also implemented this approach as a separate set of workflow definitions and virtual collections for the purpose of comparing the workflow output with other participants. The workflow for computing minimum and maximum LST values for the whole year 2020 is shown below.

```
{
   "id": "LST_ref_and_monthly_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
LST_monthly",
            "properties":
            {
               "LST_max": "AggregateMulti(LST_monthly, Max, ('time'), ('P1M'),
('R12/2020-01-01/P1M'), ('2020-01-01'))",
               "LST_min": "AggregateMulti(LST_monthly, Min, ('time'), ('P1M'),
('R12/2020-01-01/P1M'), ('2020-01-01'))",
               "LST_monthly": "LST_monthly"
            }
         }
      ]
   }
```

```
}
```

Listing D.7 — Workflow to generate reference LST minimum and maximum based on entire year 2020 (for comparison with other participants workflows)

These reference values were also made available as virtual collections at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:WholeYearRef_2020:LST_ref_and_monthly_2020, and illustrated below as rendered by the OGC API — Maps implementation.

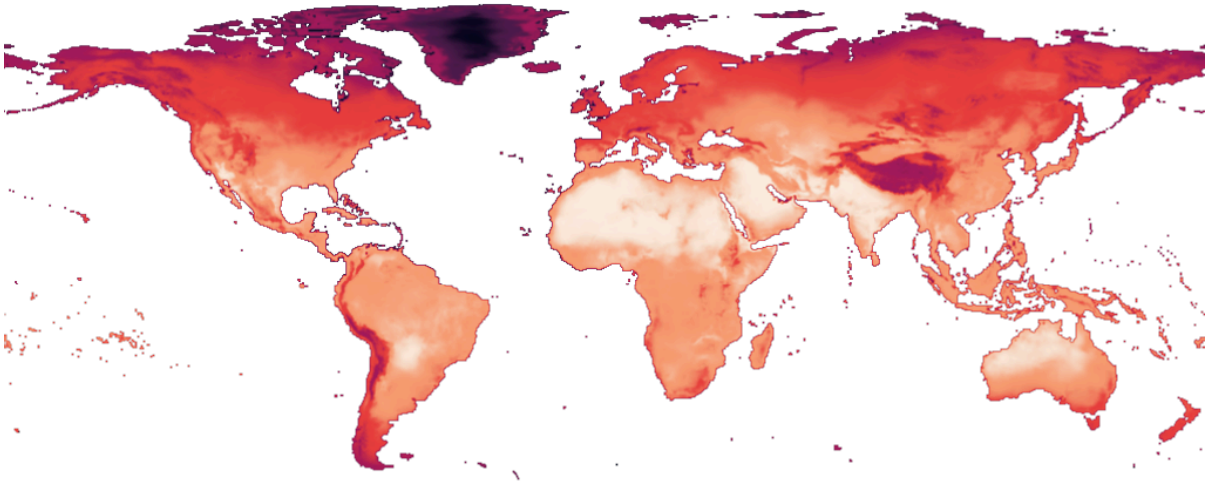

Figure D.18 — Visualization of reference LST Maximum for the world (excluding Antarctica) based on whole year 2020

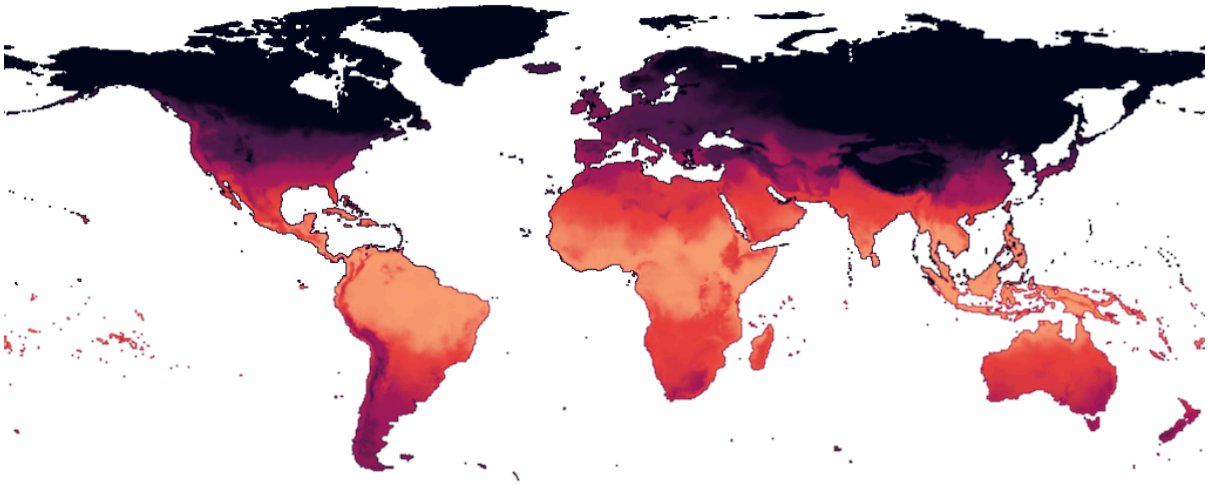

Figure D.19 — Visualization of reference LST Minimum for the world (excluding Antarctica) based on whole year 2020

D.1.7. Temperature Condition Index (TCI)

The last step for computing the Temperature Condition Index (VCI) compares the monthly LST values with the reference maximum and minimum, which were assembled together in the

previous step of the workflow. The same *PassThrough* process is used again, and this step is performed using a simple CQL2 arithmetic expression directly reflecting the formula:

$$TCI = 100\frac{LST_{max} - LST_{monthly}}{LST_{max} - LST_{min}} \tag{D.2}$$

without the need for aggregation or other special function.

Unlike the VCI, an LST value corresponding to the reference minimum will yield a score of 100, contributing positively to the VHI, whereas an LST value corresponding to the reference minimum will yield a score of 100. This is likely to compensate for the fact that vegetation is expected to have a lower NDVI in colder weather.

```
{
   "id": "TCI_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://maps.gnosis.earth/ogcapi/collections/T20-VHI:
MonthlyRef_2017_2020:LST_ref_and_monthly_2020",
            "properties": {
               "TCI": "100 * (LST_max - LST_monthly) / (LST_max - LST_min)"
            }
         }
      ]
   }
}
```

Listing D.8 — Workflow to generate reference Temperature Condition Index (TCI) (here using 2017-2020 monthly reference minimum and maximum)

A resulting virtual collection was made available at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:TCI_2020.

A request to an OGC API — Coverages endpoint for July 2020 would be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:TCI_2020/coverage?datetime=2020-07

The following figures illustrate the TCI time series as rendered for each month of 2020 using a request to an OGC API — Maps endpoint. For July 2020, a corresponding request would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:TCI_2020/map?datetime=2020-07

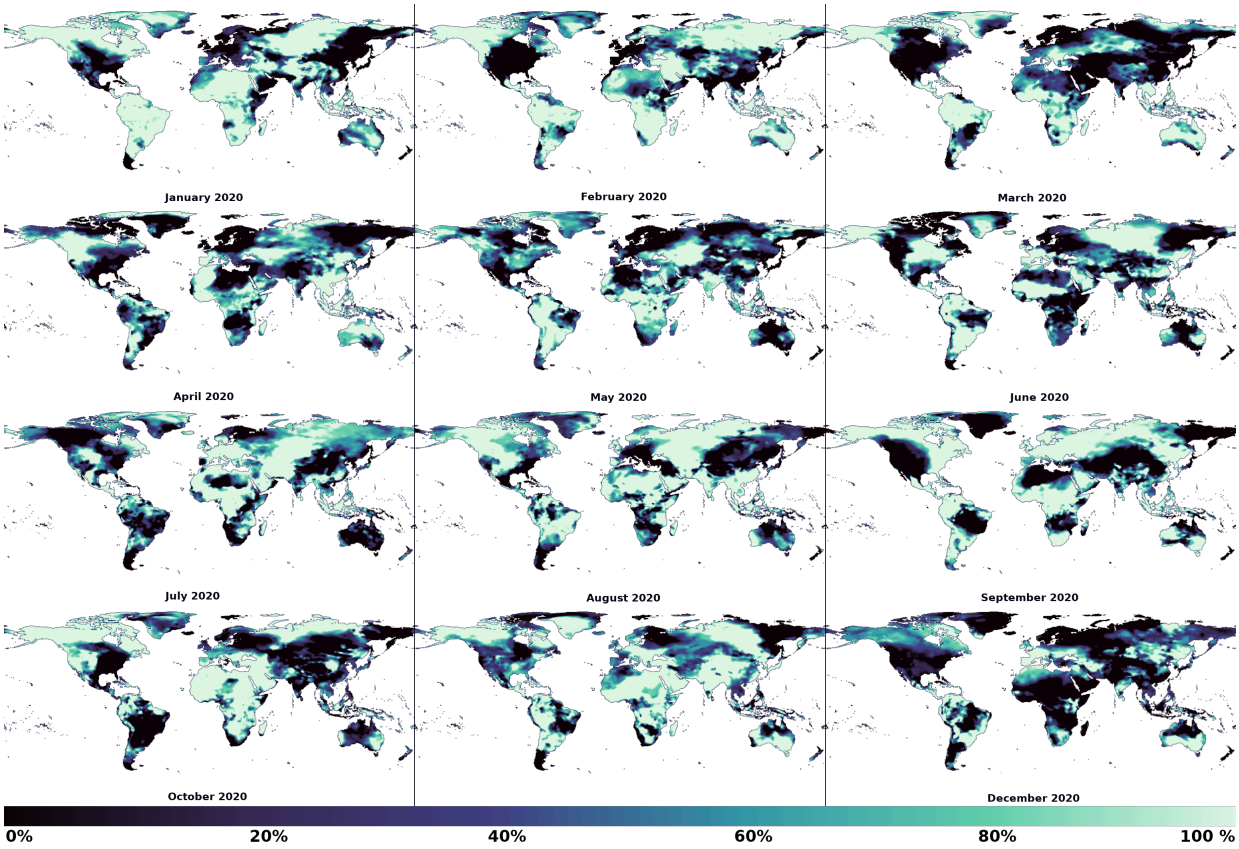

Figure D.20 — Visualization of Temperature Condition Index (TCI) for the world (excluding Antarctica) throughout 2020 using the corresponding months of 2017-2020 for reference minimum and maximum

D.1.8. Vegetation Health Index (VHI)

The final step of the VHI workflow combines the VCI and TCI computed in the previous steps with weighing factors. This operation illustrates the ability to integrate information originating from separate data sources — in this case satellite imagery with the results of climate simulation. The same *PassThrough* process is used again, and this step is performed using a simple CQL2 arithmetic expression directly reflecting the simple formula using equal weighting factors for the VCI (`α = 0.5`) and TCI (`β = 1 - α = 0.5`):

$$VHI = \alpha VCI + (1-\alpha)TCI \tag{D.3}$$

Both the VCI and TCI are used as collection inputs to the *PassThrough* process.

One particularity of this workflow definition seen below is that an outer *PassThrough* process is invoked, using the output of the first *PassThrough* process nested within as an input. Although this should ideally not be necessary, since some functionality related to the implementation of output field modifiers as defined in the OGC API — Processes — Part 3: Workflows draft Standard was not yet completed at the end of the Testbed (March 2025), this nested process

enabled performing this multi-source data integration by applying the expression as an input field modifier.

As discussed during the Testbed GDC initiative, the line between what constitutes an input field modifier vs. an output field modifier is intentionally blurred when the output of one process is used as an input to another. This allows workflows definitions to be executed in the most efficient way possible. For example, a remote nested process accessed executed on-demand using a virtual output collection mechanism through an implementation of the OGC API Standard supporting CQL2 expressions to derive or filter fields can directly be passed this expression, allowing the calculations to be done close to the data at the output level. Alternatively, if the remote data access API does not support this capability, an implementation can perform these additional steps locally, applying them at the level of the input to the outer process.

```
{
   "id": "VHI_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
            "inputs": {
               "data": [
                  { "collection": "https://maps.gnosis.earth/ogcapi/collections/
T20-VHI:MonthlyRef_2017_2020:VCI_2020" },
                  { "collection": "https://maps.gnosis.earth/ogcapi/collections/
T20-VHI:MonthlyRef_2017_2020:TCI_2020" }
               ]
            },
            "properties": { "VHI": "0.5 * VCI + 0.5 * TCI" }
         }
      ]
   }
}
```

Listing D.9 — Workflow to generate final Vegetation Health Index (VHI) (here using 2017-2020 monthly reference minimum and maximum)

A resulting virtual collection was made available at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020.

A request to an OGC API — Coverages endpoint for July 2020 for the whole of Slovenia could be as follows:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/coverage?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

The following figures illustrate the VHI time series as rendered for each month of 2020 using a request to an OGC API — Maps endpoint for the whole of Slovenia as well as for Ljubljana. For July 2020 for the whole of Slovenia, a corresponding request would be:

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/map?subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)&datetime=2020-07

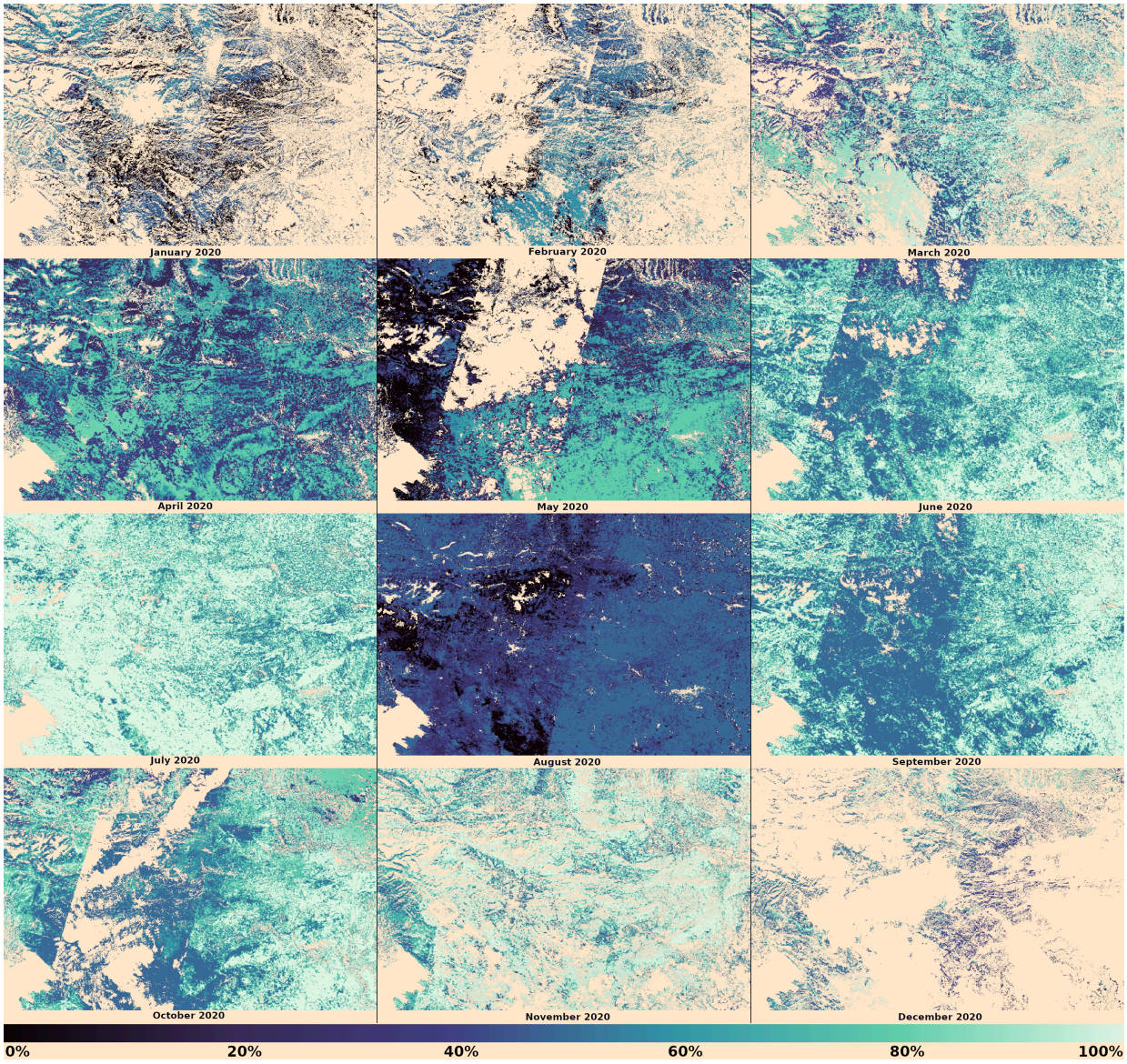

Figure D.21 — Visualization of Vegetation Health Index (VHI) for Slovenia (`subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015)`) throughout 2020 using the corresponding months of 2017-2020 for reference minimum and maximum

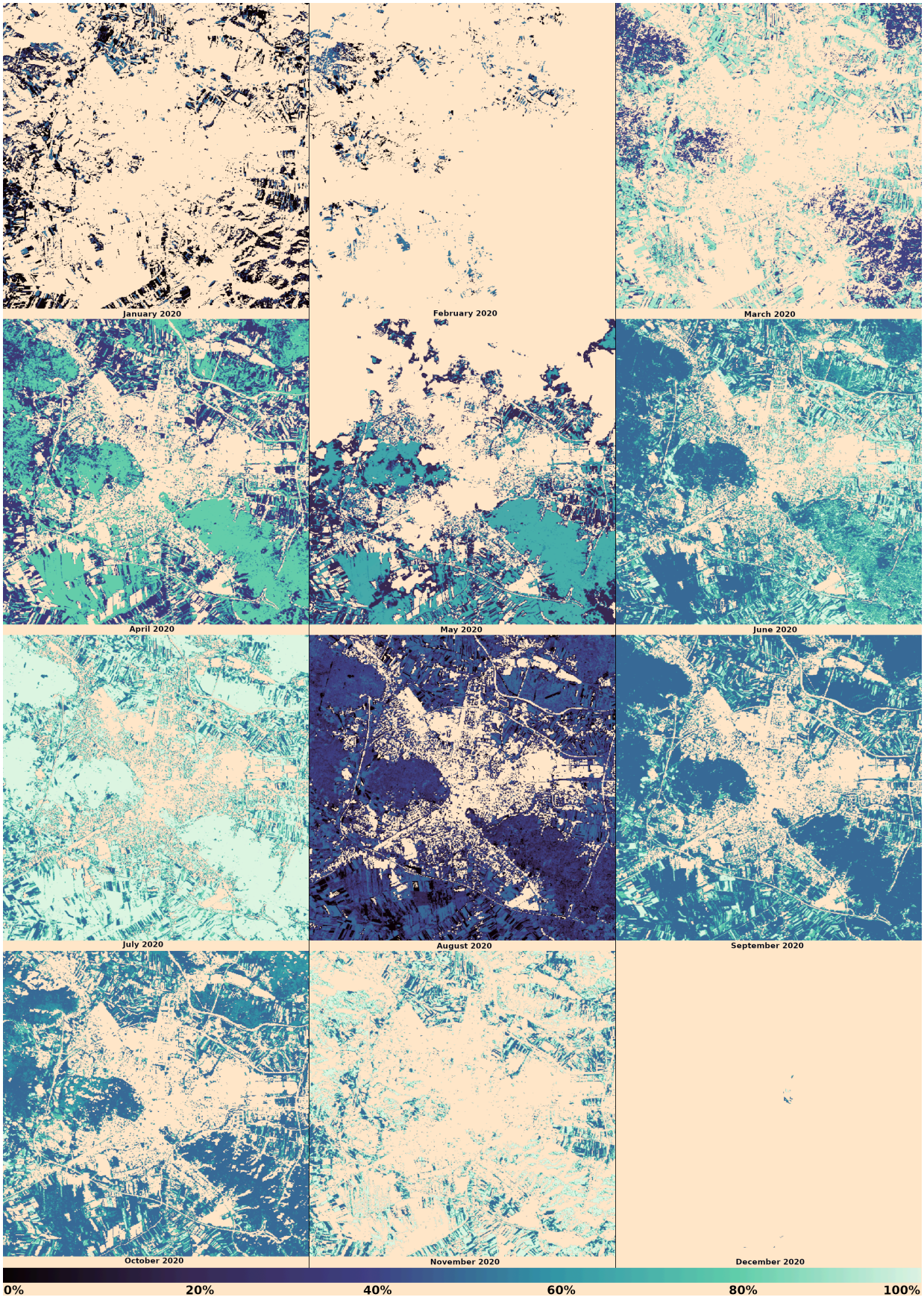

Figure D.22 — Visualization of Vegetation Health Index (VHI) for Ljubljana (`subset=Lat(46.005356:46.108536),Lon(14.431717:14.579785)`) throughout 2020 using the corresponding months of 2017-2020 for reference minimum and maximum

Figure D.23 — Visualization of Vegetation Health Index (VHI) for Slovenia with a CQL2 filter cutting out a low-resolution polygon of the country (map request) for July 2020 using July 2017-2020 for reference minimum and maximum

A virtual collection is also available based on the VCI and TCI computed using the whole year 2020 as reference minimum and maximum for the purpose of comparing the workflow output with other participants at https://maps.gnosis.earth/ogcapi/collections/T20-VHI:WholeYearRef_2020:VHI_2020.

The following figure illustrates the output of this pseudo-VHI workflow rendered using OGC API — Maps for July 2020, for Northern Europe.

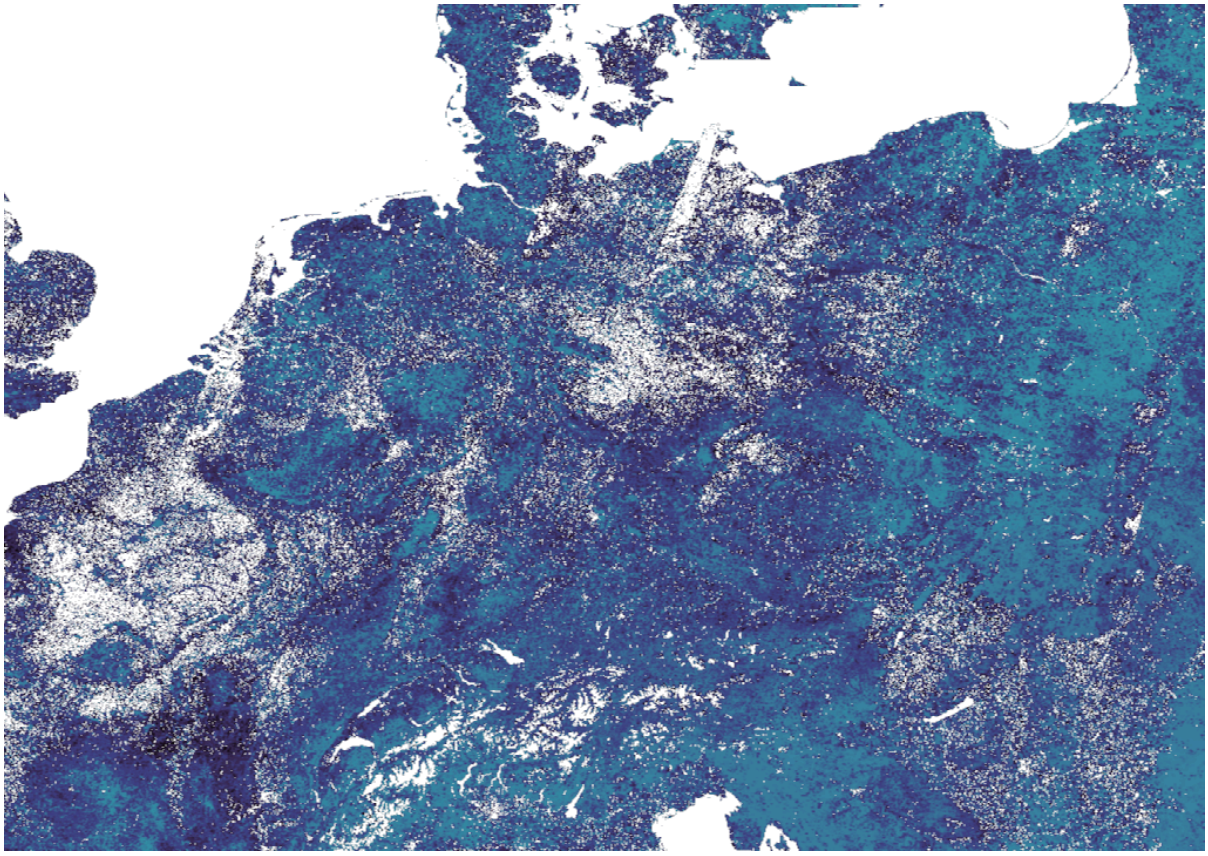

Figure D.24 — Visualization of pseudo-Vegetation Health Index (VHI) (for comparison with other participants workflows) for Northern Europe in July 2020 using entire year 2020 for reference minimum and maximum

D.1.9. Alternative definitions of the workflow

Distributed workflow

The VHI workflow was also successfully tested as a distributed workflow with different steps performed by multiple deployments of the GNOSIS Map Server at different physical locations. A local development machine in Canada performed the later steps of the workflow, while the demonstration server in Germany generated the sentinel-2 monthly time series. Communication between the two instances of the GNOSIS Map Server and parallelism was achieved using an implementation of the OGC API — Tiles Standard. Requests werde made for tiled coverage data over a time interval and using the GNOSIS Map Tiles endpoint, which already supports time intervals, unlike the GeoTIFF output. Alternatively, netCDF could be used once support for netCDF output is implemented in GNOSIS Map Server. For OGC API — DGGS endpoint data requests, DGGS-(UB)JSON which supports additional dimensions beyond those of the Discrete Global Grid Reference System (DGGRS) — would also be perfectly suitable for such tasks. DGGS-(UB)JSON would also be suitable with implementations of the OGC API — Tiles for 2D Tile Matrix Sets (2DTMS) Standard corresponding to an axis-aligned DGGRS, such as the GNOSISGlobalGrid and the proposed ISEA9R 2DTMS. Similar Technology Integration Experiments (TIEs) between GeoDataCube API deployments — where individual steps of a larger

workflow are performed by implementations provided by different participants — would be particularly interesting for future initiatives to further demonstrate interoperability.

Workflow as a single execution request referencing STAC APIs

Instead of specifying each step of the workflow as a separate execution request and referencing a virtual collection produced by the previous steps, defining the entire workflow as a single execution request should be possible. The input collection could also directly reference the source Sentinel-2 and CMIP6 STAC metadata collections. This potential alternative workflow, which was not yet working at the end of the Testbed 20 initiative (Feb. 2025), is shown below. A POST operation to `/collections` with this payload would ideally result in a virtual collection from which OGC API — Coverages, DGGS, Tiles or Maps requests can be performed to request data for an area, resolution and time of interest.

```
{
   "id": "VHI_2020",
   "process": "https://maps.gnosis.earth/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "collection": "https://earth-search.aws.element84.com/v1/collections/
sentinel-2-l2a",
            "filter": "SCL = 4",
            "properties": {
               "NDVI_monthly": "AggregateMulti((B08 - B04)/(B08 + B04), Max,
('time'), ('P1M'))",
               "NDVI_max": "AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "NDVI_min": "AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "VCI": "100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)"
            }
         },
         {
            "collection": "https://planetarycomputer.microsoft.com/api/stac/v1/
collections/nasa-nex-gddp-cmip6",
            "filter": "cmip6:model = 'GFDL-ESM4' and cmip6:scenario = 'ssp585'",
            "properties": {
               "LST_monthly": "AggregateMulti(tas, Avg, ('time'), ('P1M'))" },
               "LST_max": "AggregateMulti(LST_monthly, Max, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "LST_min": "AggregateMulti(LST_monthly, Min, ('time'), ('P1M'),
('R4/2017-01-01/P1Y'), ('2020-01-01'))",
               "TCI": "100 * (LST_max - LST_monthly) / (LST_max - LST_min)"
            }
         }
      ]
   },
   "properties": { "VHI": "0.5 * VCI + 0.5 * TCI" }
}
```

Listing D.10 — Alternative workflow for generating a Vegetation Health Index (VHI) using the *PassThrough* process, CQL2 and *AggregateMulti()* function, defined as a single execution request and directly referencing the source STAC metadata

The main challenges associated with getting this workflow definition to work as expected are as follows.

- Setting up virtual collections for the intermediate steps provides an opportunity to configure data caches. This is particularly important for the first steps of generating monthly time series, especially for the high resolution Sentinel-2 data where thousands of granules are involved, even for relatively small regions. A caching configuration in the workflow definition and/or smarter automatic caching mechanisms in implementations could be used to work around this problem.

- The ability to directly reference derived fields (properties) computed at the same processing input level greatly simplifies the workflow, requiring a single *PassThrough* process instead of seven. However, this capability would make it more difficult to implement and even more difficult to configure caching for each step and is not yet implemented in the GNOSIS Map Server (Feb. 2025).

- The implementation of OGC API — Processes — Part 3: Workflows Input Collections Standard needs client support for the STAC API and data access from assets referenced by STAC items as both Cloud Optimized GeoTIFF (for Sentinel-2) and netCDF (for CMIP6). This is partially implemented in the GNOSIS Map Server which can access a collection of STAC items and GeoTIFF assets (using range requests) as a source data store. However, work remains for supporting netCDF assets as well as for directly accessing a STAC API (as opposed to working off a local relational database of STAC items).

- Despite the STAC API being theoretically based on the OGC API — Features Standard, the STAC API deployments for the Sentinel-2 and CMIP6 metadata collections did not work as expected by the GNOSIS SDK's client implementation of OGC API — Features — Part 1 and Part 3. At the same time, the client functionality has no issue accessing OGC API — Features implementations certified compliant with the Standard. This prevented further attempts on making progress towards directly accessing STAC APIs, however a work around was used to by using local STAC metadata instead. Additional experimentation and details of conformance issues for these two deployments of the STAC API are described below.

- The netCDF assets referenced from the CMIP6 STAC metadata requires special access tokens which greatly complicate access in generic clients, significantly reducing the accessibility and interoperability of this data.

Workflow embedded within an OGC API — Coverages request

Since the VHI workflow relies on a single generic *PassThrough* process which does not actually perform any processing other than combining inputs, the workflow could alternatively be embedded directly as parameters to a request to OGC API — Coverages endpoint(s) for a large area of interest using the proposed Part 2: Filtering, Deriving and Aggregating Fields extension based on CQL2. A request for the Slovenia area of interest for July 2020 could then look as shown below. Since requests to an OGC API — Coverages endpoint must originate from a collection, the Sentinel2-l2a collection on the GNOSIS Map Server deployment is used here rather than directly referencing the STAC metadata. A new virtual collection could also be set up instead for the purpose of establishing an origin collection supporting an implementation of the OGC API — Coverages Standard on the server by using the *PassThrough* process and

specifying the Sentinel-2 STAC collection as input, without any additional filtering or deriving logic. Support for the following request was not yet implemented during the initiative and would necessitate future work.

```
GET https://maps.gnosis.earth/ogcapi/collections/sentinel2-l2a/coverage?
   subset=Lat(45.4236367:46.8639623),Lon(13.3652612:16.5153015),time=2020-07&
   joinCollections=https://planetarycomputer.microsoft.com/api/stac/v1/
collections/nasa-nex-gddp-cmip6&
   filter=SCL = 4 and cmip6:model = 'GFDL-ESM4' and cmip6:scenario = 'ssp585'&
   alias[NDVI_monthly]=AggregateMulti((B08 - B04)/(B08 + B04), Max, ('time'),
('P1M'))&
   alias[NDVI_max]=AggregateMulti(NDVI_monthly, Max, ('time'), ('P1M'), ('R4/
2017-01-01/P1Y'), ('2020-01-01'))&
   alias[NDVI_min]=AggregateMulti(NDVI_monthly, Min, ('time'), ('P1M'), ('R4/
2017-01-01/P1Y'), ('2020-01-01'))&
   alias[VCI]=100 * (NDVI_monthly - NDVI_min) / (NDVI_max - NDVI_min)&
   alias[LST_monthly]=AggregateMulti(tas, Avg, ('time'), ('P1M'))&
   alias[LST_max]=AggregateMulti(LST_monthly, Max, ('time'), ('P1M'), ('R4/2017-
01-01/P1Y'), ('2020-01-01'))&
   alias[LST_min]=AggregateMulti(LST_monthly, Min, ('time'), ('P1M'), ('R4/2017-
01-01/P1Y'), ('2020-01-01'))&
   alias[TCI]=100 * (LST_max - LST_monthly) / (LST_max - LST_min)&
   alias[VHI]=0.5 * VCI + 0.5 * TCI&
   properties=VHI
```

Listing D.11 — Alternative workflow for generating a Vegetation Health Index (VHI) embedded within an OGC API - Coverages GET request for Slovenia in July 2020

However, this approach requires re-submitting the workflow definition with each data access request and is therefore not well suited for multiple requests for different areas, times and resolutions of interest. Setting up a virtual collection, which allows the use of implementations of the OGC API — Tiles or OGC API — DGGS Standards, is preferable in those cases. This allows the server to validate and perform handshakes with remote collections only once during the initial set up, more easily cache requests, and potentially even preempt future requests based on the previous ones to minimize latency and make efficient use of idle processing times.

D.2. Issues encountered accessing STAC API deployments

Some of the issues encountered trying to access each of the two STAC API deployments for the input collections used in the VHI workflow are described below. Additionally, the importance of the availability of Executable Test Suites (ETS) for validating compliance with OGC Standards was raised during the initiative, in particular for OGC API — Features — Part 3: Filtering and OGC Common Query Language (CQL2) which can be implemented by STAC API deployments. The availability of these ETS would greatly facilitate the task of ensuring compliance to these Standards implemented by providers of these GeoDataCubes providing efficient access to large amounts of data.

IMPORTANT

The issues described here are not intended as criticisms of these implementations and deployments. Rather, the goal is to raise awareness of these challenges and encourage their resolution. Addressing these

issues could lead to significant improvements in the interoperability and accessibility of these data cubes, as well as benefit other, less visible implementations and deployments of the STAC API specification and the OGC API — Features Standard.

Furthermore, this discussion underscores the critical role of the OGC's compliance program and the importance of executable test suites (ETSs). Improved conformance among implementations would also reduce the effort required by participants in the OGC Collaborative Solutions and Innovation (COSI) Program, allowing them to make greater progress instead of spending time identifying, testing, and reporting issues in implementations that should already be conformant.

Additionally, developing software workarounds for these issues often results in code that is difficult to maintain and can lead to even greater interoperability problems in the future. This is because specially patched clients may function correctly, while standard clients do not.

D.2.1. Issues encountered with the Earth Search STAC API for the sentinel-2 data

The STAC API deployed at https://earth-search.aws.element84.com/v1 claims conformance to OGC API — Features — Part 1: Core, but not to Part 3: Filtering or CQL2.

Missing JSON API definition

The definition of the STAC API mentioned above is only available as YAML, and not as JSON as usually available from deployments of OGC API — Features. Since the deployment declares conformance to both the `oas30` and `geojson` requirement classes of OGC API — Features, according to Requirement 37 in the GeoJSON requirement class and Requirement 45 in the OpenAPI 3.0 requirement class, the server must support the `application/vnd.oai.openapi+json;version=3.0` media type (a JSON version of the OpenAPI definition) in order to conform to both of these requirements classes.

This issue was reported as Earth Search issue #61.

Errors on `limit` above 267

Requests to this STAC API with a `limit` query parameter above 267 results in an Internal Server Error, which breaks conformance with the OGC API — Features — Part 1: Core Standard.

For example, the following basic request for a maximum of 300 items results in a 500 error:

https://earth-search.aws.element84.com/v1/collections/sentinel-2-l2a/items?limit=300

To conform to Features — Part 1, the server must limit the items to the maximum it can handle instead of failing with an internal error, as specified in Requirement 22 C and Permission 6. In addition, the OpenAPI definition for this deployment actually specifies a maximum value of 10,000. Resolving this issue would greatly improve the interoperability of this STAC API.

This issue was reported as Earth Search issue #60.

A request for 300 items is a very small number of items for the use case of mirroring the STAC metadata locally to avoid further requests using computation time on the server on a regular

basis. Furthermore, a small page size prevents efficient compression, while STAC items are full of mostly redundant information. This is because STAC does not follow a relational model to minimize redundancy in assets for each item. With a page size of 300 items, over a hundred thousand separate requests are necessary to retrieve the entire metadata catalog from this collection.

New experiments during this initiative showed that using `limit=250` results in a successful response, and following the `next` links to additional items seemed to work beyond the 10,000 items limit. If recollection serves right, in the past this resulted in server errors either when the number of matched items was beyond this limit, or when following next links past this limit. Related issues may already have been resolved, making the STAC API significantly more usable directly than assessed in previous attempts — as long as a client somehow limits itself to pages of 250 items for this particular deployment.

A more efficient mechanism to request a large quantity of STAC metadata would be ideal. For example, the entire metadata catalog can be represented in a compressed SQLite database of a few gigabytes. This should require significantly fewer than a hundred thousand separate requests to mirror. This is one aspect which the proposed "Scenes" requirement class for a future revision to the OGC API — Coverages Standard aims to achieve. Alternatively, this could be as simple as HTTP access to a compressed complete collection of STAC items for a monthly baseline, augmented with compressed daily patches for created, removed or updated STAC items since. This approach could be implemented on more affordable object storage instead of using computational power for a dynamic API.

Inability to filter by cloud cover

Although the STAC API defines its own filtering extension — which seems to also be based on CQL2 — this deployment does not implement either, supporting only item-search. This *item-search* extension supports more complex queries (called *full featured queries*) as POST requests, including the option to select which properties to include in the response. During the Testbed it was clarified that not all STAC URIs are resolvable and that the use of the `#fields` anchor in the conformance declaration URIs refers to the STAC fields extension. This overlaps in functionality specified in proposed OGC API — Features — Part 6: Property Selection Standard. The idea of a *search / fields* URI initially gave the wrong impression that it allowed to search items by fields in the 2022 initiatives.

The YAML API definition available directly from the server (visualized in Swagger UI here) confirms that this STAC API supports sorting by cloud cover (`properties.eo:cloud_cover`), in an ascending or descending manner, and include or not particular properties in the response. However, not being able to filter using a maximum allowable cloud cover, which seems like a desirable capability for a catalog of almost forty million granules, and despite `eo:cloud_cover` appearing in a queryables resource.

This issue was reported as Earth Search issue #62.

D.2.2. Issues encountered with the Microsoft Planetary Computer STAC API for the CMIP6 data

The STAC API deployed at https://planetarycomputer.microsoft.com/api/stac/v1/collections/nasa-nex-gddp-cmip6 claims conformance to OGC API — Features Part 1: Core and Part 3: Filtering, as well as to CQL2, as stated on the conformance page.

Potentially caching related issues with `filter` and `datetime` query parameters

Issuing subsequent requests with a different value for `filter` or `datetime` seems to return the result of the initial request. Potentially responses are being cached irrespective of the values for the query parameters.

The following request which attempts to return all items filtered on `cmpi6:model='GFDL-ESM4'` and `cmip6:scenario='ssp585'` was found not to work as expected:

```
https://planetarycomputer.microsoft.com/api/stac/v1/collections/nasa-nex-gddp-
cmip6/items?
   filter-lang=cql2-text&
   filter=cmpi6:model='GFDL-ESM4' and cmip6:scenario='ssp585'
```

Listing

despite the `cmip6:model` and `cmip6:scenario` being declared as queryables.

The response has a 200 HTTP result code indicating a successful response and no issue with the request parameters. However, the response also includes items for models other than *GFDL-ESM4* and scenarios other than *ssp585*, indicating that the filtering does not seem to be working, with no indication to the client that this is the case.

Additionally, comparing the responses of this request with and without the `filter` parameter reveals strange disparities in the responses around a degree symbol character.

Similar issues were also encountered relating to the `datetime` parameter defined in OGC API — Features — Part 1: Core.

For example, the following subsequent requests:

```
https://planetarycomputer.microsoft.com/api/stac/v1/collections/nasa-nex-gddp-
cmip6/items?
   datetime=2020-01-01T00:00:00Z/2020-01-31T00:00:00Z

https://planetarycomputer.microsoft.com/api/stac/v1/collections/nasa-nex-gddp-
cmip6/items?
   datetime=2021-01-01T00:00:00Z/2021-01-31T00:00:00Z
```

Listing

would result in a response to the latter including items for 2020.

This made it impossible to use the STAC API directly, instead relying on manually filtering the data locally.

This seems to be related to the planetary computer APIs issue #108.

Need for special access tokens

The additional need for special access tokens to retrieve the netCDF assets mentioned previously is a further hindrance to being able to use this STAC API directly as a GeoDataCube source.

D.3. Accessing data and processing results with OGC API — DGGS

Through an implementation of the OGC API — DGGS Standard, the GNOSIS Map Server provides access to data, whether static or the result of processing workflow computations. The foundation for this OGC DGGS API Standard is the OGC Topic 21 / ISO 19170 Abstract Specification. For example, the concept of a Discrete Global Grid Reference System (DGGRS).

Implementations of the OGC DGGS API Standard enables clients to either query lists of DGGRS zones matching a particular query, return data for a particular DGGRS zone, or both. The GNOSIS Map Server implementation supports the three DGGRSs described in Annex B of the DGGS API Standard: *GNOSISGlobalGrid*, *ISEA9R* and *ISEA3H*.

D.3.1. Zone Queries

The following request and JSON response is an example of Zone Query from the VHI workflow output for all ISEA3H DGGRS zones at refinement level 8 (~7773.97 km²) within a rough polygon of Slovenia.

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/dggs/ISEA3H/zones?compact-zones=false&zone-level=8&filter=S_INTERSECTS(rec.geom,POLYGON16.377667 46.652857,16.396636 46.659123,...)

A request and JSON response for an equivalent query at a refinement level of 14 instead (~10.66 km² zones) follows.

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/dggs/ISEA3H/zones?compact-zones=false&zone-level=14&filter=S_INTERSECTS(rec.geom,POLYGON16.377667 46.652857,16.396636 46.659123,...)


```
{
   "zones" : [
      "E4-374-A", "E4-3C4-A", "E4-3C5-A", "E4-3C6-A", "E4-416-A",
      "E4-417-A", "E4-418-A", "E4-468-A", "E4-469-A", "E4-46A-A", "E4-4BA-A"
   ],
   "links" : [
      {
         "rel" : "http://www.opengis.net/def/rel/ogc/1.0/dggrs",
```

```
         "title" : "ISEA3H DGGRS for VHI_2020",
         "href" : "/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/
dggs/ISEA3H"
      },
      {
         "rel" : "http://www.opengis.net/def/rel/ogc/1.0/dggrs-definition",
         "title" : "ISEA3H DGGRS definition",
         "href" : "/ogcapi/dggrs/ISEA3H"
      },
      {
         "rel" : "http://www.opengis.net/def/rel/ogc/1.0/geodata",
         "href" : "/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020"
      }
   ],
   "linkTemplates" : [
      {
         "rel" : "http://www.opengis.net/def/rel/ogc/1.0/dggrs-zone-data",
         "title" : "ISEA3H data for T20-VHI:MonthlyRef_2017_2020:VHI_2020",
         "uriTemplate" : "/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_
2020/dggs/ISEA3H/zones/{zoneId}/data"
      }
   ]
}
```


Listing D.12 — JSON response for a non-compact zone query at level 8 for VHI workflow intersecting a low-resolution polygon of Slovenia returning 11 zones

In addition to a JSON representation and a compact 64-bit integer binary representation, the implementation also supports returning a GeoJSON representation of those zones which includes the zone geometry. This is not intended for practical clients, but for demonstration purposes. For example visualizing these zones using the geojson.io application, as illustrated below.

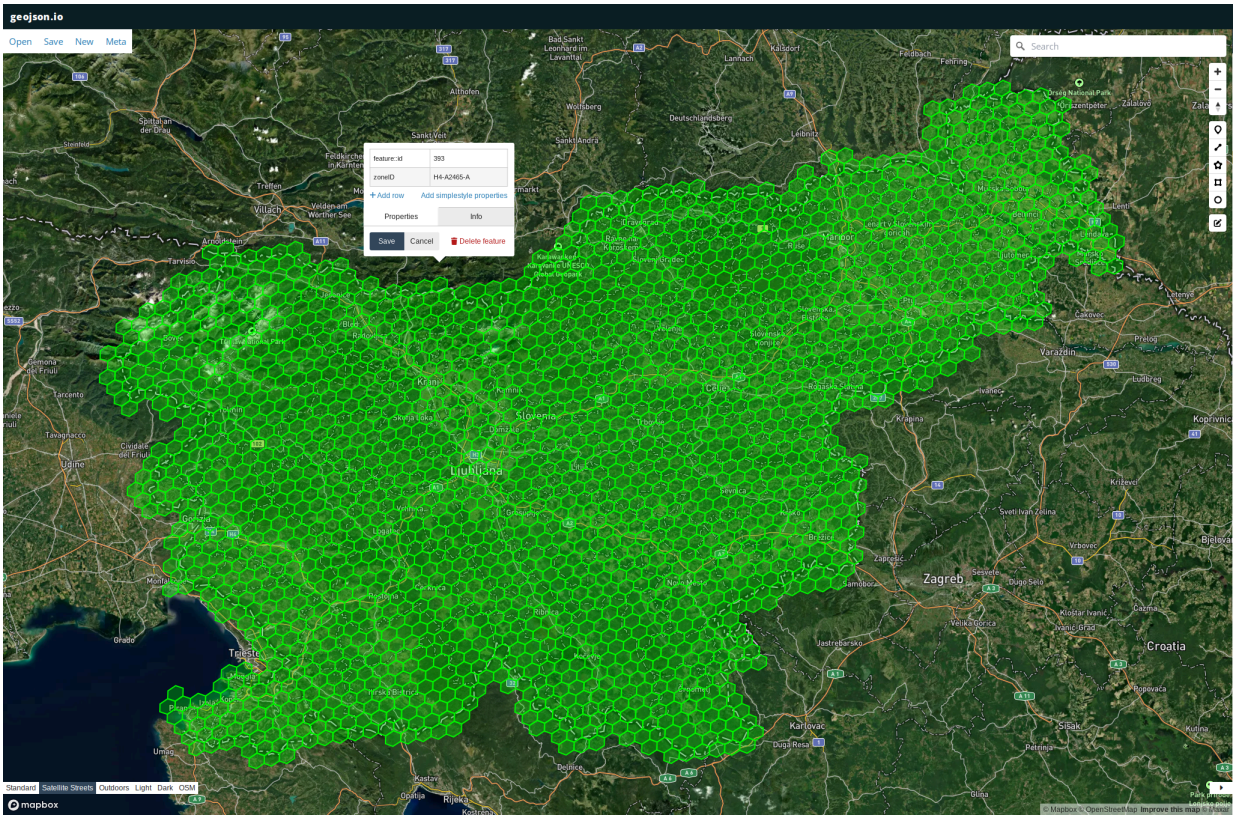

Figure D.25 — Visualization of the results of a GeoJSON zone query for ISEA3H DGGRS zones at level 14 within a rough polygon of Slovenia

D.3.2. Zone Information

The OGC API — DGGS Standard's core requirements class also specifies a zone information resource for a particular `{zoneId}`. This endpoint is not intended for practical clients, as clients should ideally understand the selected DGGRS built into them, such as using a local library implementing support for that DGGRS. Instead, this endpoint is useful for demonstration, and to explore data through a Web browser if an HTML representation is provided.

An example request and response for an HTML representation of information for ISEA3H zone H4-A825C-A for the VHI workflow follows.

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/dggs/ISEA3H/zones/H4-A825C-A

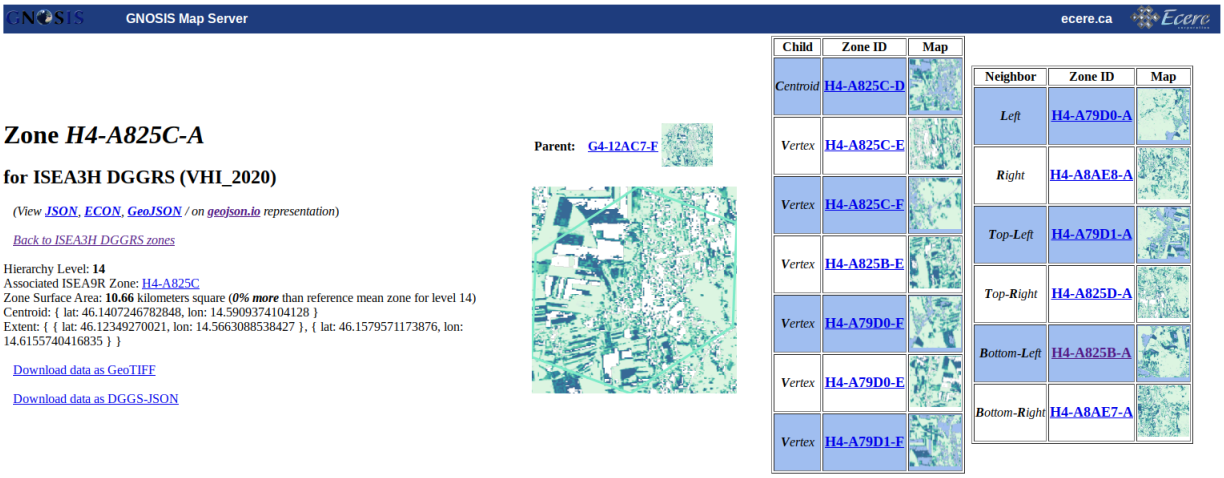

Figure D.26 — HTML representation of information for ISEA3H zone H4-A825C-A for VHI workflow

D.3.3. Zone Data

The Zone Data capability defined in the OGC API — DGGS Standard is conceptually very similar to the ability to retrieve data via an OGC API — Tiles endpoint. However, unlike 2D Tile Matrix Sets (2DTMS), DGGRSs are not restricted to rectangular tiles. This enables the use of hexagonal grids such as ISEA3H, a Discrete Global Grid Hierarchy (DGGH) with a refinement ratio of 3 based on the Icosahedral Snyder Equal Area (ISEA) projection. Because 2D Tile Matrix Sets can be defined for ISEA9R and GNOSISGlobalGrid, the GNOSIS Map Server also supports retrieving tiled data using an implementation of the OGC API — Tiles Standard for these 2D Tile Matrix Sets corresponding to DGGRSs. However, no 2DTMS can be defined for ISEA3H since zones are non-rectangular and their edges are not axis-aligned.

Implementations supporting the “Custom Depths” requirement class also allow clients to control the depth at which values are returned for the zone for which data is being requested with the `zone-depth` parameter, instead of the typical fixed depth of 2DTMS tiles (often 256×256 `tileWidth` and `tileHeight`).

While requirement classes for traditional geospatial formats such as GeoTIFF and netCDF are defined in the candidate Standard, these formats are not well suited to encode non-rectilinear data. For this reason, DGGS-JSON (as well as a binary equivalent DGGS-UBJSON based on UBJSON) is introduced for raster data. This format rely on a deterministic order of sub-zones (zones whose geometry is at least partially contained within the geometry of the parent zone for which data is being encoded), which is understood by both server/producer and client/reader/viewer.

Similarly, while vector data can be returned in GeoJSON form, the draft OGC DGGS-JSON-FG Standard extends the OGC Features & Geometry JSON (JSON-FG) Standard with a `"dggsPlace"` property supporting the encoding of vector coordinates as local sub-zone indices or global zone identifiers to return geometry in a DGGS-optimized format quantized to sub-zones of the parent zone for which data is being requested.

An example request and response for a DGGS-JSON representation of the data and an associated visualization (in ISEA projection) for ISEA3H zone *H4-A825C-A* (in the town of Domžale) from the VHI workflow output at a custom depth of 4 (91 values) follows.

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/dggs/ISEA3H/zones/H4-A825C-A/data.json?zone-depth=4&datetime=2020-07


```
{
   "dggrs" : "[ogc-dggrs:ISEA3H]",
   "zoneId" : "H4-A825C-A",
   "depths" : [ 4 ],
   "representedValue" : "centroid",
   "values" : {
      "VHI" : [
         {
            "depth" : 4,
            "shape" : { "count" : 91, "subZones" : 91 },
            "data" : [
                70.3846893310547, 50.0, 85.421630859375, 77.203353881836, 100.0,
47.401439666748, 28.0530948638916,
                null, 20.915657043457, 54.1106643676758, 46.1550483703613,
25.3251304626465, 100.0, 64.8224029541016,
                5.2464723587036, null, 53.848201751709, 99.1848526000977,
87.6873474121094, 99.616180419922, 92.522605895996,
                50.0, 84.2833557128906, 79.224349975586, 53.0078125, 50.0,
88.277442932129, 64.2528839111328,
                14.5910415649414, 88.5231323242188, 39.7393074035645,
68.5640563964844, 9.2161979675293, 93.801612854004, 92.7761611938477,
                78.2038192749023, 90.2332077026367, 45.3004417419434, 100.0,
95.8563690185547, 68.007568359375, 50.0235900878906,
                78.9492874145508, 100.0, 100.0, 39.1394233703613,
18.180597305298, 97.452896118164, 100.0,
                96.6152267456055, 90.0098876953125, 53.3945655822754,
89.0471725463867, 97.2035903930664, 49.3408088684082, 8.99792671203613,
                4.78313064575195, 100.0, 100.0, 100.0, 100.0, 79.5797805786133,
50.0,
                64.0498275756836, 47.2993087768555, 78.3489990234375,
19.2056884765625, 100.0, 100.0, 89.2680435180664,
                94.0335998535156, 80.3731842041016, 52.435214996338,
70.2813568115234, 51.3188285827637, 65.8550567626953, 100.0,
                91.2951889038086, 76.3256378173828, 56.337043762207,
94.9346237182617, 15.6606769561768, 69.9027404785156, 55.2884140014648,
                100.0, 95.680519104004, 71.243911743164, 93.3478622436523,
78.4265213012695, 72.91796875, 64.0775299072266
            ]
         }
      ]
   }
}
```


Listing D.13 — DGGS-JSON representation of zone data for ISEA3H zone H4-A825C-A at a depth of 4 for July 2020 from the output of the VHI workflow

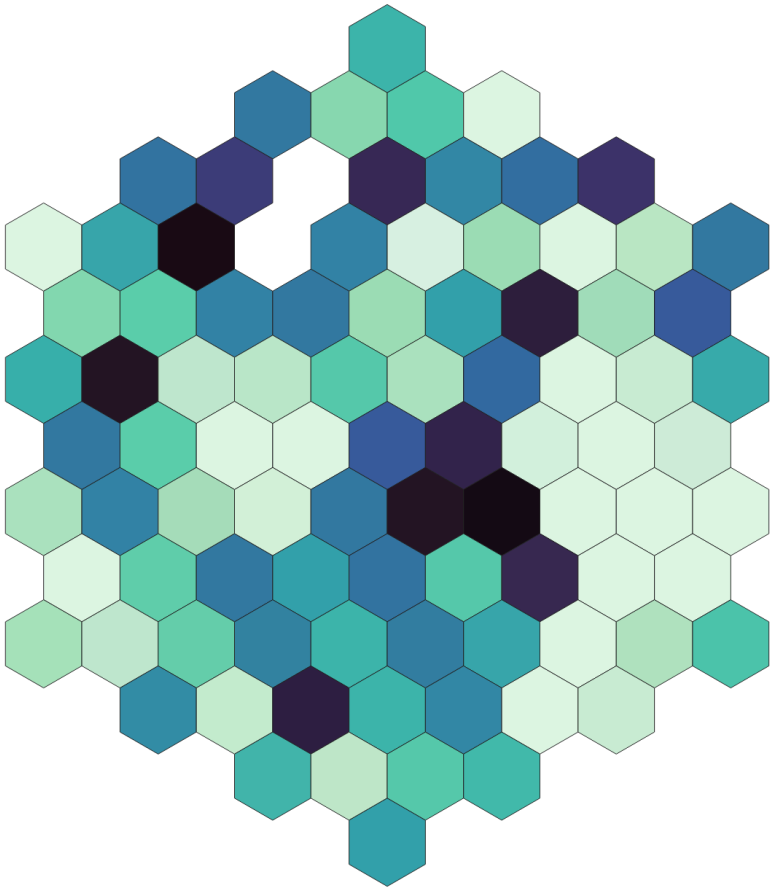

Figure D.27 — Visualization of the results of a DGGS-JSON data query for ISEA3H zone H4-A825C-A (level 14) and July 2020 at a custom depth of 4 from the output of the VHI workflow

A request and visualization of DGGS-JSON data for the same zone at a custom depth of 9 (19,927 values) is also shown below.

https://maps.gnosis.earth/ogcapi/collections/T20-VHI:MonthlyRef_2017_2020:VHI_2020/dggs/ISEA3H/zones/H4-A825C-A/data.json?zone-depth=9&datetime=2020-07

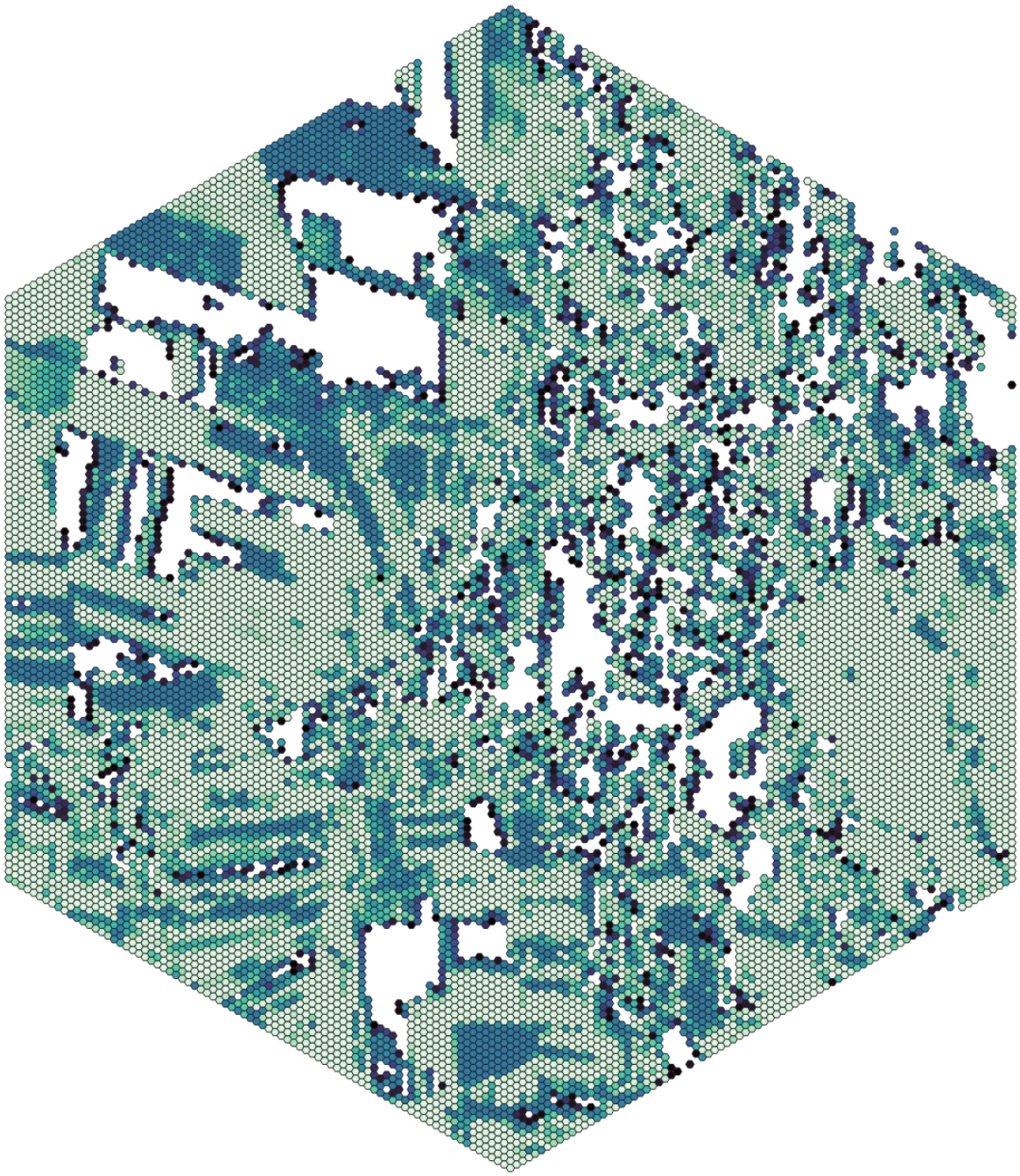

Figure D.28 — Visualization of the results of a DGGS-JSON data query for ISEA3H zone H4-A825C-A (level 14) and July 2020 at a custom depth of 9 from the output of the VHI workflow

Although not illustrated here, the DGGS-(UB)JSON format supports describing and encoding the logical schema, multiple depths, multiple fields, as well as additional dimensions beyond those of the DGGRS, such as time or atmospheric pressure. Each DGGS-(UB)JSON file can thus be a proper multi-resolution GeoDataCube, and an efficient and compact data store can be constructed as a hierarchy of DGGS-UBJSON files.

D.4. General recommendations for providing efficient access to Earth Observation data such as sentinel-2 (and other large GeoDataCubes)

D.4.1. Local to global scale with tiles and full set of overviews

Tiles and overviews, as supported by Cloud Optimized GeoTIFF, are essential to enable working with large datasets from local to global scales. These overviews should ideally go all the way to 1×1 pixels/cells. Unfortunately, the smallest overviews available for the Sentinel-2 Level-2A collection on AWS are 687×687, requiring downloading and processing a much larger volume of data than necessary to produce low resolution outputs. While image previews at lower resolutions are also available, these only reflect the red, green and blue bands, and are not useful for producing a global NDVI map for example, which requires the near-infrared band (`B08`).

> In Discrete Global Grid Systems (DGGS), tiles and overviews correspond to the concept of hierarchies of discrete global grids partitioning the globe into zones at different refinement levels.

D.4.2. Avoiding separate tile assets

Tiling of data should either be global, such as using a 2D Tile Matrix Set or Discrete Global Grid Reference System, or limited to internal tiling within e.g., GeoTIFF. For example, the splitting of scenes (capture of one or more fields/bands for a given time) into granules in the Sentinel-2 products is a major hindrance to achieving good performance. This is because it results in so many more items to process, and each granule in turn may contain a large quantity of null data values which fall outside of the granule geometry. Processing engines and cascading GeoDataCubes accessing such data sources may also select their own DGGRS or 2DTMS, making this splitting into granules unnecessary and problematic from a performance standpoint, if each granule are separately further partitioned using a different grid.

D.4.3. OGC APIs close to the data

Providing implementations of data access standards such as OGC API — DGGS, OGC API — Tiles or OGC API Coverages close to the data would greatly facilitate access to the data. This is because from a client's perspective, requests are much simpler to formulate than accessing byte ranges from cloud-optimized formats such as GeoTIFF. GeoTIFF requires either a built-in knowledge of the particular data format being accessed, or a virtual file system integrated with the format reader (which may be difficult to implement to achieve optimal performance) in order to request the correct portions of the files for a given scale (overview / refinement level) and spatial region (tile / zone).

D.4.4. Efficient scene catalog synchronization

Implementing a mechanism to support mirroring large catalogs of metadata such as STAC items using a compressed binary encoding. Ideally such a capability should be based on a relational model, as well as delta updates that would greatly facilitate the task of maintaining such mirrors by making it possible to efficiently synchronize on a regular basis. This would in turn allow for more efficient local queries which can lighten the load on dynamic API deployments.

D.4.5. Standards compliance

Ensuring conformance to standards and avoiding issues such as those described earlier encountered with the two STAC API deployments describing the input data for the VHI workflow in widely used production systems would greatly improve accessibility. The availability of Executable Test Suites (ETS) for standards greatly facilitates testing and achieving this conformance.

E

ANNEX E (INFORMATIVE) CRIM GDC API

ANNEX E (INFORMATIVE) CRIM GDC API

E.1. Implementation of OGC APIs for GeoDataCubes in CRIM Server

For the D140 GeoDataCube API Profile deliverable of the *OGC Testbed 20 – GeoDataCubes* task, CRIM provided updates to its Hirondelle development server implementation. This includes amongst other services, a *SpatioTemporal Asset Catalog* (STAC) endpoint (https://hirondelle.crim.ca/stac/) and an *OGC API – Processes* endpoint (https://hirondelle.crim.ca/weaver/), and access control management of individual resources on a per service, endpoint, HTTP method and user/group basis.

The STAC service and its companion STAC Browser interface are employed to publish and serve relevant data collections, either as *GeoDataCubes* or other structures, according to applied STAC metadata and extensions. The main purpose of these services is to retrieve the metadata from collections and visualize their items, allowing for advanced search and filtering capabilities. To access and manipulate the contained data referenced within a STAC Collection, Item, or Asset, other services offered by the server are employed (Implementations of OGC Web Services, GeoServer, etc.), according to inspected data types, domains, desired access mechanisms, and visualization requirements. Because most operations needed in the context of the GeoDataCube API Profile for *Data Discoverability* and *Data Access* capabilities are already offered by the STAC API, very few adjustments were applied to this service beyond what was already offered.

The implementation of *OGC API – Processes* Standard on the CRIM server is the service that received the most updates during Testbed 20. The service was implemented by CRIM using the Weaver open-source code. At the time of writing the report (Feb. 2025), the service implements most requirements defined in the *OGC API – Processes* extensions (*Part 1: Core*, *Part 2: Deploy, Replace, Undeploy*, *Part 3: Workflows and Chaining*, *Part 4: Job Management*). This is for both the older and newer revisions of supported requests/responses (i.e.: Core v1.0 and v2.0 representations), including draft proposals and unreleased exploratory work billing, quoting, and for OGC APIs and openEO alignments. Under the hood, the service employs cwltool to define processes such as *Application Packages* using the *Common Workflow Language* (CWL), which allows the service to deploy virtually any script or containerized application as an executable process, and chains them into advanced workflows for parallel and distributed computing. The service is also capable of chaining processing across multiple remote servers,

including interfaces for the OGC *Web Processing Service* (WPS) and the *OGC API — Processes* and *Application, Deployment and Execution Service* (ADES) Standards.

During testbed-20 efforts, the main updates applied to Weaver were:

1. Implementing the *Nested Processes* functionality of *Part 3: Workflows and Chaining*, therefore enabling clients to submit ad-hoc workflow execution directly rather than developing and deploying an equivalent CWL definition. Requiring fewer steps than the CWL deploy+execute apparoach, this allows for quicker chaining of simple processes together. However, this approach is more limited in regard to workflow advanced capabilities such as auto-scattering of parallel operations.

2. Implementing the *Collection Input* and *Remote Collection* requirements of *Part 3: Workflows and Chaining*, which supports directly submitting a process execution with a STAC or OGC compliant `collection` reference input. This update allows better interaction of processing capabilities offered by the service with data access mechanisms that align with GeoDataCube API Profile. The remote collection also supports making use of external catalogs from other servers, rather than limiting access to only local collections. Given that multiple agencies provide precomputed and *Analysis-Ready Data* (ARD) collections, supporting remote collections is typically more convenient to compute derived products on GDC data.

3. Implementing the job execution endpoint (`POST /jobs`) from *OGC API Processes — Part 4: Job Management*, such that a similar interface can be employed for submitting jobs either as an *OGC API — Processes* or openEO interface. This update provides better support of the sync execution, async execution or job queueing approach, as well as further job metadata definitions for provenance validation (see the *D022 Report* of the *D144 Provenance Demo* for more details).

While implementing all the above, CRIM directly contributed to the submission, review and update of the corresponding OGC Standards within the reference GitHub repository.

E.2. GDC Capabilities Demonstration

During Testbed 20, the Vegetation Health Index (VHI) use case was selected for implementation and testing. This selection was made because VHI simultaneously depends on Earth Observation data (raster imagery bands) and Climate data (air-surface temperature, TAS). As such, any GDC API Profile capabilities must be adaptable to the specific domains and constraints of each data type. These datasets are encoded in different formats (e.g., COGs, NetCDF), have varying resolutions, and represent different variables (imagery bands versus climate variables). This introduces multiple challenges when aggregating these distinct data sources.

For both collections, the same Area of Interest (AOI) and Time of Interest (TOI) are employed, namely around `bbox=13.3652612,45.4236367,16.5153015,46.8639623` and `datetime=2020-01-01T00:00:00Z/2021-01-01T00:00:00Z`, representing an area over Slovenia for the year 2020. The reference Earth Observation data that is obtained from *Earth-Search by Element-84*

using the sentinel-2-l2a collection, while the Climate data is obtained from *Copernicus Climate Data Store* with the ERA-5 Land Monthly Averaged collection.

For all following code snippets, process executions are submitted with the `POST /jobs` operation, unless stated otherwise, to comply with the GDC API Profile definitions.

E.2.1. Earth Observation GDC

In the case of the Earth Observation data, a pre-deployed `echo-files` process is employed to illustrate retrieval of the matched collection contents against filter criteria, using the *Collection Input* and *Remote Collection* requirements.

```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/echo-files",
  "inputs": {
     "files": {
         "collection": "https://earth-search.aws.element84.com/v1/collections/
sentinel-2-l2a",
         "datetime": "2020-01-01T00:00:00Z/2021-01-01T00:00:00Z",
         "bbox": [13.3652612, 45.4236367, 16.5153015, 46.8639623],
         "format": "stac-items",
         "type": "application/geo+json"
     }
  }
}
```

Listing E.1 — Earth Observation GDC Data Access

Obtained results are the matched STAC Items, as represented below (output clipped to keep the response concise).

```
{
  "files": [
    {
      "href": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/67c49a19-
d4dd-4bd4-b88a-1e0fe9766d81/files/S2B_33TUL_20200101_1_L2A.geojson",
      "type": "application/geo+json"
    },
    {
      "href": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/67c49a19-
d4dd-4bd4-b88a-1e0fe9766d81/files/S2B_33TUL_20200101_0_L2A.geojson",
      "type": "application/geo+json"
    }
  ]
}
```

Listing E.2 — Earth Observation GDC Data Result (Subset)

In order to compute NDVI, a `calculate-band` process is employed. This process is based on the gdal_calc utility, wrapped in a Docker container with all necessary runtime requirements and managed by CWL Application Package deployed onto the server. A sample execution body is presented below.

```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/calculate-band",
  "inputs": {
    "calc": "(B - A)/(B + A)",
    "file_a": {
```

```
      "href": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/c2e8517d-
4d39-477e-9772-351491a6b64e/B04.tif",
      "type": "image/tiff; application=geotiff; profile=cloud-optimized"
    },
    "file_b": {
      "href": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/a66d2047-
afef-428e-ad5a-ecad4a9ee5e4/B08.tif",
      "type": "image/tiff; application=geotiff; profile=cloud-optimized"
    },
    "name": "output"
  }
}
```

Listing E.3 — NDVI on a single patch

Note that the above `calculate-band` process takes the supplied bands for the NDVI calculation directly. However, the collection input from Earth-Search returns STAC Items. Therefore the relevant STAC Assets (B04 — red and B08 — NIR) must be extracted to operate on them. To do so, another `select-assets` process was employed. The job consists of simply extracting the requested Assets by name from the STAC Item JSON.

For example, the processes can be chained as follows.

```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/calculate-band",
  "inputs": {
    "calc": "(B - A)/(B + A)",
    "file_a": {
      "process": "https://hirondelle.crim.ca/weaver/processes/select-assets",
      "inputs": {
        "stac_item": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/
67c49a19-d4dd-4bd4-b88a-1e0fe9766d81/files/S2B_33TUL_20200101_1_L2A.geojson",
        "assets": ["B04"]
      }
    },
    "file_b": {
      "process": "https://hirondelle.crim.ca/weaver/processes/select-assets",
      "inputs": {
        "stac_item": "https://hirondelle.crim.ca/wpsoutputs/weaver/users/24/
67c49a19-d4dd-4bd4-b88a-1e0fe9766d81/files/S2B_33TUL_20200101_1_L2A.geojson",
        "assets": ["B08"]
      }
    },
    "name": "output"
  }
}
```

Listing E.4 — Chaining of Processes for NDVI from STAC Item

Given the expectation that many STAC Items will match the search criteria to form the GDC, it is more efficient to parallelize the chain of processes. For this, taking advantage of CWL `scatter` capability worked. The resulting CWL workflow deployed as process `ndvi-batch` is presented below. It takes the original collection URI as input (by chaining the `stac_items` references yielded from it) and performs the Asset data selection and NDVI calculation for all matched STAC Items, with auto-scaling of the processing pipeline as necessary. Its execution result is an array of NDVI tiles in COG format (inferred from the input Assets passed to `calculate-band`) for all applicable STAC Items.

```
#!/usr/bin/env cwl-runner
```

```yaml
cwlVersion: "v1.2"
class: Workflow
id: ndvi-batch

s:softwareVersion: "1.0.0"
s:codeRepository: "https://gitlab.com/crim.ca/clients/ogc/ogc-tb20"
s:license: "https://spdx.org/licenses/CC-BY-NC-SA-4.0"

s:author:
  - class: s:Person
    s:identifier: "https://orcid.org/0000-0003-4862-3349"
    s:email: francis.charette-migneault@crim.ca
    s:name: Francis Charette-Migneault

requirements:
  ScatterFeatureRequirement: {}
  StepInputExpressionRequirement: {}
  SubworkflowFeatureRequirement: {}
  InlineJavascriptRequirement: {}

inputs:
  stac_items:
    type: string[]
    format: "iana:application/geo+json"

outputs:
  ndvi:
    type: File[]
    format: "ogc:geotiff"
    outputSource: ndvi/result

steps:
  select_bands:
    run: select-assets.cwl
    scatter: [ stac_item ]
    scatterMethod: flat_crossproduct
    in:
      stac_item: stac_items
      assets:
        valueFrom: ${ return ["red", "nir"] }
    out: [ result ]

  ndvi:
    run: calculate-band.cwl
    scatter: [file_a, file_b]
    scatterMethod: dotproduct
    in:
      calc:
        # (B08 - B04) / (B08 + B04)
        valueFrom: '(B - A)/(B + A)'
      file_a:
        source: select_bands/result
        valueFrom: ${ return self[0] }
      file_b:
        source: select_bands/result
        valueFrom: ${ return self[1] }
    out: [result]

$namespaces:
  s: "https://schema.org/"
  ogc: "http://www.opengis.net/def/media-type/ogc/1.0/"
```

```
  iana: "https://www.iana.org/assignments/media-types/"
```

Listing E.5 — NDVI batch CWL Workflow

Following is an example invocation of the process that will trigger the entire workflow procedure:

```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/ndvi-batch",
  "inputs": {
    "stac_items": {
      "collection": "https://earth-search.aws.element84.com/v1/collections/
sentinel-2-l2a",
      "datetime": "2020-01-01T00:00:00Z/2021-01-01T00:00:00Z",
      "bbox": [13.3652612, 45.4236367, 16.5153015, 46.8639623],
      "format": "stac-items",
      "type": "application/geo+json"
    }
  }
}
```

Listing E.6 — Chaining of Processes for NDVI batch from STAC Collection

E.2.2. Climate GDC

In the case of the meteorological collection data, fewer operations are needed. The Copernicus Climate Data Store (CDS) offers an *OGC API — Processes* interface (**note:** v1.0!) which can retrieve specific subsets of the climate data the user is interested in. More specifically, the process reanalysis-era5-land-monthly-means can be employed.

Since the CDS process uses the v1.0 interface, the `POST /jobs` operation specified by the GDC API Profile does **NOT** apply. Instead, the `POST /processes/reanalysis-era5-land-monthly-means/execution` operation must be used. Also, this process enforces `async-execute` as per its *Process Description*. Executing this process would be performed as follows.

```
{
  "inputs": {
    "product_type": ["monthly_averaged_reanalysis"],
    "variable": ["skin_temperature"],
    "year": ["2020"],
    "month": ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11",
"12"],
    "time": ["00:00", "01:00", "02:00", "03:00", "04:00", "05:00", "06:00",
"07:00", "08:00", "09:00", "10:00", "11:00", "12:00", "13:00", "14:00", "15:00",
"16:00", "17:00", "18:00", "19:00", "20:00", "21:00", "22:00", "23:00"],
    "area": [13.3652612, 45.4236367, 16.5153015, 46.8639623],
    "data_format": "netcdf",
    "download_format": "unarchived"
  }
}
```

Listing E.7 — Original Copernicus Execution for Climate Data (non GDC compliant)

And the result obtained from it after monitoring of the job until completion should be similar to the following.

```
{
  "asset": {
```

```
    "value": {
      "type": "application/netcdf",
      "href": "https://object-store.os-api.cci2.ecmwf.int:443/cci2-prod-cache/ebb
8dcabf65ca763767806c71f51639e.nc",
      "file:checksum": "dadbc824755486423fd9df634eb1f648",
      "file:size": 37523,
      "file:local_path": "s3://cci2-prod-cache/ebb8dcabf65ca763767806c71f51639e.
nc"
    }
  }
}
```


Listing E.8 — Results from the Original Copernicus Execution for Climate Data

To make this process compliant with GDC API Profile (i.e.: using the `POST /jobs` operation), the Weaver's capability to invoke remote processes can be taken advantage of. The process is simply "deployed" using the following wrapping approach by sending the reference to `POST /processes`.


```
{
  "process": {
    "id": "cds-reanalysis-era5-land-monthly-means"
  },
  "executionUnit": [
    {
      "href": "https://cds.climate.copernicus.eu/api/retrieve/v1/processes/
reanalysis-era5-land-monthly-means",
      "type": "application/json; profile=ogcapi-processes"
    }
  ]
}
```


Listing E.9 — Deploy the Copernicus Climate Data Store process as GDC compliant

With this deployment, it is now possible to employ the GDC compliant `cds-reanalysis-era5-land-monthly-means` process. The execution request is essentially the same, except that it can be sent to the `POST /jobs` operation and must also provide additional `process` reference(s) in the body. The inputs will be automatically chained to the underlying remote process on CDS, its execution will be monitored until completion, after which it will stage the results under the local job.


```
{
  "process": "https://hirondelle.crim.ca/weaver/processes/cds-reanalysis-era5-
land-monthly-means",
  "inputs": {
    "product_type": ["monthly_averaged_reanalysis"],
    "variable": ["skin_temperature"],
    "year": ["2020"],
    "month": ["01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11",
"12"],
    "time": ["00:00", "01:00", "02:00", "03:00", "04:00", "05:00", "06:00",
"07:00", "08:00", "09:00", "10:00", "11:00", "12:00", "13:00", "14:00", "15:00",
"16:00", "17:00", "18:00", "19:00", "20:00", "21:00", "22:00", "23:00"],
    "area": [13.3652612, 45.4236367, 16.5153015, 46.8639623],
    "data_format": "netcdf",
    "download_format": "unarchived"
  }
}
```


Listing E.10 — Wrapped Copernicus Execution for Climate Data (GDC compliant)

This allows the wrapped process to seamlessly take advantage of all GDC capabilities that Weaver provides, including Provenance monitoring and using synchronous execution, although the original process did not implement them. Furthermore, by defining a local process, the obtained results from the remote process will be automatically staged onto the local server, making the combined of this climate result with other GDC collections more practical.

E.2.3. Combining GDC Collections

Due to a change of scope during Testbed 20 (i.e.: a change of interpretation of the TOI and monthly-averaging strategy of VHI), the final step to complete the calculation could not be processed in time for the Testbed. However, given that Weaver supports the deployment of practically any script, calculating the VHI would generally consist of performing the previous GDC collection retrieval steps, both for Earth Observation and Climate data, over the extended TOI ranges needed by the VHI definition, and aggregating the results as defined by the VHI formula.

Please note that the `calculate-band` process could be reused for most of the underlying operations, since the underlying gdal_calc that it utilizes can make use of any NumPy operation (`min`, `max`, `avg`, etc.), supports both COGs and NetCDF (corresponding to our GDC collections), and supports multiple options to handle the necessary resolution scaling and interpolation between them.

E.3. Observations and Recommendations

While working on Testbed 20 to define the GDC API Profile, work in collaboration with other participants quickly highlighted a need for better definitions regarding provided functions. For example, the CRIM implementation employs the `calculate-band` process extensively to perform various band-math operations. Other participants have opted for specific process definitions (`min`, `max`, etc.), some as openEO functions, others as CQL2-like `properties` operations submitted along the `collection` input or output. Regardless of the selected approach, each of which has its advantages and disadvantages, all these processes and approaches are not standard across all implementations. Therefore, they cannot be employed in an interoperable manner by multiple servers.

Similarly, while the issue of scope adjustment from the TOI monthly-averaging of VHI appeared, the involved processes in the workflow were often updated to accommodate the new requirements, without need to change any process ID. The insertion of additional parameters and alternate behaviors depending on the set of arguments provided as input to a process to adapt to the new use case, mixed with the high-dimensionality complexity of GDC, made interpretation of the processing and opportunities to interoperate even harder.

Given the limited information about the specific functions of each process in a workflow processing chain, it is strongly recommended that the OGC establish a registry of “well-known functions” as part of its processing services. Providing a standardized set of operations—with clearly defined inputs and outputs—will significantly enhance the understanding of data manipulation when working with data processed by the GDC API. This initiative will also support

efforts to improve data provenance comprehension, ensure data integrity, and build trust in the resulting analyses.

F

ANNEX F (INFORMATIVE) MMS GDC API — OPENEO GOOGLE EARTH ENGINE

ANNEX F (INFORMATIVE) MMS GDC API — OPENEO GOOGLE EARTH ENGINE

The implementation of a GDC API on top of Google Earth Engine (GEE) is unique compared to other implementations, as it acts as a "proxy" over an existing API/service without direct access to the underlying data, processing engine, or infrastructure. As a result, this approach presented a distinct set of challenges compared to the other implementations in Testbed 20. The following sections outline the technical details and challenges encountered during the implementation.

The implementation can be found on GitHub: <https://github.com/Open-EO/openeo-earthengine-driver/pull/92>

A couple of potential issues/improvements were reported to the OGC API — Coverages SWG: https://github.com/opengeospatial/ogcapi-coverages/issues?q=+is%3Aissue+author%3Am-mohr>

F.1. GDC Core (Full Data Access)

The implementation of `GET /` and `GET /conformance` is straightforward.

The implementation of `GET /collections` and `GET /collections/{collectionId}` is more difficult. The source STAC metadata is missing parts of the metadata that implementations of the OGC API — Coverages Standard requires. As such, the implementation has to add additional metadata to make it compliant with the requirements stated in the OGC API — Coverages Standard. This required additions by the Google Earth Engine team. Otherwise the client needs to request the missing metadata from their processing API which does not scale due to the number of collections and rate limits of the GEE API.

Although not directly part of any of the GDC profiles, the output CRS for Coverage requests can be set through the `crs` parameter.

F.2. GDC Partial Access

The domain subsetting can mostly be implemented on top of Google Earth Engine. One potential issue is that it is not clear from the STAC metadata whether the values are “value-is-point” or “value-is-area”, which makes the result of the subsetting ambiguous.

The field selection can be implemented on top of Google Earth Engine. Note: GEE generally has no other fields than bands. The bands are not necessarily spectral and should better be seen as variables that are always named “bands”.

The openEO GEE implementation only implements `bbox` and `datetime` subsetting in addition to field selection. The `subset` parameter is not implemented due to the complexity of parsing it. It would not add much value on top of the other parameters and thus focus was put on the other more important implementation details.

F.3. GDC Resampled Access

Resampled access / scaling is difficult to implement on top of Google Earth Engine due to the lack of related information in their STAC metadata. The conversion between the various parameters in the Scaling requirement class of the OGC API — Coverages Standard necessitates a lot of conversions based on such as the resolution must take place, which are not possible due to the lack of related uniform metadata. Thus, this functionality was not implemented during Testbed 20.

F.4. Authentication

Authentication is available through both OpenID Connect (Google accounts) and HTTP Basic (demo accounts).

F.5. GDC Data Processing

Requires: GDC Core

Supports submitting a workflow definition to one endpoint for advanced data processing: POST /tasks

The implementation may process data in different modes: synchronously, asynchronously or on-demand (i.e. create a new GDC API or collection). Which mode to choose is specified through a mode query parameter, which can be either sync, async, collection, or api.

A workflow definition could be anything, for example:

OGC API — Processes — Part 3 ("MOAW") — conformance class: http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/moaw CWL — conformance class: http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/cwl openEO process — conformance class: http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/openeo WCPS — conformance class: http://www.opengis.net/spec/ogcapi-gdc-1/0.0/conf/wcps

F.5.1. GDC Data Processing via openEO

The openEO implementation was only slightly updated as it was available before Testbed 20. The use case used in Testbed-20 (Vegetation Health Index) would require additional processes, but due to the uncertainties around the use case, focus was put on unplanned work items related to general alignment of openEO and the related OGC API Standards.

F.5.2. GDC Data Processing via OGC API — Processes

As planned, data processing through an OGC API — Processes endpoint was not implemented on top of Google Earth Engine.

F.6. Usecase

The use case was available rather late in the Testbed 20 activity and uncertainties were around until the end of Testbed 20. An openEO process (graph) for the usecase was created: https://github.com/clausmichele/OGC-T20-D144/blob/main/VHI_local.ipynb> The use case would require additional openEO processes, primarily `merge_cubes`, which is a complex process that has no direct equivalent in GEE. Due to the uncertainties around the use case, they were not implemented, and focus was instead put on work items related to general alignment of openEO and the related OGC APIs.

G

ANNEX G (INFORMATIVE) MMS — GDC WEB EDITOR — GDC CLIENT

ANNEX G (INFORMATIVE) MMS — GDC WEB EDITOR — GDC CLIENT

The GDC Client is a web-based application that allows users to interact with an implementation of a GDC API through a user-friendly no-code interface. It is an implementation based on the openEO ecosystem, more specifically the openEO Web Editor and the openEO JS client. The client is available at https://m-mohr.github.io/gdc-web-editor/>. The source code is available at:

- <https://github.com/m-mohr/gdc-web-editor>
- <https://github.com/Open-EO/openeo-js-client/pull/62>

The following section highlights the implementations and potential issues based on the profiles defined for a GDC API.

G.1. GDC Core (Full Data Access)

Fortunately, openEO and OGC APIs share the same core endpoints as defined by the OGC API — Common Standard. Thus, most endpoints can already be read by the GDC Web Editor with just some minor changes.

The following differences exist:

1. openEO APIs host a well-known document that supports discovery and versioning of openEO instances. For example, multiple versions of the openEO API could be hosted, including production and development versions. openEO clients can discover the available versions and choose the one that fits best. This functionality is not available in implementations of OGC API Standards and as such the openEO clients have to skip this step when working with OGC API endpoints. Unfortunately, the only way to discover this is by sending a request and waiting for either a successful response or a 404 Not Found error.
2. The landing page has an endpoint list in openEO instead of specifying implemented endpoints via conformance classes. This is due to legacy reasons. This means the openEO JS client was extended to convert the endpoints list of openEO to a conformance class list so that a generic mechanism for accessing conformance across APIs exists. The openEO API plans to switch to conformance classes in the future, see https://github.com/Open-EO/openeo-api/issues/506>.

3. The collection endpoints are similar and follow the general definitions in the OGC API — Common Standard. openEO follows the STAC requirements and requires the STAC datacube extension. On the other hand, the OGC API — Coverages Standard extends the OGC API — Common Standard in a different way. As such the general collection metadata is similar, but the details about data cubes are different and have to be duplicated. The OGC API — Coverages Standard uses the UMD model and a schema endpoint to describe data cubes, while openEO uses the STAC datacube extension. The client will use the UMD model and the schema endpoint when necessary but not by default.

Generally, the client can show process and collection metadata for all APIs. Functionality for data access through an OGC API — Coverages endpoint is available through a step-by-step wizard in the GDC Web Editor. Data access in openEO is available similarly, also through a step-by-step wizard.

The OGC API — Coverages Standard is a draft and as such the implementation support is still limited while the community waits for the final version of the standard and its additional parts. Based on the implementation, extensive feedback was provided to the OGC API — Coverages Standard Working Group (SWG) through GitHub issues and discussions in the OGC SWG meetings to improve the draft standard: https://github.com/opengeospatial/ogcapi-coverages/issues?q=is%3Aissue+author%3Am-mohr>

G.2. GDC Partial Access

As the GDC Partial Access profile consists of a conformance class specific to the draft OGC API — Coverages Standard, the conformance class is implemented as part of the step-by-step wizard in the GDC Web Editor.

The following features are available:

- Spatial subsetting based on bounding boxes
- Temporal subsetting

Field selection is not implemented yet.

G.3. GDC Resampled Access

As the GDC Resampled Access profile consists of a conformance class specific to the draft OGC API — Coverages Standard, the conformance class is implemented as part of the step-by-step wizard in the GDC Web Editor.

The following features are available:

- Specifying a scale factor for resampling

Other scaling options are not implemented yet.

G.4. Authentication

For sake of the practicality during Testbed-20, the authentication methods specified for a GDC API are adopted from the openEO API. OGC does not define or recommend any specific authentication methods, so the openEO methods are used as a reference for this activity and for the previous testbed.

The GDC Web Editor supports all authentication methods, i.e. OpenID Connect and Basic authentication.

G.5. GDC Data Processing

Various data processing and especially workflow options are discussed:

- openEO process
- OGC API — Processes — Part 3 workflows ("MOAW")
- CWL
- WCPS

The focus in the GDC Web Editor is on openEO processes. There is also basic support for OGC API — Processes — Part 3 workflows ("MOAW"). CWL is planned for later. Implementation of the OGC Web Coverage Processing Service (WCPS) Standard is not planned for now.

G.5.1. GDC Data Processing via openEO

The GDC Web Editor, due to its roots in the openEO project, supports most of the functionality of the openEO API, including asynchronous batch processing, synchronous processing, and secondary web services (on-demand processing).

G.5.2. GDC Data Processing via OGC API — Processes

The openEO JS client was extended to support working with OGC API — Processes implementation through a similar high-level interface as openEO. Internally, it converts the OGC API — Processes endpoint responses to openEO responses. For example, the descriptions of processes from an implementation of the OGC API Processes Standard are converted to openEO processes for consumption by the GDC Web Editor. The GDC Web Editor as such does not have any specific code for OGC API — Processes endpoints as it relies on the openEO JS client for this functionality.

The current functionality is limited to OGC API — Processes — Part 1, version 1.0 and a small subset of the draft OGC API — Processes — Part 3 workflows ("MOAW"). The OGC API — Processes Standard is in flux. As such support is still limited while waiting for the final version of the standard and its additional parts. Based on the implementation, extensive feedback was provided to the OGC API — Processes SWG through GitHub issues and discussions in SWG meetings to improve the draft standard: https://github.com/opengeospatial/ogcapi-processes/issues?q=is%3Aissue+author%3Am-mohr>