OGC® DOCUMENT: 21-027

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D012

Open Geospatial Consortium

OGC TESTBED 17: GEO DATA CUBE API ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2022-02-09
Approval Date: 2022-03-04
Publication Date: 2022-04-08
Editor: Jérôme Jacovella-St-Louis

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I ABSTRACT

This OGC Testbed 17 Engineering Report (ER) documents the results and recommendations of the Geo Data Cube API task. The ER defines a draft specification for an interoperable Geo Data Cube (GDC) API leveraging OGC API building blocks, details implementation of the draft API, and explores various aspects including data retrieval and discovery, cloud computing and Machine Learning. Implementations of the draft GDC API are demonstrated with use cases including the integration of terrestrial and marine elevation data and forestry information for Canadian wetlands.

II EXECUTIVE SUMMARY

II.A. Key findings

An architecture for a Geo Data Cube API framework is proposed. The framework is built using approved OGC standards and draft OGC API specifications. *OGC API — Common* provides a cohesive consistency for presenting an API landing page, conformance declaration and API description in *Part 1: Core* OGC 19-072, while defining collections of spatiotemporal data corresponding to the GDC API data cube resources in *Part 2: Geospatial data* OGC 20-024. Multi-resolution data can be stored, indexed, and represented as such Geo Data Cube resources. These resources can be transformed by performing different operations such as resampling, subsetting, aggregation, filtering, band arithmetic calculations, or processing algorithms. This also includes using complex workflows, queried by multiple data access mechanisms defined in OGC API building blocks, and returned as outputs in suitable negotiated formats.

The following OGC standards and specifications were considered and/or used in defining the GDC API.

- *OGC API — Coverages* is considered a key GDC capability with its subsetting, range (fields) subsetting, scaling and tiles conformance classes, as well as proposed extensions for supporting filtering expressions, band arithmetic calculations and varying resolution.
- Cloud Optimized GeoTIFF (COG) was considered as both a backend data store and an efficient distribution mechanism.
- *OGC API — Tiles* and *OGC API — Features* are also considered as data access mechanisms.
- *OGC API — Environmental Data Retrieval (EDR)* which offers queries for typical meteorological use cases such as data along a trajectory or within a corridor.

- *OGC API — Discrete Global Grid Systems* is suggested as an important component to integrate within a GDC API framework.
- Complex analytics can be achieved using *OGC API — Processes — Part 1: Core*, while simpler analytics capabilities should be conveniently integrated directly within *OGC API — Coverages* and *EDR* data requests.
- The *OGC API — Processes — Part 2: Deploy, Replace, Update* draft specification is highlighted as a way to deploy new complex algorithms close to data.
- The *OGC API — Processes — Part 3: Workflows & Chaining* draft specification is highlighted as a way to present the output of a process or workflow as a data cube, while supporting integration of distributed data cubes and analytics capabilities.
- The *OGC API — Maps* and *OGC API — Tiles* specifications were identified as ways to directly integrate server-side visualization capabilities within a GDC framework.
- The role of *OGC API — Records*, *STAC* and *OGC API — Common* for data discovery was explored.

Some overlap between the *OGC API — EDR* Standard and draft *OGC API — Coverages* specification were identified, particularly in terms of describing a data cube and the EDR cube queries. Some current incompatibilities between the APIs specified in *OGC API — EDR* and *OGC API — Common* were also identified.

A *Scenes API* is proposed as a way to provide a unified data cube while still providing direct access to individual scenes making it up as well as to their metadata. The *Scenes API* is also proposed as a mechanism to manage multiple scenes making up a data cube. This approach is based on the work from the Testbed 15 — *Images API*.

The new analytics capabilities defined by Testbed 16 — *Data Access & Processing API (DAPA)* are proposed as extensions for the *Coverages* and *EDR* APIs rather than as a new separate API. The definition of well-known processes supporting convenient processing languages is suggested. The need for identifying data cubes for use as input to particular processes was identified.

II.B. Results

The initiative participants developed four servers (provided by Wuhan University, 52°North, MEEO, and Ecere) and three clients (provided by Solenix, Ethar and Ecere) implementing selected Geo Data Cube API capabilities based on OGC API standards and specifications:

- OGC API — Common — Part 1: Core;
- OGC API — Common — Part 2: Geospatial data;
- OGC API — Coverages — Part 1: Core, supporting subsetting, range subsetting, i.e. fields selection, scaling;

- OGC API — Processes — Part 1: Core, supporting synchronous and asynchronous execution;
- OGC API — Processes — Part 3: Workflows and Chaining, supporting collection input and collection output.

52°North demonstrated the use of the GDC API in the context of machine learning for land cover prediction from Earth Observation imagery. Ethar additionally demonstrated the use of the GDC API in the context of Augmented Reality.

II.C. Business value

This Engineering Report (ER) describes the results of discussions and experiments evaluating OGC API standards and draft specifications. Further, other data cube implementations developed outside of the OGC were evaluated in the context of a Geo Data Cube API for data access, analytics and discovery. The ER makes recommendations for the OGC Standards Program to improve interoperability of data cubes. The ER also highlights the interoperability drawbacks of defining different specifications for the same functionality within the same family of OGC API standards. Capabilities missing from OGC standards for accessing and performing analytics on data cubes are also identified which should be standardized in a uniform manner by extending the current approved standards and draft specifications. This should in turn facilitate the rapid implementation of interoperable spatiotemporal data cube capabilities within various technologies and spur further innovation.

II.D. Requirements addressed

In the Testbed-17 GDC task, the participants addressed requirements for defining an OGC API for Geo Data Cubes, leveraging existing building blocks, which would support:

- access and processing in the cloud,
- data discovery and querying information of diverse collections of data,
- interoperability with STAC, registries & catalogs,
- interoperability of data formats and access methods,
- interoperability across different cloud providers,
- interoperable workflows,
- machine learning for detection from Earth Observation imagery and deriving insights from spatiotemporal data, and

- interoperability between different Geo Data Cubes and APIs.

II.E. Motivation for defining a Geo Data Cube API

The motivation for defining a GDC API was to provide efficient access to data cubes, performing analytics close to the data ranging from some simple aggregation and band arithmetic to more complex algorithms, discovering data and analytics capabilities, as well as potentially integrating visualization and analytics management capabilities. Such an API will enable the use of these capabilities in client applications, allowing to derive useful insights from very large collections of data, in particular multi-spectral Earth Observation imagery, which are an important source of information in the context of solving global challenges such as climate change.

II.F. Recommendations for future work

The GDC task demonstrated the value of the OGC API family of standards, including those already approved (*Features Part 1: Core & Part 2: CRS by reference*, *EDR*, and *Processes*), and those still in draft stage (e.g. Common Part 1: Core & Part 2: Geospatial Data, Features — Part 3: Filtering, CQL2, Tiles, Coverages, Maps, DGGS, Records/STAC, Processes — Part 2: Deploy, Replace, Update & Part 3 — Workflows & Chaining), and recommends prioritizing their completion.

The importance of completing *OGC API — Common — Part 1* & *Part 2* as a framework for integrating capabilities in particular is highlighted. For example, resolving some incompatibilities that already identified with the EDR and *OGC API — Common — Part 2* specifications. These could be resolved, allowing to offer the same data cube using the EDR API plus additional access mechanisms.

The role of the draft *OGC API — Coverages* specification as a baseline for describing data cubes and providing a simple and convenient data access mechanism should be clarified. This includes support for subsetting domain and range (fields / bands), and resampling. Further, this also includes support for accessing coverage data as tiles, following a fixed pyramidal multi-dimensional tiling scheme. Additional capabilities for filtering based on CQL expressions should be considered as an extension for coverages.

If possible, an attempt should be made to re-align and harmonize the EDR specification's data description mechanism and its cube query with *OGC API — Coverages*. The analytics capabilities defined in the Testbed 16 — *DAPA* specification should be integrated directly within *OGC API — Coverages* and possibly *OGC API — EDR* as well as extensions rather than defining a new specification. OGC should ensure separate OGC API standards do not re-define the same capabilities with only superficial variations that could reduce interoperability. This also introduces a significant burden on implementers of clients & services in terms of additional standards to implement.

A *Scenes API* should be defined making it possible to support both a unified data cube while providing direct access to the data and metadata of individual scenes, thereby enabling integrated discovery, as well as scenes management capabilities.

Defining well-known processes expecting specific inputs — including a particular convenient processing language — to facilitate flexible coverage processing should be considered.

The need for Executable Test Suites for the different OGC API standards was highlighted.

The value of defining a set of standardized OGC API building blocks as a GDC meta-standard should be considered.

Further defining and leveraging the draft *OGC API — Processes — Part 3: Workflows and Chaining* specification would support presentation of the results of analytics capabilities as a virtual data cube and facilitating the integration of analytics capabilities in visualization clients, as well as facilitating the integration of remote data cubes with processing algorithms. This should be an important priority.

III KEYWORDS

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

ogcdoc, OGC document, API, OpenAPI, OGC API, Coverage, Data Cube

IV PREFACE

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.

V SECURITY CONSIDERATIONS

No security considerations have been made for this document.

VI SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Ecere Corporation

VII SUBMITTERS

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

NAME	ORGANIZATION	ROLE
Jérôme Jacovella-St-Louis	Ecere	Editor
Tony Hodgson	Ethar, Inc.	Contributor
Karri Ojala	Solenix GmbH	Contributor
Alexander Lais	Solenix GmbH	Contributor
Peng Yue	Wuhan University	Contributor
Martin Pontius	52°North GmbH	Contributor
Eike Hinderk Jürrens	52°North GmbH	Contributor
Sufian Zaabalawi	52°North GmbH	Contributor
Joshua Lieberman	OGC	Contributor
Patrick Dion	Ecere	Contributor
Diego Caraffini	Ecere	Contributor
Fabio Govoni	MEEO	Contributor
Fan Gao	Wuhan University	Contributor

NAME	ORGANIZATION	ROLE
Shuaifeng Zhao	Wuhan University	Contributor
Colin Steinmann	Ethar, Inc., Open AR Cloud	Contributor
Nazih Fino	Ethar, Inc., Open AR Cloud	Contributor
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.	Contributor

1

NORMATIVE REFERENCES

1 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r3, *OGC API — Features — Part 1: Core*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

Clements Portele, Panagiotis (Peter) A. Vretanos: OGC 18-058, *OGC API — Features — Part 2: Coordinate Reference Systems by Reference*. Open Geospatial Consortium (2020). https://docs.ogc.org/is/18-058/18-058.html

Peter Baumann: OGC 17-089r1, *OGC Web Coverage Service (WCS) 2.1 Interface Standard — Core*. Open Geospatial Consortium (2018). http://docs.opengeospatial.org/is/17-089r1/17-089r1.html

Matthias Mueller: OGC 14-065r2, *OGC® WPS 2.0.2 Interface Standard: Corrigendum 2*. Open Geospatial Consortium (2018). http://docs.opengeospatial.org/is/14-065/14-065r2.html

OGC API — Environmental Data Retrieval Standard, https://www.opengis.net/doc/IS/ogcapi-edr-1/1.0

OGC API — Processes — Part 1, https://docs.ogc.org/is/18-062r2/18-062r2.html

OGC: OGC 07-011, *Topic 6 — Schema for coverage geometry and functions*. Open Geospatial Consortium (2007). https://portal.ogc.org/files/?artifact_id=19820

Peter Baumann, Eric Hirschorn, Joan Masó: OGC 09-146r6, *OGC Coverage Implementation Schema*. Open Geospatial Consortium (2017). http://docs.opengeospatial.org/is/09-146r6/09-146r6.html

Joan Masó: OGC 17-083r2, *OGC Two Dimensional Tile Matrix Set*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/is/17-083r2/17-083r2.html

Matthew Purss: OGC 15-104r5, *Topic 21 — Discrete Global Grid Systems Abstract Specification*. Open Geospatial Consortium (2017). http://docs.opengeospatial.org/as/15-104r5/15-104r5.html

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

2.1. Terms and definitions

2.1.1. **cell**

unit of a coverage’s domain set (potentially spanning multiple direct positions), of a fixed resolution in the case of gridded coverages, for which a specific set of range values (e.g. a pixel in an image, or a set of measurements) is returned

2.1.2. **collection**

(in the context of OGC API specifications) resource consisting of geospatial data that may be available as one or more sub-resource distributions that conform to one or more OGC API standards.

(SOURCE: https://github.com/opengeospatial/ogcapi-common/issues/140#issuecomment-642239475)

(in a general computer science context) grouping of some variable number of data items (possibly zero) that have some shared significance to the problem being solved and need to be operated upon together in some controlled fashion

(SOURCE: https://en.wikipedia.org/wiki/Collection_(abstract_data_type))

2.1.3. coordinate reference system

coordinate system that is related to the real world by a datum

(SOURCE: ISO 19111:2019 Geographic information — Referencing by coordinates)

2.1.4. coordinate reference system

coordinate system that is related to the real world by a datum term name (source: ISO 19111)

2.1.5. coverage

feature that acts as a function to return values from its range for any direct position within its spatio-temporal domain

2.1.6. data cube

multi-dimensional data store

Multi-dimensional (*n-D*) array of values

(SOURCE: OGC 18-095r7)

Note 1 to entry: The term is also sometimes used to refer to a service or platform providing access to such data cube, or to a federation of such services or platforms.

Note 2 to entry: Even though it is called a 'cube,' it can be 1- dimensional, 2-dimensional, 3-dimensional, or higher-dimensional. The dimensions may be coordinates or enumerations, e.g., categories.

2.1.7. **dataset**

A dataset is a collection of data, published or curated by a single agent. Data comes in many forms including numbers, words, pixels, imagery, sound and other multi-media, and potentially other types, any of which might be collected into a dataset.

(SOURCE: W3C Data Catalog Vocabulary (DCAT) — Version 2, 2020)

Note 1 to entry: There is an important distinction between a dataset as an abstract idea and a distribution as a manifestation of the dataset

2.1.8. **data store**

A data store is a repository for persistently storing and managing collections of data which include not just repositories like databases, but also simpler store types such as simple files, metadata, models, etc.

(SOURCE: https://www.information-management.com/glossary/d.html:2020)

2.1.9. **direct position**

position described by a single set of coordinates within a coordinate reference system

(SOURCE: OGC Abstract Topic 6 — Schema for coverage geometry and functions)

2.1.10. **domain**

well-defined set [ISO/TS 19103]

(SOURCE: OGC Abstract Topic 6 — Schema for coverage geometry and functions)

Note 1 to entry: Domains are used to define the domain and range of operators and functions.

2.1.11. **elevation**

synonym for “height”

(SOURCE: Clause 4.16 of ISO/TS 19159:2016, https://www.iso.org/obp/ui/#iso:std:iso:ts:19159:-2:ed-1:v1:en)

2.1.12. **geo data cube**

a data cube for which some dimensions are geospatial (e.g. latitude and longitude, or projected easting and northing; elevation above the WGS84 ellipsoid)

A (geo) data cube is a discretized model of the earth that offers estimated values of certain variables for each partition of the Earth’s surface called a cell. A data cube instance may provide data for the whole Earth or a subset thereof. Ideally, a data cube is dense (i.e., does not include empty cells) with regular cell distance for its spatial and temporal dimensions. A data cube describes its basic structure, i.e., its spatial and temporal characteristics and its supported variables (also known as ‘properties’), as metadata. It is further defined by a set of functions. These functions describe the available discovery, access, view, analytical, and processing methods that are supported to interact with the data cube.

(Source: OGC 21-067)

Note 1 to entry: From a functionality perspective, it can be considered a multi-dimensional field including spatial dimensions, and often temporal dimensions as well (much like a coverage).

Note 2 to entry: As documented in OGC 21-067, this definition was proposed as an outcome of a Workshop and is thus still the subject of discussion.

2.1.13. **height**

Distance of a point from a chosen reference surface measured upward along a line perpendicular to that surface.

(SOURCE: ISO 19111:2019 Geographic information — Referencing by Coordinates)

Note 1 to entry: A height below the reference surface will have a negative value, which would embrace both gravity-related heights and ellipsoidal heights.

2.1.14. **job**

instance of a process execution

2.1.15. **metadata**

information about a resource.

(SOURCE: ISO 19115-1:2014)

Note: The US National System for Geospatial Intelligence (NSG) Metadata Foundation (NMF) Version 3.0 defines metadata as information that captures the characteristics of a resource to represent the 'who', 'what', 'when', 'where', 'why', and 'how' of that resource.

2.1.16. **platform**

computer hardware, software and/or network services providing a set of defined capabilities

2.1.17. **process**

series of computing operations to be executed, which may produce one or more output (and/or result in some other side effects), and may take one or more inputs.

2.1.18. **range**

(coverage) set of feature attribute values associated by a function with the elements of the domain of a coverage

(SOURCE: OGC Abstract Topic 6 — Schema for coverage geometry and functions)

2.1.19. **resource**

identifiable asset or means that fulfills a requirement

(SOURCE: ISO:19115-1:2014 Geographic information — Metadata — Part 1: Fundamentals)

Note 1 to entry: A web resource, or simply resource, is any identifiable thing, whether digital, physical, or abstract.

2.1.20. **slice**

subset of a coverage for a single coordinate along a dimension axis, for which the resulting coverage is reduced by one dimension

2.1.21. **subsetting**

operation whose result is a subset of the original set (e.g. trim or slice operations on a coverage)

2.1.22. **tile**

geometric shape with known properties that may or may not be the result of a tiling (tessellation)process. A tile consists of a single connected "piece" without "holes" or "lines" (topological disc).

(SOURCE: OGC 19-014r1: Core Tiling Conceptual and Logical Models for 2D Euclidean Space)

Note 1 to entry: "tile" is NOT a packaged blob of data to download in a chunky streaming optimization scheme!

2.1.23. **tiling**

in mathematics, a tiling (tessellation) is a collection of subsets of the space being tiled, i.e. tiles that cover the space without gaps or overlaps.

(SOURCE: OGC 19-014r1: Core Tiling Conceptual and Logical Models for 2D Euclidean Space)

2.1.24. **trim**

subset of a coverage between lower and upper bound coordinates along a dimension axis which does not reduce the dimensionality of the resulting coverage

2.1.25. **workflow**

sequence of processes (whether local or remote) to be executed, possibly with pre-defined and/or external input values, whose output(s) may serve as input(s) to subsequent processes part of the same workflow, whereas those subsequent processes have a dependency on the completion of the operations generating their inputs.

Note 1 to entry: The workflow as a whole may itself take inputs and generate outputs, and may also be encapsulated as a single process.

Note 2 to entry: A workflow (or part of it) may be executed in a distributed manner (e.g. for specific area and/or resolution of interest) if some or all processes involved can be computed in a localized manner.

2.2. Abbreviated terms

ADES	Application Deployment Execution System
API	Application Programming Interface
COG	Cloud Optimized GeoTIFF
CRS	Coordinate Reference System
CWL	Common Workflow Language
DEM	Digital Elevation Model
DGGS	Discrete Global Grid System
EO	Earth Observation
ESA	European Space Agency

EVI	Enhanced Vegetation Index
FOSS	Free and Open Source Software
GDAL	Geospatial Data Abstraction Library
GDC	Geo Data Cube
GPKG	GeoPackage
GPU	Graphical Processing Unit
JSON	JavaScript Object Notation
LoD	Level of Detail
ML	Machine Learning
MOAW	Modular OGC API Workflows
NDVI	Normalized Difference Vegetation Index
NRCan	Natural Resources Canada
OGC	Open Geospatial Consortium
STAC	SpatioTemporal Asset Catalog
TIE	Technology Integration Experiment
TIFF	Tagged Image File Format
TMS	Tile Matrix Set
UML	Unified Modeling Language

3

INTRODUCTION

3 INTRODUCTION

Section 4 introduces the concept of a Geo Data Cube. It describes the situation prior to the Testbed-17 work and discusses the requirements set by the sponsoring organizations.

Section 5 discusses the approach to standardizing a GDC API. This includes exploring different OGC API specifications selected for experimentation during the Testbed. These APIs included *OGC API — Common*, *OGC API — Coverages* and *OGC API — Processes*. These current and draft API standards form the basis for the GDC API. Additional specifications providing a basis for the GDC API include the *OGC API — Environmental Data Retrieval* standard, as well as the draft *Data Access and Processing API (DAPA)* specification and the draft *OGC API — Records* specification. These additional specifications could also be integrated within this framework.

Section 6 describes the experimentation and results pertaining to the integration and use of a Machine Learning model within a GDC API.

Section 7 provides an overview of the GDC API services developed and improved for the Testbed-17 GDC task.

Section 8 provides an overview of the GDC API clients developed and improved for the Testbed-17 GDC task, also relating experiences with the use of Augmented Reality and GeoPose together with a GDC API.

Section 9 lays out a path forward for standardization of a GDC API.

Annex A selects GDC API capabilities consisting of current and draft OGC API standards and conformance classes implemented by the Testbed participants.

Annex B summarizes the Technology Integration Experiments conducted between the different server and client components.

4

GEO DATA CUBE CONCEPTS

4 GEO DATA CUBE CONCEPTS

This chapter introduces the concept of a Geo Data Cube and the requirements provided by sponsoring organizations guiding this initiative. Literature consulted to inform these concepts includes

- reports from past OGC initiatives (OGC 21-013 OGC 21-008 OGC 20-016 OGC 20-025r1 OGC 20-035 OGC 20-018 OGC 20-039r2 OGC 20-041 OGC 20-091 OGC 20-073 OGC 19-070 OGC 19-027r2 OGC 19-026 OGC 18-038r2 OGC 18-049r1 OGC 18-050r1 OGC 18-046),
- an OGC community best practice (OGC 18-095r7),
- an OGC discussion paper (OGC 21-033),
- articles (datacubeManifesto viewBasedModelDataCube doiPavingIncreased copernicusEarthSystem),
- documentation for data cubes and APIs (openEOAPI sentinelhubAPI up42Doc climateDataStoreAPI roocsTools earthSystemDataCube), as well as
- Wikipedia entries (wikiDataCube wikiOLAPcube).

4.1. What is a Geo Data Cube?

Before considering what functionality a Geo Data Cube (GDC) API should provide, clarifying what is meant by a *Geo Data Cube* is important:

- A *data cube* is a multi-dimensional (“n-D”) array of values (OGC 18-095r7).
- A data cube persistently stores and provides efficient access to multi-dimensional information (although this is not meant to exclude one-dimensional information).
- A *Geo Data Cube* is a data cube for which some dimensions are geospatial in nature (such as latitude and longitude, projected easting and northing, or elevation above the WGS84 ellipsoid).
- In terms of functionality, a geo data cube can be considered a multi-dimensional field including spatial dimensions, and often temporal dimensions as well.

Conceptually, this is essentially the same as a coverage as defined in ISO 19123 / OGC Abstract Topic 6:

- A coverage is a feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain (OGC 07-011).

Where a Geo Data Cube is established on the basis of a coverage, it may be referred to as a Geospatial Coverage Data Cube. Section 4.2 of the Community Practice (OGC 18-095r7) provides a definition of the term Geospatial Coverage Data Cube. For the purpose of this ER, the term Geospatial Coverage Data Cube is considered a specialization of the term Geo Data Cube.

An API may offer access to information from a particular dataset organized as separate data cubes. Each cube could, for example, represent a different type of information, or a different imagery product or collection, and provide an integrated access to these multiple data cubes. Each of these GDCs would be equivalent to an individual *coverage* in the draft *OGC API — Coverages* specification and to a *collection* in the draft *OGC API — Common — Part 2: Geospatial Data* specification as well as in the other OGC API standards and draft specifications for data access (*Features*, *Tiles*, *Maps*, *EDR*…).

The data cube and Geo Data Cube terms are also sometimes used to refer to a service or platform providing access to such data cubes, or to a federation of such services or platforms. For the purpose of this ER, unless explicitly stated otherwise a GDC refers to a single collection of multi-dimensional data.

This figure from openEO illustrates a multidimensional data cube:

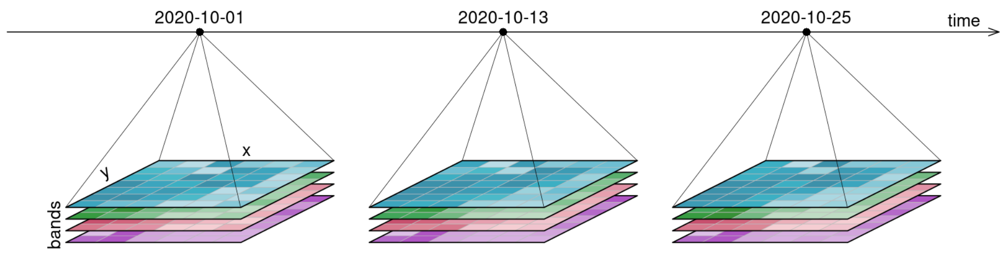


Figure 1 — Illustration of a data cube with multiple imagery bands and time axis

NOTE: The focus of openEO is developing an open API to connect R, Python, JavaScript and other clients to big Earth observation cloud back-ends in a simple and unified way.

4.2. Goals of a Geo Data Cube API

In addition to providing efficient access to the data, a GDC API may also enable performing analytics close to the data. Analytics could range from simple aggregation and arithmetic to more complex algorithms such as machine learning predictions. A GDC API may also allow discovering data or processing capabilities available either from within the same API or elsewhere.

There may also be interest in integrating visualization and/or data or analytics management capabilities in some deployments.

The purpose of an OGC GDC API is to enable the use of these capabilities in client applications to derive useful insights from very large collections of data, in particular multi-spectral imagery routinely collected by Earth Observation satellites such as the US Landsat, EU Sentinel-2 and Canadian RADARSAT. Such insights are of particular importance in the context of solving global challenges like climate change.

4.2.1. Needs of end-users and application developers

Two main categories of users must be considered in the design of an OGC GDC API. The first category is that of end-users, such as climate researchers. These end-users are less concerned with the technical aspects of the API as they will likely be using the API indirectly through client applications. Their primary concern is that a standardized GDC API enables interoperability between multiple client applications and services providing datasets and analytics for a common baseline of functionalities meeting their needs. However, server-side OGC API implementations are also intended to be directly accessible by end-users, such as implementing an HTML representation of resources which may readily offer a minimum amount of functionality typical of a client. This should be considered in the design to ensure that it is possible to present the API in a user-friendly manner.

The second category of users is the developers. Developers, who will be using the GDC API to build client applications, are the primary users of the API. These users expect a uniform API that can be used with different services and datasets. They are concerned primarily with the API providing the functionality needed for their application. The ease with which they can learn how to access that functionality and how interoperable and efficient this functionality is in different implementations of the API is also important.

Finally, the back-end developers, although technically not users of the API, must implement the API functionality and are often concerned with the amount of effort required to understand and develop a service conforming to the API specification, with how easy it is to map each operation to their capabilities to provide access to data and analytics, and how possible it is to efficiently map this functionality.

Clearly, all of these targeted users and developers desire a convenient, simple and uniform API. Several of the OGC APIs being considered for use in the GDC API are still at a draft stage. If these draft APIs do not currently satisfy requirements for convenience, simplicity and uniformity, attempts should be made to improve them to address those needs. This approach is better than defining yet another completely distinct API that would further fragment the OGC API standards base and reduce interoperability.

4.2.2. Requirements from sponsors

From the *Testbed 17 Call for Participation*, the following sponsor requirements for the *Geo Data Cube API* task were identified:

- Define an OGC API leveraging existing building blocks for Geo Data Cubes.
- Support access and processing in the cloud.

- Support data discovery and querying information of diverse collections of data, including spatial and temporal resolution, interoperability with STAC, registries and catalogs.
- Support interoperability of data formats and access methods: Cloud Optimized GeoTIFF which supports direct HTTP range requests, OGC WxS, OGC APIs.
- Support interoperability across different cloud providers.
- Support interoperable workflows for terrestrial & marine elevation, forestry information that can:
  - Process / extract information from forestry imagery;
  - Handle formats that enable interoperability such as for images/point clouds;
  - Derive insights & change prediction from spatiotemporal data.
- Support interoperability between different Geo Data Cubes / APIs, as well as between GDC API and offline.
- Support integration of terrestrial & marine elevation data from separate Geo Data Cubes.
- Support for integration with advanced technology such as Machine Learning.

4.2.3. Data access

A GDC API should support accessing different types of data. Examples are data cubes for regular and irregular gridded raster data and data defined by vector features geometries of different dimensionality (including point clouds with a large number of points). The domain of the data cube should be capable of supporting one or more spatial and/or temporal dimensions, and possibly additional types of dimensions.

The values associated with a direct position in the data cube (*range values* in coverages terminology) should support both discrete (e.g. land cover category) and continuous (e.g. radiance) observed properties (e.g. the bands / sensor type in EO imagery).

During the GDC work in Testbed-17, some confusion was noted as to what should be presented as a field / property / value of the range vs. what should be presented as an axis of the domain. Topic 6 of the OGC Abstract Specification makes a clear distinction between the two. A dimension is part of the direct position for which values are available, and must be defined in the Coordinate Reference System (CRS) for the overall domain. Most often dimensions are limited to the spatiotemporal domain. Another use case for an additional dimension would be a parameter for which properties were observed at several different values or at a continuous range of values, throughout the other aspects (e.g. spatiotemporal) of the domain.

A data cube may itself be made up of smaller data cube pieces (e.g. imagery scenes or granules). Having the GDC API providing direct access to these scenes would be useful. This could enable an application to more accurately reflect the original data characteristics of those scenes making up the data cube. An example is supporting their native CRS (e.g. Universal Transverse Mercator

(UTM) coordinate system zones in Landsat-8) while providing an easier-to-access unifying data cube for the different scenes through a single CRS and a fixed resolution.

Describing these aspects of the data and providing convenient and efficient access to it in its raw form are two key capabilities for a GDC API.

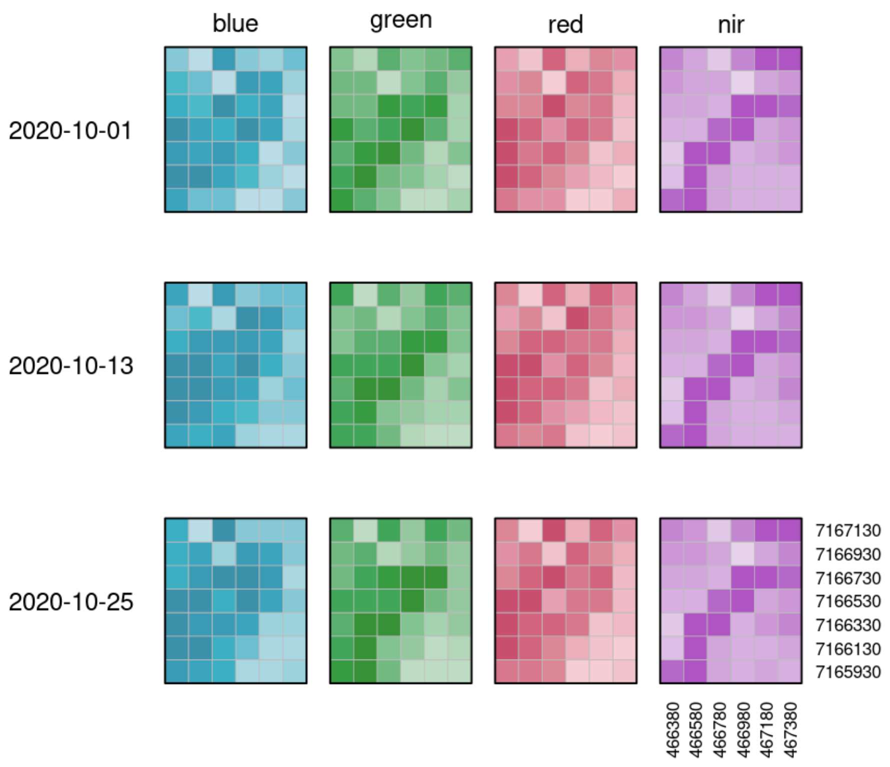


Figure 2 — Figure from openEO showing layers of a data cube across imagery bands and time axis.

4.2.3.1. Data description

A GDC API needs a mechanism to describe the domain (e.g. the spatiotemporal extent) and the range (the type of values, or observed properties, or fields) defined for each direct position within the data cube. For describing the domain, one or more CRS must be clearly identified and associated with the axes to fully cover the spatiotemporal continuum of the data. For regular axes of a grid, a resolution must be specified, while for irregular axes of a grid, direct positions must be enumerated along the axis.

For describing the range, a list of fields must be enumerated, each ideally annotated with a semantic association, a unit of measure and additional metadata if appropriate. Statistics for the values found in the data cube for each field would also be very useful information, as well as clarifications as to how the data is encoded (e.g. for encodings where it is not possible to provide these additional clarifications internally).

4.2.3.2. Data retrieval

A GDC API needs a simple mechanism to retrieve data in convenient encodings, without imposing a particular logical data model on those physical encodings.

Although retrieving an entire data cube as a single operation is possible, a common use case is to retrieve only a certain portion of interest. This is often a particular spatial area which also corresponds to a useful resolution (native resolution for small areas, but down-sampled for larger areas), resulting in a constant maximum response size. Support for retrieving only part of the data is critical for large collections of data for which retrieving everything is unnecessary and a waste of processing and bandwidth resources at both ends of the API, and often impractical or impossible. Such subsetting and down-sampling capability can be implemented efficiently with backing data stores supporting overviews / tile pyramids, as in Cloud Optimized GeoTIFF (COG) and Tile Matrix Sets (OGC 17-083r4). Directly exposing the multi-resolution tiles through the API to clients may improve performance by aligning requests with the data store's internal organization, and thus enable efficient caching of responses on both the server and client side. Requesting a subset of a temporal dimension may also be a desirable capability.

In coverages terminology, a subsetting operation reducing dimensions (e.g. from 3D space + time to 2D space only) is called *slicing*. A subsetting operation preserving the same number of dimensions is called *trimming* (i.e. requesting a range of values for each axis in the subsetting operation). A GDC API may also support supersampling, but this is of less value for accessing raw data as it wastes bandwidth and processing resources, and could always be done on the client-end if necessary. However, supersampling may be necessary to present a data cube of a uniform resolution where the resolution of the data source in fact is variable.

Additionally, a client may only be interested in requesting values only for some of the range (observed properties / fields, e.g. specific imagery bands of interest).

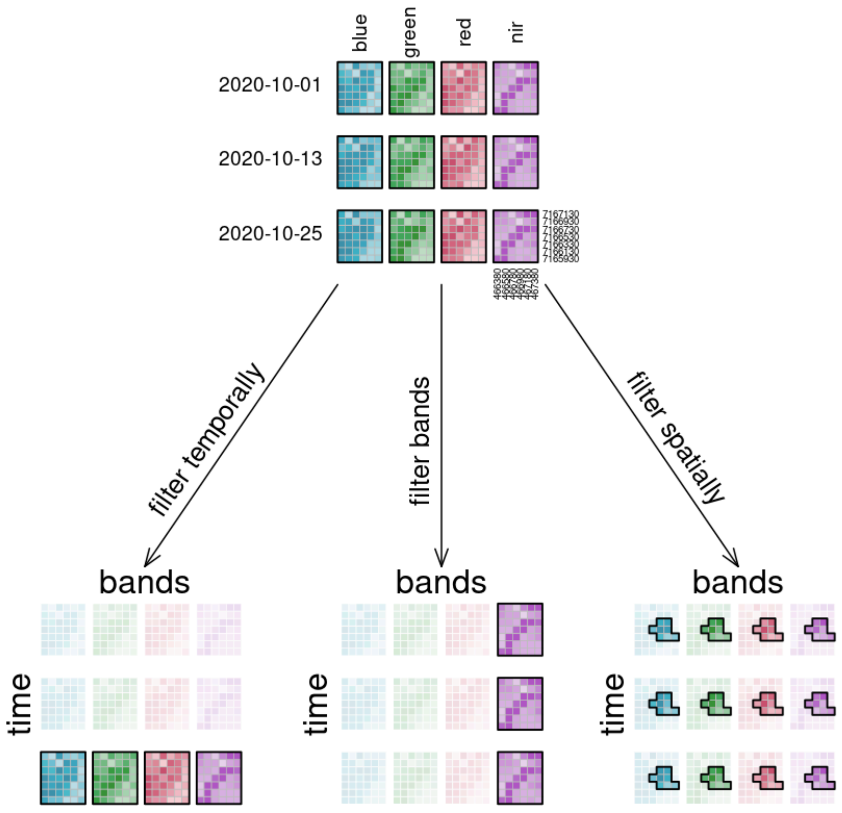


Figure 3 — Figure from openEO illustrating data trimming by time, range subsetting (selecting a single band) and intersection with a spatial area.

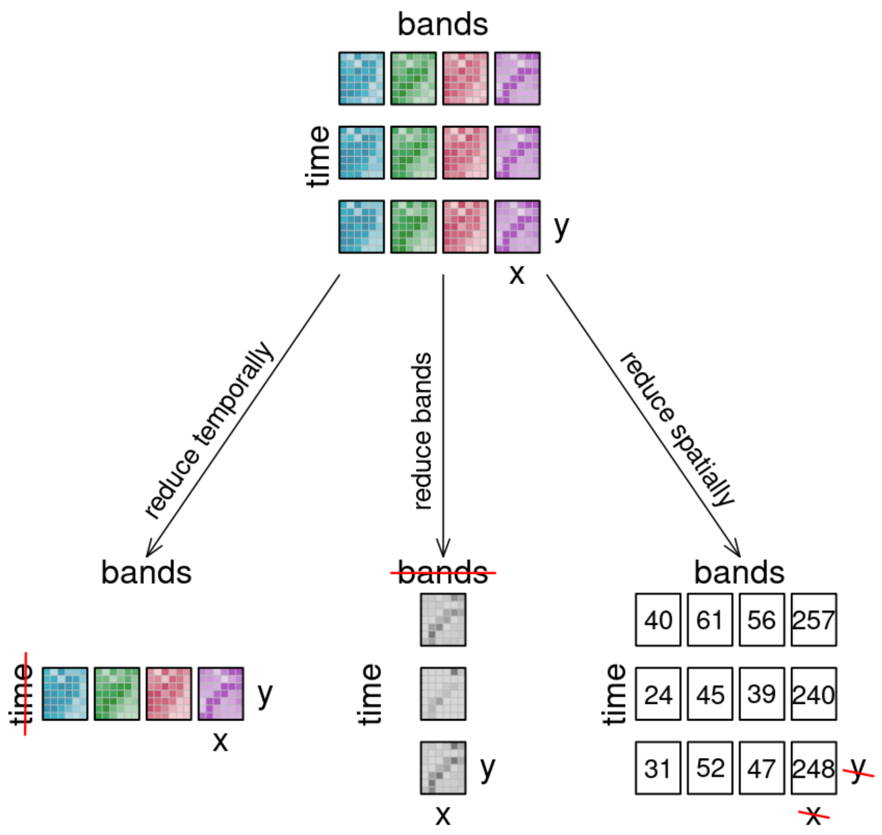


Figure 4 — Figure from openEO illustrating data slicing, reducing dimensions of the data.

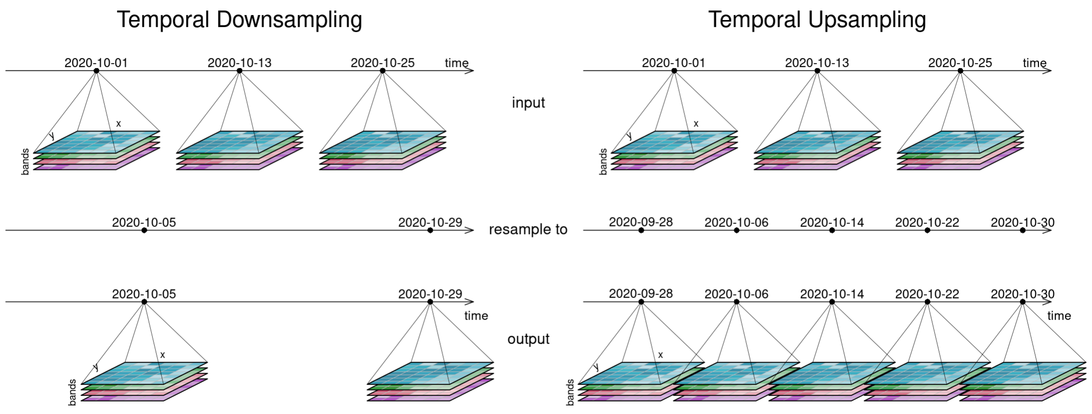


Figure 5 — Figure from openEO illustrating temporal resampling.

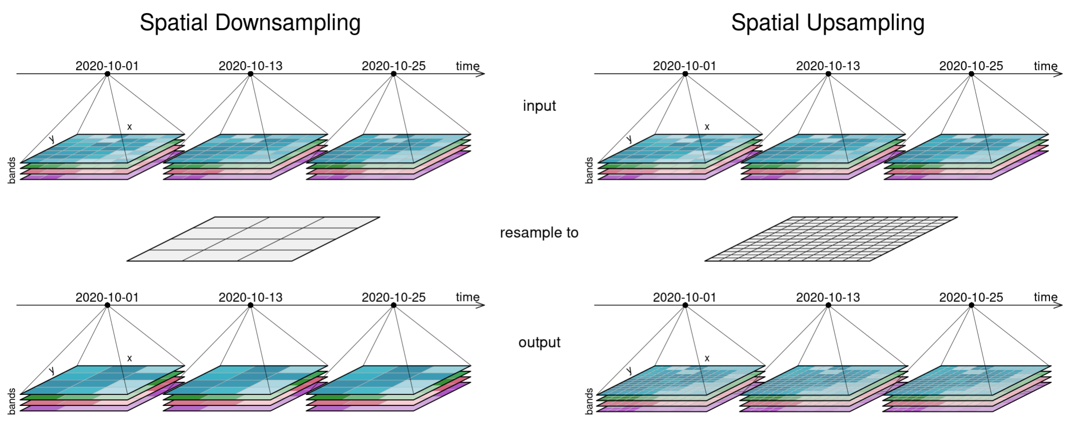


Figure 6 — Figure from openEO illustrating spatial resampling.

The participants also discussed the need for more advanced data filtering capabilities. Examples are returning only portions of the data by comparing values from particular fields (e.g. Quality of data band with cloud cover information) and/or properties of metadata (e.g. cloud cover) associated with a particular scene as a whole (for coarser but faster filtering). This is particularly useful in the context of reducing dimensionality such as when wishing to retrieve a cloud-free 2D mosaic comprised of multiple EO imagery scenes, from the same spatiotemporal data cube, which were captured closest to or before a specific slicing time.

Another use case, of particular relevance to meteorology for example, is subsetting data based on more complex patterns, such as trajectories or corridors, providing the API with a detailed geometry of the area of interest and retrieving only relevant values.

In a sense, these advanced retrieval capabilities such as filtering, re-sampling and even subsetting to some extent could be considered a form of simple analytics. They are discussed here because they still integrate well within a data retrieval request. This is because they can be expressed as simple query parameters and combined together, with the resulting response sharing many of the characteristics of a plain request for the original data.

4.2.4. Analytics

The ability to perform data analytics close to the data is an important capability of a GDC API, improving performance, saving bandwidth, time and costs. The participants noted that the community has struggled so far in defining and adopting a simple and interoperable approach to analytics, with several implementors coming up with their own approach.

A comprehensive GDC API should cover both simple analytics such as aggregation over time or band arithmetic calculations (e.g. for vegetation index calculation), as well as more complex analytics such as prediction from machine learning.

Commonly accessed and simple to express analytics capabilities should be very simple and convenient to use.

For simple cases such as aggregation and band arithmetic calculations, it may be possible to standardize a simple language to express those aspects which could also be integrated with data retrieval requests. For slightly more complex analytics, the concept of a well-known process, standardized to expect a specific set of inputs and return a specific type of output, might be useful. Some well-known processes could be defined to expect a particular language defining the analytics to perform expressed in popular raster expression or coverage processing languages adopted by particular communities. Others could be defined to implement specific algorithms, such as calculation of the Normalized Difference Vegetation Index (NDVI) or contour generation. Other examples of well-known processes previously experimented with in the context of OGC APIs include for example calculating a route OGC 21-000 and rendering a map.

4.2.5. Discovery

In the context of using a single API to discover and access data, a large number of data cubes may be available which could be filtered based on specific aspects (e.g. spatial or temporal extent, resolution, sensor types). Also if a single data cube is made up of several scenes, which typically also have associated metadata, it would be useful to query for scenes of interests, possibly directly integrated as part of data retrieval requests. Similarly, discovering relevant algorithms from a large set of analytics capabilities available would be very useful. Establishing which data cubes and which algorithms can be used together is also of great interest. The possibility to catalog data and algorithms, as well as provide the ability to discover relevant data and analytics capabilities, would also be very useful for end-users.

4.2.6. Visualization

Although not an essential capability because clients can implement their own visualization capabilities with better performance and using less bandwidth, a GDC API implementation could also decide to provide access to server-side visualization capabilities. However with the ubiquity of Graphical Processing Units (GPUs) in modern hardware, leaving visualization to the client has many benefits. The HTML representation for the raw data resources of a GDC API could still incorporate visualization capabilities, without requiring the definition of those capabilities in the API itself.

4.2.7. Managing data and algorithms

The ability to manage data and algorithms is not necessarily a capability required by end-users, especially if the GDC API supports the creation of ad-hoc workflows referencing arbitrary data sources and processes (as researched in the *Modular OGC API Workflows project* and defined in OGC API — Processes — Part 3). However, this is a useful capability allowing larger organization or multiple parties to manage and maintain data cubes.

4.3. Nature of a Geo Data Cube API

While providing most of the functionality for the GDC API with existing or currently ratified OGC APIs is possible, specific cases may require complex queries or specific processes being available on a particular server. The goal of the GDC API is the common functional baseline across different servers, which enables common capabilities for multiple distributions and data sources. By providing this common functionality, GDC API implementations can close functional gaps between different internal backends.

By using the existing OGC standards and OGC APIs “behind the scenes”, it is also very likely and desired by implementers to reuse existing processing algorithms and other infrastructure in order to provide the GDC capabilities. The current and draft OGC API Standards are highly flexible in terms of data access, processing, dynamic registration and clear discovery of data and functionality. When paired with a well-defined minimum set of functionality for the GDC API, there is room for reuse of components for filling functional gaps and room for competition for larger-scale processing, integration of old and new data sources and implementation diversity.

5

AN API FOR GEO DATA CUBES

5 AN API FOR GEO DATA CUBES

5.1. OGC API framework for providing GDC capabilities

The Geo Data Cube API is proposed as a profile with extensions of the OGC API standards baseline enabling efficient discovery, access and analytics capabilities for use with multi-dimensional geospatial data.

Since the OGC API baseline already provides most of the required capabilities for defining the GDC API, the participants considered leveraging existing OGC API approved standards and draft specifications as potential building blocks. Some desired capabilities currently missing or not yet fully specified were identified. New extensions to be further defined are proposed as additional OGC API building blocks.

A limited selection of these current and/or future building blocks should provide coherent and convenient access to multi-dimensional data sets, as well as frequently used analytics capabilities (e.g. resampling, filtering, pre-defined processes), from different providers. These building blocks could also be implemented as a façade to integrate other, possibly distributed, implementations of these same building blocks. Alternatively, a façade could also be built on top of other pre-existing types of data sources, such as WxS compliant services (in particular the Web Coverage Services (WCS) and the Web Processing Service (WPS)) or static HTTP servers offering Cloud Optimized GeoTIFF (COG) with support for HTTP range requests.

The following diagram illustrates the GDC API architecture showing how multi-dimensional data can be:

- ***stored*** in a variety of backends such as databases, local files and cloud-based object storage,
- ***indexed*** for discovery and efficiency using e.g. spatial indexes, tile pyramids, and Discrete Global Grid Systems,
- ***represented*** as a Geo Data Cube resource corresponding to the spatiotemporal field (as well as the *OGC API — Common — Part 2: Geospatial data collection* and often to a *coverage*),

- ***transformed*** by performing operations such as:
    - *resampling* to better suit the resolution of interest,
    - *subsetting* along axes for the area or time of interest,
    - *aggregation* in different ways across one or more dimensions,
    - *filtering* based on range values, scene metadata, or spatiotemporal comparisons with supplied geometry or other data sources,
    - *band arithmetic calculations* creating new values based on existing ones,
    - *processing* algorithms which are pre-defined, customizable via expressions or a processing language, or by custom user-deployed processes,
    - complex *workflows* either pre-registered or supplied by clients in an ad-hoc manner by referencing local or remote processes and data cubes,
- ***queried*** by data access mechanisms specified in OGC API building blocks specifications which defines and triggers those transformations, and
- returned as ***outputs*** to the end-user in a negotiated suitable format.

The draft *OGC API — Processes — Part 3: Workflows & Chaining* specification enables the chaining of data cubes as an input to processes and the output of a process to be presented as a data cube resource, including between remote processes and data cubes distributed across federated services.

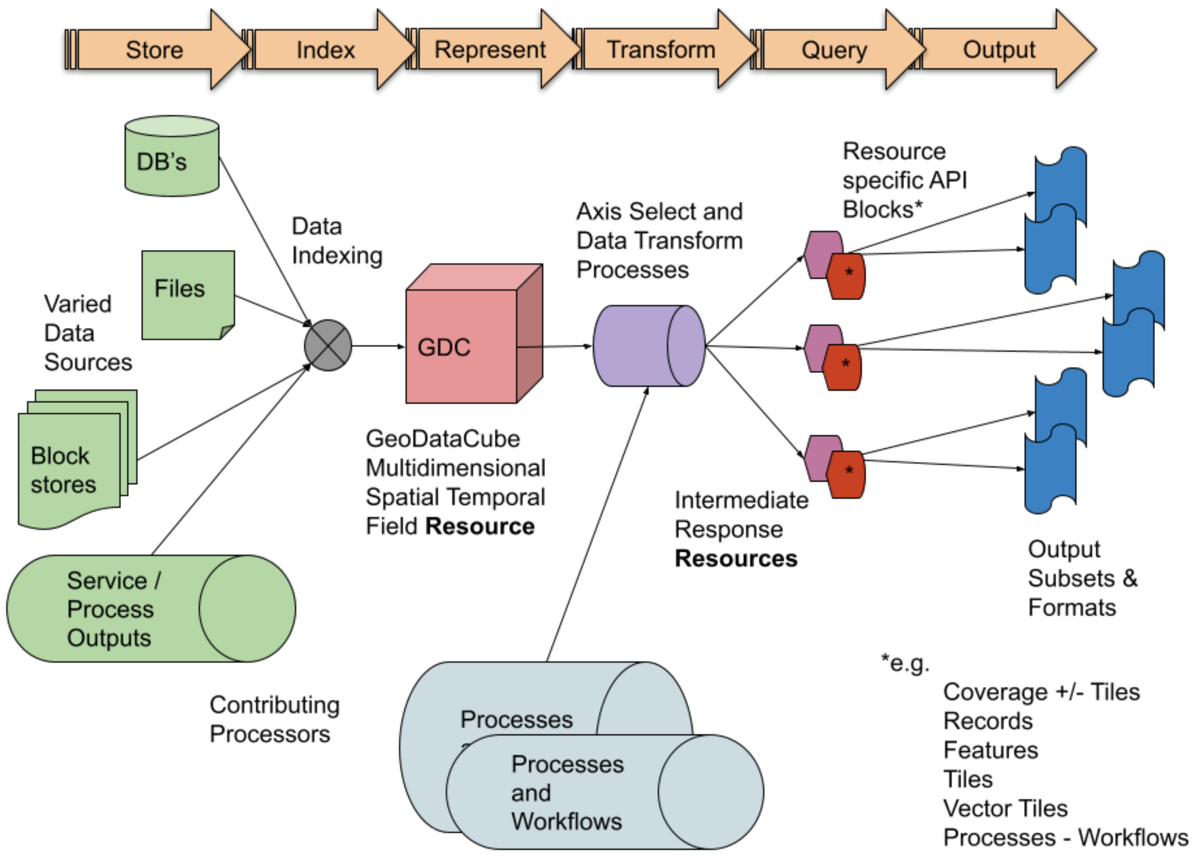


Figure 7 — Geo Data Cube API Architectural Framework

The *OGC API — Common* specification provides a cohesive framework to integrate these different building blocks into an API and a minimum capability to be implemented as part of a GDC API. Part 1 of *OGC API — Common* (OGC 19-072) defines the concept of a landing page, conformance declaration and API description. Part 2 (OGC 20-024) defines the listing and basic description of available collection of spatiotemporal data.

5.2. Data access

A number of OGC API specifications enable efficient access to the raw values from a spatiotemporal dataset:

- *OGC API — Coverages*,
- *OGC API — Tiles*,
- *OGC API — Features*,

- *OGC API — Environmental Data Retrieval (EDR)* and
- *Data Access & Processing API (DAPA)*.

As of March 4th, 2022, *OGC API — Features* and *OGC API — EDR* were approved OGC standards. *OGC API — Tiles* is a draft specification nearing completion. *OGC API — Coverages* is a relatively stable draft specification which has been proven to be interoperable through several successful Technology Integration Experiments (TIEs) in previous initiatives. *DAPA* is in the form of an OGC Testbed-16 Engineering Report.

Participants noted that although *OGC API — EDR* and *DAPA* define some new capabilities not yet covered by *OGC API — Coverages*, functionality specific to data cube access overlaps between the three. This overlap is especially in terms of describing the domain and range (fields) of those data cubes and in requesting a potentially downsampled subset of the data. This may be problematic within the OGC API framework where different building blocks should be complementary rather than competing for adoption. Therefore, the testbed participants recommend that some re-alignment be considered between *OGC API — EDR* and *OGC API — Coverages* for the overlapping aspects (data description and cube queries), and that the new functionality covered by DAPA be integrated within *OGC API — EDR* and/or *OGC API — Coverages* rather than defining a whole new API that would further reduce interoperability by fragmenting OGC API specifications.

Although an argument for separate *EDR* and *DAPA* standards is convenience and simplicity, those are also goals of *Coverages*. The successful implementations built and the TIEs performed in a short time towards the end of this initiative demonstrate that these goals were achieved reasonably well. Furthermore, there is still a possibility to improve the *Coverages* specification since it is still at a draft stage. Having to re-implement the same functionality multiple ways to achieve interoperability with different systems will always introduce more complexity, regardless of how simple an individual API happens to be.

5.2.1. OGC API — Coverages

The draft OGC API — Coverages OGC 19-087 specification defines building blocks for describing and retrieving multi-dimensional data from a coverage. The Coverages API work is based on Topic 6 of the OGC Abstract Specification (ISO 19123). A common type of coverage is gridded (raster) data, but coverages are not limited to gridded data and can include, for example, point clouds.

A description of a coverage's *DomainSet* always includes a Coordinate Reference System, as well as the lower and upper bound for which data can be retrieved. Coverage grids can either be spaced regularly (described by a fixed resolution for a regular axis) or irregularly (in which case positions along the irregular axis must be explicitly listed when describing the domain).

The fields (range values, e.g. the different bands available for imagery, or observed properties) available for retrieval make up the *RangeType*, and include a name, a semantic link and a unit of measure where applicable.

The description of the coverage's *DomainSet* and *RangeType* are linked from the *OGC API — Common* collection, and can be embedded within the collection description itself by using a

JSONPointer. In addition to possibly being able to retrieve the entire raw data in a single request at `/coverage`, additional conformance classes are defined in *Part 1 — Core* allowing to retrieve only the area and resolution interest.

5.2.1.1. Subsetting

The subsetting conformance class supports retrieval of a subset of the whole coverage by trimming (retrieving values between a lower and upper bound coordinates along an axis, which maintain the same dimensionality) or by slicing (retrieving values for specific coordinates along an axis, which reduces dimensionality) the coverage.

5.2.1.2. Scaling (re-sampling)

The scaling conformance class supports down-sampling or up-sampling data by defining a scale factor, either for all axes or for an individual axis, or a desired number of resulting cells in the response along specific dimensions.

5.2.1.3. Tiles

A conformance class for *Coverage Tiles* is defined, leveraging the *OGC API — Tiles* and *2D Tile Matrix Sets* specifications. A coverage tile request is equivalent to a coverage request with specific combination of subsetting and scaling parameters.

5.2.1.4. Range subsetting

A range subsetting conformance class defines how only some of the fields can be selected for retrieval, e.g. only a particular band or observed property.

5.2.1.5. Filtering

The *Coverages* API does not yet define filtering capabilities, but this was identified as a very useful extension that could leverage the draft CQL2 specification that is being developed by the Features API SWG. An issue was already filed on this topic. In addition to filtering based on the range values themselves, filtering could also be done based on the metadata of scenes making up the coverage. These *queryables* that can be used in a filter expression could all be described together as in the filtering extension for *OGC API — Features* (Part 3). A sorting capability would also be useful when flattening a 3D coverage (2D space + time) to a 2D image mosaic, so that e.g. the scene and/or cell with the least cloud cover is what gets returned. An example of such a request could look similar to:

```
/collections/landsat8/coverage?
filter=scene.CLOUD_COVER<30&sort=L8CloudCover(BQA)
(desc),scene.SCENE_CENTER_TIME(asc)
```

assuming content-negotiation for a 2D coverage format (e.g. GeoTIFF) and the server supports flattening multiple ordered overlapping scenes to a 2D coverage, with `L8CloudCover()` a function to grab the cloud cover for individual cells from the BQA band bits).

5.2.1.6. Varying resolution

The *Coverage API* does not currently support specifying different resolutions for different areas or for different fields of a data cube. This may be useful for example in the context of polar regions when using a CRS such as the EPSG:4326 CRS, or for a data cube made up of different scenes which may be in a different CRS than the unified data cube, or for bands of varying resolution such as the panchromatic bands in Landsat-8 which is twice the resolution of other bands. The participants suggest that extensions be developed for these capabilities. For example, Coverages API #142 proposes supporting bands of varying resolution by allowing a `range-subset` query parameter on the domainset resource. Including a `subset` query parameter when requesting the domainset could potentially also result in a resolution specific to the area being subset. This approach shares similarity with the suggestion to support a `crs` parameter in issue #129 for returning a domainset specific to that CRS.

5.2.1.7. Cloud Optimized GeoTIFF (COG)

In addition to providing a suitable back-end to facilitate support for the subsetting and scaling conformance classes with tiles and overviews, an *OGC API — Coverages* implementation may potentially expose its `/coverage` resource as a Cloud Optimized GeoTIFF with support for HTTP range requests, as suggested in this comment.

5.2.2. OGC API — Tiles

The draft OGC API — Tiles specification (OGC 20-057) specifies how to retrieve data on a tile-by-tile basis. The Tiles API is based on the 2D Tile Matrix Set and TileSet metadata draft specification (OGC 17-083r4), which is a revision of the 2D Tile Matrix Set Standard. With the Core conformance class of *OGC API — Tiles*, tiles can be retrieved according to registered TileMatrixSets using a simple URL template with three variables: a *tile matrix* identifier (usually a zoom level), a *tile row* and a *tile column*. The content of the tiles can be rendered map tiles, raw gridded coverage values, vector features, or other things like point clouds or 3D meshes. The *TileSet* conformance class adds support for describing a tileset by providing such a template, indicating limits for these variables, a link to the TileMatrixSet definition, a URI for registered TileMatrixSets, as well as additional metadata.

Having a pre-defined pyramidal tiling scheme as opposed to clients requesting arbitrary scale factors and subsetting bounds facilitates caching on both the client & server side. For example, different visualization clients panning and zooming in the same area would make the exact same fewer requests only reaching the next binary zoom level or bringing in the next 256×256 pixels tiles, as opposed to potentially making misaligned requests for their exact viewport configuration any time it changes.

5.2.3. OGC API — Features

The OGC API — Features Standard specifies an API to retrieve vector features. Features can be retrieved individually by ID, or as a feature collection up to a certain limit beyond which pagination into multiple requests is necessary. The Core standard supports filtering by intersection with a bounding box and or a temporal range. *Part 2* of *OGC API — Features* is also an approved standard defining supports for CRSes other than CRS84. *Part 3* of *OGC API — Features* (OGC 19-079r1) implements filtering capabilities. Future extensions may also define the ability to create, replace, update and delete features, as well as to retrieve generalized (simplified) and clipped features.

5.2.4. OGC API — Environmental Data Retrieval

OGC API — Environmental Data Retrieval (EDR) is an approved OGC Standard specifying multiple ways to query a data cube, including for typical meteorological use cases such as data along a trajectory or within a corridor. The Testbed-17 Participants did not implement support for the EDR API, but performed a quick comparative analysis with other OGC API specifications for accessing data cubes. EDR leverages building blocks of OGC API — Common Part 1 & 2. However some discrepancies (see EDR issues #331, #332, and #333) were identified in how the spatiotemporal extents are described which would have made it impossible to offer the same collection of data through both *OGC API — EDR* and other draft specification or OGC API standard (such as *OGC API — Coverages*, *OGC API — Tiles*, or *OGC API — Features*), unless the collection of data (data cube) is declared as a separate collection resource. As a result of those findings, the EDR SWG has corrected the standard to address most of those issues.

In order to resolve issue #332, the EDR SWG opted to introduce a property separate from *interval* called *values* to describe the *DomainSet* of the collection of data. *OGC API — EDR* also specifies a mechanism to describe the fields (*RangeType*) directly in the collection description, which differs from the way it is being specified in *Coverages*. The participants also noted that the *cube* query defined by *OGC API — EDR* overlaps greatly with the *OGC API — Coverages* specification and its subsetting and scaling conformance classes, and that the two could perhaps be harmonized and re-integrated, similarly to how *OGC API — EDR* references *OGC API — Features* for its *items* query. Although *OGC API — EDR* defines several types of queries, it does not define separate conformance classes for each of those queries. Splitting these into multiple conformance classes stood out as something that could facilitate testing compliance or requiring the capability for specific types of queries. Another observation was that the `parameter-name` query parameter supporting the selection of properties to be returned (range subsetting) could be defined the same way as `properties=` across *OGC API — Features*, *OGC API — Coverages* and *OGC API — EDR*. An issue was filed to propose changing `range-subset=` to `properties=` in *Coverages*. These aspects would greatly benefit from a re-alignment to maintain consistency within the OGC API family of standards, and avoid re-defining the same functionality in different ways.

5.2.5. Data Access & Processing API (DAPA)

The Data Access & Processing API (DAPA) Engineering Report (ER) (OGC 20-025r1) provides a draft specification and documents work performed in the OGC Testbed-16 task. The DAPA ER suggests a number of different designs for accessing data cubes. This includes:

- a capability to describe the fields of a data cube, overlapping with those defined in *OGC API — EDR* (parameter names) and *OGC API — Coverages* (RangeType),
- point and area sampling capabilities, overlapping with similar capabilities in *OGC API — EDR* and *OGC API — Coverages*,
- some filtering capabilities, similar to the proposed filtering extensions for coverages,
- the ability to query for data within a specified geometry also found in *OGC API — EDR* queries using a *geom* query parameter (which could also be implemented as an extension to *OGC API — Coverages* as proposed in issue #52),
- the ability to select fields / observable properties to be returned also found in the other OGC APIs (those suggested to be unified as `properties=`), as well as
- some analytics capabilities covered in a separate section below. Rather than defining a completely separate new API, the proposed capabilities not already present in *OGC API — EDR* and *OGC API — Coverages* could instead be integrated in these core specifications or in extensions. Testbed-17 participants did not get the chance to explore DAPA in details, however Wuhan University initiated support for the API in its service.

5.2.6. Scenes API

Large data cubes such as those resulting from Earth Observation satellites that are constantly capturing new data are often made up of individual *scenes* with more being added continuously. Each of these individual scenes may be stored in a different native CRS (such as UTM zones) and/or spatial resolution. The ability to present these multiple scenes as a unified data cube / coverage, while still being able to directly access the individual scenes as well would prove useful and forms the basis for proposing a Scenes API. This API would have both a data access as well as a discovery aspect (of individual scenes of interest but from within a single collection / coverage / data cube). As discussed above, preserving and exposing the metadata for these scenes as queryables would also enable integrating scene discovery directly in a data request (e.g. at `/coverage`), facilitating generating a cloud-free mosaic. This is an important use case for which the SpatioTemporal Asset Catalog (STAC) has been used but without this suggested level of integration. The management aspect of these scenes is discussed in a section below.

Two approaches were suggested for a *Scenes API*:

- One using hierarchical collections (e.g. `/collections/landsat8` and `/collections/landsat8:LC81400402013123LGN01`), which could work with the existing `/collections/`

`{collectionId}/coverage` and OGC API — Records — Part 2: Collections extension (for queries) to filter `/collections`, and

- One introducing `/scenes/` where the coverage endpoint could be transported to `/collections/{collectionId}/scenes/{sceneId}/coverage` and records queries could also be made available at `/collections/{collectionId}/scenes` (in this case you would get back a GeoJSON list of scenes, as with STAC & **OGC API — Records — Part 1: Core**).

This would still make it possible to access a unified (native) CRS84 coverage at `/collections/landsat8/coverage`, while the native CRS for e.g. `/collections/landsat8:LC81400402013123LGN01/coverage` (or `/collections/landsat8/scenes/LC81400402013123LGN01/coverage`) would be in the original UTM zone CRS.

A Scenes API would facilitate serving while supporting the management, access and filtering of the individual scenes making up a coverage. The discovery capability could also be fully integrated within the data access itself (e.g. as a query parameter to a `/coverage` request). This integrated discovery / data access capability could unlock a lot of potential functionality, such as easily generating a cloud-free mosaic.

The pros and cons, as well as the challenges to resolve with each approach are as follows. None of the drawbacks are seen as showstoppers, so either approach would be suitable.

5.2.6.1. Hierarchical Collections

PROS:

- *OGC API — Coverages* endpoint and *OGC API — Records* — Collections extension work out of the box
- Obvious where to advertise supported CRS for individual scenes

CONS:

- Hierarchical Collections proposal not yet widely adopted in OGC APIs
- Endpoints for scenes/images CRUD slightly different from Testbed 15 *Images API*.

5.2.6.2. /scenes

PROS:

- Greater similarity with Testbed 15 *Images API*
- Concept of Scenes more explicit than general concept of hierarchical collections

CONS:

- Individual scenes coverages would not work out of the box for the *OGC API — Coverages* `/collections/{collectionId}/coverage` endpoint.
- Would not directly match either Part1 or Part 2 of *OGC API — Records* (although `/items` could also be provided as the scenes catalog using *Records — Part 1*, unless the multi-scenes collection is also available as a Feature Collection)
- Where would the supported CRS of each Scene be listed?

5.2.7. OGC API — DGGS

The OGC API — DGGS draft specification (OGC 21-038) enables clients and servers understanding the same Discrete Global Grid System to exchange information in terms of that reference system, i.e. as DGGS zones. The *What is here?* capability of a DGGS is a data access mechanism which can return data for one or more specific DGGS zone in a compact representation which could be integrated within the GDC API framework. In addition to selecting data by explicitly listing zones, it could also be possible to specify a CQL expression and/or geometry to filter what data gets returned.

5.3. Analytics

The *OGC API — Processes* approved OGC standard provides a very flexible framework for providing any type of analytics capabilities. Although the final version 1.0 of the specification is relatively simple to use and implement, integrating convenient analytics capabilities directly into *OGC API — Coverages* and *EDR* data requests, such as aggregation and band arithmetic calculations, similar to how it is done in DAPA, would prove beneficial.

5.3.1. OGC API — Processes — Part 1: Core

The core *OGC API — Processes* standard defines how an API can list and describe available processes (including their inputs and outputs), how to submit execution requests for available processes, and how to retrieve results from these processes. Processes can support synchronous and/or asynchronous execution. An execution request is a JSON document usually containing a list of inputs, and optionally a list of specific outputs to be returned. For example, inputs may be scalar parameters, spatial or spatiotemporal data. Inputs can be provided embedded into the execution requests (encoded as base64 for binary inputs) or referenced as a URL.

5.3.2. OGC API — Processes — Part 3: Workflows & Chaining

The OGC API — Processes — Part 3 — Workflows & Chaining draft specification (OGC 21-009) defines how to chain multiple processes and data sources together, whether they are local or remote. Part 3 of *OGC API — Processes* also enables triggering processing on-the-fly as a result of regular data requests (e.g. *OGC API — Coverages* or *Tiles*) for a particular area and resolution of interest. This facilitates the integration of processing capabilities into visualization libraries and clients.

In addition to synchronous and asynchronous processing, the draft specification introduces a new mode of execution: the response to submitting an execution request is an OGC API landing page or collection including links to supported data access endpoints for the execution results. Clients can then submit requests to these endpoints with parameters specifying the area and resolution of interest (e.g. tile identifiers, or coverage subsetting and scaling) to both trigger processing and retrieve the response. By submitting multiple small requests (e.g. 256×256 pixels tiles) which can be requested and processed in parallel and leveraging caching along the workflow chain, real-time processing workflows are achievable.

Part 3 of *OGC API — Processes* also adds three new types of inputs that can be used in an execution request:

- The first new type of input is an OGC API collection (which can be local to the server or remote), that points to a data source without hardcoding a particular area, resolution or format, leaving it up to the two nodes at each end of a hop in the workflow to negotiate which to use for accessing the data, and facilitating the re-usability of a defined workflow with different data sources or area of interests.
- The second new type of input is a nested process (which can also be local or remote), which is defined as the same object as the top-level object of the execution request document.
- Finally, the third new possibility for an execution request input allows chaining an external input in the context of deploying a workflow as new opaque process.

In addition to the possibility of being deployed as new processes, workflows could also be deployed as virtual collections / data cubes, which would appear to a client as any other data cubes, except that they may additionally expose the source workflow that generates the data (which would include a reference to all processes and data sources part of the workflows), which would facilitate the re-usability and reproducibility. Another advantage of workflows is the ability to always use the latest available data without requiring batch processing but by responding to data requests while still able to support intelligent caching.

For more details on workflows, see the *Flexible real-time data processing and visualization workflows emerging from OGC API modules* (OGC 21-033).

5.3.3. *DAPA* capabilities as *Coverages* and *EDR* extensions

The draft DAPA specification includes some pre-defined analytics capabilities such as aggregation and band arithmetic calculations. There are already plans to integrate aggregation capabilities in a future version of *OGC API — EDR* as well as in OGC API — Coverages. The participants suggest that instead of defining a separate API, similar aggregation capability be integrated directly in the *OGC API — EDR* queries and *OGC API — Coverages* request, in a harmonized manner, possibly with an `aggregate` parameter. This likely requires further investigation to address the variety of ways data can be aggregated over different dimensions. Similarly, the ability to define a new set of fields for a coverage based on a simple expression (as in derived `fields` proposed in DAPA) could also be available directly in *OGC API — EDR* queries and *OGC API — Coverages* requests, and could possibly use the same suggested `properties=` syntax. This was previously suggested for coverages here.

5.3.4. Profiles for coverage processing

OGC API — Processes offers unlimited analytics possibilities, but it may sometimes be useful to define well-known processes expecting specific types of inputs and returning specific outputs. A particular well-known process could, for example, support a specific coverage processing language, which may be useful for more complex use cases which cannot be expressed in very simple query parameters to express filtering, aggregation or computed fields.

5.3.5. Compatible data cubes and processes

The concept of a *GeoDataClass* is proposed as a mechanism to identify the compatibility of any given data cube as an input to a particular process, which may be available from separate providers. A *GeoDataClass* would be a URI corresponding to a particular data schema to which a data cube could ascribe. A process would be able to tag a particular input with this URI, allowing to easily identify compatible data sources which can be used for this input. For example, a GeoDataClass could be defined for an elevation data cube with a single field representing elevation in meters above the WGS84 ellipsoid, and a process generating contours could specify this URI for its single input elevation data source. Another example would be a GeoDataClass defined for Landsat-8 Collection 2 Level 2, specifying all sensor bands and their characteristics, and a land cover prediction process could tag its input with this URI as expecting such a data source.

5.3.6. OGC API — DGGS

The *Where is it?* capability of the OGC API — DGGS draft specification could enable analytics queries, returning a list of DGGS zones satisfying the query. For example, this query could be expressed using CQL, which has the benefit of being reusable for performing analytics with other OGC API specifications.

5.4. Data discovery, queries and catalogs

The *OGC API — Records* draft specification and the *SpatioTemporal Asset Catalog specifications (STAC)*, which may be considered a profile of *OGC API — Records*, support the discovery of resources offered by a GDC API deployment and any associations that exist between those resources. For example, an *OGC API — Records* endpoint can catalogue all the data offerings of a GDC API deployment, all the processing capabilities offered by the GDC API deployment and any associations that exist between those two resources (e.g. particular processes can operate on particular data offerings of the GDC API).

The *OGC API — Records* draft specification defines three building blocks:

- the schema for a core *record*,
- a *collection* resource that describes a set of related records (i.e. a catalogue),
- an API that allows catalogues (i.e. collections of records) to be searched.

Using these building blocks, various deployment patterns can be envisioned but the most relevant deployment patterns for a GDC API deployment are:

- a crawlable catalogue,
- a searchable catalogue,
- and local resources catalogue.

A *crawlable catalogue* is a static and linked deployment of *collection* and *record* objects that can be crawled by a browser or search-engine-crawler to navigate the hierarchy of resource offered by a GDC API deployment. This allows browsing the resources offered by a GDC API deployment in a hierarchical, tree-like manner in the same way one would browse the pages of a web site.

A *searchable catalogue* is a defined endpoint in a GDC API deployment where searches can be performed to discover resources offered by the deployment. The core search API of an *OGC API — Records* deployment is *OGC API — Features*. That is, *OGC API — Records* supports searches using spatial (i.e. `bbox`), temporal (i.e. `datetime`), full text (i.e. `q`) and resource type (i.e. `type`) predicates.

NOTE: the `q` parameter is currently defined in the draft *OGC API — Records* specification but it has been decided to move it over to the *OGC API — Features* specification.

An *OGC API — Records* deployment can also implement additional conformance classes to provide more advanced search capabilities. Specifically, an *OGC API — Records* deployment can implement CQL2 that offers search capabilities using a broad range of logically connected predicates. Having discovered a resource from the catalogue, a client can then bind to that resource using links found in the record describing the resource. Those links can be static links or *templated* links that allow for run-time, parameterized resolution of the binding link.

A *local resources catalogue*, enhances any existing endpoint in a GDC API deployment (e.g. /collections, /scenes) to add catalogue-like query capabilities to that endpoint. For example, the proposed *Scenes API* endpoint can be enhanced with catalogue search capabilities allowing scenes to be searched. For example, "Find all the scenes that intersect a specific bounding box, for a specific time period where the cloud cover of each scene is less than 10%".

The *OGC API — Collection — Part 2: Geospatial data* specification itself also provides some basic discovery capabilities in its *Core* conformance class, as well as more advanced capabilities with its *Simple Query* conformance class. Proposals for hierarchical collections (Common#11, Common#298) would also facilitate browsing through a large number of data cubes from a single API in a tree-like manner.

5.5. Visualization

To integrate visualization capabilities directly within a GDC API, the OGC API — Maps OGC 20-058 draft specification can be implemented to render maps of data cubes or the results of process. Map tiles can also be provided in combination with the *OGC API — Tiles* specification.

5.6. Managing data and algorithms

The functionality defined in the Testbed 15 — *Images API* prototype could be integrated within a Scenes API to Create/Update/Replace/Delete scenes making up a data cube, e.g. `POST` to `/collections/landsat8/scenes` (or `/collections/landsat8`) to upload a new scene, `DELETE /collections/landsat8/scenes/{sceneId}` (or `/collections/landsat8:{sceneId}`) to remove an existing scene. The *OGC API — Styles* draft specification (OGC 20-009) allows to manage styles, retrieve styles for client-side rendering, as well as providing a mechanism to select styles for use in server-side rendering in conjunction with *OGC API — Maps* and *OGC API — Tiles*.

For managing algorithms, *OGC API — Processes — Part 2 — Deploy, Replace, Update* (OGC 20-044) defines how to deploy new processes. These processes could be defined in different ways, including as a workflow as defined in Part 3, as a Jupyter notebook, or using the OGC Earth Observation Application Packages. This deployment functionality corresponds to the *Application Deployment and Execution System (ADES)* experiments from previous OGC Testbeds.

6

MACHINE LEARNING WITHIN A GEO DATA CUBE API

6 MACHINE LEARNING WITHIN A GEO DATA CUBE API

This chapter describes the experiments and findings from 52°North's D126 deliverable, integrating Machine Learning model within a Geo Data Cube.

6.1. Use case

In Machine Learning (ML) and Data Science most of the work is dedicated to preparing a meaningful dataset for the model, so that the model comprehends and extracts useful information for the prediction, which could be directly or indirectly leading to the objective of the ML task. As the known quote in computer science says "Garbage in — Garbage out", 70 % of ML engineers work is to prepare a clean dataset for the model. Therefore, having less clean data affects the accuracy and robustness of the model.

52°North had a free choice for the objective of the ML model. The main focus was on the API and the interaction between the ML model and Geo Data Cube. 52°North had the possibility to receive label data from the National Forest Information System (NFIS). 52°North chose the forest land cover for Canada 2015 Dataset as label data, and classifying Landsat data as the objective for the ML model. Hence, Landsat-8 Level 2 Collection 2 image data was retrieved through the EarthExplorer web interface. The Landsat-8 data set consists of multiple bands over a wide spectrum of wavelengths, which contains comprehensive spectral and spatial information over different vegetation species and land cover.

6.2. Data preparation

Generally, data preparation in Data Science is the process of collecting data from several data sources and then profiling, cleaning, enriching and combining those into a derived dataset to use in the analytics process. The input data of the Landsat multispectral images consists of spectral bands with a spatial resolution of 30 meters for Bands 1 to 7, where:

- Band 1 (ultra-blue) is useful for coastal and aerosol studies and ranges between 0.43-0.45 µm,
- Band 2 is the visual blue band and ranges 0.45-0.51 µm,
- Band 3 is the visual green band and ranges between 0.53-0.59 µm,
- Band 4 is the visual red band and ranges between 0.64-0.67 µm,
- Band 5 is the Near Infrared (NIR) band and ranges between 0.85-0.88 µm,

- Band 6 and Band 7 are the short wave Infrared (SWIR) 1 and 2 and range between 1.57-1.65 µm and 2.11-2.29 µm, respectively.

52°North used 6 bands of Landsat, namely, Band 2 through Band 7. Therefore, the multispectral image consists of Blue, Green, Red, NIR, SWIR 1 and SWIR 2 corresponding to Band 2, 3, 4, 5, 6 and 7, respectively. The projection of the Landsat bands uses the UTM with zones, and the projection unit is in meters. 52°North only obtained satellite scenes with 1% maximum cloud cover, since clouds affect spectral information and lead to distortions in the multispectral images. Furthermore, the pixel values were normalized by the maximum reflectance for each band. This normalization ensures that different scenes of Landsat images have similar reflectance values between 0 and 1.

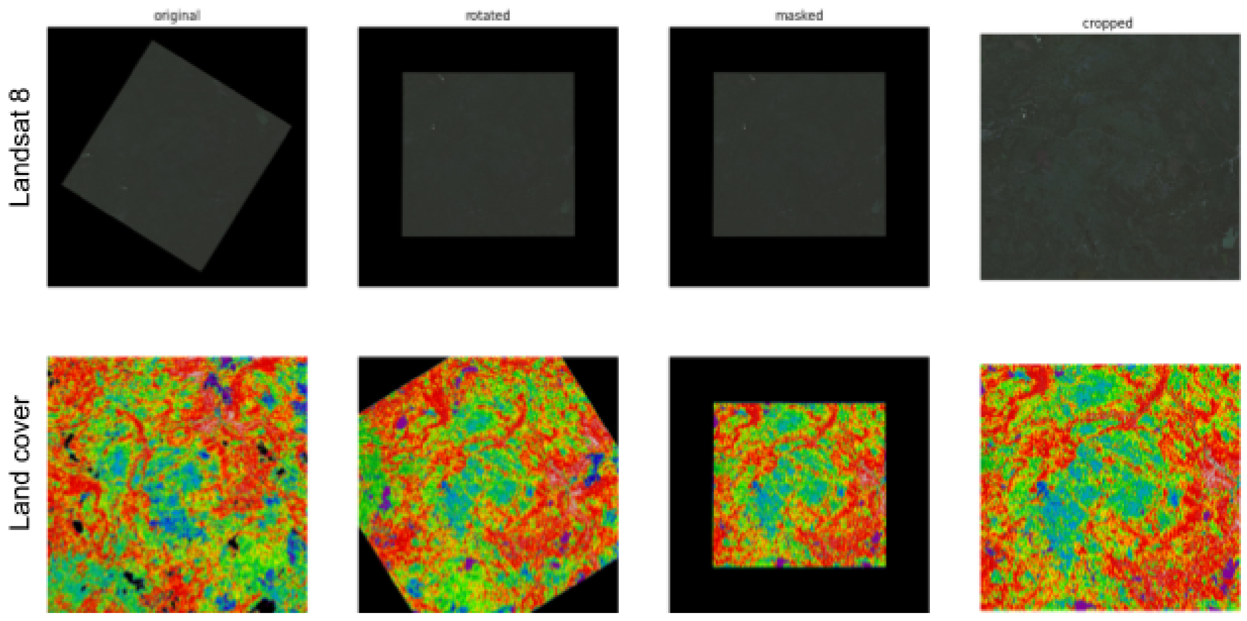


Figure 8 — Data Preparation steps to obtain registered label and input datasets.

Label Data of the dataset from NFIS consists of a large GeoTiff of 13 different classes of land cover data in Canada for the year 2015. The raster data CRS is the Lambert conformal conic projection (LCC), and the projection unit is in meters. The 13 classes are assigned to different integers with the following data codes:

```
  0 = no change
 20 = water
 31 = snow_ice
 32 = rock_rubble
 33 = exposed_barren_land
 40 = bryoids
 50 = shrubland
 80 = wetland
 81 = wetland-treed
100 = herbs
210 = coniferous
220 = broadleaf
230 = mixedwood
```

52°North implemented a configurable class selection to choose different classes for different tasks. For the next step, it is required that both datasets (remote sensing image and labeled data) have the same spatial reference system. Therefore, the Landsat images were reprojected to match the spatial reference system of the land cover dataset. It was possible to obtain georeferenced data from different sources, because the reprojection of the images guarantees overlapping of both datasets and hence facilitates image registration. As shown in Figure 8, after reprojecting the datasets, we rotated both datasets and cropped out the *nodata* edges to extract the most of the input images. The preparation of the training data consists of extracting pairs of input and output patches of the training and label data. 52°North extracted patches with a fixed size of 256×256 as input for the ML model.

6.3. Model training

Since the number of classes plays a major role in the time needed for obtaining the data and training the model, the number of classes for the task should be reasonably selected. Within the scope of Testbed-17, the participants decided to choose only four classes as a prototype for a trained model. The selected classes were *no_change*, *water*, *coniferous* and *herbs*.

For the model architecture the participants implemented a semantic segmentation model using convolutional neural network layers. The model has a u-net architecture consisting of five convolution and deconvolution layers. The binary cross entropy loss function was applied and the dice coefficient to evaluate accuracy was used.

The Landsat-Water Classifier is a semantic segmentation model that was trained on 12 different Landsat scenes of Canada in 2015. The model has reached a total accuracy of 87.4% after learning for 40 epochs.

6.4. Model prediction

After the model was trained, Landsat scenes or subsets were processed to estimate land cover classes. The classes can be obtained by predicting a sliding window with a size of 256×256 over the input image. However, the lack of information about the other nearby image windows results in seams and artifacts between the estimated window images. Therefore, the parameter trim function was used to overcome this issue. For instance, when trim equals 80, 80 pixels are trimmed from each side of each window image to create a small rectangle in the center of each window of size 96×96. By attaching only the center windows, smooth and seamless predictions of the entire scene are obtained.

The model prediction was provided as an *OGC API — Processes* process using pygeoapi and integrated in the service implementation of Clause 7.1. One type of plugin that is currently supported by pygeoapi is a processor. Extending pygeoapi this way can be done by implementing a subclass of `pygeoapi.process.base.BaseProcessor` with the `execute()` method. The process description (see landsat predictor process description listing) is provided as a python dictionary and handled by pygeoapi. All process metadata and job management is handled by pygeoapi including synchronous or asynchronous execution. The process implementation itself needs only to handle its specific inputs:

- *bbox*
  A spatial bounding box defining the area of interest, the model prediction should be performed on. It must be encoded using WGS84 coordinates in the following form: [min lon, min lat, max lon, max lat] with unitless decimal numbers

- *collection*
  The URL of the *OGC API — Coverages* collection providing the Landsat-8 Collection 2 Level 2 data. Must start with http or https if referring to a coverage hosted in an *OGC API — Coverages* service instance. Support for file links to local GeoTIFFs is implemented for testing, too. In addition, the following bands in the given order must be contained if no value is specified for the *bands* input: blue, green, red, nir, swir1, swir2.

- *bands*
  A comma-separated list of band names containing the same bands in the same order as described in the collection input.

The workflow of the implemented process after being processed by pygeoapi looks as follows:

```
{
    "version": "0.1.0",
    "id": "landcover-prediction",
    "title": "Land cover prediction",
    "description": "Land cover prediction with Landsat 8",
    "keywords": ["land cover prediction", "landsat 8", "tb-17"],
    "jobControlOptions": "async-execute",
    "outputTransmission": ["value"],
    "links": [
        {
            "type": "text/html",
            "rel": "canonical",
            "title": "Processor Repository",
            "href": "https://github.com/52North/Landsat-classification/blob/mai
n/README.md",
            "hreflang": "en-US"
        },
        {
            "type": "text/html",
            "rel": "canonical",
            "title": "Landsat 8 Collection 2 Level 2",
            "href": "https://www.usgs.gov/core-science-systems/nli/landsat/
landsat-collection-2-level-2-science-products",
            "hreflang": "en-US"
        }
    ],
    "inputs": {
        "collection": {
            "title": "Coverage collection",
            "description": "url of the OGC API Coverages collection providing
the Landsat 8 Collection 2 "
                           "Level 2 data (must start with http or https and
include the following bands:"
                           " blue, green, red, nir, swir1, swir2)",
            "schema": {
                "oneOf": [
                    {
                        "type": "string",
                    },
                    {
                        "type": "string",
                        "contentEncoding": "binary",
                        "contentMediaType": "image/tiff; application=geotiff"
                    }
                ]
            },
            "minOccurs": 1,
            "maxOccurs": 1,
            "metadata": null,
            "keywords": ["landsat"]
        },
        "bbox": {
            "title": "Spatial bounding box",
            "description": "Spatial bounding box in WGS84",
            "schema": {
                "allOf": [
                    {"format": "ogc-bbox"},
                    {"$ref": "https://raw.githubusercontent.com/opengeospatial/
ogcapi-processes/master/core/openapi/schemas/bbox.yaml"}
                ],
                "default": {"bbox": [-104.6, 51.8, -103.7, 52.6], "crs": "http:
//www.opengis.net/def/crs/OGC/1.3/CRS84"}
            },
            "minOccurs": 0,
            "maxOccurs": 1,
            "metadata": null,
            "keywords": ["bbox"]
        },
        "bands": {
            "title": "Bands",
            "description": "Landsat 8 bands (comma-separated list, e.g.
\"blue, green, red, nir, swir1, swir2\")",
            "schema": {
                "type": "string"
            },
            "minOccurs": 0,
            "maxOccurs": 1,
            "metadata": null,
            "keywords": ["bands"]
        },
    },
    "outputs": {
        "prediction": {
            "title": "Land cover prediction",
            "description":
                "Land cover prediction with Landsat 8 Collection 2 Level 2 for
no change (=1), "
                "water (=2), coniferous (=3) and herbs (=4) (no data=0)",
            "schema": {
                "type": "string",
                "format": "byte",
                "contentMediaType": "image/tiff; application=geotiff"
            }
        }
    },
    "example": {
        "inputs": {
            "collection": {
                "collection": "https://17.testbed.dev.52north.org/geodatacube/collections/landsat8_c2_l2"
            },
            "bbox": {
                "bbox": [-104.6, 51.8, -103.7, 52.6],
                "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
            }
        },
        "jobControlOptions": ["async-execute"],
        "outputTransmission": ["value"],
        "response": "raw"
    }
}
```

Landsat predictor process description

This additional resource of the service instance needs to be configured in the service configuration in the `resources` section. In addition, activating a manager plugin for handling the jobs/executions of the process (see pygeoapi configuration listing) is recommended. The default implementation provided by pygeoapi based on TinyDB (https://github.com/geopython/pygeoapi/blob/master/pygeoapi/process/manager/tinydb_.py) is used in the service instance. Using the /tmp folder in the service does not prevent losing job results and state during pod restarts. In production and under high load, another database driven job manager should be used. For the requirements within Testbed-17, the performance of the TinyDB job manager was sufficient.

```
Landcover-prediction:
     type: process
     processor:
         Name: landsatpredictor.pygeoapi_processor.LandcoverPredictionProcessor
```

Process resource to be added to pygeoapi configuration in resources section

```
manager:
    name: TinyDB
    connection: /tmp/pygeoapi-process-manager.db
    output_dir: /tmp/pygeoapi-process-outputs/
```

Activate TinyDB job manager in pygeoapi's configuration server section

The workflow as outlined in Figure 9:

1. Receive input from pygeoapi
2. Validate inputs:
    a) bbox is formatted correct and valid
    b) Collection is url
    c) All band names are included in the coverage rangetype

3. Process collection:
    a) Download coverage
    b) Generate required model input
        i) Normalize all bands
        ii) Visual light reflectance mask
        iii) Output GeoTIFF metadata
4. Submit data “to model”
5. Process response from model and return data

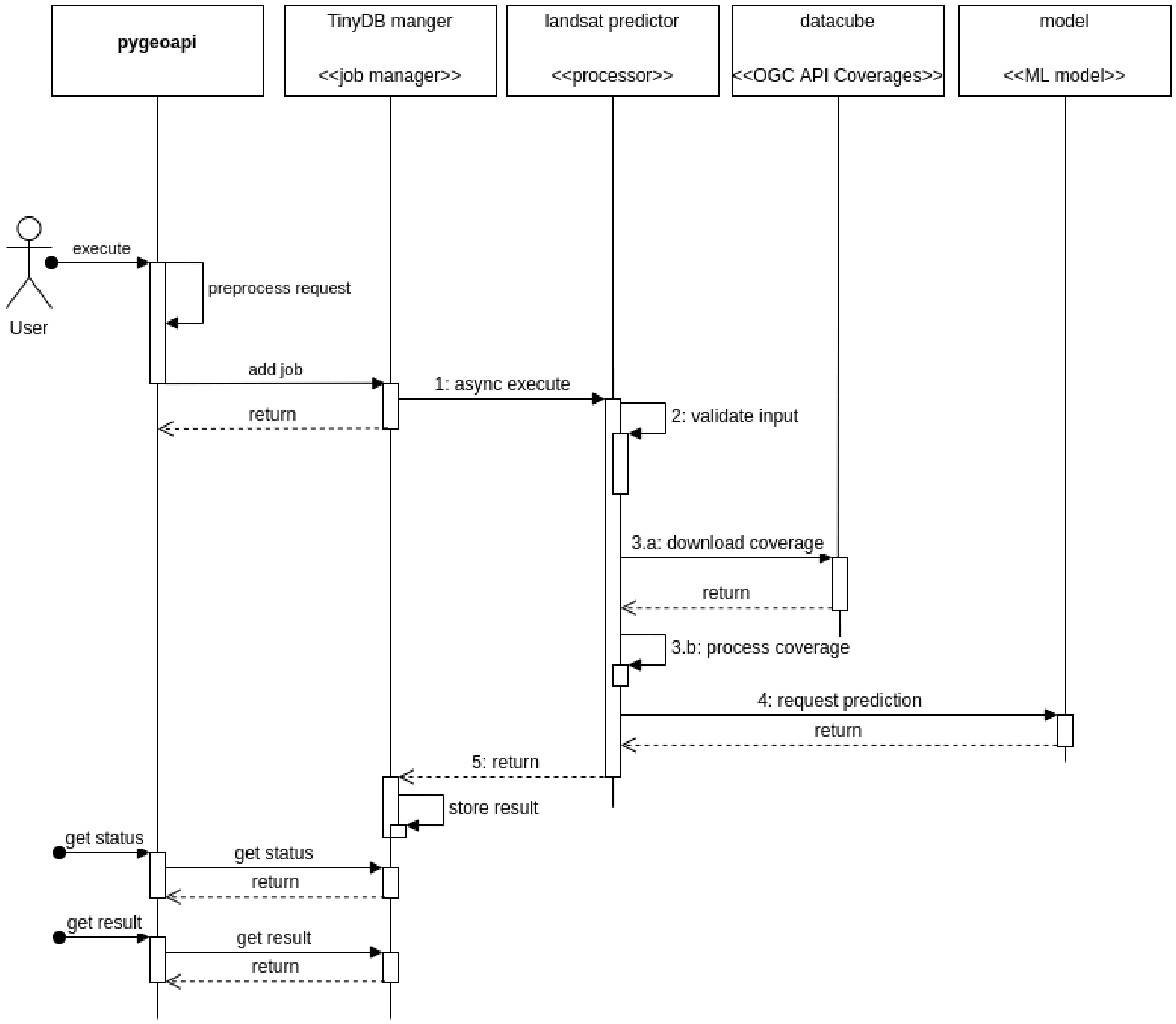


Figure 9 — Process workflow

The ability to return the model output directly encoded as GeoTIFF required some adjustments to pygeoapi. These suggested changes were provided as a pull request in the corresponding repository and discussed with the developers of pygeoapi. These adjustments are:

- Changing the result encoding in pygeoapi depending on the MIME type returned by the processor execute method
- Changing the result storing in TinyDB process manager depending on the MIME type of process result

The result of the process is a single band GeoTIFF with 5 categories:

- 0: no data
- 1: no change
- 2: water
- 3: coniferous
- 4: herbs

For executing the process, one can use the curl call from curl call listing.

```
curl -X POST "https://17.testbed.dev.52north.org/geodatacube/processes/landcove
r-prediction/execution" \
   -H "Content-Type: application/json" \
   -d "{\"mode\": \"async\", \"response\": \"raw\", \"outputTransmission\":
[\"value\"], \
   \"inputs\":{\"collection\": {\"collection\": \"https://17.testbed.dev.
52north.org/geodatacube/collections/landsat8_c2_l2\"}, \
   \"bbox\": {\"bbox\": [-104.6, 51.8, -103.7, 52.6], \"crs\": \"http://www.
opengis.net/def/crs/OGC/1.3/CRS84\"}}}"
```

curl call to execute Landsat prediction process

The call sends a JSON document using the HTTP POST method to the execution endpoint of the “landcover-prediction” process in the OGC API Processes instance listening behind the endpoint https://17.testbed.dev.52north.org/geodatacube/. A more readable version of the document is provided in Listing Figure 10. This document provides the required process inputs bbox and collection, accompanied by some job management options:

- Mode (https://docs.ogc.org/is/18-062r2/18-062r2.html#sc_execution_mode): async → the job should be executed asynchronously. The results of the jobs can be downloaded later, when available, using the jobs endpoint of the process.
- Response (https://docs.ogc.org/is/18-062r2/18-062r2.html#_response_type): raw → the job result should be provided in the encoding of the output and not as json document
- outputTransmission (https://docs.ogc.org/is/18-062r2/18-062r2.html#sc_process_outputs): value → the job result should be provided as value and not as reference

The status of the job (https://docs.ogc.org/is/18-062r2/18-062r2.html#sc_retrieve_status_info) and its final results (https://docs.ogc.org/DRAFTS/18-062.html#sc_retrieve_job_results) can be requested using the according endpoints as outlined in the specification.

```
{
    "mode": "async",
    "response": "raw",
    "outputTransmission": ["value"],
    "inputs": {
        "collection": {
            "collection": "https://17.testbed.dev.52north.org/geodatacube/colle
ctions/landsat8_c2_l2"
        },
        "bbox": {
            "bbox": [-104.6, 51.8, -103.7, 52.6],
            "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        }
    }
}
```

Figure 10 — Execute request example json body

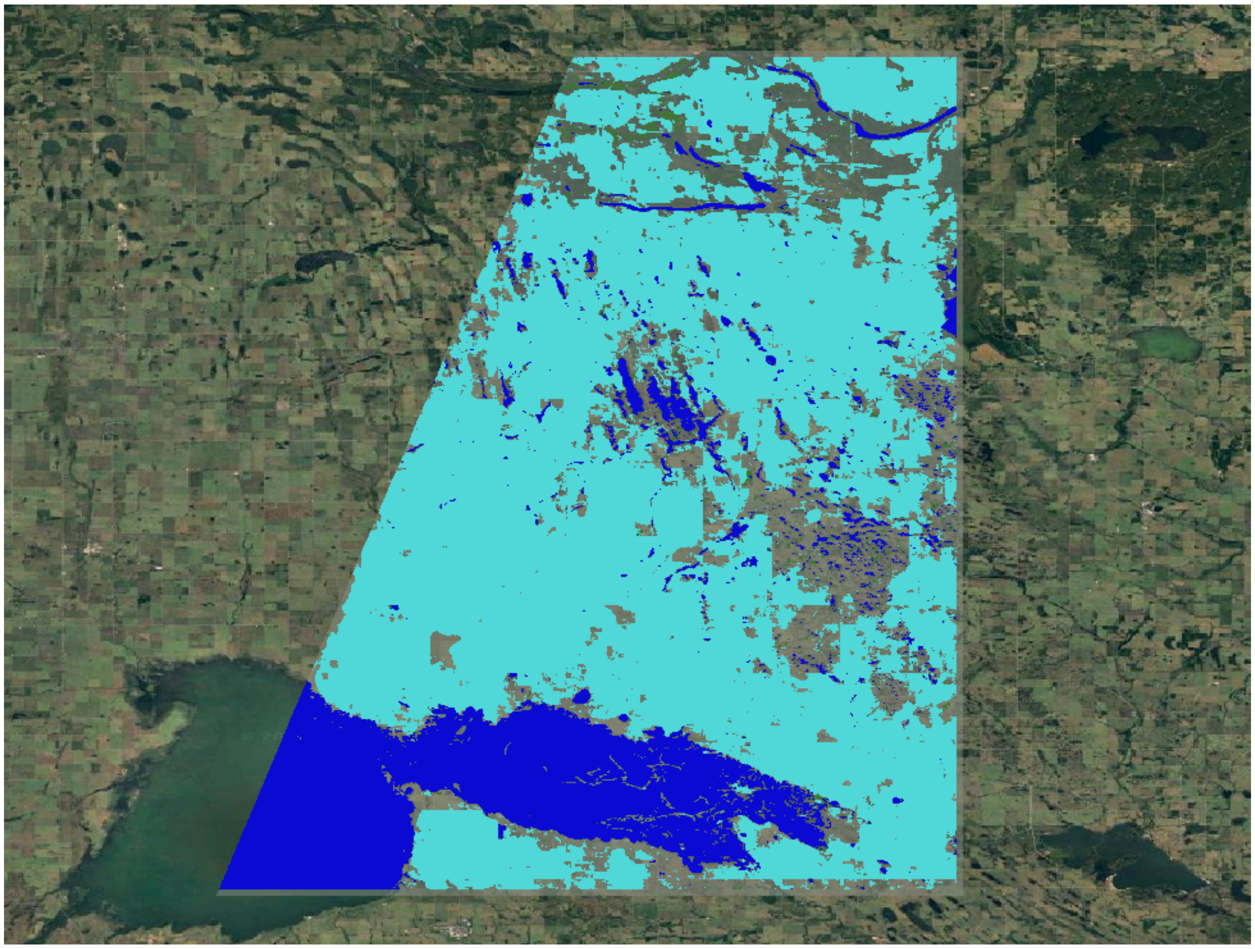


Figure 11 — Land cover prediction for the bounding box "[-104.6, 51.8, -103.7, 52.6]" (dark blue=water, turquoise=herbs, dark green=coniferous, transparent grey=no change)

6.5. Technology Integration Experiments with D123 — GDC API Service (Wuhan University)

In addition to using Landsat-8 coverage data from their service to run a ML prediction, 52°North tested the process with Landsat-8 coverage data provided by Wuhan University's service.

The process is invoked in the same way as before:

```
curl -i -H "Content-Type: application/json" -X POST -d @landcover_async_api_
wuhan.json \
   "https://17.testbed.dev.52north.org/geodatacube/processes/landcover-
prediction/execution"
```

However, this time the collection URL and the `bands` input were changed, as the coverage collection includes more than the six expected bands:

```
{
    "mode": "async",
    "response": "raw",
    "outputTransmission": ["value"],
    "inputs": {
        "collection": {
            "collection": "http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/
collections/LC08_L1TP_ARD_EO_20180915025555"
        },
        "bbox": {
            "bbox": [113.05,30.85,113.2,30.95],
            "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        },
        "bands": "Blue,Green,Red,Near-Infrared,SWIR1,SWIR2"
    }
}
```

landcover_async_api_wuhan.json

Figure 12 shows the land cover prediction from this process execution which can be downloaded as GeoTIFF after the asynchronous execution has finished.

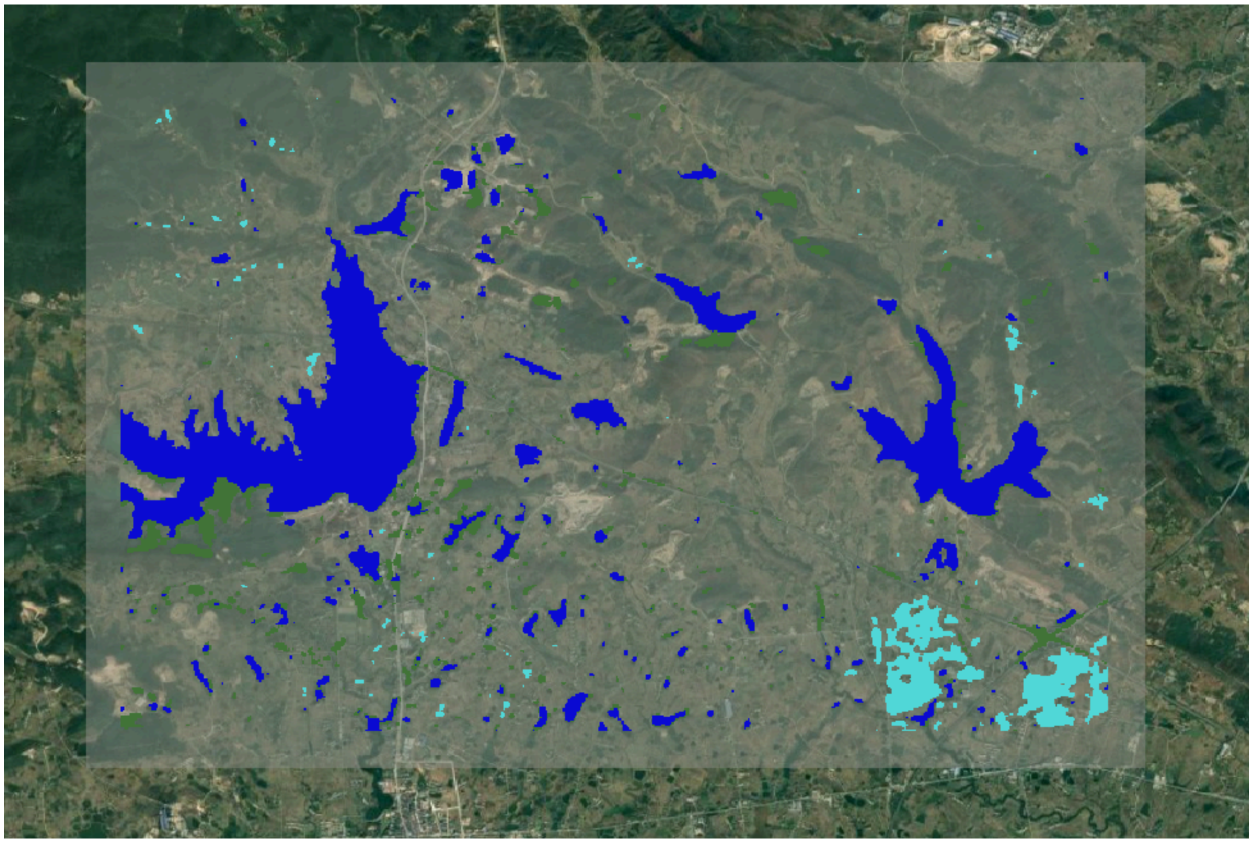


Figure 12 — Land cover prediction for the bounding box "[113.05, 30.85, 113.2, 30.95]" (dark blue=water, turquoise=herbs, dark green=coniferous, transparent grey=no change)

6.6. Future work

In order to use ML models in practice, multiple preparatory steps need to be taken of as described above:

- Identifying and understanding the use case,
- Collecting and preparing the data (label + feature data),
- Implementing the model,
- Training the model,
- Evaluating and optimizing the model, and finally
- Applying the trained model.

The work completed in this testbed, demonstrated how a ML model prediction can be part of a GDC API by making use of *OGC API — Processes*. By defining the model input as a coverage collection URL, interoperability between different OGC service instances offering Landsat-8 data is achieved. In contrast to the model prediction, the model training including data preparation was not performed on the server, but locally. The resulting model weights were then uploaded as part of the prediction process.

In the future, it should be further investigated how the training itself can be simplified by accessing data from a GDC API or by performing it directly as part of a GDC API. The most obvious benefit of a GDC/GDC API concerns the image registration (the provision of analysis-ready data). The capability of a GDC API to subset different variables (`/bands/layers/`…) that serve as label and feature data for the same area, time and in the same resolution and CRS has the potential to significantly simplify the data preparation. Moreover, with the GDC API, data could be normalized to make sure it has a common scale which is important for ML models. Use of the *OGC API — Tiles* draft specification might make it even easier by providing tiles that can be directly loaded as training patches for the model training.

As ML model training is generally heavy on resources, performing the training close to the data is advantageous. Using DAPA could be a promising approach as it directly connects data and processing. Triggering a new model training automatically when new data is available would be an interesting feature. One important aspect of ML model training is that it is typically performed by a group of skilled users/experts, as the quality of the trained model needs to be sensitively assessed. Thus, it would be necessary to add authorization to respective servers that also support model training through dedicated processes.

In addition to including label and feature data as dimensions in the GDC, the prediction results could also be persisted. For this capability, support for transactional coverage collections could be used. However, this opens up new challenges such as overlapping model runs and possible updates of an underlying model or its weights.

Initial efforts towards transaction capabilities for coverages took place as part of drafting an *Images API* specification in OGC Testbed-15, and that work is being considered as one aspect of the potential Scenes API investigated in this Testbed-17 task. The *OGC API — Create, Replace, Update and Delete* draft specification extending *OGC API — Features* also lays out a foundation for transactions intended to be common across different OGC API specifications.

7

IMPLEMENTED SERVER COMPONENTS

7 IMPLEMENTED SERVER COMPONENTS

This chapter describes the different Geo Data Cube service components developed and enhanced during this project by Wuhan University, MEEO, 52°North and Ecere.

The following table summarizes the capabilities implemented by each server:

Table 1 — Server components

PROVIDER	CAPABILITIES
Wuhan University	Coverages (1,2,3,4,5), Processes (8, e.g. ndvi, aspect), Records (11), DAPA (12)
MEEO	Common/Core (1) & Collections (2), started: Coverages (3) & Processes (8)
52°North (pygeoapi)	Coverages (1,2,3,4,5), Processes (7, 8, Workflows: 9, Land Cover ML prediction), Records (11)
Ecere (GNOSIS Map Server)	Coverages (1,2,3,4,6), Processes (7, Workflows: 9, 10)

Table 2 — GDC API Capabilities

1. Common-1: Core
2. Common-2: Collections
3. Coverages-1
4. Coverages-1 (Subsetting)
5. Coverages-1 (Range subsetting)
6. Coverages-1 (Scaling)
7. Processes-1 (Sync execution)
8. Processes-1 (Async execution)
9. Processes-3: Workflows (Collection Input)
10. Processes-3: Workflows (Collection Output)

11. Records

12. DAPA

7.1. 52°North Geo Data Cube API Server Implementation (D122)

This section focuses on 52°North's server implementation. The implementation follows the approach discussed during the Testbed that a Geo Data Cube API should implement existing (draft) OGC APIs (see Clause 5). The following OGC APIs are partially or fully implemented by 52°North's service:

- OGC API — Common
- OGC API — Records
- OGC API — Coverages
- OGC API — Processes

For functionality that is not yet supported by the existing (draft) specifications, new API specifications or extensions could be developed. Specific challenges faced in this task will be discussed in Section Clause 7.1.5.

The service is based on two existing open source projects, Open Data Cube (ODC) (https://www.opendatacube.org/) and pygeoapi (https://pygeoapi.io/). Before explaining the developed service in more detail, the next two sections will briefly introduce ODC and pygeoapi.

7.1.1. Open Data Cube (ODC)

ODC is an open source geospatial data management and analysis software project that supports the efficient use of earth observation data. At its core it offers uniform access to heterogeneous data sets and sources as data cubes in an analysis-ready way. Figure 13 illustrates the basic concept. Data is stored on a native file system or a cloud platform. First, the data is indexed in a PostgreSQL database with some metadata. This index is the central metadata store that allows querying and accessing data. For the indexing, metadata documents need to be prepared and registered using the datacube-core Python library (https://github.com/opendatacube/datacube-core) which is the heart of ODC. These documents are yaml files in the ODC-specific metadata format "eo3" (https://datacube-core.readthedocs.io/en/latest/about-core-concepts/dataset-documents.html#dataset-metadata-doc-eo3) or its predecessor "eo". ODC distinguishes between products and datasets. A dataset is for example a single Landsat scene while a product is a collection of datasets. All datasets of a single product share the same measurements and some basic metadata, e.g. sensor type. In contrast, coordinate reference systems can vary among different datasets. The datacube-core library also offers a simple uniform Python API to retrieve data. Spatial, temporal and thematic (band) filtering is possible and also reprojection and down- and upsampling of the data.

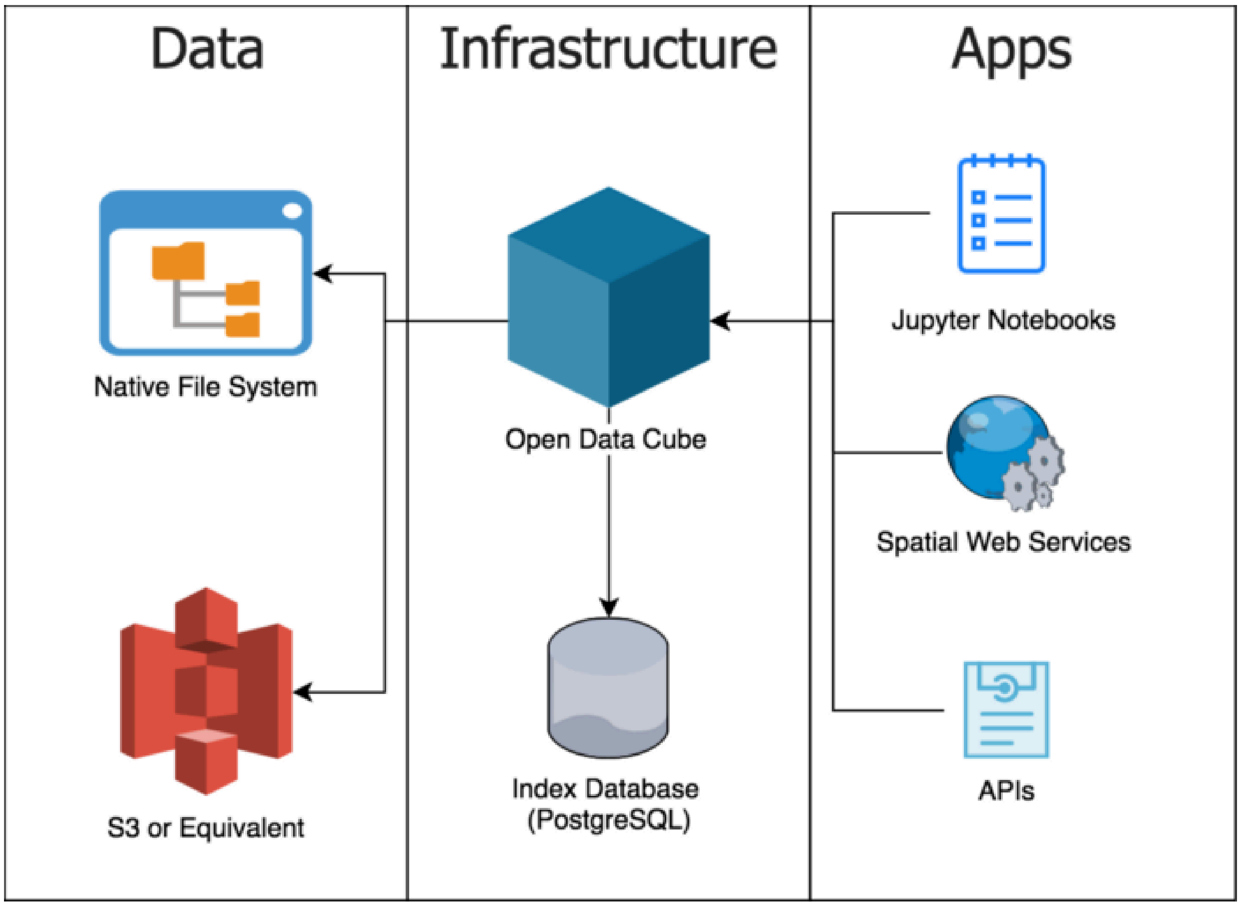


Figure 13 — ODC architecture (source: https://medium.com/opendatacube/what-is-open-data-cube-805af60820d7)

ODC is continuously improved and extended by a large community. Recently, new features like 3D datasets and STAC (Spatio Temporal Asset Catalog, https://stacindex.org/) support have been added which were not available at the beginning of Testbed-17. By interfacing with a STAC API it will be possible to retrieve data without indexing it in a database beforehand (https://github.com/opendatacube/odc-stac). Some features regarding STAC can already be used, however, there isn't a comprehensive public documentation yet.

Once data is indexed, it can be retrieved and used in applications like Jupyter Notebooks or web services. While there is already a web service implementation offering classical OGC web services (OWS) on top of ODC's index and core library, there is — to the best of the participants' knowledge — no service implementation offering OGC APIs.

7.1.2. pygeoapi

pygeoapi is a Python server implementation of the OGC API family of standards. It offers a core Python API and an HTTP API on top of it. Publishing of data is organized with a provider framework. Providers implement the logic of handling specific data resources, e.g. geotiff files or databases, and return data to the pygeoapi API framework. Every resource (collections, processes, catalogs) which is served needs to be configured with a suitable provider.

7.1.3. Service architecture

52°North's service implementation builds on ODC and pygeoapi. ODC serves as a Geo Data Cube resource and is responsible for storing and managing data and metadata. pygeoapi uses this resource and publishes the included data via OGC APIs. The connection between ODC and pygeoapi is achieved by a provider plugin for pygeoapi called pygeoapi-odc-provider (https://github.com/52North/pygeoapi-odc-provider), which has been developed by 52°North in this Testbed. The basic service architecture is shown in Figure 14 and explained in more detail in the following.

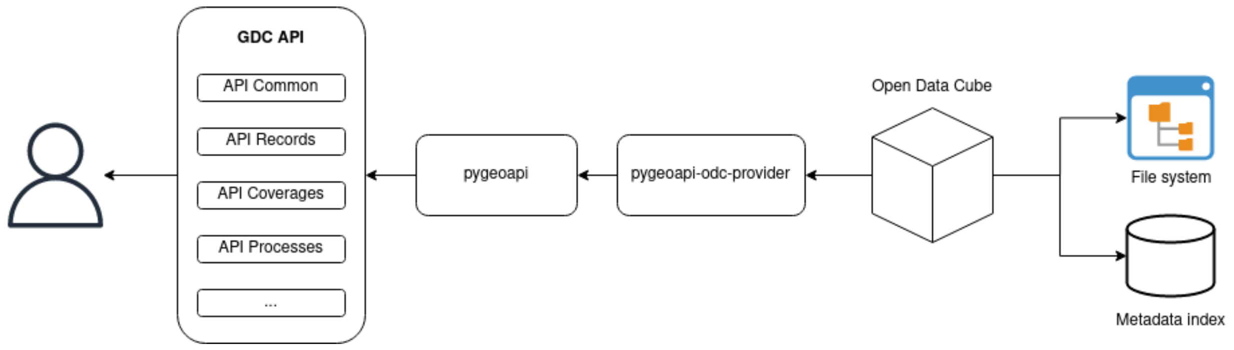


Figure 14 — Architecture of 52°North's service implementation

Data is first downloaded from various platforms, stored in a file system and added to the metadata index using ODC. The process of downloading and indexing data was automatized for NRCan's DEM datasets and for Landsat 8 Collection 2 Level 2 data (https://github.com/52North/ogc-tb-17_datacube-service_odc/tree/main/downloader, make repo public). It is also possible to add custom metadata under the "metadata" key in the product definition (see example yaml) and parse these later to complete pygeoapi's resource configuration. 52°North added product, provider, project, category, links and keywords. An example for a product definition for NRCan's DEM data is presented in the following. It provides Digital Surface Model (DSM) data collected during the project "The Pas" in 2014:

```
metadata_type: eo3
name: dsm__MB__The_Pas_2014
description: '"dsm" data created by "MB" within the project "The_Pas_2014"'
metadata:
  product:
    name: dsm__MB__The_Pas_2014
  provider:
    name: MB
  project:
    name: The_Pas_2014
  category:
    name: dsm
  keywords:
  - MB
  - The_Pas_2014
  - dsm
  - NRCAN
  - Canada
  links:
  - type: text/html
    rel: canonical
    title: High Resolution Digital Elevation Model (HRDEM) - CanElevation
Series
    href: https://open.canada.ca/data/en/dataset/957782bf-847c-4644-a757-
e383c0057995
    hreflang: en-CA
measurements:
- name: dsm
  units: m
  dtype: float32
  nodata: -32767.0
```

A dataset for this product looks like this:

```
$schema: https://schemas.opendatacube.org/dataset
id: e8dc8680-08d8-5aa5-a05c-70c2f9b85ee9
product:
  name: dsm__MB__The_Pas_2014
provider:
  name: MB
project:
  name: The_Pas_2014
category:
  name: dsm
keywords:
- MB
- The_Pas_2014
- dsm
- NRCAN
- Canada
links:
- type: text/html
  rel: canonical
  title: High Resolution Digital Elevation Model (HRDEM) - CanElevation Series
  href: https://open.canada.ca/data/en/dataset/957782bf-847c-4644-a757-
e383c0057995
  hreflang: en-CA
crs: EPSG:2957
geometry:
  type: Polygon
  coordinates:
  - - - 710000.0
      - 5940000.0
    - - 710000.0
      - 5950000.0
    - - 720000.0
      - 5950000.0
    - - 720000.0
      - 5940000.0
    - - 710000.0
      - 5940000.0
grids:
  default:
    shape:
    - 10000
    - 10000
    transform:
    - 1.0
    - 0.0
    - 710000.0
    - 0.0
    - -1.0
    - 5950000.0
    - 0
    - 0
    - 1
measurements:
  dsm:
    path: /ogc-tb-17/DATA/dsm/MB/The_Pas_2014/dsm_1m_utm13_e_21_194.tif
    layer: dsm
properties:
  datetime: '1970-01-01T00:00:00+00:00'
  platform: na
  instrument: na
  odc:processing_datetime: '2021-09-22T14:28:57+00:00'
  odc:file_format: GeoTIFF
  odc:product_family: dsm__MB__The_Pas_2014
  dea:dataset_maturity: final
  providers:
  - MB
  mission: The_Pas_2014
lineage: {}
```

Once ODC is set up, the connection between ODC and pygeoapi has to be established. The new pygeoapi provider plugin pygeoapi-odc-provider facilitates this. It implements all the logic of retrieving metadata and data using ODC's Python API and maps it to the specific OGC API. For *OGC API — Coverages*, it maps ODC products to OGC coverage collections. These two concepts are similar and the mapping is relatively straightforward. To finally use ODC in pygeoapi, it is necessary to configure the resources with the correct provider class. A resource entry for a coverage collection in pygeoapi`s configuration file looks like this:

```
dsm__MB__The_Pas_2014:
    type: collection
    title: dsm__MB__The_Pas_2014
    description: '"dsm" data created by "MB" within the project "The_Pas_2014"'
    keywords:
    - MB
    - The_Pas_2014
    - dsm
    - NRCAN
    - Canada
    links:
    - type: text/html
      rel: canonical
      title: High Resolution Digital Elevation Model (HRDEM) - CanElevation
Series
      href: https://open.canada.ca/data/en/dataset/957782bf-847c-4644-a757-
e383c0057995
      hreflang: en-CA
    extents:
      spatial:
        bbox:
        - -101.88198091547551
        - 53.491466538566925
        - -101.04177012496181
        - 53.94941224261376
        crs: http://www.opengis.net/def/crs/OGC/1.3/CRS84
    providers:
    - type: coverage
      name: odcprovider.OpenDataCubeCoveragesProvider
      data: dsm__MB__The_Pas_2014
      format:
        name: GeoTIFF
        mimetype: application/geotiff
```

The *data* field in the *provider* section corresponds to the product name in ODC. The name field declares the provider class implementation. The pygeoapi-odc-provider library also provides a script to automatically generate a configuration file for pygeoapi including resource entries for all ODC products.

The service is deployed in the cloud using kubernetes. It consists of two pods: a database (image: postgres:13-buster with 10Gi storage) and a service pod created from a 52°North image (https://github.com/52North/ogc-tb-17_datacube-service_odc/blob/main/pygeoapi/Dockerfile). The data is currently stored on persistent volume claims. Using the s3 capabilities of ODC is a useful next step in the development of the provider, but could not be fulfilled during this testbed. The datasets (see Section Clause 7.1.4.1) are downloaded using two init-containers sharing the same storage volume. During this download process, each dataset is enriched with the required metadata documents and added to the ODC index.

7.1.4. API Structure

- Service root
  - Collections
    - Catalog collection providing records, which provides links to
      - Items
        - List of items with properties in html, json, ld-json
          - Item with properties, e.g. associations (aka links) to the coverage itself in the three formats html, json, ld-json
      - Queryables (not implemented)
      - Different formats as html, json, ld-json
    - For each product in the ODC one collection with
      - Bounding box on map
      - Keywords
      - Links from links metadata
      - Links to
        - Collection as html, json, ld-json
        - Domain set as html, json, ld-json
        - Coverage Domain as html, json, ld-json
        - Coverage data, e.g. as GeoTIFF
  - Processes
    - Described in Machine Learning within a Geo Data Cube API

7.1.4.1. Datasets

For the testbed and TIEs, several datasets are loaded into the data cube; on the one hand, four scenes from the Landsat 8 Collection 2 Processing Level 2 as outlined in the following list. On the other hand, 198 datasets from the High Resolution Digital Elevation Model (HRDEM) — CanElevation Series (https://open.canada.ca/data/en/dataset/957782bf-847c-4644-a757-

e383c0057995) of the testbed sponsor NRCan providing terrain and surface model data of Canada, are loaded into the service instance.

- Landsat 8 Collection 2 Processing Level 2
  - Selected bands: blue, green, red, nir, swir1, swir2
  - Scenes:
    - LC08_L2SP_016021_20150824_20200908_02_T1
    - LC08_L2SP_035024_20150813_20200909_02_T1
    - LC08_L2SP_039015_20150809_20200908_02_T1
    - LC08_L2SP_040023_20150731_20200909_02_T1
- NRCan
  - Two projects providing
    - Digital Surface Model (DSM),
    - Digital Terrain Model (DTM), and
    - DSM Colour Hill Shade (CHS)
  - data are selected:
    - Port_Hawkesbury_2016,
    - The_Pas_2014

→ 198 datasets The NRCan data is downloaded from their servers using a shape file (https://ftp.maps.canada.ca/pub/elevation/dem_mne/highresolution_hauteresolution/Datasets_Footprints.zip) providing the metadata and the download locations. This process is implemented as a script for each datasource: NRCan (https://github.com/52North/ogc-tb-17_datacube-service_odc/blob/main/downloader/nrcan.py) and Landsat (https://github.com/52North/ogc-tb-17_datacube-service_odc/blob/main/downloader/landsat8.py).

7.1.5. Specific challenges

7.1.5.1. Multiple CRS within one coverage collection

The chosen datasets revealed a challenge regarding the handling of coordinate reference systems. For example, Landsat 8 uses UTM zones and a single Landsat collection might contain data for several UTM zones. The question is how different CRS within one collection can be handled by API Coverages. The draft specification "OGC API — Coverages — Part 1: Core" currently allows only a single CRS. An “OGC API — Coverages — Part X — CRS” extension that addresses the usage of different CRS for storing, subsetting and outputting coverage data will likely be developed in the future but is not specified yet (see https://github.com/opengeospatial/ogcapi-coverages/issues/144, https://github.com/opengeospatial/ogcapi-maps/issues/82, https://github.com/opengeospatial/ogcapi-common/labels/CRS). A solution could be the usage of coverage partitioning (http://docs.opengeospatial.org/is/09-146r6/09-146r6.html#53), however, this would add an undesired level of complexity. Instead, the data is simply reprojected to WGS84 on-the-fly (in ODC it is still in the native projection) and allow only WGS84 for subsetting and as output CRS. One disadvantage of this approach is that the original data cannot be retrieved and there might be a loss of information due to the reprojection.

7.1.5.2. Scenes API

One possible solution to the CRS challenge could be the introduction of a Scenes API. This would allow access to multiple scenes as a single coverage but also as individual scenes in the original projection. An additional advantage would be the option to filter and sort scenes by scenes metadata. This would be specifically useful in the ML use case (Clause 6). For Landsat 8 large percentages of cloud cover would reduce the quality of the trained model significantly, thus filtering scenes that should be included in the training is very important.

7.1.5.3. Improving pygeoapi

In order to successfully perform TIEs, a few improvements of pygeoapi were necessary, such as binary output for processes and the usage of the “Prefer:” header for asynchronous process execution. These can be found in 52°North’s pygeoapi fork: https://github.com/52North/pygeoapi/tree/deployment/testbed-17.

Additionally github issues were filed in the original pygeoapi repository:

- https://github.com/geopython/pygeoapi/issues/705
- https://github.com/geopython/pygeoapi/issues/772
- https://github.com/geopython/pygeoapi/issues/809
- https://github.com/geopython/pygeoapi/issues/838
- https://github.com/geopython/pygeoapi/issues/845

7.2. Wuhan University Geo Data Cube API Server Implementation (D123)

This section describes the services provided “on top” of the Wuhan University GeoCube infrastructure including *OGC API — Common*, *OGC API — Coverages* and *OGC API — Processes*.

This section also gives some thoughts about a definition for Geo Data Cube (GDC) and the API design from Wuhan University's perspective.

7.2.1. GDC definition from Wuhan University perspective

A geospatial data cube is defined as a time-series multidimensional data model, where multi-source geospatial data can be organized as spatially aligned analysis ready data in a high-performance form. The multi-source geospatial data is not limited to EO data such as remote sensing images, but also can be the vector, trajectory, or tabular data.

7.2.2. Deployment infrastructure

The Wuhan deployment relies on the GeoCube infrastructure. The infrastructure is established on a private cloud environment comprising three physical servers. These servers are connected to a high-performance storage array of PB level.

The GeoCube infrastructure supports the accommodation of multi-source geospatial data including raster and vector data in the cube, these data are segmented into tiles and persisted in an HBase database, which allows a user to perform efficient multi-source data analysis using a high-performance tile form. In the infrastructure, cloud computing technology such as Apache Spark is used to enable large-scale analysis for cube tiles.

7.2.3. Datasets

- Landsat-8 L1TP
- NRCAN DEM
- Gaofen-1
- OpenStreetMap

Please note: The API might not work with every dataset. The API has been tested successfully with Landsat-8 L1TP.

7.2.4. Coverages API Implementation

This implementation establishes how to access data cube through the official WHU GDC endpoint http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/

This section lists some examples of how to access the data cube using the implemented Geo Data Cube API.

Example: `/collections`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections?limit=10&bbox=112.65942,29.23223,115.06959,31.36234&time=2016-08-30T02:55:50Z/2018-08-30T02:55:50Z

The API endpoint for retrieving dataset collections. Query parameters including limit, bbox, and time can be used.

This returns a list of collections in the data cube. The response is shown below.

```
{
    "collections": [
        {
            "id": "NRCAN_DEM_ARD_EO_19780101080000",
            "title": "DEM_ARD",
            "description": "Satellite images of DEM",
            "extent": {
                "spatial": [
                    -95.0,
                    76.4450135938,
                    -55.6861621295,
                    85.1591224444
                ],
                "temporal": [
                    "1978-01-01T08:00:00Z"
                ]
            },
            "crs": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
                "http://www.opengis.net/def/crs/EPSG/0/4326"
            ],
            "links": [
                {
                    "href": "/geocube/gdc_api_v2/collections/NRCAN_DEM_ARD_EO_
19780101080000",
                    "rel": "self",
                    "type": "application/json",
                    "title": "NRCAN_DEM_ARD_EO_19780101080000(as PNG; Note:
requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/NRCAN_DEM_ARD_EO_
19780101080000/coverage?f=png",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
                    "type": "image/png",
                    "title": "NRCAN_DEM_ARD_EO_19780101080000(as PNG; Note:
requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/NRCAN_DEM_ARD_EO_
19780101080000/coverage?f=tif",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
                    "type": "image/tiff; application=geotiff",
                    "title": "NRCAN_DEM_ARD_EO_19780101080000(as geoTiff; Note:
 requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/NRCAN_DEM_ARD_EO_1
9780101080000/coverage/domainset",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-
domainset",
                    "type": "application/json",
                    "title": "NRCAN_DEM_ARD_EO_19780101080000(domain set of
the coverage for this collection)"
```

```
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/NRCAN_DEM_ARD_EO_1
9780101080000/coverage/rangetype",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-
rangetype",
                    "type": "application/json",
                    "title": "NRCAN_DEM_ARD_EO_19780101080000(range type of
the coverage for this collection)"
                }
            ]
        },
        {
            "id": "LC08_L1TP_ARD_EO_20171217025629",
            "title": "Landsat8_ARD",
            "description": "Satellite images of Landsat8",
            "extent": {
                "spatial": [
                    112.62546,
                    29.23323,
                    115.03488,
                    31.36008
                ],
                "temporal": [
                    "2017-12-17T02:56:29Z"
                ]
            },
            "crs": [
                "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
                "http://www.opengis.net/def/crs/EPSG/0/4326"
            ],
            "links": [
                {
                    "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629",
                    "rel": "self",
                    "type": "application/json",
                    "title": "LC08_L1TP_ARD_EO_20171217025629(as PNG; Note:
requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629/coverage?f=png",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
                    "type": "image/png",
                    "title": "LC08_L1TP_ARD_EO_20171217025629(as PNG; Note:
requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629/coverage?f=tif",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
                    "type": "image/tiff; application=geotiff",
                    "title": "LC08_L1TP_ARD_EO_20171217025629(as geoTiff; Note:
 requesting large extent may result in generalized data)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_2
0171217025629/coverage/domainset",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-
domainset",
                    "type": "application/json",
                    "title": "LC08_L1TP_ARD_EO_20171217025629(domain set of
the coverage for this collection)"
                },
                {
                    "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_2
0171217025629/coverage/rangetype",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-
rangetype",
                    "type": "application/json",
                    "title": "LC08_L1TP_ARD_EO_20171217025629(range type of
the coverage for this collection)"
                }
            ]
        },
        ...
    ]
}
```

Example: `/collections/{collectionId}`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629

The API endpoint for describing the data cube collection "LC08_L1TP_ARD_EO_20171217025629".

The response is shown below.

```
{
    "id": "LC08_L1TP_ARD_EO_20171217025629",
    "title": "Landsat8_ARD",
    "description": "Satellite images of Landsat8",
    "extent": {
        "spatial": [
            112.62546,
            29.23323,
            115.03488,
            31.36008
        ],
        "temporal": [
            "2017-12-17T02:56:29Z"
        ]
    },
    "crs": [
        "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
        "http://www.opengis.net/def/crs/EPSG/0/4326"
    ],
    "links": [
        {
            "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629",
            "rel": "self",
            "type": "application/json",
            "title": "LC08_L1TP_ARD_EO_20171217025629(as PNG; Note: requesting
large extent may result in generalized data)"
        },
        {
            "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629/coverage?f=png",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
            "type": "image/png",
            "title": "LC08_L1TP_ARD_EO_20171217025629(as PNG; Note: requesting
large extent may result in generalized data)"
        },
        {
            "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_
20171217025629/coverage?f=tif",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage",
            "type": "image/tiff; application=geotiff",
            "title": "LC08_L1TP_ARD_EO_20171217025629(as geoTiff; Note:
requesting large extent may result in generalized data)"
        },
        {
            "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_201712170
25629/coverage/domainset",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-domainset",
            "type": "application/json",
            "title": "LC08_L1TP_ARD_EO_20171217025629(domain set of the
coverage for this collection)"
        },
        {
            "href": "/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_201712170
25629/coverage/rangetype",
            "rel": "http://www.opengis.net/def/rel/ogc/1.0/coverage-rangetype",
            "type": "application/json",
            "title": "LC08_L1TP_ARD_EO_20171217025629(range type of the
coverage for this collection)"
        }
    ]
}
```

Example: `/collections/{collectionId}/coverage`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629/coverage

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629/coverage?f=tif&bbox=114.21,30.37,114.41,30.57&rangeSubset=Red,SWIR1,Coastal,Pan,Near-Infrared

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629/coverage?f=png&subset=Lon(114.23:114.45)&scale-factor=3

The API endpoint for accessing a data cube. Query parameters including format(png,geotiff), bbox, subset,rangeSubset,scale-size,scale-axes,scale-factor.

This returns true color png or mutli-band geotiff of the coverage according to the in specified area, bands, size and scale.

Example: `/collections/{collectionId}/coverage/rangetype`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629/coverage/rangetype

The API endpoint for accessing cube range type.

This returns the range type of the multi-band coverage as the dimensions of the cube.

```
{
    "type": "DataRecord",
    "field": [
        {
            "type": "Quantity",
            "name": "SWIR1",
            "description": "SWIR1 channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Coastal",
            "description": "Coastal channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Blue",
            "description": "Blue channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Cirrus",
            "description": "Cirrus channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Red",
            "description": "Red channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Green",
            "description": "Green channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Near-Infrared",
            "description": "Near-Infrared channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "Pan",
            "description": "Pan channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        },
        {
            "type": "Quantity",
            "name": "SWIR2",
            "description": "SWIR2 channel",
            "uom": {
                "type": "UnitReference",
                "code": "1"
            },
            "encodingInfo": {
                "dataType": "http://www.opengis.net/def/dataType/OGC/0/float32"
            }
        }
    ]
}
```

Example: `/collections/{collectionId}/coverage/domainset`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/collections/LC08_L1TP_ARD_EO_20171217025629/coverage/domainset

The API endpoint for accessing cube domain set.

This returns the domain set of a coverage.

```
{
    "type": "DomainSet",
    "generalGrid": {
        "type": "GeneralGridCoverageType",
        "srsName": "http://www.opengis.net/def/crs/OGC/1.3/CRS84",
        "axisLabels": [
            "Lon",
            "Lat"
        ],
        "axis": [
            {
                "type": "RegularAxis",
                "axisLabel": "Lon",
                "lowerBound": 112.62546,
                "upperBound": 115.03488,
                "resolution": 27.5
            },
            {
                "type": "RegularAxis",
                "axisLabel": "Lat",
                "lowerBound": 29.23323,
                "upperBound": 31.36008,
                "resolution": 27.5
            }
        ],
        "gridLimits": {
            "type": "GridLimits",
            "srsName": "http://www.opengis.net/def/crs/OGC/0/Index2D",
            "axisLabels": [
                "i",
                "j"
            ],
            "axis": [
                {
                    "type": "IndexAxisType",
                    "axisLabel": "i",
                    "lowerBound": 0.0,
                    "upperBound": 15999.0,
                    "resolution": null
                },
                {
                    "type": "IndexAxisType",
                    "axisLabel": "j",
                    "lowerBound": 0.0,
                    "upperBound": 11999.0,
                    "resolution": null
                }
            ]
        }
    }
}
```

7.2.5. Processes API Implementation

This implementation establishes how to perform analysis on a data cube through the implemented GDC API.

This section lists some examples.

Example: `/processes`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/processes

The API endpoint for retrieving the processes list.

This returns a list of processes implemented in the GeoCube. The response is shown below.

```
[
    {
        "id": "aspect",
        "title": "aspect",
        "version": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "value",
            "reference"
        ],
        "links": [
            {
                "href": "/geocube/gdc_api_v2/processes/aspect",
                "rel": "self",
                "type": "application/json",
                "title": "process description"
            }
        ]
    },
    {
        "id": "slope",
        "title": "slope",
        "version": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "value",
            "reference"
        ],
        "links": [
            {
                "href": "/geocube/gdc_api_v2/processes/slope",
                "rel": "self",
                "type": "application/json",
                "title": "process description"
            }
        ]
    },
    {
        "id": "ndvi",
        "title": "ndvi",
        "version": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "value",
            "reference"
        ],
        "links": [
            {
                "href": "/geocube/gdc_api_v2/processes/ndvi",
                "rel": "self",
                "type": "application/json",
                "title": "process description"
            }
        ]
    },
    {
        "id": "ndwi",
        "title": "ndwi",
        "version": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "value",
            "reference"
        ],
        "links": [
            {
                "href": "/geocube/gdc_api_v2/processes/ndwi",
                "rel": "self",
                "type": "application/json",
                "title": "process description"
            }
        ]
    }
]
```

Example: `/processes/{processId}`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/processes/ndwi

The API endpoint for retrieving a process description (e.g. ndwi).

This returns the description of “ndwi” process. The response is shown below.

```
{
    "id": "ndwi",
    "title": "ndwi",
    "version": "1.0.0",
    "jobControlOptions": [
        "async-execute"
    ],
    "outputTransmission": [
        "value",
        "reference"
    ],
    "inputs": {
        "rasterProductName": {
            "title": "raster input",
            "schema": {
                "type": "string",
                "default": "LC08_L1TP_ARD_EO"
            },
            "minOccurs": 1,
            "maxOccurs": 1
        },
        "extent": {
            "title": "extent",
            "description": "Bounding box of the extent to process",
            "schema": {
                "allOf": [
                    {
                        "format": "ogc-bbox"
                    },
                    {
                        "$ref": "https://raw.githubusercontent.com/
opengeospatial/ogcapi-processes/master/core/openapi/schemas/bbox.yaml"
                    }
```

```
                        ],
                        "default": {
                            "bbox": [
                                113.3,
                                30.5,
                                113.5,
                                30.7
                            ],
                            "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
                        }
                    },
                    "minOccurs": 0,
                    "maxOccurs": 1
                },
                "startTime": {
                    "title": "startTime",
                    "schema": {
                        "type": "string",
                        "default": "2017-01-01"
                    },
                    "minOccurs": 1,
                    "maxOccurs": 1
                },
                "endTime": {
                    "title": "endTime",
                    "schema": {
                        "type": "string",
                        "default": "2018-01-01"
                    },
                    "minOccurs": 1,
                    "maxOccurs": 1
                }
            },
            "outputs": {
                "ndwiResult": {
                    "title": "NDWI Result",
                    "description": "Normalize Difference Water Index",
                    "schema": {
                        "oneOf": [
                            {
                                "type": "string",
                                "contentEncoding": "binary",
                                "contentMediaType": "image/tiff; application=geotiff"
                            },
                            {
                                "type": "string",
                                "contentEncoding": "binary",
                                "contentMediaType": "image/png"
                            }
                        ]
                    }
                }
            },
            "links": [
                {
                    "href": "geocube/gdc_api_v2/processes/ndwi/execution",
                    "rel": "http://www.opengis.net/def/rel/ogc/1.0/execute",
                    "title": "NDWI Execute endpoint"
                }
            ]
        }
```

Example: `/processes/{processId}/execute`

http://geos.whu.edu.cn:8097/geocube/processes_api/processes/ndwi/execution

The API endpoint for creating a new job.

Request body example:

```
{
   "process" : "http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/processes/ndwi",
   "inputs" : {
      "rasterProductName" : "LC08_L1TP_ARD_EO",
      "startTime" : "2017-01-01",
      "endTime" : "2018-01-01",
      "extent" : {
         "bbox" : [ 114.3, 30.5, 114.5, 30.7 ],
         "crs" : "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
      }
   }
}
```

The response is as follows:

```
{
    "jobID": "60fe95e1-a077-40fe-b05e-346bd6373880",
    "progress": 40,
    "links": {
        "href": "/geocube/gdc_api_v2/processes/ndwi/jobs/60fe95e1-a077-40fe-
b05e-346bd6373880",
        "rel": "self",
        "type": "application/json",
        "title": "ndwi"
    },
    "type": "processes",
    "message": "Process is running",
    "status": "running"
}
```

Example: `/processes/{processId}/jobs/{jobId}`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/processes/ndwi/jobs/{jobId}

The API endpoint for retrieving status of a job, which retrieves same returns as the above.

Example: `/processes/{processId}/jobs/{jobId}/results`

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/processes/ndwi/jobs/{jobId}/results

The API endpoint for retrieving results of a job, which are encoded as follows:

```
[
    {
        "id": "2017_01_24",
        "value": {
            "inlineValue": "/geocube/gdc_api_v2/results/view/F59E2F8CE40897CFB0
3CA50512096317/60fe95e1-a077-40fe-b05e-346bd6373880/NDWI_2017_01_24.png"
        }
    },
    {
        "id": "2017_12_17",
        "value": {
            "inlineValue": "/geocube/gdc_api_v2/results/view/F59E2F8CE40897CFB0
3CA50512096317/60fe95e1-a077-40fe-b05e-346bd6373880/NDWI_2017_12_17.png"
        }
    },
    {
        "id": "2017_08_27",
        "value": {
            "inlineValue": "/geocube/gdc_api_v2/results/view/F59E2F8CE40897CFB0
3CA50512096317/60fe95e1-a077-40fe-b05e-346bd6373880/NDWI_2017_08_27.png"
        }
    }
]
```

7.2.6. Geo Data Cube (GDC) API design

NOTE: Here `{cubeID}` is equivalent to the *OGC API — Common — Part 2: Geospatial data* `{collectionID}` and highlights the fact that this resource is a representation of a multi-dimensional data cube.

- `/collections`: to get the list of GDC instances. Each cube instance is described by multi-dimensions including extent, time range, list of available products, and resolution if it is a raster product. The products can contain various datasets like raster data, vector data, and point cloud data. These cube instances can be filtered by the dimensions.
- `/collections/{cubeID}`: to describe the cube. Returns the dimension information of the cube. Continuous dimension (e.g. space and time) is comprised by a range, while discrete dimension (e.g. product and resolution) consists of a list of discrete members.
- `/collections/{cubeID}/coverage`: to access data from the collection using *OGC API — Coverages*.
- `/collections/{cubeID}/processes`: to perform analytics functions using *OGC API — Processes*.
- `/collections/{cubeID}/dapa`: to get the list of available data retrieval patterns including (based on DAPA): for example
  - `/collections/{cubeID}/dapa/area:aggregate` + dimension: space,time,product,resolution (aggregate along these dimensions)
  - `/collections/{cubeID}/dapa/position:aggregate` + dimension: time,product,resolution (aggregate along these dimensions)

The following endpoints are specific to the Wuhan University implementation:

- `/collections/{cubeID}/dimensions`: to describe the shared dimensions formalizing that cube, which can be space, time, product, theme and so on.
- `/collections/{cubeID}/sources`: to describe the data sources composing that cube, which can be a variety of products.

- `/collections/{cubeID}/cells`: to get the list of cells/facts in the cube, each cell contains a measure (single pixel/point or collection of pixels/points) which is characterized by the dimensions. These cells can be filtered by the dimensions (product, space, time, and theme), allowing multi-dimensional subsetting.
- `/collections/{cubeID}/cells/{cellId}`: to get the detailed information of one particular cell/fact in the cube.

7.3. MEEO Geo Data Cube API Server Implementation (D177)

The MEEO server has been developed to follow or encapsulate the set of OGC API specifications for the Geo Data Cube API.

This server supports the following specifications: OGC API — Common, Records, Tiles and Processes (to be finalized).

The datasets are collected and organized on an NFS volume at MEEO Cloud Infrastructure; the indexing of data stored on public s3 buckets is also supported.

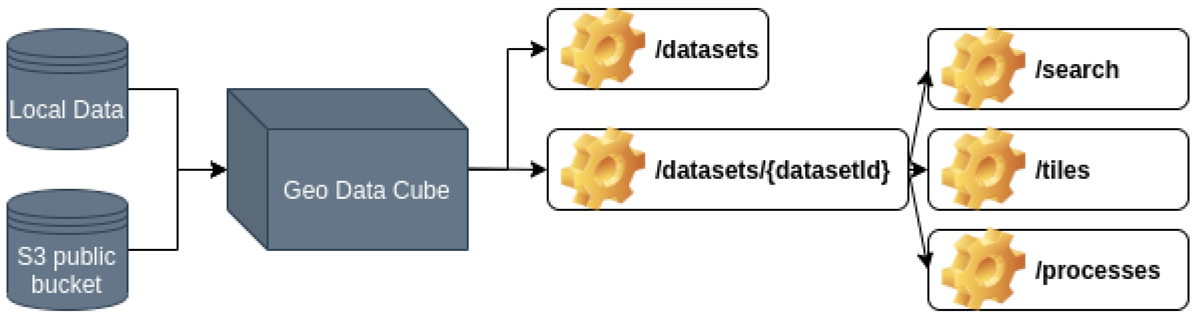


Figure 15 — Architecture of MEEO service implementation

7.3.1. MEEO Server Endpoint

https://testbed17.adamplatform.eu/datacube/api/v0

7.3.2. Idea for a Geo Data Cube

The idea behind the MEEO Geo Data Cube service is to index the datasets (data) suggested in the CFP, and to expose a series of services per datasets, namely: discovery (e.g. OGC API — Records or equivalent), access (e.g. OGC API — Tiles or equivalent) and processing (e.g. OGC API — Processes or equivalent).

The idea is not to define a closed list of services and standards, but rather to provide human/machine readable services to support the exploitation of the data.

7.3.3. Geo Data Cube discovery

https://testbed17.adamplatform.eu/datacube/api/v0/datasets

This API provides the list of datasets indexed in the Geo Data Cube instance.

7.3.4. Geo Data Cube dataset fetch

https://testbed17.adamplatform.eu/datacube/api/v0/datasets/CA_harvest_year

This API provides details about the dataset = datasetID (e.g. CA_harvest_year), including description and services.

In the description subdocument, all the information about geolocation, time range, data type, etc. can be found.

In the services subdocument all the enabled services can be found.

7.3.5. Geo Data Cube dataset Records API

https://testbed17.adamplatform.eu/datacube/api/v0/datasets/CA_harvest_year/search

This API exposes the discovery service to list the actual resources (i.e. raster) available for the dataset `{datasetID}` (e.g. *CA_harvest_year*).

This API relies on the OpenSearch conformance class, with Geo and Time extensions.

7.3.6. Geo Data Cube dataset Tiles API

https://testbed17.adamplatform.eu/datacube/api/v0/datasets/CA_harvest_year/tiles/gist_rainbow_mpl;nodata=0.000000;colorrange=(0.000000,115.000000)/1984-12-31T00:00:00Z/1985-01-01T00:00:00Z/EPSG:3857/9/157/171.png

This API exposes the access and visualization services for exploiting the actual resources (i.e. raster data) available for the dataset `{datasetID} (e.g. *CA_harvest_year*).

This API relies on the Dataset Tile Sets conformance class, for what concerns the Style, and on the Geo Data Resource Selection conformance class, for what regards Tiles.

7.4. Ecere Geo Data Cube API Server Implementation (in-kind)

Ecere’s GNOSIS Map Server is an OGC certified compliant implementation of *OGC API — Features*, and supports a number of additional OGC API standards and draft specifications, including Common, Coverages, Features — Part 2: CRS and Part 3: Filtering, Processes (including Workflows & Chaining), Tiles, Maps, Styles, GeoVolumes and Routes.

7.4.1. Elevation datasets

Elevation datasets from NRCan’s HRDEM of the Red River area in Manitoba, and of the Ottawa river were loaded onto Ecere’s OGC API demonstration server.

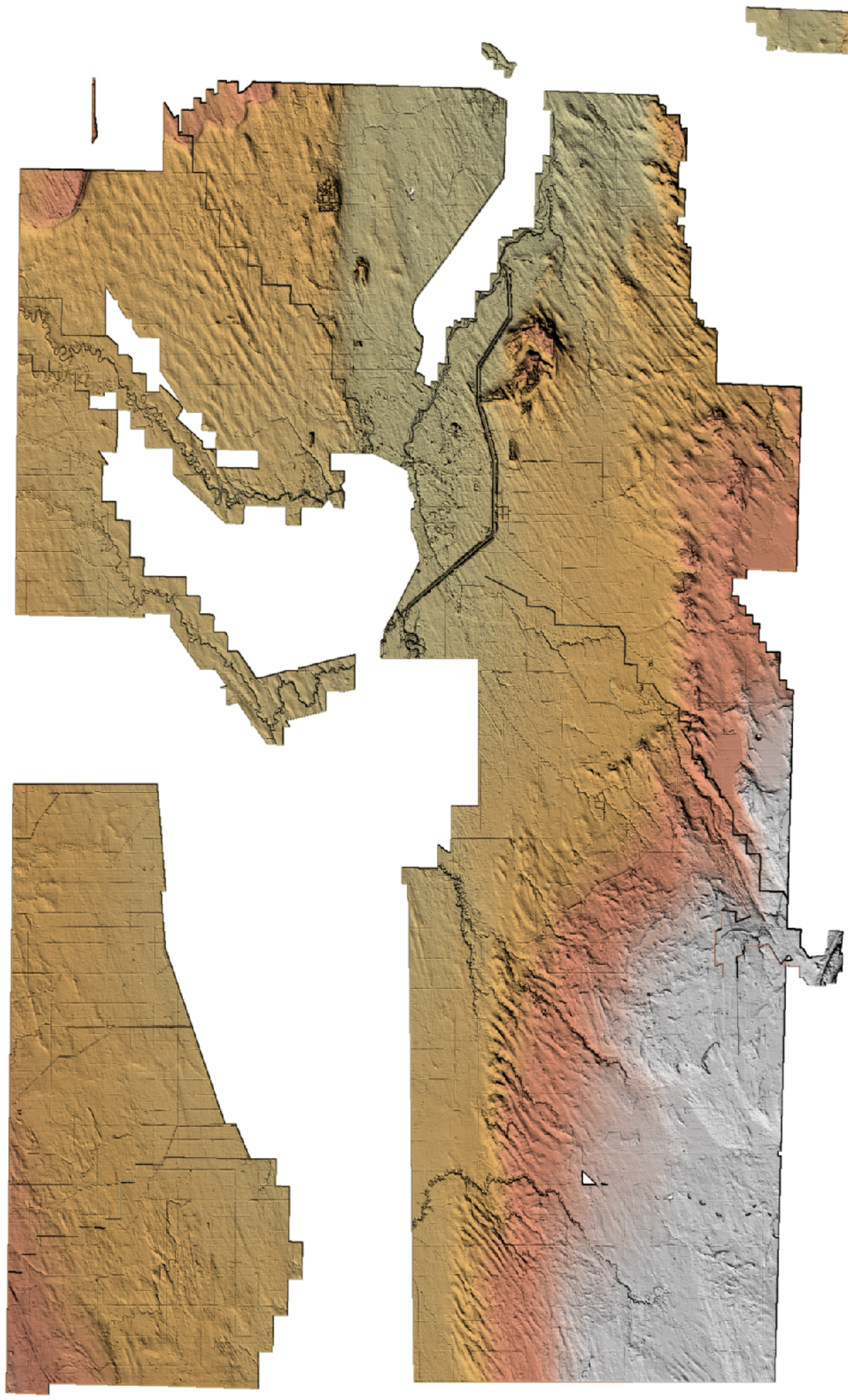


Figure 16 — NRCan HRDEM for Red River area in Manitoba at 1 and 2 m resolution rendered by Ecere's GNOSIS Map Server

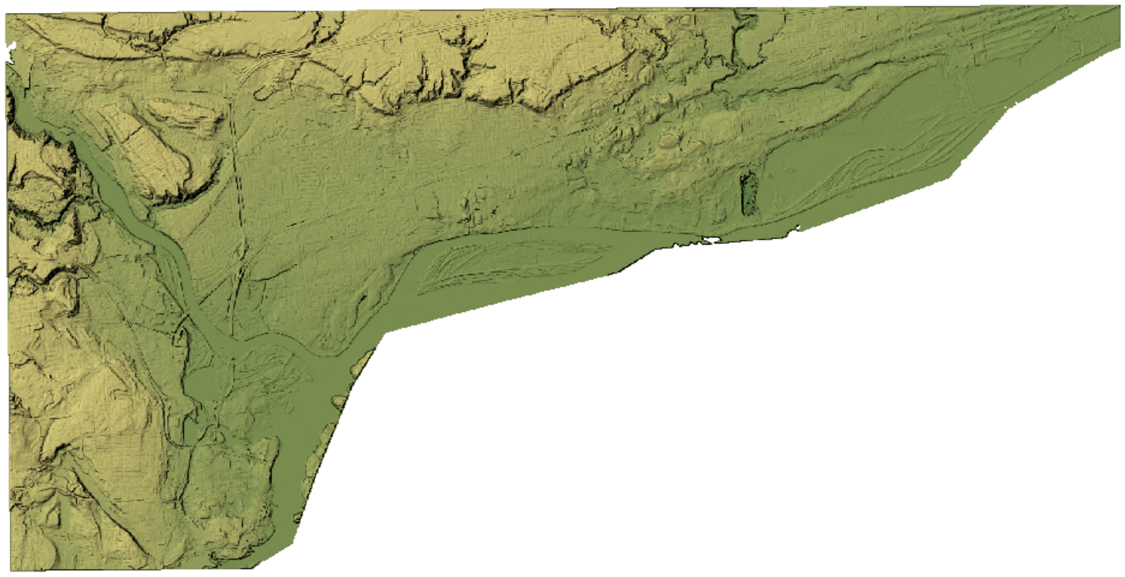


Figure 17 — NRCan HRDEM for Ottawa river area rendered by Ecere's GNOSIS Map Server

Bathymetry datasets from CHS NONNA, were also loaded for resolutions of 10 m (partial dataset) and 100 m (whole dataset).

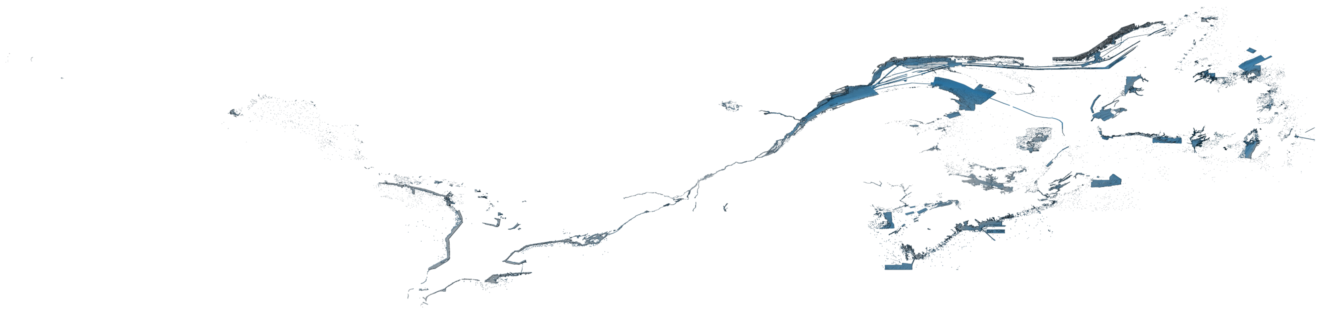


Figure 18 — CHS NONNA at 10m resolution rendered by Ecere's GNOSIS Map Server

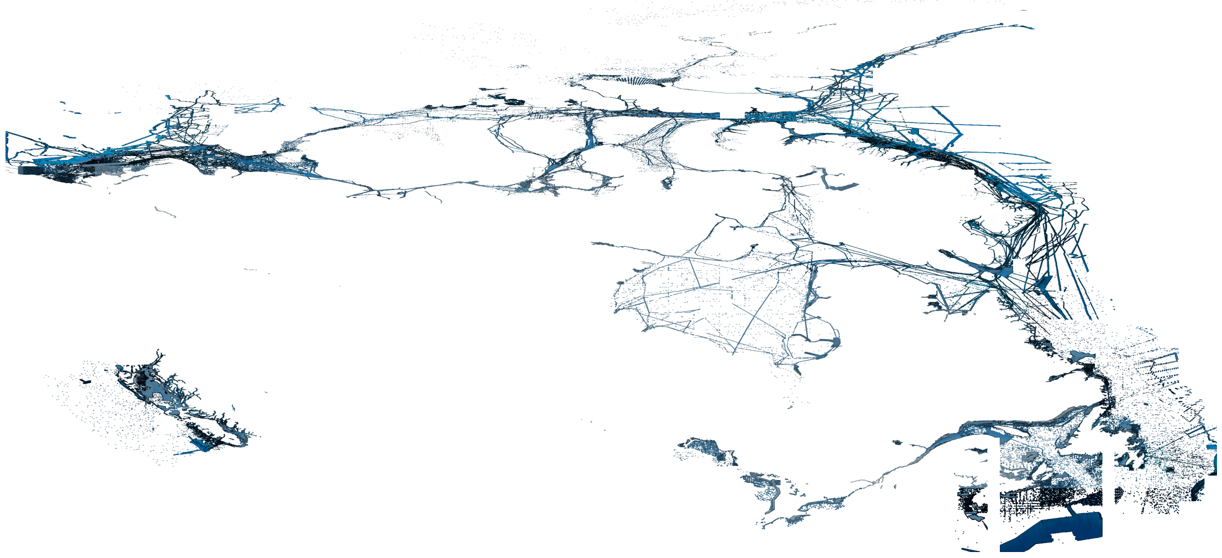
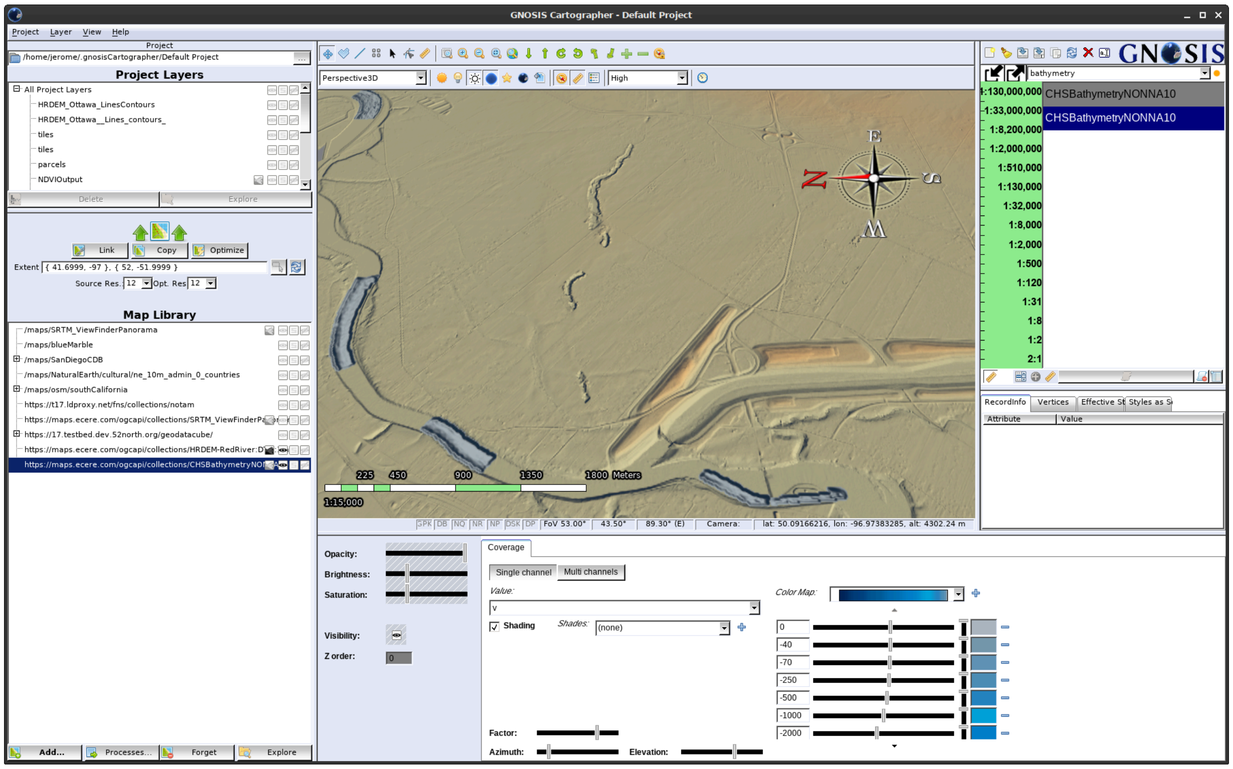


Figure 19 — CHS NONNA at 100m resolution rendered by Ecere's GNOSIS Map Server

In the following screenshot, both terrestrial elevation from the HRDEM and bathymetry from the 10 m NONNA are accessed and visualized together in Ecere's GNOSIS Cartographer client.



Project: /home/jerome/.gnosisCartographer/Default Project

Project Layers:
- All Project Layers
  - HRDEM_Ottawa_LinesContours
  - HRDEM_Ottawa__Lines_contours_
  - tiles
  - tiles
  - parcels
  - NDVIOutput

Extent { 41.6999, -97 }, { 52, -51.9999 }

Source Res. 12 Opt. Res 12

Map Library:
- /maps/SRTM_ViewFinderPanorama
- /maps/blueMarble
- /maps/SanDiegoCDB
- /maps/NaturalEarth/cultural/ne_10m_admin_0_countries
- /maps/osm/southCalifornia
- https://t17.ldproxy.net/fns/collections/notam
- https://maps.ecere.com/ogcapi/collections/SRTM_ViewFinderPa
- https://17.testbed.dev.52north.org/geodatacube/
- https://maps.ecere.com/ogcapi/collections/HRDEM-RedRiver:D
- https://maps.ecere.com/ogcapi/collections/CHSBathymetryNO

bathymetry

1:130,000,000 CHSBathymetryNONNA10
1:33,000,000 CHSBathymetryNONNA10
1:8,200,000
1:2,000,000
1:510,000
1:130,000
1:32,000
1:8,000
1:2,000
1:500
1:120
1:31
1:8
1:2
2:1

225 450 900 1350 1800 Meters

1:15,000

GPK DB NQ NR NP DSK DP FoV 53.00° 43.50° 89.30° (E) Camera: lat: 50.09166216, lon: -96.97383285, alt: 4302.24 m

Opacity:
Brightness:
Saturation:
Visibility:
Z order: 0

Value: v

☑ Shading Shades: (none)

Factor:
Azimuth: Elevation:

Color Map:

0
-40
-70
-250
-500
-1000
-2000

Figure 20 — HRDEM from RedRiver and CHS NONNA at 10m resolution visualized accessed from Ecere's GNOSIS Map Server and visualized in GNOSIS Cartographer

7.4.2. Coverages API Implementation

The server implements support for *OGC API — Coverages*, including the subsetting and scaling conformance classes. Efforts were spent during the initiative to enable the ability to serve multi-band data cubes made up of multiple scenes such as Landsat-8 imagery. However, these capabilities, which will also support range subsetting and the proposed Scenes API capability, remain to be completed.

The following query URL demonstrates accessing raw values from an elevation data cube served by Ecere's OGC API demonstration server sourced from a high resolution Digital Elevation Model provided by NRCan, downsampled (by a factor of 2) and subset along both the latitude and longitude axes, using the *Coverages API* as a GeoTIFF.

Downsampled and subset coverage request

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/coverage?scale-factor=2&subset=Lat(45.44:45.47),Lon(-75.7:-75.6)&f=geotiff

The following URLs point to the collection description resource for this data cube, which itself provides a link to the description of the domain and range, which in this case are also available separately.

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/coverage/domainset?f=json

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/coverage/rangetype?f=json

The server also supports a prototype of what a CRS extension could look like, based on an approach similar to *OGC API — Features — Part 2*. For example, the following returns the same coverage projected as World Mercator (EPSG:3395) rather than the default in Plate carrée (EPSG:4326):

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/coverage?crs=epsg:3395&scale-factor=8&f=tiff

7.4.3. Processes API Implementation

Ecere's GNOSIS Map Server supports *OGC API — Processes — Part 1: Core* using synchronous execution. The Map Server also supports the draft *Part 3: Workflows & Chaining* specification allowing first setting up an execution of an individual process or a workflow, and then triggering execution for an area and resolution of interest, and retrieval of the result in a negotiated format. The retrieval can be done using *OGC API — Tiles* regardless of whether the result of the execution is a coverage, a collection of vector features or a map, or using *OGC API — Features*, *Coverages* or *Maps* depending on the type of output. If the result of the execution is vector data, requests can still be made using *OGC API — Maps* or *Tiles* which will additionally trigger server side rendering of the output.

A list of processes is available at the `https://maps.ecere.com/ogcapi/processes` end-point.

Process description example

https://maps.ecere.com/ogcapi/processes/RFClassify?f=json

Requesting the *RFClassify* process description will return the following JSON response.

```
{
  "id" : "RFClassify",
  "title" : "Random Forest Classification",
  "version" : "1.0.0",
  "jobControlOptions": [
    "sync-execute", "workflow-collection"
  ],
  "outputTransmission" : [ "value" ],
  "description" : "This process outputs a random-forest classified image using
imagery and training feature dataset",
  "links" : [ {
    "href" : "https://maps.ecere.com/ogcapi/processes/RFClassify/execution",
    "rel" : "http://www.opengis.net/def/rel/ogc/1.0/execute",
    "title" : "Execution endpoint"
  } ],
  "inputs" : {
   "data" :
    {
      "title" : "The data set",
      "description" : "The collection containing the imagery for the
randomforest process.",
      "minOccurs" : 1,
      "maxOccurs" : 1,
      "schema" : {
        "oneOf": [
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "image/tiff; application=geotiff"
          },
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "image/png"
          }
        ]
      }
    }
  },
  "outputs" : {
    "classification" :
    {
      "title" : "Classified image",
      "description" : "Classified image",
      "schema" : {
        "oneOf": [
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "image/tiff; application=geotiff"
          },
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "image/png"
          }
        ]
      }
    }
  }
}
```

Process execution

A process execution is submitted at `/processes/{processId}/execution` for synchronous execution. At the publication date of this ER, a separate endpoint (`/processes/{processId}`) was used to set up a workflow and retrieve a description of the resulting virtual collection. In the future, this capability will likely move to the same `/execution` endpoint, differentiated from a regular synchronous or asynchronous execution request by using a query parameter to request a collection description or landing page to be returned.

Inputs to the process are specified as part of the execution request. All defined outputs are returned by default, if not explicitly including an "outputs" section in the request. The HTML representation of the process endpoint provides a form to easily submit an execution request. For example, the following request can be submitted to the *RenderMap* process located at `https://maps.ecere.com/ogcapi/processes/RenderMap`:

```
{
   "process" : "https://maps.ecere.com/ogcapi/processes/RenderMap",
   "inputs" : {
      "background" : "navy",
      "transparent" : false,
      "layers" : [
         { "collection" : "https://maps.ecere.com/ogcapi/collections/CHSBathyme
tryNONNA100" },
         { "collection" : "https://maps.ecere.com/ogcapi/collections/CHSBathyme
tryNONNA10" }
      ]
   }
}
```

Upon execution the output is generated and exists as a temporary virtual collection, e.g. .../`ogcapi/scratch/31FDA020`. These can be accessed and used like any other collection.

7.4.4. Tiles API Implementation

The server supports vector and coverage data as well as map tiles.

The tile URL template for coverages is:

```
/collections/{collectionID}/coverage/tiles/{tileMatrixSetId}/{tileMatrix}/{tile
Row}/{tileCol}.{format}
```

Multiple tiling schemes are supported. This example requests the HRDEM-Ottawa coverage tiles as GeoTIFF with the WebMercatorQuad tiling scheme.

HRDEM-Ottawa coverage tiles

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/coverage/tiles/WebMercatorQuad/13/2931/2374.tif

The request for the same dataset as map tiles would be as follows:

HRDEM-Ottawa map tiles

https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa/map/tiles/WebMercatorQuad/13/2931/2374.png

8

IMPLEMENTED CLIENT COMPONENTS

8 IMPLEMENTED CLIENT COMPONENTS

This chapter describes the different Geo Data Cube client components developed and enhanced during this project by Solenix, Ethar, and Ecere.

The following table summarizes the capabilities implemented by each client:

Table 3 — Clients components

PROVIDER	CAPABILITIES
Solenix (Web WorldWind)	Coverages (1,2,3,4), Processes (Workflows: 10)
Ethar	Coverages (1,2,3,4), Augmented Reality
Ecere (GNOSIS Cartographer)	Coverages (1,2,3,4,5,6), Processes (7,8, Workflows: 9,10)

Table 4 — GDC API Capabilities

1. Common-1: Core
2. Common-2: Collections
3. Coverages-1
4. Coverages-1 (Subsetting)
5. Coverages-1 (Range subsetting)
6. Coverages-1 (Scaling)
7. Processes-1 (Sync execution)
8. Processes-1 (Async execution)
9. Processes-3: Workflows (Collection Input)
10. Processes-3: Workflows (Collection Output)

8.1. Solenix Geo Data Cube API Client Implementation (D124)

The Solenix Geo Data Cube API client consists of three main components:

1. The API client that is generated based on the preliminary Geo Data Cube API definition and used to communicate with corresponding servers
2. A connector for Web WorldWind that bridges between the experimental GDC API and ties in the API responses into visual layers on WorldWind using existing means, such as placemarks, image layers, 3D models (e.g. for DEMs)
3. A demonstrator that uses Web WorldWind to showcase the end-to-end capabilities.

The demonstrator is deployed at https://ogctb17-gdc-breithorn.solenix.ch/#/globe

The most interesting aspects about this experiment are:

- Execution in a browser exhibits particular challenges for the execution and communication with remote services as it is executed in the restricted JavaScript sandbox in a browser. Considerations such as HTTP vs. HTTPS access, CORS and limitations in processing of data are to be considered.
- Exploration of the API Client in a real-world scenario, using Web WorldWind
- Focus on a constrained environment (web browser, JavaScript runtime) that puts specific emphasis for features on the server, e.g. sub-setting, resampling.

Clients are an integral part of the execution, analysis and improvement of TIE experiments as they identify the intended use cases and possible limitations in the backend, middleware or frontend applications.

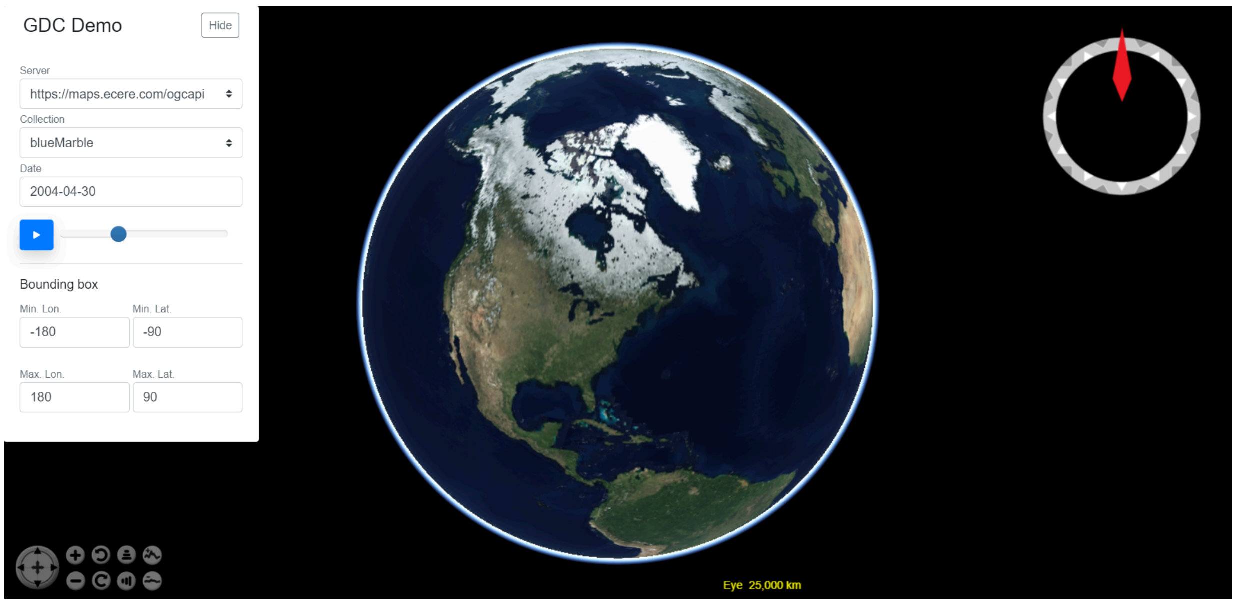


Figure 21 — WebWorldWind client visualizing Blue Marble spatio-temporal data cube from Ecere GDC API

8.1.1. Scope and functional Focus

The client can connect to different data back-ends and uses the same API client implementation for all of them. In the course of the thread described in this ER, various flavors of a Data Cube API were discussed. Consequently, the servers may implement specific subsets or deviate in certain aspects from the REST resource structure that this GDC client expects.

The TIE experiments were thus focused on covering the basics such as initial connection, collection discovery, and metadata retrieval for all servers. Ideally, access to the data provided by resources is then available in a systematic fashion, where the URL pattern / template is part of the metadata and allows the client to retrieve the tiles of the resource independent of the server's implementation.

8.1.2. Goal and Challenges

The following goals and challenges were tackled with the development of this client:

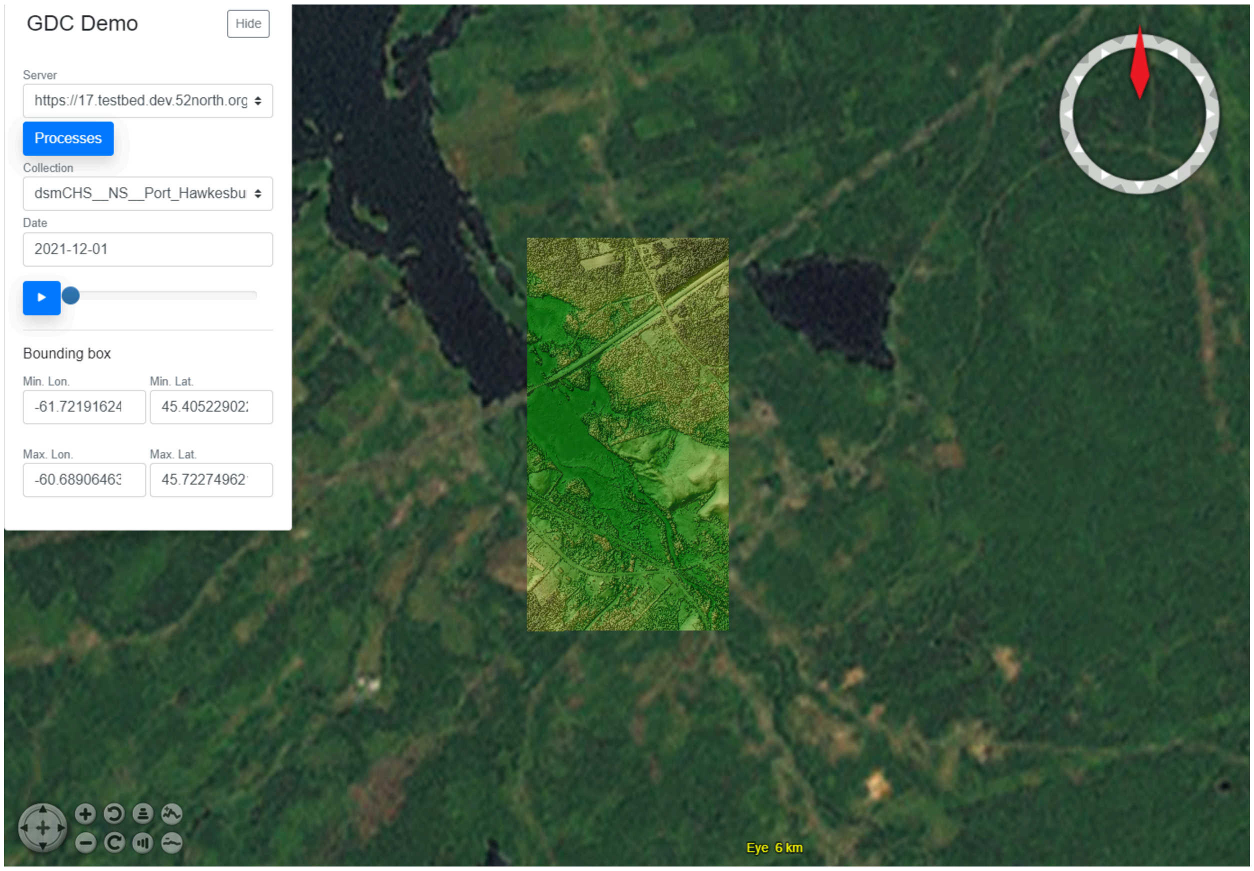
- The client API should be as uniform as possible, allowing connection to different servers that provide the standard API endpoints (landing page, collection enumeration, links to those collections).
  A connection should be possible to any server that provides these basic endpoints.
- The GDC API is not strictly defined yet and could potentially contain data in any OGC format (e.g. coverages, features, records, etc).
  - Response formats should be supported for the demonstrator to a reasonable degree, i.e. not necessarily with all depth that the standard has to offer but with enough

    functionality that the concept can be shown. Accordingly, the client's development was started with using other OGC APIs and consuming their respective responses in order to visualize them and adapted to use the correct resource URLs and requests specific to the GDC API.

  - Servers that deviate too far from the majority consensus of presenting data may not be supported. There is not enough time to implement multiple extensive client libraries.

- The servers should support HTTPS, as the demonstrator is hosted on an HTTPS server and HTTP resources are considered insecure and will be blocked.
- The GDC API should support tiling / sub-setting / mip-mapping of data in order to fetch reasonable sized partial resources to overlay on the globe.
  - Reasonable size means 1-3x the final rendered size at a particular zoom level. A little bit of oversampling is ok and manageable, but gigabyte-sized images or vector collections are not.
  - Reasonable size for images / raster data means common OpenGL texture buffer sized images, e.g. 1024×1024 — 4096×4096 (depending on hardware and driver support. The lower bound limit might be preferred for resource constrained environments.)

Larger images need to be tiled either way in order to render them on screen. Preferably, this tiling is already done at the data source. Even though it is possible to do more complex processing of downloaded files, to retrieve specific slices of a file and to do more complex processing in a browser, it is not an efficient use of resources. A more efficient approach in terms of processing power is to pre-compute results on the server and provide them via a Tiling capability on the server instead of downloading gigabytes of data into a resource constrained browser environment, possibly via slow internet connection, to then possibly discard most of it to render the visible pixels on screen.

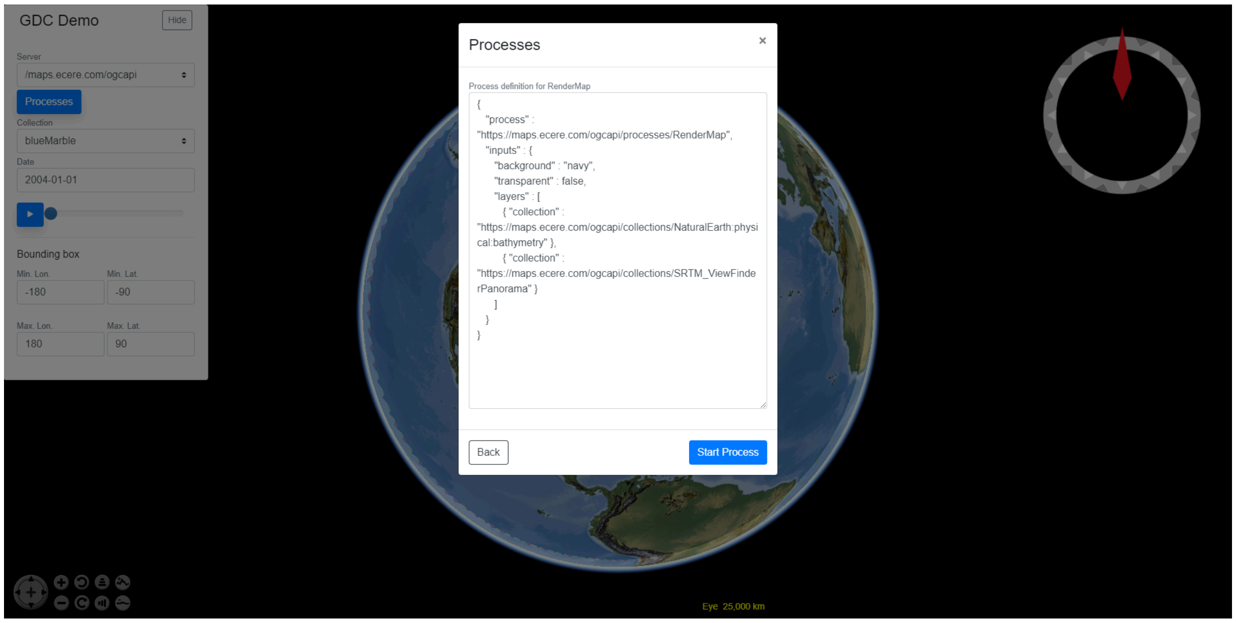


Server: https://17.testbed.dev.52north.org
Collection: dsmCHS__NS__Port_Hawkesbu
Date: 2021-12-01
Min. Lon.: -61.72191624
Min. Lat.: 45.40522902
Max. Lon.: -60.68906463
Max. Lat.: 45.72274962

Figure 22 — WebWorldWind client visualizing elevation data from the 52°North GDC API

8.1.3. Implementation

A Typescript API was generated using the OpenAPI specification from Ecere's server implementation. This generated API provides Typescript bindings for maps, coverages and processes. The client application written in Angular uses the API to retrieve data and displays it on the WebWorldWind globe.



Server: /maps.ecere.com/ogcapi

Collection: blueMarble

Date: 2004-01-01

Bounding box

Min. Lon.	Min. Lat.
-180	-90

Max. Lon.	Max. Lat.
180	90

Process definition for RenderMap

```
{
   "process" :
"https://maps.ecere.com/ogcapi/processes/RenderMap",
   "inputs" : {
      "background" : "navy",
      "transparent" : false,
      "layers" : [
         { "collection" :
"https://maps.ecere.com/ogcapi/collections/NaturalEarth:physi
cal:bathymetry" },
         { "collection" :
"https://maps.ecere.com/ogcapi/collections/SRTM_ViewFinde
rPanorama" }
      ]
   }
}
```

Figure 23 — WebWorldWind client visualizing processing results from Ecere GDC API using *Workflows & Chaining* (collection output)

The following functionality is implemented on the client:

- Retrieving list of collections
- Defining bounding box
- Defining time using a time slider
- “Play” button to automatically move the time forward
- Rendering output PNG or GeoTIFF on the globe
- Retrieving list of processes
- Defining process input as JSON
- Executing process synchronously
- Rendering process output PNG on the globe
- Retrieving heightmap data (SRTM_ViewFinderPanorama from Ecere API)

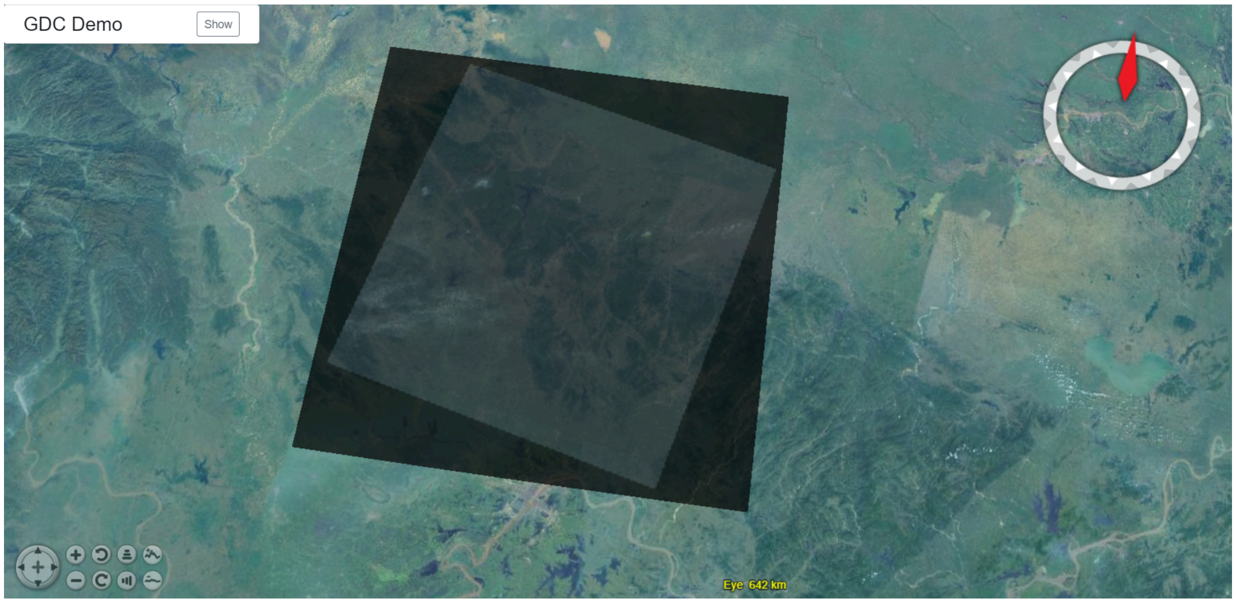
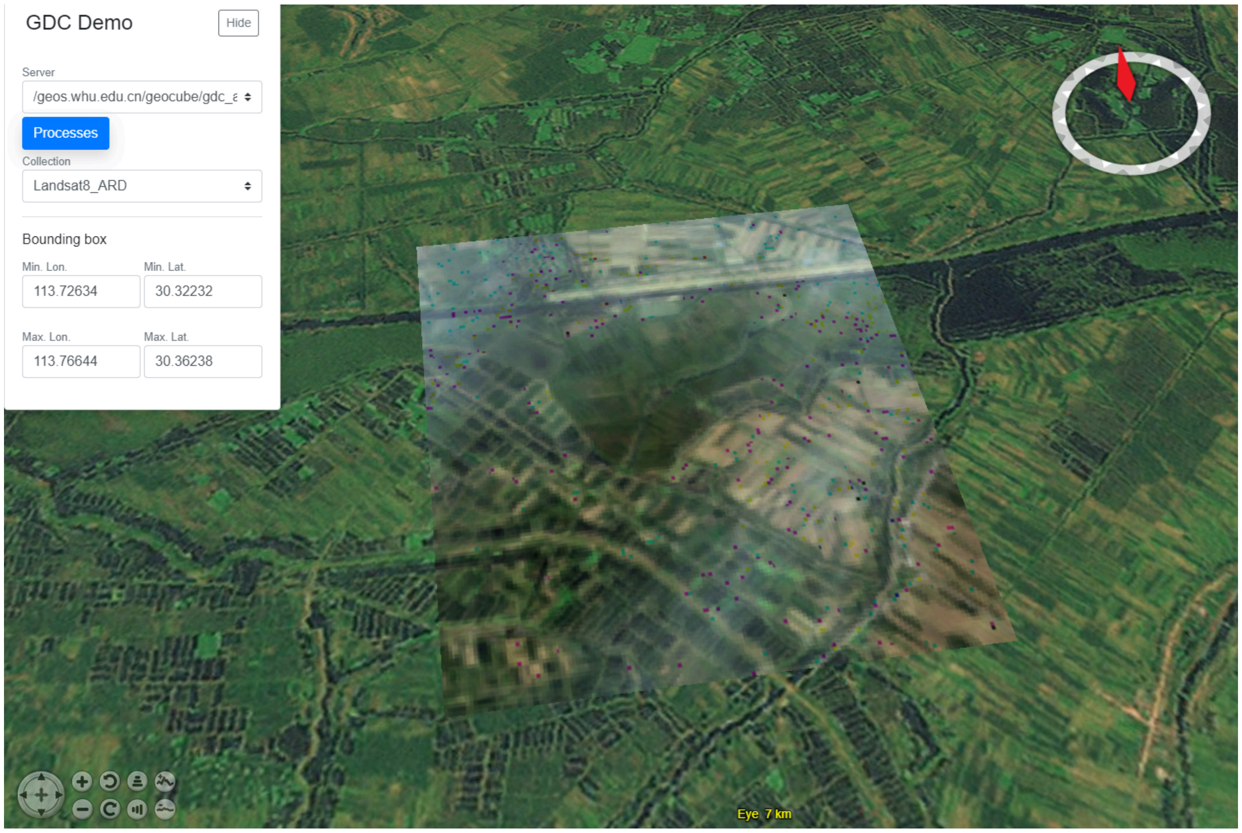


Figure 24 — WebWorldWind demonstrator application showing successful Coverages TIE with Wuhan University GDC API



Server: /geos.whu.edu.cn/geocube/gdc_a

Collection: Landsat8_ARD

Bounding box

Min. Lon.	Min. Lat.
113.72634	30.32232
Max. Lon.	Max. Lat.
113.76644	30.36238

Figure 25 — WebWorldWind demonstrator application showing successful Coverages TIE with Wuhan University GDC API at a smaller scale

8.1.4. Future improvements

The following improvements are recommended for implementation in future testbeds:

- Visualizing CoverageJSON data
- Asynchronous process execution
- Band subsetting
- Scaling
- UI for defining heightmap inputs

8.2. Ethar Geo Data Cube API Client Implementation (D125)

The Ethar Geo Data Cube API client is a web-based geospatial client with WebXR-based immersive visualization, based on the preliminary Geo Data Cube API definition.

We believe that our WebXR-based approach to geo data visualization enjoys the benefits of an open web platform. By leveraging emerging web-standards for XR we reduce the risk of obsolescence, reduce the risk of platform lock-in and reduce maintenance costs. Building for the web also offers superior control over app publishing: app functionality can be updated instantly, the architecture of an app enjoys the interoperable flexibility of a modern web-stack, and deployment can be centralized for many platforms. Using web technologies also offers the benefit that it is a trusted platform for user-data, and avoids some of the pitfalls and subversion that can be encountered with on-device apps or user data.

Our client uses Web Assembly which allows us to bring optimized hardware native code to the web:

- Allows use of C/C++/Rust and other languages
- Supports many system APIs
- Compiles to bytecode, which at runtime compiles to machine code via JIT for near-native performance
- Integrates with JavaScript and web APIs including WebGL, Fetch
- Typically 70-95% native performance on benchmarks

The example client implementation can be found here:

https://au.gmented.com/app/OGCT17/?usePolyfill=0

The client requires either an Oculus Quest, Hololens 2, Windows PC VR with Google Chrome, or Android mobile phone running Google Chrome (this app depends on experimental browser features and needs a flag set).

Additional detail and examples may be found here:

https://www.ethar.com/ogc17/

The novel technical challenges addressed by Ethar's client implementation include:

- Client is web-based, leveraging recent and experimental browser features including WebAssembly and WebXR.
- This implementation operates within the relatively restrictive code-execution environment of the web browser and must contend with the constraints of web-stack, in particular the orchestration of JavaScript, WebAssembly and cross-origin isolation.
- Cross-browser, and cross-platform support for web-based augmented reality is nascent, and in practice much compatibility coding is required to support as many web-browsers as possible.

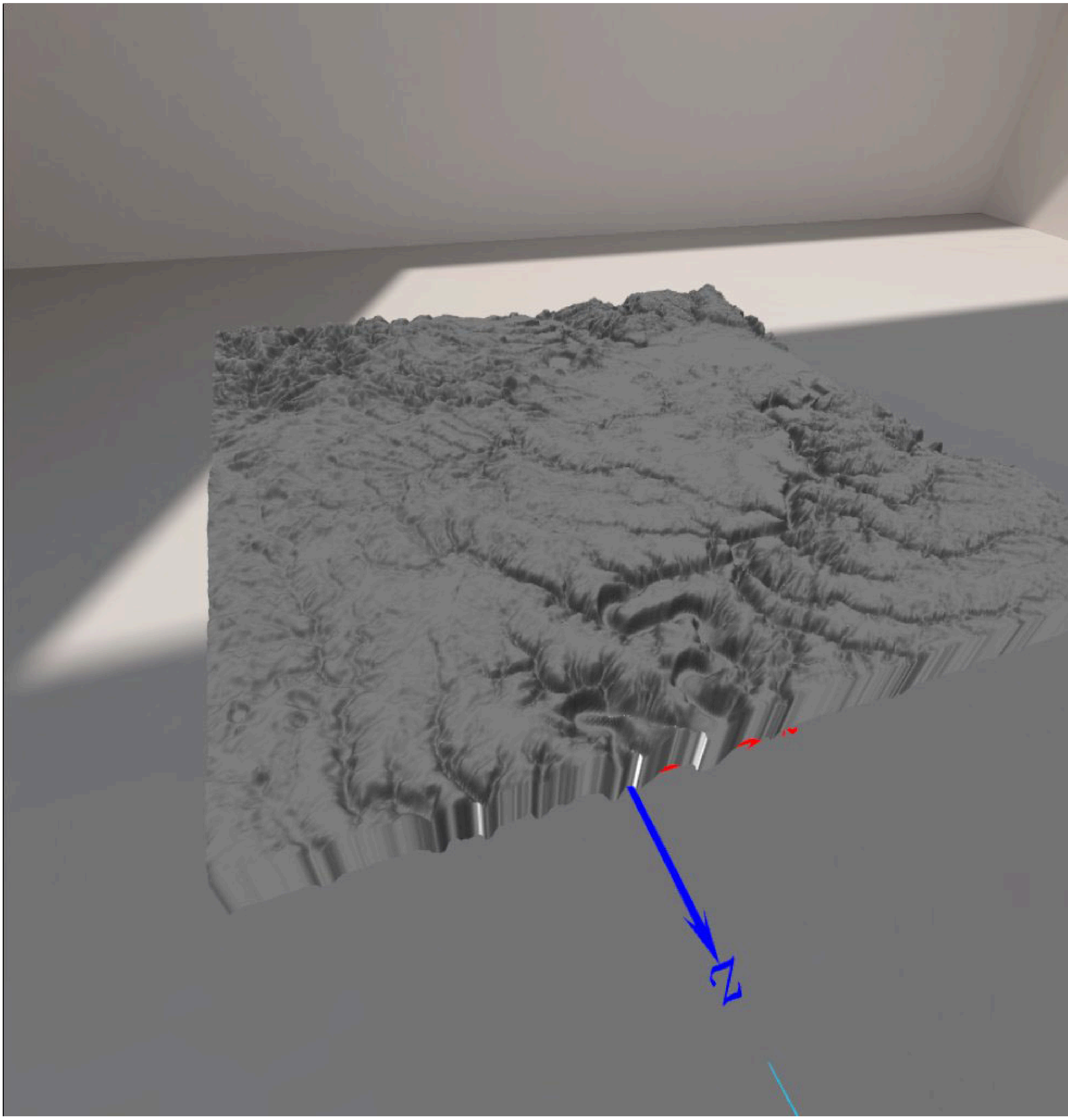


Figure 26 — Ethar GDC WebXR Client Screenshot

8.2.1. Scope and functional Focus

The client can connect to any service that uses the draft Geo Data Cube API definition. As this initial definition is still rather provisional, and there has been some debate on how exactly a GDC service differs from the Coverages API, the client implements only a subset of the potential endpoints considered over the course of the testbed. With this in mind, the goal was to implement example data-visualizations that make use of each of the draft implementations of the GDC API. Subsequently the TIE testing for this client focused on a key subset of the possible endpoints described in the initial GDC API. The client can render visualizations using various data-types which may be served via GDC, including OpenSceneGraph (OSG) native, but also GeoTIFF (.tif) and glTF binary format (.glb).

8.2.2. Implementation goals

The Ethar client was built with the following goals in mind:

- To create a geospatial data visualization tool specifically for an Augmented Reality context.
- To create a visualization environment that can make use of and integrate traditional geospatial data formats alongside new and emerging formats that are purpose-built for AR and VR applications (such as supporting both established formats like GeoTIFF and emerging formats such as glTF).
- To promote open and interoperable tooling by demonstrating the potential for a device-agnostic geospatial data visualization client for AR-enabled devices.
- Initially the participants proposed that the Ethar client would implement support for the Spatial Discovery Service (SDS) and the draft OGC GeoPose specification. These are emerging draft standards for Augmented Reality which might nicely compliment that functionality of a Geo Data Cube service. However, as testbed work progressed, it became apparent that these feature additions were premature (especially in the case of SDS).
  - The GeoPose approach was initially proposed by Ethar as a means of querying for data present in a Geo Data Cube. However, after some discussion, it was determined that GeoPose limits the scope of a query based on orientation (and potentially field-of-view).
  - GeoPose may still be useful as part of records submitted to a Geo Data Cube Service, such as a dataset describing the current position of field-workers actively connected to a GDC service.

- To demonstrate augmented reality contexts that could be useful both on-site and off-site. Thus two discrete ‘modes’ were implemented;
  - An ‘in-situ’ mode where data visualizations are scaled to align against the real-world as a field-worker might see it while on-site.
  - A ‘tabletop’ mode where data-visualizations are scaled for off-site planning purposes.

8.2.3. Implementation

The Ethar client is implemented using a suite of web-friendly technologies. The client is written primarily with JavaScript and C/C++. These very different languages can work in concert thanks to WebAssembly, and some of the experimental WebXR features now present in select web-browsers. To maximize the varieties of geospatial data the client can support the OpenSceneGraph library is used.

The following functionality is implemented on the client:

- Retrieving a list of n-dimensional data associated with a given location.
- Exploring change over time via a slider input control.
- Visualizing GeoTIFF or glTF datasets.

8.2.4. Future improvements

The following improvements are recommended for implementation in future testbeds:

- Visualizing additional common geospatial data formats in an Augmented Reality context.
- Interfacing with additional data APIs besides the preliminary Geo Data Cube API definition.
- Support additional web-browsers/hardware-platforms:
  - Safari and Firefox for macOS/iOS
  - Helio for Magic Leap
  - Firefox Reality for XR1/XR2 platforms (e.g. Oculus, HTC, Pico VR, etc)
- Additional User Interface components:
  - Scaling
  - Heightmap variables
  - Expanded Pan, Tilt, Zoom, Rotate (PTZR) controls

8.3. Ecere Geo Data Cube API Client Implementation (in-kind)

Ecere's *GNOSIS Cartographer* client is capable of requesting, processing and rendering multidimensional geospatial data in accordance with the OGC API specifications being considered as building blocks for the Geo Data Cube API. The GNOSIS Cartographer client supports the following specifications: OGC API — Common, Coverages (including subsetting, scaling and range subsetting), Features (including Part 2: CRS), Processes (including Part 3: Workflows & Chaining), Tiles, Maps and Styles. During the Testbed initiative, efforts were spent to improve support for multi-band coverages as well of coverages made up of multiple scenes.

For any OGC API data source, an end-user only needs to point to the URL of a collection or landing page after clicking an *Add data source* button to start visualizing a data cube.

8.3.1. Coverages API Implementation

The client is capable of requesting and rendering coverages from OGC API endpoints, with support for subsetting, range subsetting and scaling conformance classes. The coverage data can be styled by either assigning band (fields) values to red, green, blue or alpha channels with the option to perform band arithmetic calculations, or as a single channel for which a colormap and/or hillshading can be applied.

In the Testbed 17 Geo Data Cube API task, successful *Coverages* TIEs were achieved with services provided by 52°North and Wuhan University. In previous projects and code sprints, the client could also successfully visualize data cubes served by the EuroDataCube and by rasdaman using the *Coverages* API.

Accessing and visualizing data cubes using the *Coverages API*

The following screenshot is visualizing the 10-meter resolution bathymetry from CHS NONNA hosted by Ecere's GNOSIS Map Server at `https://maps.ecere.com/ogcapi/collections/CHSBathymetryNONNA10`.

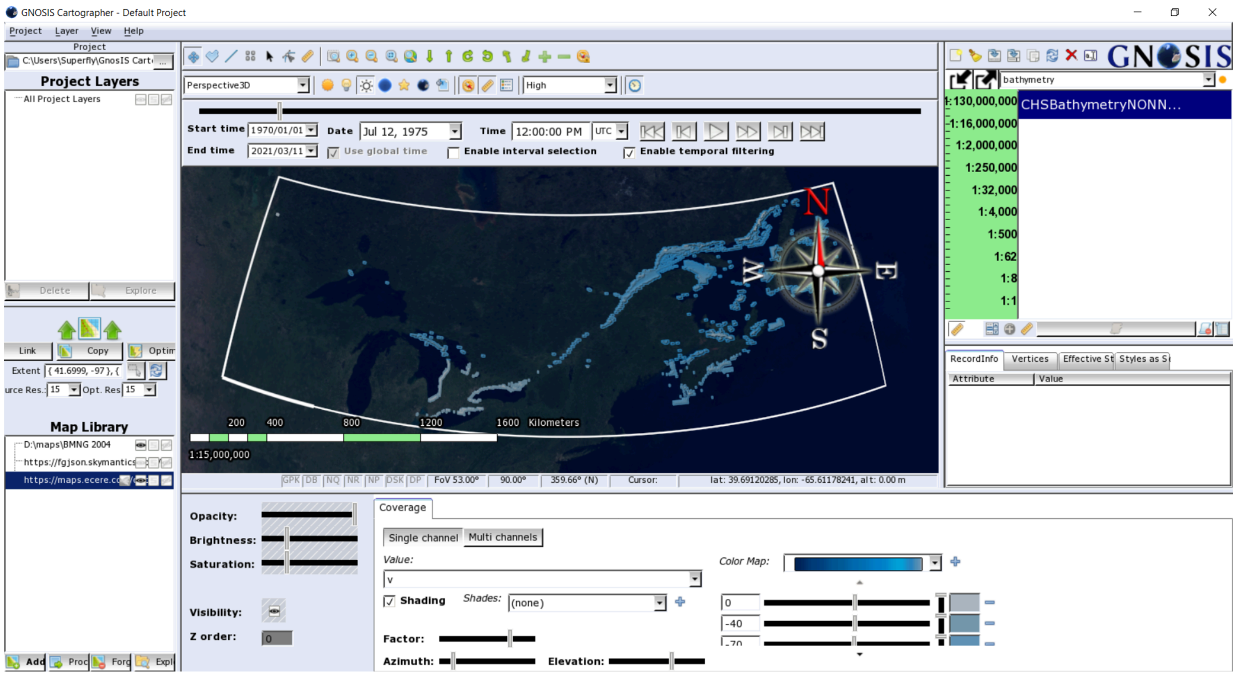


Figure 27 — Ecere's GNOSIS Cartographer client visualizing 10-meter resolution bathymetry from CHS NONNA

The following screenshots demonstrate visualizing data cubes provided by 52°North's GDC API implementation at https://17.testbed.dev.52north.org/geodatacube/.

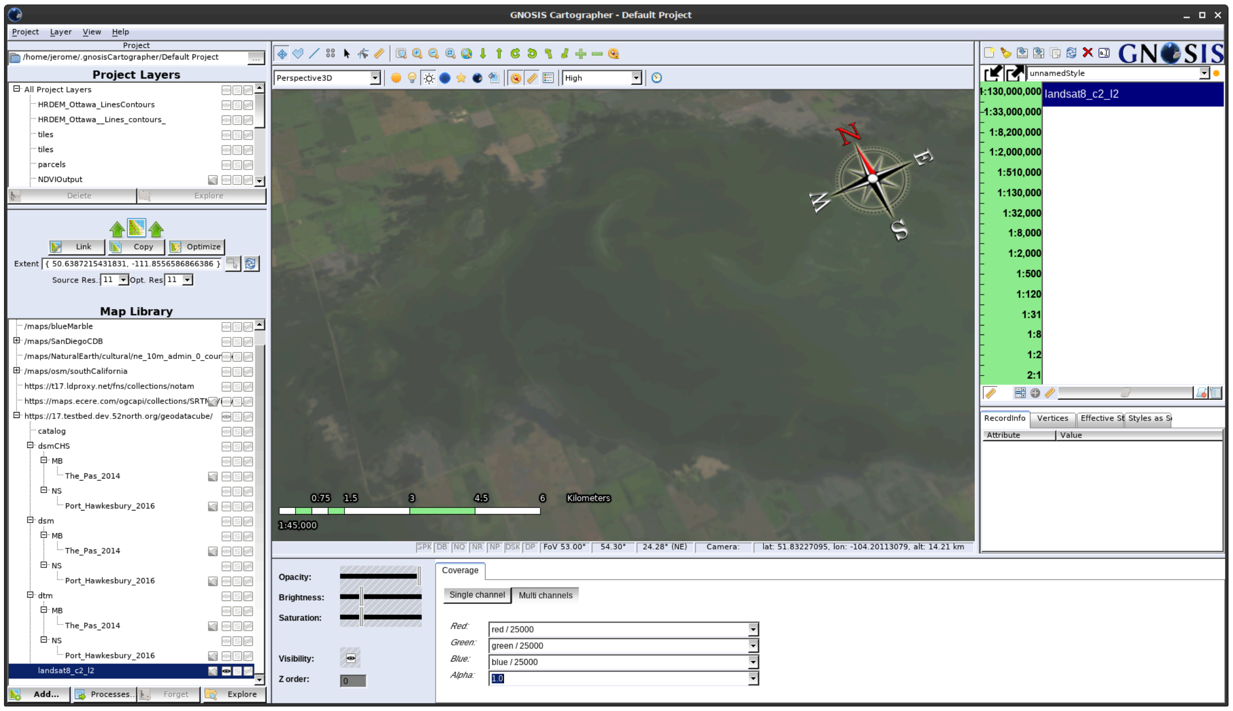
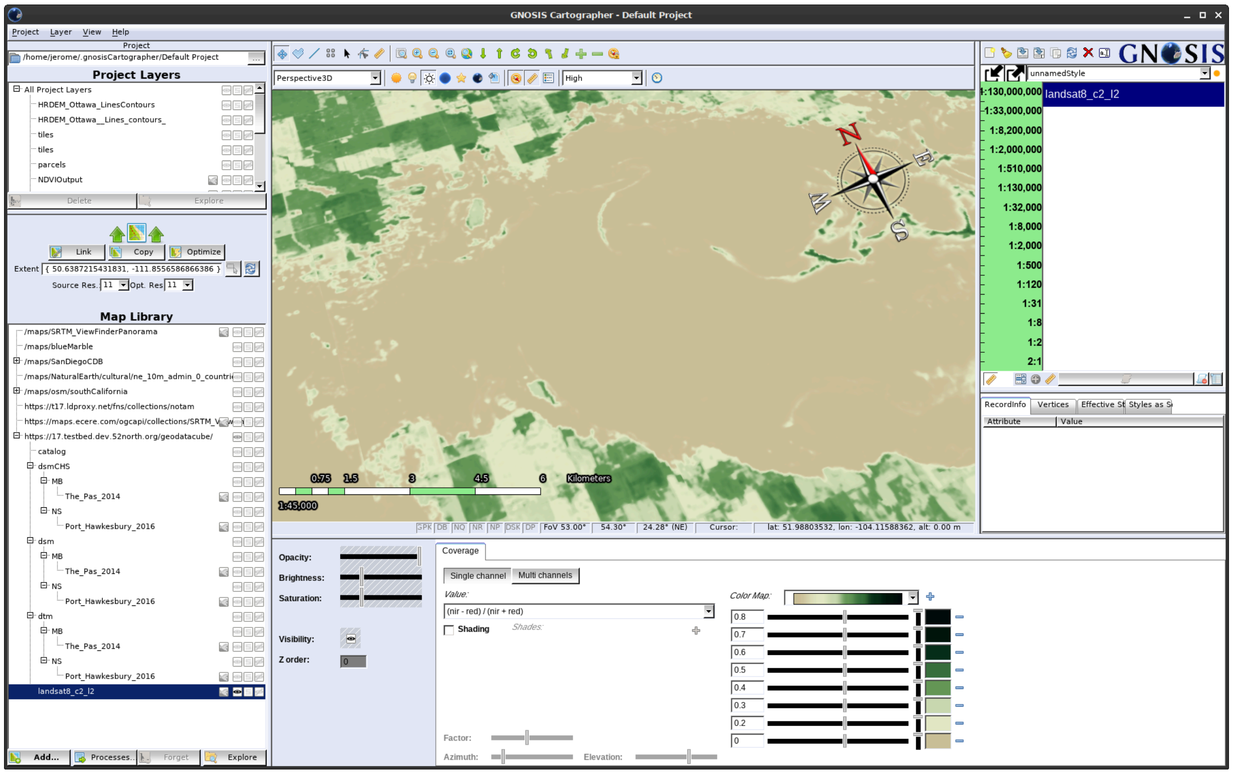


Figure 28 — Ecere's Cartographer client visualizing Landsat-8 imagery from 52°North Coverages implementation



Project: /home/jerome/.gnosisCartographer/Default Project

Extent { 50.6387215431831, -111.8556586866386 }
Source Res. 11 Opt. Res 11

Perspective3D High

FoV 53.00° 54.30° 24.28° (NE) Cursor: lat: 51.98803532, lon: -104.11588362, alt: 0.00 m

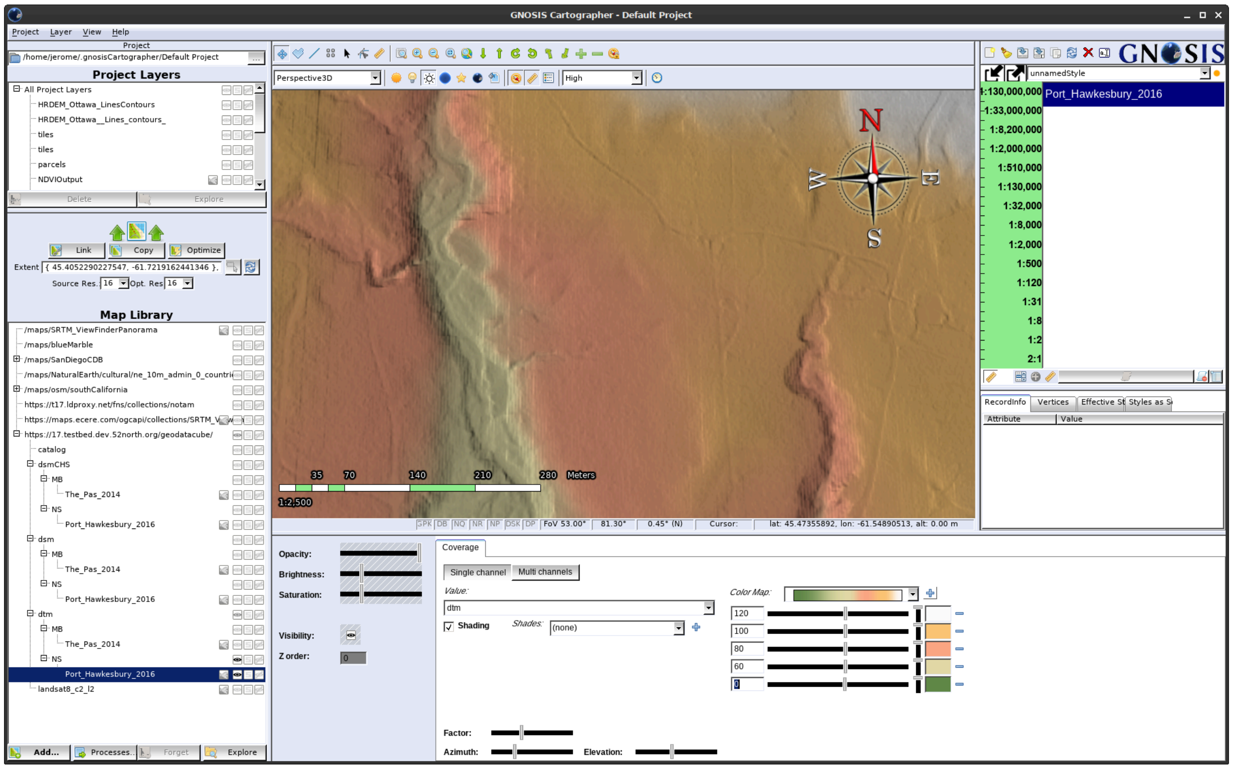
Style: unnamedStyle landsat8_c2_l2

Value: (nir - red) / (nir + red)

Color Map: 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0

Z order: 0

Figure 29 — Ecere's Cartographer client visualizing Landsat-8 imagery as NDVI from 52°North Coverages implementation



Project: /home/jerome/.gnosisCartographer/Default Project

All Project Layers
- HRDEM_Ottawa_LinesContours
- HRDEM_Ottawa__Lines_contours_
- tiles
- tiles
- parcels
- NDVIOutput

Extent { 45.4052290227547, -61.7219162441346 },
Source Res. 16 Opt. Res 16

- /maps/SRTM_ViewFinderPanorama
- /maps/blueMarble
- /maps/SanDiegoCDB
- /maps/NaturalEarth/cultural/ne_10m_admin_0_countri
- /maps/osm/southCalifornia
- https://t17.ldproxy.net/fns/collections/notam
- https://maps.ecere.com/ogcapi/collections/SRTM_V
- https://17.testbed.dev.52north.org/geodatacube/
  - catalog
  - dsmCHS
    - MB
      - The_Pas_2014
    - NS
      - Port_Hawkesbury_2016
  - dsm
    - MB
      - The_Pas_2014
    - NS
      - Port_Hawkesbury_2016
  - dtm
    - MB
      - The_Pas_2014
    - NS
      - Port_Hawkesbury_2016
  - landsat8_c2_l2

1:130,000,000 Port_Hawkesbury_2016
1:33,000,000
1:8,200,000
1:2,000,000
1:510,000
1:130,000
1:32,000
1:8,000
1:2,000
1:500
1:120
1:31
1:8
1:2
2:1

GPK DB NQ NR NP DSK DP FoV 53.00° 81.30° 0.45° (N) Cursor: lat: 45.47355892, lon: -61.54890513, alt: 0.00 m

Z order: 0

Value: dtm
[✓] Shading Shades: (none)

Color Map:

120	⬜
100	🟧
80	🟥
60	🟨
0	🟩

Figure 30 — Ecere's Cartographer client visualizing Digital Terrain Model from 52°North Coverages implementation

The following screenshot demonstrates visualizing a data cube provided by Wuhan University's GDC API implementation at http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/.

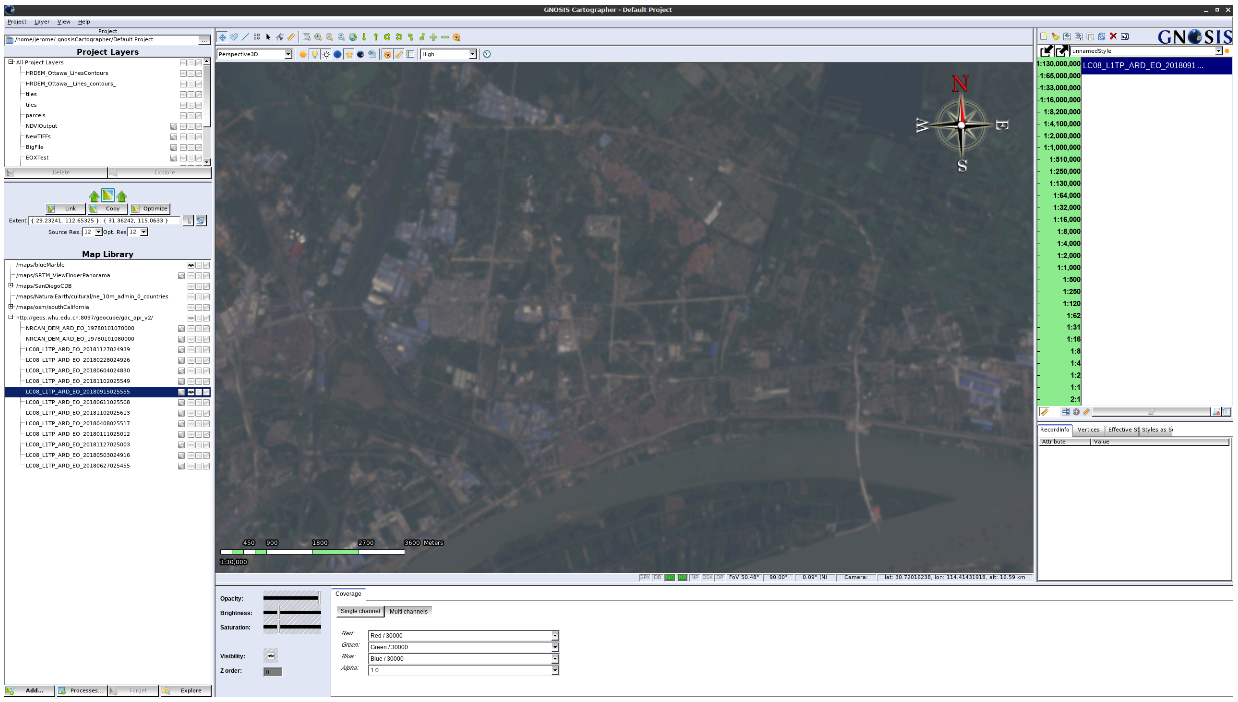


Figure 31 — Ecere's Cartographer client visualizing Landsat-8 from Wuhan University's Coverages implementation

The following screenshot demonstrates initial progress listing the data cubes available from MEEO's *OGC API — Common* implementation at https://testbed17.adamplatform.eu/datacube/api/v0 .

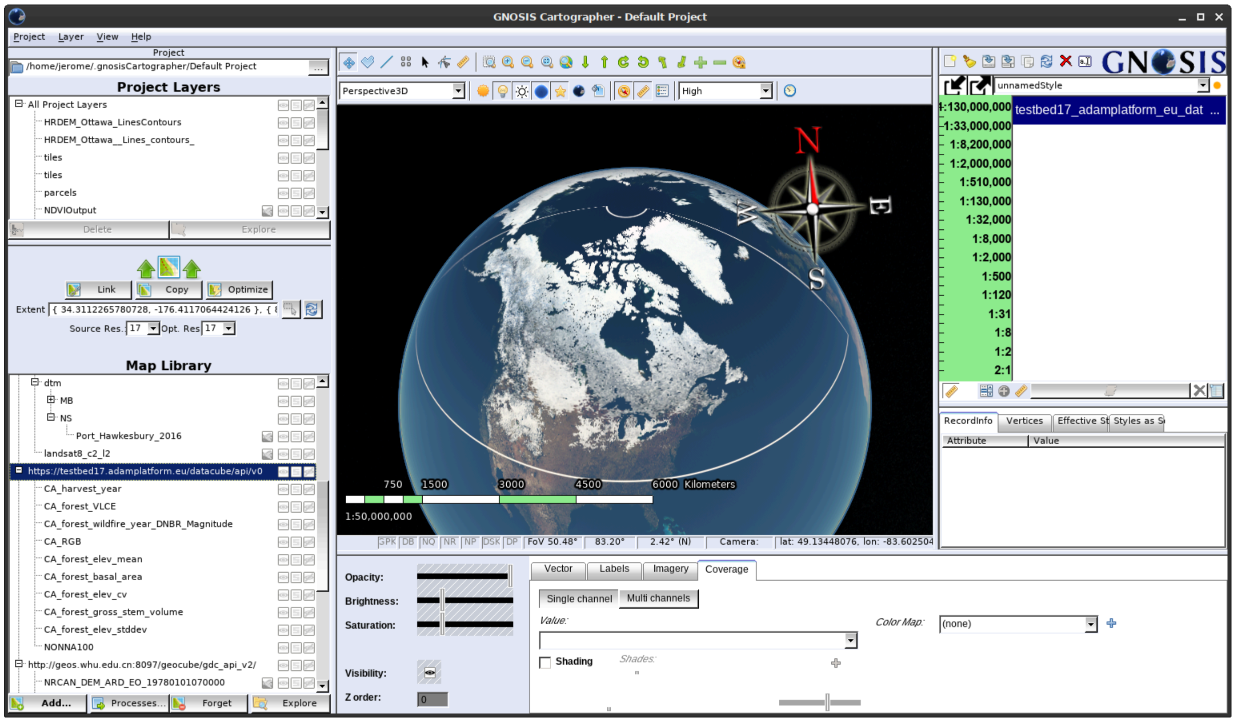


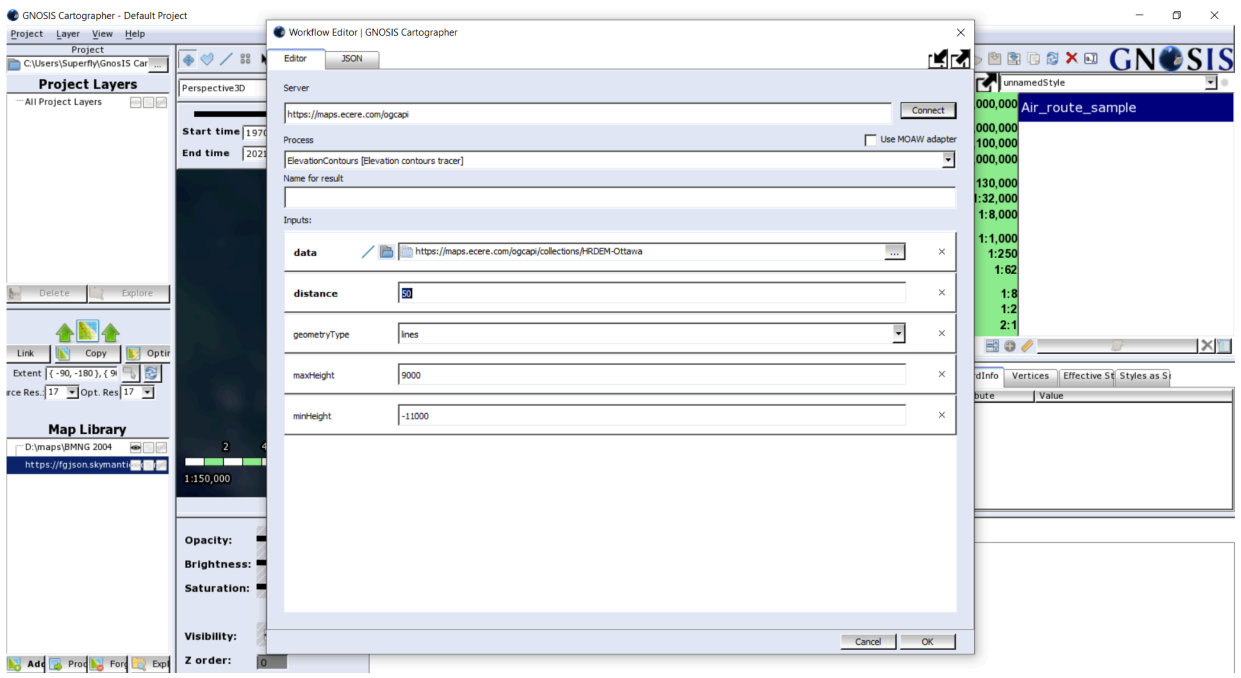
Figure 32 — Ecere's Cartographer client accessing MEEO's OGC API implementation

8.3.2. Processes API Implementation

GNOSIS Cartographer’s workflow editor allows a user to access an *OGC API — Processes* instance and discover the list of available processes that can be executed either stand-alone or as part of a workflow. Inputs and parameters can be configured with the user interface, or directly crafting the execution request in a JSON editor. Executing the workflow adds the output as a data source which can be visualized by the client in the viewport. Virtual collections resulting from a workflow, as implemented in Ecere’s GNOSIS Map Server, can also be directly added as data sources without the client having to submit an execution request.

Executing processes and workflows

Pictured below is workflow configuration of a contour generation process executed on Ecere’s GNOSIS Map Server and its output visualized as vector tiles in GNOSIS Cartographer. The processing is performed on demand and on a tile-by-tile basis, considering the area and resolution of interest.



Server: https://maps.ecere.com/ogcapi

Process: ElevationContours [Elevation contours tracer]

Name for result:

Inputs:

data	https://maps.ecere.com/ogcapi/collections/HRDEM-Ottawa
distance	50
geometryType	lines
maxHeight	9000
minHeight	-11000

Figure 33 — Configuration of workflow using GUI tool in Cartographer client to process HRDREM-Ottawa dataset elevation contours

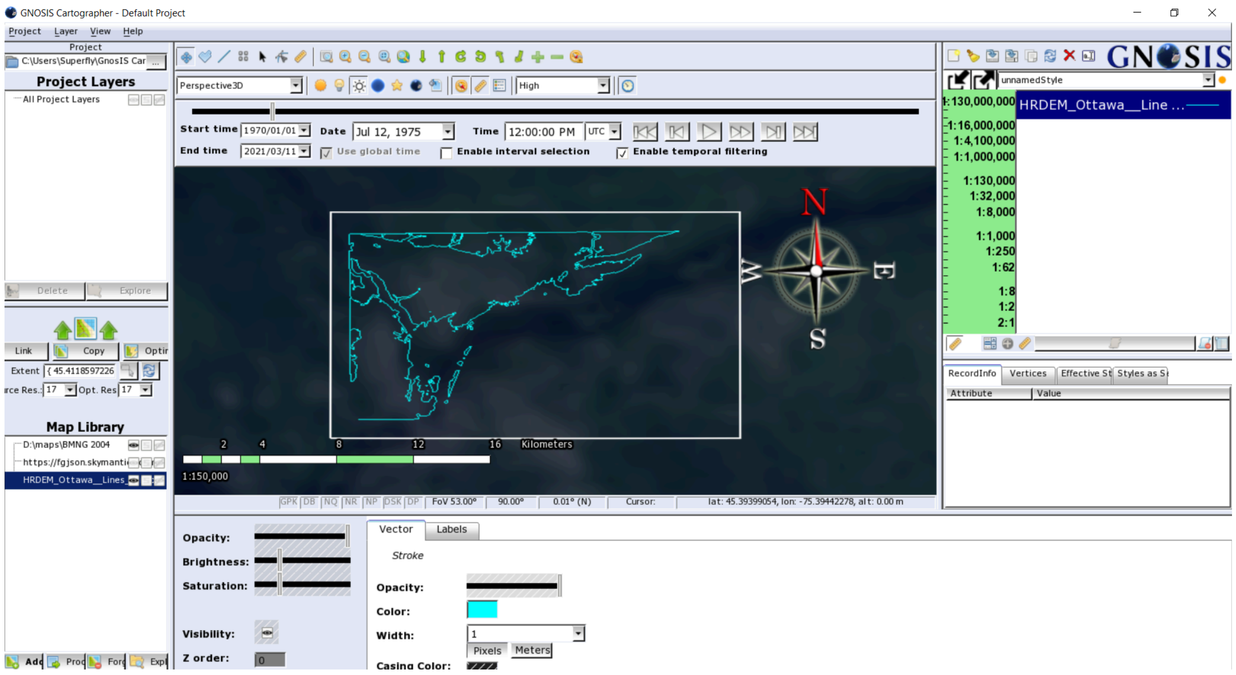
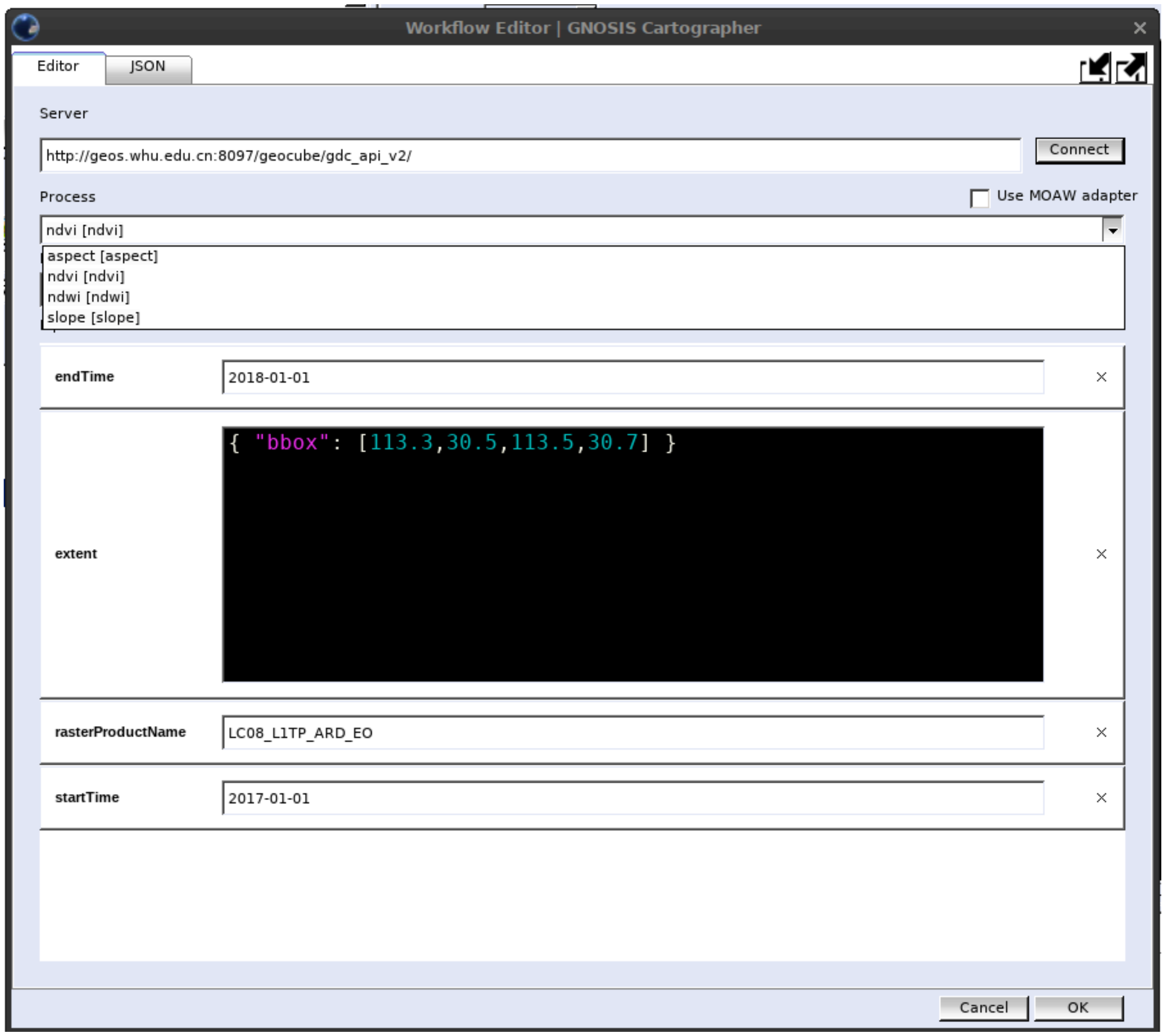


Figure 34 — Visualizing vector lines output of contour generation process

The following screenshot shows the workflow dialog accessing the Wuhan University's Processes implementation.



Server

http://geos.whu.edu.cn:8097/geocube/gdc_api_v2/

Process

Use MOAW adapter

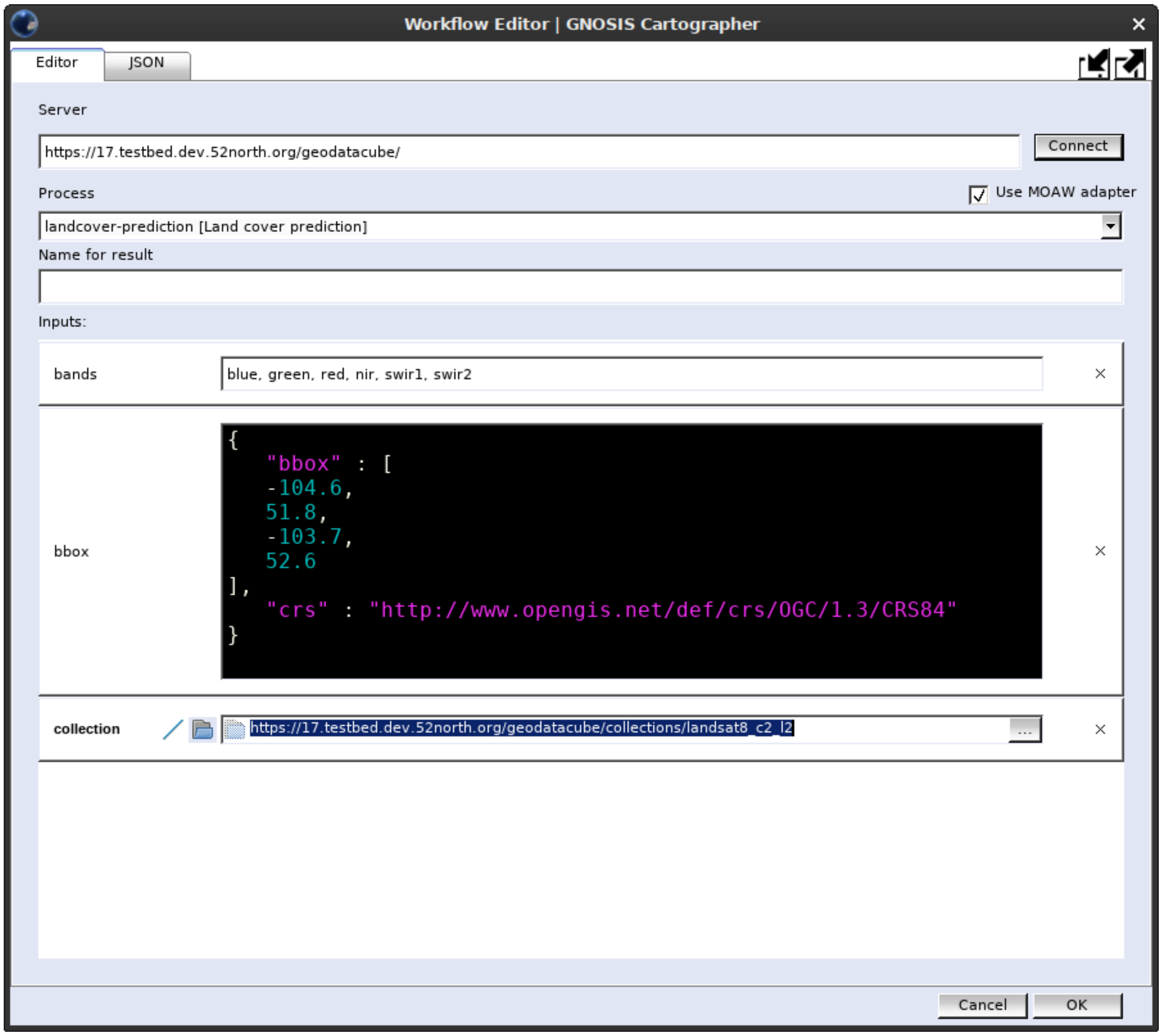
ndvi [ndvi]

aspect [aspect]
ndvi [ndvi]
ndwi [ndwi]
slope [slope]

endTime	2018-01-01
extent	`{ "bbox": [113.3,30.5,113.5,30.7] }`
rasterProductName	LC08_L1TP_ARD_EO
startTime	2017-01-01

Figure 35 — Accessing Wuhan University Processes Implementation in Workflow Editor

The following screenshot shows the workflow dialog accessing the 52°North's Processes implementation.



Server

https://17.testbed.dev.52north.org/geodatacube/

Process

Use MOAW adapter [✓]

landcover-prediction [Land cover prediction]

Name for result

Inputs:

bands	blue, green, red, nir, swir1, swir2
bbox	see below
collection	https://17.testbed.dev.52north.org/geodatacube/collections/landsat8_c2_l2

```
{
   "bbox" : [
   -104.6,
   51.8,
   -103.7,
   52.6
],
   "crs" : "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
}
```

Figure 36 — Accessing 52°North Processes Implementation in Workflow Editor

8.3.3. Tiles API Implementation

The client may request tiles of vector features or coverage values and render them client-side, or request and display map tiles pre-rendered or rendered server-side.

Visualization of map tiles and vector tiles

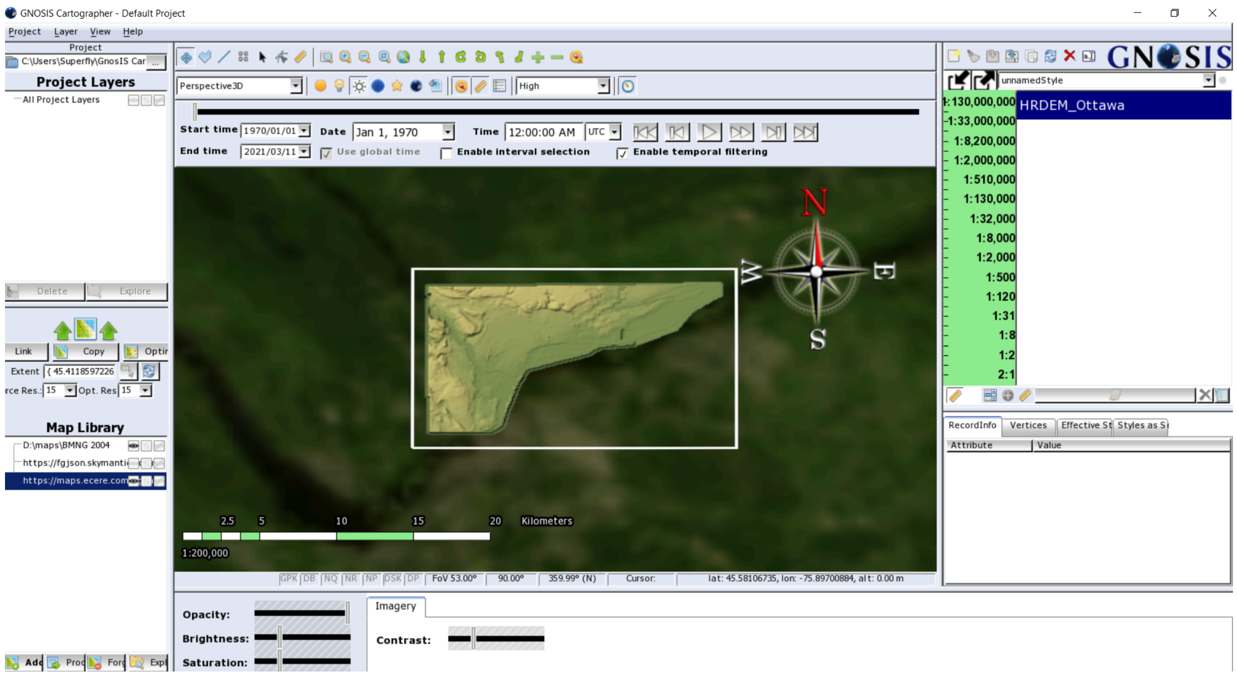


Figure 37 — Visualizing of HRDEM-Ottawa as map tiles

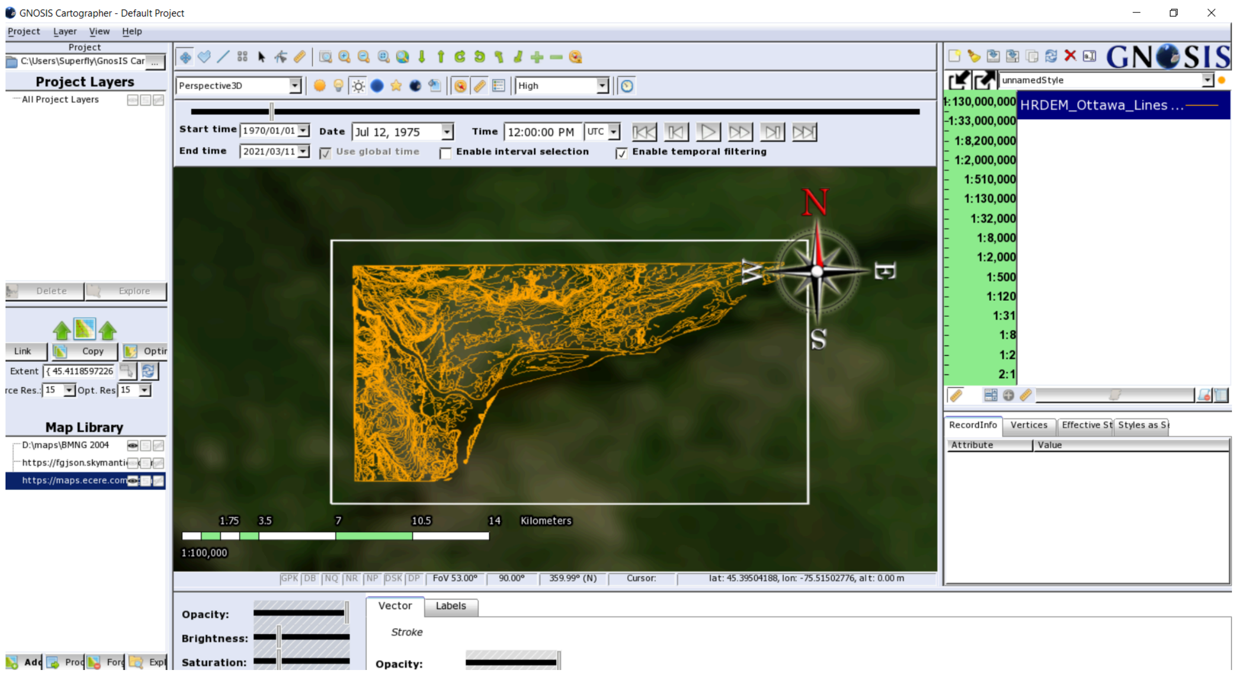


Figure 38 — Visualizing HRDEM-Ottawa-LinesContours as vector tiles

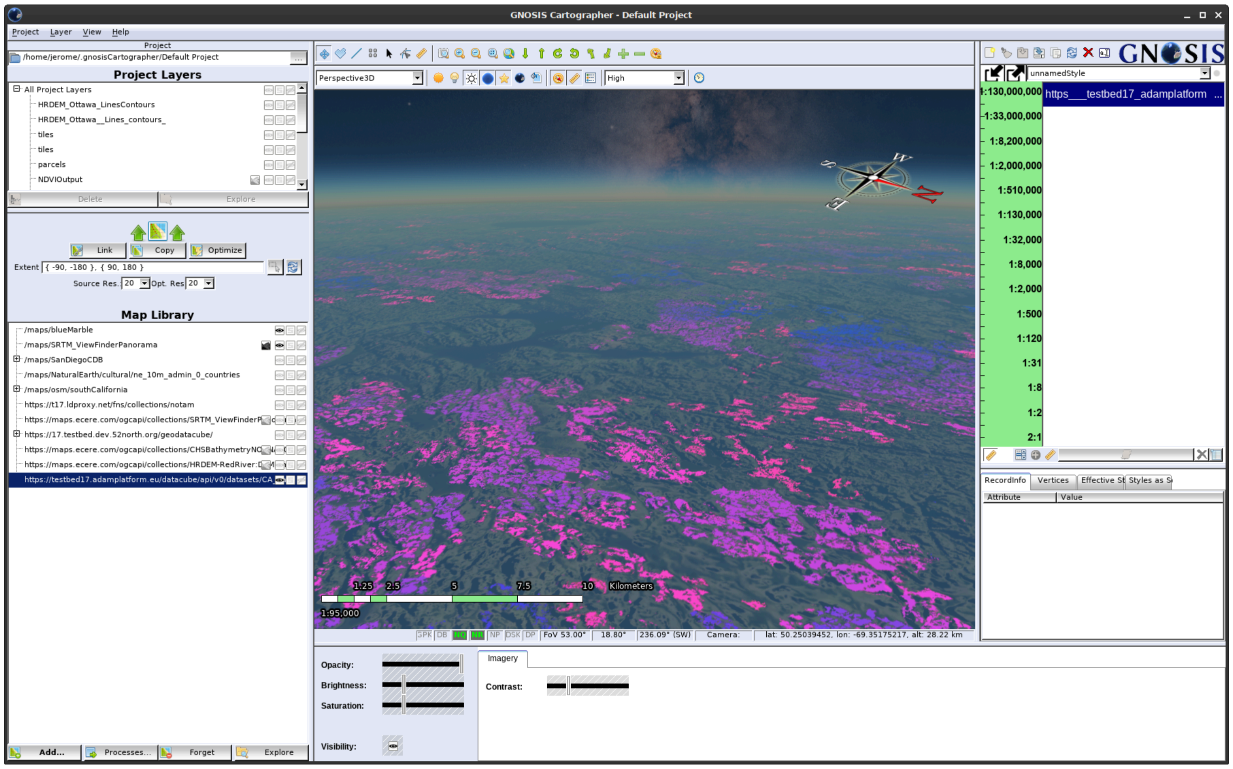

GNOSIS Cartographer - Default Project
Project
Layer
View
Help
/home/jerome/.gnosisCartographer/Default Project
Project Layers
All Project Layers
HRDEM_Ottawa_LinesContours
HRDEM_Ottawa__Lines_contours_
tiles
tiles
parcels
NDVIOutput
Delete
Explore
Link
Copy
Optimize
Extent { -90, -180 }, { 90, 180 }
Source Res. 20
Opt. Res 20
Map Library
/maps/blueMarble
/maps/SRTM_ViewFinderPanorama
/maps/SanDiegoCDB
/maps/NaturalEarth/cultural/ne_10m_admin_0_countries
/maps/osm/southCalifornia
https://t17.ldproxy.net/fns/collections/notam
https://maps.ecere.com/ogcapi/collections/SRTM_ViewFinderP
https://17.testbed.dev.52north.org/geodatacube/
https://maps.ecere.com/ogcapi/collections/CHSBathymetryNO
https://maps.ecere.com/ogcapi/collections/HRDEM-RedRiver
https://testbed17.adamplatform.eu/datacube/api/v0/datasets/CA
Add...
Processes...
Forget
Explore
Perspective3D
High
1:25
2.5
5
7.5
10
Kilometers
1:95,000
FoV 53.00°
18.80°
236.09° (SW)
Camera:
lat: 50.25039452, lon: -69.35175217, alt: 28.22 km
Opacity:
Brightness:
Saturation:
Visibility:
Imagery
Contrast:
GNOSIS
unnamedStyle
https___testbed17_adamplatform
1:130,000,000
1:33,000,000
1:8,200,000
1:2,000,000
1:510,000
1:130,000
1:32,000
1:8,000
1:2,000
1:500
1:120
1:31
1:8
1:2
2:1
RecordInfo
Vertices
Effective St
Styles as S
Attribute
Value

Figure 39 — Visualizing map tiles from MEEO service's *CA_harvest_year* data cube, together with NASA's *Blue Marble Next Generation*, SRTM elevation from Jonathan de Ferranti's *View Finder Panoramas* and ESA's *Gaia Sky in Color*

9

STANDARDIZING A GEO DATA CUBE API

9 STANDARDIZING A GEO DATA CUBE API

This chapter describes the next steps and recommendations towards standardizing a Geo Data Cube API.

9.1. Next steps for a Geo Data Cube API

- The draft *OGC API — Common* specification was recognized as a valuable framework for integrating API capabilities, with *Part 2: Geospatial data* providing multiple access mechanisms for a particular data cube resource (collection of multi-dimensional spatiotemporal data). Completing the standardization of both Part 1 & Part 2 of *Common* should be an important priority.
- The value of defining a set of standardized OGC API building blocks (e.g. specifications and conformance classes) as a Geo Data Cube standard or profile of an OGC API Standard should be considered.
- The need for Executable Test Suites for the different OGC API standards and draft specifications was highlighted. The presence of such test suites would have greatly facilitated the achievement of successful TIEs in this initiative. In particular, the completion of test suites for *OGC API — Processes* and *OGC API — Coverages* should be prioritized.

9.2. Recommendations relating to data discovery

- The completion of the draft *OGC API — Records* specification and integration with STAC is an important standardization task to facilitate data discovery.
- The idea of a *Scenes API* offering both a unified data cube while providing direct access to the data and metadata of individual scenes (which may be in different CRSes) should be investigated. The use of scene metadata properties may also be useful in filtered data access queries, providing integrated discovery. The API could also provide the capability to manage scenes, based on previous work from the Testbed 15 — *Images API*.

9.3. Recommendations relating to data access

9.3.1. Recommendations relating to *OGC API — Coverages* and extensions

- The draft *OGC API — Coverages* specification was demonstrated as a relatively simple access mechanism for data cubes. This API can also be implemented with a minimum amount of effort on top of various data cube technologies. Support for both the *subsetting* and *scaling* conformance classes leveraging a multi-resolution data store was identified as key capabilities to adequately support large data cubes. Because overlapping functionality for describing and accessing data cubes was defined in *OGC API — EDR* and the Testbed 16 — *DAPA* specification with an argument for additional convenience, the perceived inconvenience of the *Coverages* API should be re-evaluated, corrections made if necessary, and the possibility of defining extensions providing the convenience or required capabilities directly in the *Coverages* API should be considered. Completing the standardization of Part 1 of *OGC API — Coverages* should be an important priority.
- A greatly simplified and better modularized future version of the Coverage Implementation Schema (CIS) should be considered as suggested here.
- Defining extensions to the SWE Common DataRecord fields, used in the Coverage Implementation Schema and *OGC API — Coverages* for describing the *RangeType*, for providing information for statistics (e.g. min and max values) and encoding information (e.g. offsets and scale factors).

9.3.2. Recommendations relating to *OGC API — Tiles* and *Cloud Optimized GeoTIFF*

- The value of the *OGC API — Tiles* specification in providing pre-defined multi-resolution tile pyramids of raw data values (e.g. coverage tiles) was identified as important. Completing the standardization of Part 1 of *OGC API — Tiles* should be a priority.
- A conformance class of *Coverages* for integrating HTTP range access to a Cloud Optimized GeoTIFF representation of a coverage resource should be investigated.
- The use of Cloud Optimized GeoTIFF and/or TileMatrixSets pyramids as a backing data store to support the subsetting and scaling conformance classes of *Coverages* should be considered in implementations.

9.3.3. Harmonization of OGC API data cube access

- Implementing support for the *OGC API — EDR* standard should be considered when the particular types of queries it defines are desired as access mechanisms.

- The compatibility issues between *OGC API — EDR* and *OGC API — Common* (as well as other OGC API specifications leveraging *Common*) in terms of the description of the spatiotemporal extent should be corrected so as to allow providing a particular collection using *OGC API — EDR* as well as other OGC API specifications.
- The overlapping functionality between the *OGC API — Coverages* draft specification and the *OGC API — EDR* standard, particularly in how they both provide mechanisms to describe the domain and range of a data cube and requesting a subset of a data cube, should be considered for a potential re-alignment and harmonization.
- The OGC process for ensuring a harmonized set of complementary OGC API standards should be re-evaluated to avoid occurrences of re-defining the same capability with only superficial variation in different standards of the OGC API family, which reduces interoperability by introducing a significant burden on implementers of clients & services in terms of additional standards to implement.
- The draft *OGC API — DGGS* specification should be completed and considered for integration within a Geo Data Cube API.

9.4. Recommendations relating to analytics

9.4.1. Recommendations relating to *OGC API — Processes* and extensions

- The draft *OGC API — Processes — Part 2: Deploy, Replace, Update* specification should be completed to enable deploying analytics capabilities close to the data.
- The concept of a *GeoDataClass* URI to indicate the compatibility of a particular data cube as an input to a process should be investigated.
- The value of *OGC API — Processes — Part 3: Workflows and Chaining* supporting presentation of the results of analytics capabilities as a virtual data cube was demonstrated as facilitating the integration of analytics capabilities in visualization clients, as well as facilitating the integration of remote data cubes with processing algorithms. Completing its standardization should be an important priority.
- Defining well-known processes expecting specific inputs including a particular convenient processing language to facilitate flexible coverage processing should be considered.

9.4.2. Recommendations relating to *OGC API — Coverages*, *OGC API — EDR* and *DAPA*

- Extensions to *OGC API — Coverages* for filtering based on CQL expressions, for specifying simple band arithmetic expressions for fields to return, for different types of aggregation based on some of the dimensions, and for support for varying resolution across a data cube should be considered.

- The functionality defined in the Testbed 16 — *DAPA* draft specification, such as for aggregation and derived field expressions, should be considered for integration directly within the *OGC API — EDR* and/or *OGC API — Coverages* specifications or as extensions, rather than defining yet another completely separate specification.

A

ANNEX A (INFORMATIVE) SELECTED GEO DATA CUBE API CAPABILITIES

A ANNEX A (INFORMATIVE) SELECTED GEO DATA CUBE API CAPABILITIES

This appendix details the selection of building blocks in the form of specifications and conformance classes that were chosen for implementation in clients and servers by participants during the initiative to enable Technology Integration Experiments (TIEs), and provides a short overview of the operations that must be implemented for each capability, which could be considered a GDC API quickstart guide for implementers of a client or service.

The following set of 12 OGC API specifications and conformance classes were selected by participants based on what could reasonably be implemented during the Testbed initiative, and deployed in at least one API. However, no clients implemented capabilities for Records or DAPA, therefore those capabilities were not included in the TIEs.

A.1. Common-1: Core

1. Able to retrieve a landing page at /
2. Able to retrieve conformance declaration by following a rel: `conformance` to `/conformance`
3. Able to retrieve an API description by following a rel: `service-desc` link (e.g. to `/api`).

A.2. Common-2: Collections

1. Able to follow the landing page rel: `data` to `/collections`
2. Able to retrieve a list of collections at `/collections`, which contains a subset of individual collection description

3. Able to follow links with rel: `self` from the elements of the `"collections"` array at `/collections` to retrieve individual collections at `/collections/{collectionId}`. See schema for collection description.

A.3. Coverages-1

1. Able to follow a rel: `http://www.opengis.net/def/rel/ogc/1.0/coverage-domainset` to a coverage's domainset which could either be embedded in the collection description itself (in which case the link will be a JSON pointer or an external resource e.g. at `/collections/{collectionId}/coverage/domainset`. The schema for the JSON representation should conform to the CIS DomainSet.

2. Able to follow a rel: `http://www.opengis.net/def/rel/ogc/1.0/coverage-rangetype` to a coverage's range type which could either be embedded in the collection description itself (in which case the link will be a JSON pointer or an external resource e.g. at `/collections/{collectionId}/coverage/rangetype`. The schema for the JSON representation should conform to the CIS RangeType.

3. Able to retrieve data from the coverage by following a link with rel: `http://www.opengis.net/def/rel/ogc/1.0/coverage` from the collection description to `/collections/{collectionId}/coverage`. Content negotiation is used to select a coverage format (e.g. NetCDF: `application/netcdf`, GeoTIFF: `image/tiff; application=geotiff`, CoverageJSON `application/prs.coverage+json`, CIS JSON: `application/json`). If the negotiated format can encode it, the response should include in addition to the rangeset (the actual data), the domainset, rangetype and if applicable, the coverage metadata.

A.4. Coverages-1 (Subsetting)

1. Able to use the `subset=` query parameter to trim (lower & upper bound, not reducing dimensionality) or slice (single axis value, reducing dimensionality) as part of a coverage data request at `/collections/{collectionId}/coverage`. One or multiple axes (as described in the DomainSet) are supported in the same subset query parameter, e.g. `subset=Lat(10:30),Lon(100:120),time("2020-01-01":"2020-12-31")`

2. Able to use the `bbox=` query parameter to spatially trim along 2 or 3 geospatial dimensions, if applicable to the coverage

3. Able to use the `datetime=` query parameter to temporally slice (instant value) or trim (interval value) along the primary temporal dimension, if applicable to the coverage

NOTE: Coverage Tiles requests conformance class (leveraging *OGC API — Tiles*) can be mapped to a scaling + subsetting request. e.g.`/coverage/tiles/WorldCRS84Quad/1/0/0` is equivalent to either `/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=257,257` (for a ValueIsPoint coverage) or `/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=256,256` (for a ValueIsArea coverage)_

A.5. Coverages-1 (Range subsetting)

1. Able to use the `range-subset=` query parameter to select fewer values (e.g. imagery bands) from all those listed in the range type and returned by default.

A.6. Coverages-1 (Scaling)

1. Able to use the `scale-factor=` query parameter to up-sample or down-sample from the maximum resolution of the coverage. `scale-factor=2` means downsamples 2x from the maximum resolution.
2. Able to use the `scale-axes=` query parameter to specify different scaling factor for each axis, e.g. `scale-axes=Lat(4),Lon(2)`
3. Able to use the `scale-size=` query parameter to specify the desired number of cells returned for each axis, e.g. `scale-size=Lat(512),Lon(1024)`

NOTE:Coverage Tiles requests (leveraging *OGC API — Tiles*) can be mapped to a scaling + subsetting request. e.g.`/coverage/tiles/WorldCRS84Quad/1/0/0` is equivalent to either `/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=257,257` (for a ValueIsPoint coverage) or `/coverage?subset=Lat(0:90),Lon(-180:-90)&scale-size=256,256` (for a ValueIsArea coverage)_

Note that without e.g. `scale-factor=1`, the default response may not necessarily be the maximum resolution (e.g. to avoid returning too much data for default responses).

A.7. Processes-1 (Sync execution)

1. Able to follow a rel: `http://www.opengis.net/def/rel/ogc/1.0/processes` link from the landing page to a list of processes at `/processes`

2. Able to retrieve the process description for individual processes by following the rel: `self` links from `/processes` `"processes"` array elements to `/processes/{processId}`, following the process description schema.

3. Able to `POST` a process execution request (following the process execution schema) to the execution endpoint (linked from process description using rel: `http://www.opengis.net/def/rel/ogc/1.0/execute`) at `/processes/{processId}/execution`. No response preference should be specified to execute synchronously (**omitting** `Prefer: respond-async`).

4. Execution response should be successful

5. Able to the retrieve results successfully. The response currently depends on a number of things in the execution request (`response` mode, `transmissionMode`, number of `outputs`). Table 7 in the specification illustrates the different possibilities. There is a proposal to greatly simplify this in an upcoming minor revision of *OGC API — Processes — Part 1*, which would also enable HTTP content negotiation, and directly accessing individual outputs.

6. Able to access the individual outputs successfully, if they are available separately (response: `document`), or from within the execution response.

A.8. Processes-1 (Async execution)

1. Able to follow a rel: `http://www.opengis.net/def/rel/ogc/1.0/processes` link from the landing page to a list of processes at `/processes`

2. Able to retrieve the process description for individual processes by following the rel: `self` links from `/processes` `"processes"` array elements to `/processes/{processId}`, following the process description schema.

3. Able to `POST` a process execution request (following the process execution schema) to the execution endpoint (linked from process description using rel: `http://www.opengis.net/def/rel/ogc/1.0/execute`) at `/processes/{processId}/execution`. If the server supports both Sync and ASync mode for the process, a `Prefer: respond-async` header should be included for async execution.

4. Able to parse an execution response conforming to the statusInfo.yaml schema, which at minimum requires a `"type": "process"`, a `jobID` and a `status` (`accepted`, `running`, `successful`, `failed` or `dismissed`).

5. Execution response should be successful (returning a **201** HTTP status as per Requirement 34 A) and include a *Location* header containing a link to the new job (i.e. at `/jobs/{jobId}`)

6. Able to list running jobs at `/jobs`

7. Able to retrieve the status of an executed job at `/jobs/{jobId}`

8. Able to `DELETE` a running job at `/jobs/{jobID}` to cancel execution

9. Able to retrieve the results of a job at `/jobs/{jobId}/results`. The response of that endpoint currently depends on a number of things in the execution request (`response` mode, `transmissionMode`, number of `outputs`). Table 8 in the specification illustrates the different possibilities. There is a proposal to greatly simplify this in an upcoming minor revision of *OGC API — Processes — Part 1*, which would also enable HTTP content negotiation, and directly accessing individual outputs.

10. Able to access the individual outputs successfully, if they are available separately (response: `document`), or from within the results response.

A.9. Processes-3: Workflows (Collection Input)

1. Able to execute a process either synchronously or asynchronously, which accepts as one or more of its execution requests inputs an OGC API collection, e.g. `{ "collection" : "http://example.com/ogcapi/collections/example" }`. The server may either accept only local collections, or remote collections as well. The *Area / Resolution of Interest* as well as the format are left out of the workflow/ process execution (facilitating re-use), as they can be inferred from the OGC API data access (e.g. Coverages) requests to both the output collection and the input collection, thus allowing to chain a Workflows Collection Output (or a nested process) as a Workflows Collection Input.

A.10. Processes-3: Workflows (Collection Output)

1. Able to `POST` a process execution request (following the process execution schema) to the execution endpoint (linked from process description using rel: `http://www.opengis.net/def/rel/ogc/1.0/execute`) at `/processes/{processId}/execution`. (NOTE: Currently in Ecere's implementation the *Collection Output* execution mode is using a different endpoint directly at `/processes/{processId}`. This may change to become a query parameter (e.g. `response=collection`, `response=landingPage`) instead to keep to the usual `/execution` endpoint.)

2. Able to parse the response as an OGC API collection description, and follow links to access the resulting data using Coverages requests (e.g. rel: `http://www.opengis.net/def/rel/ogc/1.0/coverage`). Note that the actual execution of

the process may happen on-demand as requests for different *Area* or *Resolution of Interest* are made by the client.

A.11. Records-1: Core

OGC API — Records was not tested in TIEs but was implemented by 52°North and Wuhan University.

A.12. DAPA

DAPA was not tested in TIEs but was partially implemented by Wuhan University.

B

ANNEX B (INFORMATIVE) TECHNOLOGY INTEGRATION EXPERIMENTS

B

ANNEX B (INFORMATIVE) TECHNOLOGY INTEGRATION EXPERIMENTS

B.1. TIEs Table

Table B.1 — Successful Geo Data Cube API Technology Integration Experiments

CLIENTS → ▼ SERVERS	SOLENIX	ETHAR	52°NORTH (CASCADING SERVER)	ECERE
52°North	1,2,3,4		1,2,3,4,5,9	1,2,3,4,5,9
Wuhan University	1,2,3,4		1,2,3,4,5,9	1,2,3,4,5
MEEO				1,2,Map tiles
Ecere	1,2,3,4,10			1,2,3,4,6,7,9,10

Table B.2 — GDC API Capabilities

1. Common-1: Core
2. Common-2: Collections
3. Coverages-1
4. Coverages-1 (Subsetting)
5. Coverages-1 (Range subsetting)
6. Coverages-1 (Scaling)
7. Processes-1 (Sync execution)

8. Processes-1 (Async execution)
9. Processes-3: Workflows (Collection Input)
10. Processes-3: Workflows (Collection Output)

B.2. Summary

- Successful TIEs were performed with all server implementations for *OGC API — Common* **Core** (1) and **Collections** (2) capabilities.
- Successful TIEs were performed with most servers for *OGC API — Coverages* (3) functionality, including support for **subsetting** (4) and **range subsetting** (fields selection) (5). Support for multi-bands coverages and range subsetting was initiated in the Ecere service but not completed in time for the TIEs.
- The *Coverages* **scaling** capability (6) was found to be something lacking from a client perspective in order to explore datasets covering a large spatial extent at different resolutions, but requires a backend with support for tile pyramids / overviews to implement efficiently and will require more work to implement in some servers.
- Both 52°North and Wuhan University deployed an *OGC API — Processes* implementation supporting **asynchronous** processing (8), with 52°North additionally supporting **synchronous** processing (7). However there were still some issues left to resolve preventing successful TIEs.
- *Workflows* were demonstrated to provide a mechanism by which an external data cube could be used with a cascading server using the **collection input** capability (9), and by which a visualization client could easily trigger processing through the **collection output** capability (10).
- Ethar implemented support for *OGC API — Coverages* in its Augmented Reality client, but did complete TIEs with the three different services.
- The Ecere client was able to list available collections as well as separately visualize map tiles from a URL template from the MEEO service.

C

ANNEX C (INFORMATIVE) REVISION HISTORY

C

ANNEX C (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
May 31, 2021	.1	J. St-Louis	all	initial version
Dec 20, 2021	.2	J. St-Louis	all	first complete draft
Feb 9, 2022	.3	J. St-Louis	all	first version posted to pending documents
Mar 7, 2022	.4	J. St-Louis	all	contributions from P. Vretanos, applied changes from review by OGC staff, restore bibliography (final editor draft)

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Baumann, P., The datacube manifesto, (2017). (https://earthserver.eu/tech/datacube-manifesto/The-Datacube-Manifesto.pdf)

2. Nativi S., Mazzetti P., Craglia M., A view-based model of data-cube to support big earth data systems interoperability, Big Earth Data, no. 1, vol. 1-2, pp. 75-99, DOI 10.1080/20964471.2017.1404232, (2017). (https://doi.org/10.1080/20964471.2017.1404232)

3. Giuliani, G., Masó, J., Mazzetti, P., Nativi, S., Zabala, A., Paving the Way to Increased Interoperability of Earth Observations Data Cubes, DOI 10.3390/data4030113, ISSN 2306-5729, (2019). (https://www.mdpi.com/2306-5729/4/3/113)

4. George Percivall : OGC 18-095r7, *Geospatial Coverages Data Cube Community Practice*. Open Geospatial Consortium (2020). https://portal.ogc.org/files/18-095r7

5. Mahecha, M.D. et al., Earth system data cubes unravel global multivariate dynamics, Earth System Dynamics, vol. 11, pp. 201-234, DOI 10.5194/esd-11-201-2020, (2020). (https://esd.copernicus.org/articles/11/201/2020/)

6. openEO API specification (https://api.openeo.org/)

7. sentinel hub APIs (https://www.sentinel-hub.com/develop/api/)

8. up42 documentation (https://docs.up42.com/index.html)

9. Climate Data Store API (https://cds.climate.copernicus.eu/api-how-to)

10. Roocs Tools — Remote Operations on Climate Simulations (https://github.com/roocs)

11. Earth System Data Cube (https://cablab.readthedocs.io/en/latest/)

12. Data cube on Wikipedia (https://en.wikipedia.org/wiki/Data_cube)

13. Online analitycal processing cube on Wikipedia (https://en.wikipedia.org/wiki/OLAP_cube)

14. Jacovella-St-Louis, J., Flexible real-time data processing and visualization workflows emerging from OGC API modules, Open Geospatial Consortium, (2021). (http://docs.opengeospatial.org/dp/21-033.html)

15. Heazel, C., OGC API — Common — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/19-072.html)

16. Heazel, C., OGC API — Common — Part 2: Geospatial Data (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/20-024.html)

17. Vretanos, P. A., Portele, C., OGC API — Features — Part 3: Filtering (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/19-079r1.html)

18. Heazel, C., OGC API — Coverages — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/19-087.html)

19. Masó, J., Jacovella-St-Louis, J., API — Tiles — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/20-057.html)

20. Masó, J., OGC API — Maps — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/20-058.html)

21. Masó, J., Jacovella-St-Louis, J., OGC Two Dimensional Tile Matrix Set and Tile Set Metadata (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/17-083r4.html)

22. Vretanos, P. A., OGC API — Processes — Part 2: Transactions for deployment (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/20-044.html)

23. Jacovella-St-Louis, J., Vretanos, P. A., OGC API — Processes — Part 3: Workflows and chaining (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/21-009.html)

24. Portele, C., OGC API — Routes — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/21-000.html)

25. Portele, C., OGC API — Styles — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://docs.opengeospatial.org/DRAFTS/20-009.html)

26. Purss, M., OGC API — DGGS — Part 1: Core (draft specification), Open Geospatial Consortium, (2021). (https://github.com/opengeospatial/ogcapi-discrete-global-grid-systems)

27. Robert Thomas, Josh Lieberman: OGC 21-013, *Modernizing SDI: Enabling Data Interoperability for Regional Assessments and Cumulative Effects CDS*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/21-013.html

28. Gobe Hobona, Angelos Tzotsos, Tom Kralidis, Martin Desruisseaux: OGC 21-008, *Joint OGC OSGeo ASF Code Sprint 2021 Summary Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/21-008.html

29. Panagiotis (Peter) A. Vretanos: OGC 20-016, *OGC Testbed-16: Data Access and Processing Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-016.html

30. Luis Bermudez: OGC 20-025r1, *OGC Testbed-16: Data Access and Processing API Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-025r1.html

31. Christophe Noël: OGC 20-035, *OGC Testbed-16: Earth Observation Application Packages with Jupyter Notebooks*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-035.html

32. Guy Schumann: OGC 20-018, *OGC Testbed-16: Machine Learning Training Data ER*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-018.html

33. Robert Gibb, Byron Cochrane, Matthew Purss: OGC 20-039r2, *OGC Testbed-16: DGGS and DGGS API Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-039r2.html

34. Joan Maso: OGC 20-041, *OGC Testbed-16: Analysis Ready Data Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-041.html

35. Gobe Hobona: OGC 20-091, *OGC API – Common and OGC API – Features Sprint 2020: Summary Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-091.html

36. Ingo Simonis: OGC 20-073, *OGC Earth Observation Applications Pilot: Summary Engineering Report*. Open Geospatial Consortium (2020). https://docs.ogc.org/per/20-073.html

37. Joan Maso Pau: OGC 19-070, *OGC Testbed-15:Images and ChangesSet API Engineering Report*. Open Geospatial Consortium (2020). http://docs.opengeospatial.org/per/19-070.html

38. Sam Meek: OGC 19-027r2, *OGC Testbed-15: Machine Learning Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/19-027r2.html

39. Pedro Gonçalves: OGC 19-026, *OGC Testbed-15: Federated Clouds Analytics Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/19-026.html

40. Tom Landry: OGC 18-038r2, *OGC Testbed-14: Machine Learning Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/18-038r2.html

41. Paulo Sacramento: OGC 18-049r1, *OGC Testbed-14: Application Package Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/18-049r1.html

42. Paulo Sacramento: OGC 18-050r1, *OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report*. Open Geospatial Consortium (2019). https://docs.opengeospatial.org/per/18-050r1.html

43. Ingo Simonis: OGC 18-046, *OGC Earth Observation Exploitation Platform Hackathon 2018 Engineering Report*. Open Geospatial Consortium (2018). http://docs.opengeospatial.org/per/18-046.html