ESA EOPF 101

Table of contents

1 ESA EOPF 101

Your community guide for working with EOPF Sentinel Zarr data in the cloud

Explore EOPF 101, an open community resource designed to help Sentinel data users explore EOPF Sentinel Zarr data in the cloud. With our step-by-step and hands-on tutorials, you'll learn how to effectively use EOPF Sentinel Zarr products and build Earth Observation workflows that scale.

Ready to explore EOPF 101?

EOPF 101 is designed for Sentinel data users who are new to cloud-optimised geospatial formats and cloud-based workflows. It introduces you to fundamental cloud-native geospatial concepts, the Earth Observation Processing Framework (EOPF) activities by ESA, re-processed EOPF Sentinel Zarr data, as well as tools and libraries to work with EOPF Sentinel Zarr data in the cloud.

Across five chapters, EOPF 101 gradually introduces you to the EOPF Sentinel Zarr products, how you can search and access these, relevant tools and plugins to use EOPF Sentinel Zarr data in different working environments, as well as practical end-to-end application workflows highlighting the benefits of EOPF Sentinel Zarr data.

- **Chapter 1 - About EOPF**
  - Introduction to the EOPF
  - About Cloud-Optimised Formats
  - EOPF Sentinel Zarr products
- **Chapter 2 - About EOPF Zarr**
  - Overview of the EOPF Zarr format
  - Discover EOPF Zarr - Sentinel-2 L2A
- **Chapter 3 - EOPF and STAC**
  - Introduction to STAC
  - Explore the web interface of the EOPF Zarr STAC Catalog
  - Access the EOPF Zarr STAC API with Python
  - From STAC to Data: Accessing EOPF Zarr with xarray
- ***[COMING SOON] Chapter 4 - Tools to work with EOPF Zarr***

  - *Get hands-on with different libraries and tools facilitating the use of EOPF Sentinel Zarr data*

- ***[COMING SOON] Chapter 5 - EOPF Zarr in Action***
  - *See end-to-end workflows leveraging EOPF Sentinel Zarr data across different application domains*

How best use EOPF 101

You can use EOPF 101 as a reference online resource to get example code and workflows for working with Zarr data, the EOPF STAC Catalogue, and different libraries and plugins facilitating the use of EOPF Sentinel Zarr data. Beyond this browsable version, you can also set up the required environment to execute the notebooks.

Instructions on local setup and Docker container on CDSE are coming soon.

How to get involved

EOPF 101 is an open community resource under active development. Our activities are designed to engage with Sentinel users and to gather feedback on EOPF Sentinel Zarr products. There are different ways you can get involved and engaged:

Join the EOPF Toolkit Notebook Competition

Get ready and participate in the EOPF Toolkik Notebook Competition! The competition will kick off in October 2025 and run till January 2026. It is your chance to get hands-on with EOPF Zarr products, get expert input and guidance and show the community the great work you do.

Express your interest today and do not miss any updates related to the notebook competition.

More details coming soon.

Ideas & Feedback?

Is there a plugin or library missing that you would like to see integrated? Do you have feedback on EOPF 101? Please submit an issue and we will review your request.

About the ESA EOPF Toolkit project

EOPF 101 is a community resource developed as part of the EOPF Toolkit project funded by the European Space Agency. EOPF 101 is brought to you by Development Seed, thriveGEO and SparkGeo.

Part I

About EOPF

2 Introduction to the EOPF

2.0.1 Introduction

In this chapter, we will introduce the European Space Agency's (ESA) Earth Observation Processor Framework (EOPF) initiative. This project marks a significant step towards modernising how we handle satellite data, specifically by moving away from the traditional `.SAFE` data format to a more efficient, cloud-optimised format. We will take a closer look at the new structure ESA has developed for this purpose.

2.0.2 What we will learn

- What exactly is ESA's EOPF initiative, and why is it important?
- Which metadata is crucial for the EOPF data?
- Which new encoding format has ESA proposed?

2.1 What is EOPF?

The Earth Observation Processor Framework (EOPF) is an initiative led by the European Space Agency (ESA) designed to modernise and harmonise data from the Copernicus Sentinel Missions.

With the upcoming Copernicus Expansion missions in 2028, the amount of data produced daily will significantly increase. EOPF is ESA's solution to organise Sentinel data in a way that works seamlessly with modern cloud technology. This will make it easier to find, access, and process the information you need. The new approach provides user-friendly access, simplifies maintenance, and helps keep costs down, guaranteeing reliable access to Sentinel data in the long run.

The Sentinel-1, Sentinel-2, and Sentinel-3 missions are the first to be updated with this new system.

2.2 The EOPF Data Model

The EOPF data model has been defined by following a set of principles:

- **Open standards:** Following common and community-approved data standards ensures sustainability and user uptake.
- **Interoperability:** Harmonised with a clear and organised structure that describes the data itself.
- **Cloud optimisation:** Designed for efficient access and handling in cloud environments.
- **Conversion flexibility:** Providing tools to adjust the data for different applications.

Under EOPF, there are four key areas of activities: (i) EOPF product structure, (ii) EOPF metadata structure, (iii) EOPF encoding structure and (iv) the EOPF Processor Framework:

2.2.1 EOPF product structure

As part of the EOPF, ESA is actively working on a common data structure for Sentinel data products to define a common meta-model that can be used across all Sentinel and other EO missions. This approach ensures that data from several missions is consistent.

The EOPF product structure consists of the following components:

- **Measurements:** The actual sensor readings (like how much light is reflected or the temperature), at different levels of detail.
- **Quality indicators:** Details that help understand how reliable the measurements are.
- **Conditions:** Information about the environment or technical aspects when the data was collected.
- **Attributes:** Global metadata, such as when it was acquired and the sensor's orbit.

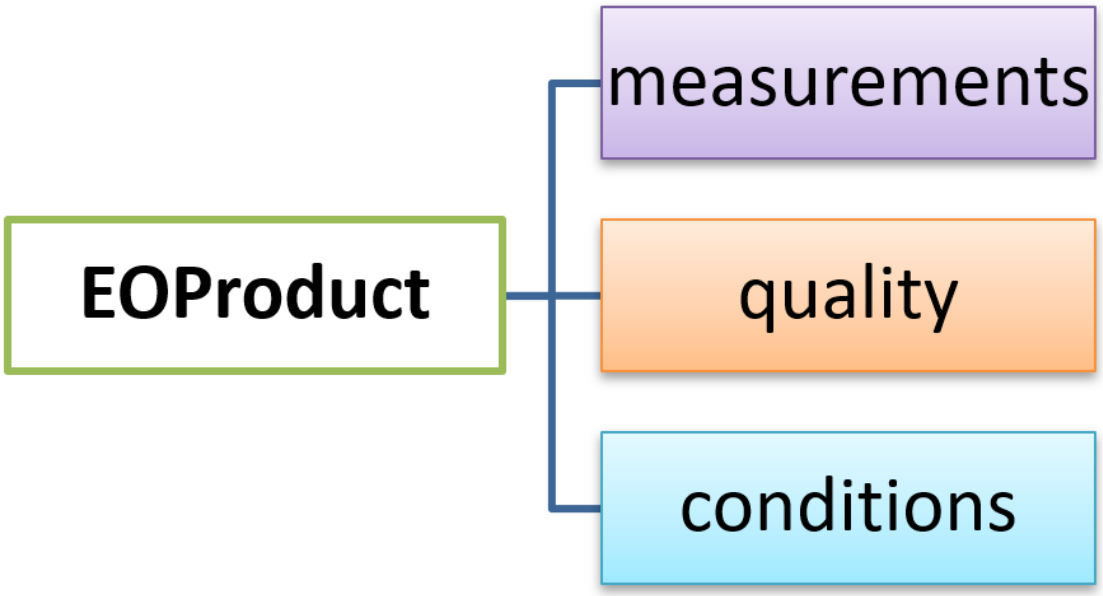


Figure 2.1: EOPF product structure

> **Note**
>
> Learn more about the EOPF Zarr product structure here.

2.2.2 EOPF metadata structure

Metadata provides all relevant information required to uniquely describe each Sentinel product. The EOPF metadata structure is organised as follows:

- **Discovery Metadata**: Following the metadata structure defined by the SpatioTemporal Asset Catalogue (STAC), which helps to keep things consistent across different missions.
- **Processing History Metadata**: Keeping a record of how the data has been processed.
- **Other Metadata**: Information like the status of the sensor and details about the satellite's orbit.

> **Note**
>
> EOPF and STAC: Learn more about EOPF and STAC here.

2.2.3 EOPF encoding structure

An encoding structure can be seen as the specific method used to package and store data and its associated metadata in a digital format. Building on the consistent data structure and clear metadata, the new storage system must be capable of handling various aspects of current Sentinel data (such as manifest files and tile structures from the SAFE format) while remaining fully compatible with cloud environments.

ESA chose `.zarr` as encoding format as it allows for instant access to data, efficient processing of massive amounts of data, and seamless integration with other datasets. The EOPF Sentinel Zarr data encoding allows you to work with data from multiple missions more effectively.

> **Note**
>
> Learn more about the EOPF Sentinel Zarr format here. And learn more about cloud-optimised geospatial data formats in general in the Cloud-Optimised Geospatial Data Formats Guide

2.2.4 EOPF processor framework

The way Sentinel data is processed is being updated to take advantage of modern cloud computing. This will make the processing faster and more efficient, and at the same time ensure the scientific quality and accuracy of the Sentinel data remains the same.

> **Note**
>
> To learn more about the EOPF processor framework, visit https://eopf.copernicus.eu/eopf/

2.3 Conclusion

Throughout this section, we explored the EOPF initiative and its adoption of the `.zarr`format. This new approach is set to significantly improve and simplify how we access and work with data from the satellite missions Sentinel-1, Sentinel-2, and Sentinel-3.

2.4 What's next?

In the following [section] (./12_about_cloudoptimized_formats.qmd), we learn why EO data needs to be cloud-optimised when processed in the cloud.

3 About cloud-optimised formats

3.0.1 Introduction

In this section, we will dive into **cloud-optimised geospatial formats**. We explore why these new formats are important and will introduce you to two common cloud-optimised data formats specifically for raster files.

3.0.2 What we will learn

- What do we need to cloud-optimise geospatial data?
- Differences between traditional and cloud-native workflows
- What are the main characteristics of cloud-optimised formats?
- What are **COGs** and **Zarr**, and how do they differ?

3.1 Why to cloud-optimise geospatial data formats?

The volume of EO data has grown exponentially in recent years. The Copernicus programme alone generates ~16TB daily from the Sentinel missions. Traditional file formats, like `.SAFE` (where each file can be hundreds of megabytes), are optimised for efficient archiving and distributing data. This means that we often download the data from an entire overpass, even if we only need to access a small part of it. For example, if we want to do an analysis of the area of a single city over a decade.

With growing data volumes, this becomes a challenge. To picture the different nature of challenges we come across, let us compare a traditional local workflow with a cloud-based workflow:

- **Traditional local workflow**: When working locally, we download much more data than we need, and we are constrained by the compute and storage capacity of the local system. However, an advantage of working locally is that data and compute are close together, meaning that there is not much delay in accessing the data.

- **Cloud-based workflow**: Cloud environments overcome the limitations local workflows have. A cloud environment offers limitless storage and compute capacity. On the contrary, data storage, compute, and you the destination are far apart. There is an additional time for data to travel between the storage location, processing resources and us. This time is referred to as **data latency**.

> **Note**
>
> **Data latency** refers to the time it takes for data to be transmitted or processed from cloud storage to your computer. In local workflows, data latency is minimal, whereas in cloud-based workflows, data latency needs to be optimised.

3.1.1 Analogy: Comparing local and cloud-based workflows with ordering a pizza

To understand the principal concept, let us compare local and cloud-based workflows with ordering a pizza. Local workflows are similar to placing an order at a pizza store on your street. It is quick since the 'data' (pizza) is easily accessible, but we can only choose from what the local pizza store offers.

On the other hand, cloud-based workflows are comparable to ordering a pizza from a pizza store in a different city or even country. This option allows you to order different types of pizzas, which are not available in the pizza store on your street. While we might have more options to choose from, the time between order and delivery can become a challenge. The time until your pizza from a different town or country arrives at your house is called **data latency**.

Hence, the overall goal with cloud-based workflows is to minimise **data latency** as much as possible. This is why traditional data formats need to be cloud-optimised.

3.2 Characteristics of cloud-optimised formats

Cloud-optimised formats are optimised to minimise data latency. By allowing for an efficient retrieval of smaller, specific chunks of information rather than downloading an entire file. Accessing a smaller data subset also reduces the costs associated with data transfer and data processing.

Cloud-optimised geospatial data formats have the following characteristics:

- Data is **accessible over an HTTP protocol**.
- **Read-Oriented**, as it supports partial and parallel reads.
- Data is **organised in internal groupings (such as chunks, tiles, shards)** for efficient subsetting, distributed processing and data access in memory.

- **Metadata** can be accessed in one read.

> **Note**
>
> When accessing data over the internet (e.g. through object stores in the cloud), latency is high compared to local storage, so it is recommended to fetch lots of data in fewer reads.

3.3 Cloud-Optimised Geospatial Raster Formats

For satellite data, two main cloud-optimised formats are being used: - **Cloud-Optimised GeoTIFF (COG)**: Optimised for 2D image data and originates from the traditional `GeoTIFF` format, and - **Zarr**: Used and designed for complex, n-dimensional data structures and originates from the traditional formats `netCDF` and `HDF5`.

3.3.1 Cloud-optimised GeoTIFF (COG)

COGs have been widely used as a cloud-native format for satellite imagery and improve the standard GeoTIFF format by: - Organising data into **tiles**: Dividing the data into smaller, manageable squares (like 512x512 pixels). - Including lower-resolution previews: Having pre-generated, less detailed versions of the data. This allows for fast and efficient data visualisations.

A key feature of COGs is the **Internal File Directory** (IFD), which acts like an internal index. This allows for retrieving only the parts of the data needed using simple web requests. For example, it is possible to access just the tiles covering Paris from a large Sentinel-2 image of Europe.

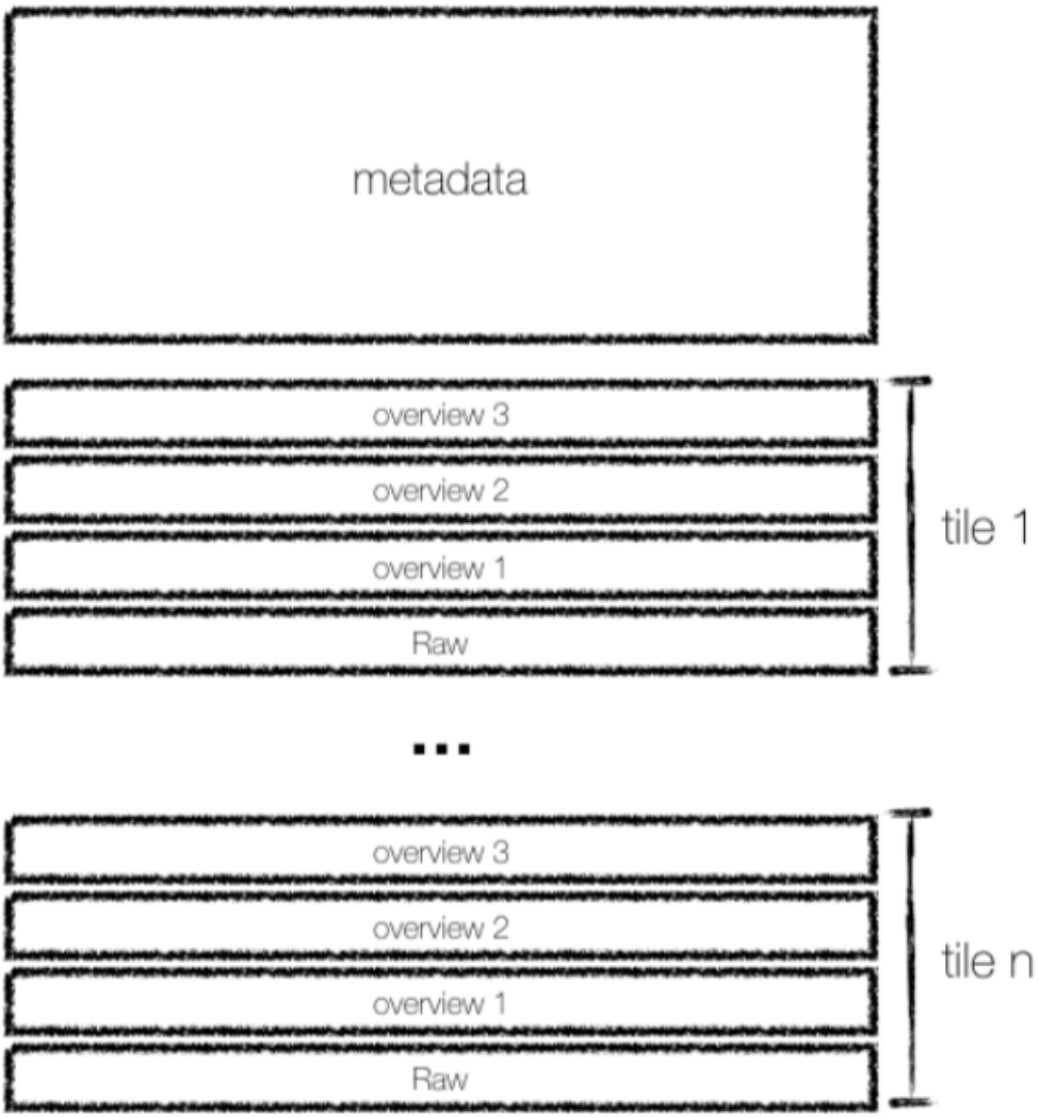


Figure 3.1: COG structure. Retrieved from CNG documentation

3.3.2 Multi-dimensional Array Storage with Zarr

Zarr is the cloud-optimised version for the traditional formats `netCDF` and `HDF5`, and is specifically designed for storing and accessing large n-dimensional arrays in the cloud by:

- **Chunking**: Breaking large arrays into smaller pieces that can be accessed independently

- **Compression**: Each chunk can be compressed individually for efficient storage
- **Hierarchical Organisation**: Arrays are organised in groups, similar to folders in a filesystem
- **Cloud-Native Access**: Optimised for reading partial data over HTTP
- **Parallel I/O**: Multiple chunks can be read or written simultaneously
- **Self-Description**: Rich metadata is stored alongside the data using JSON

This makes Zarr particularly well-suited as a storage format for processing Earth observation data in the cloud.

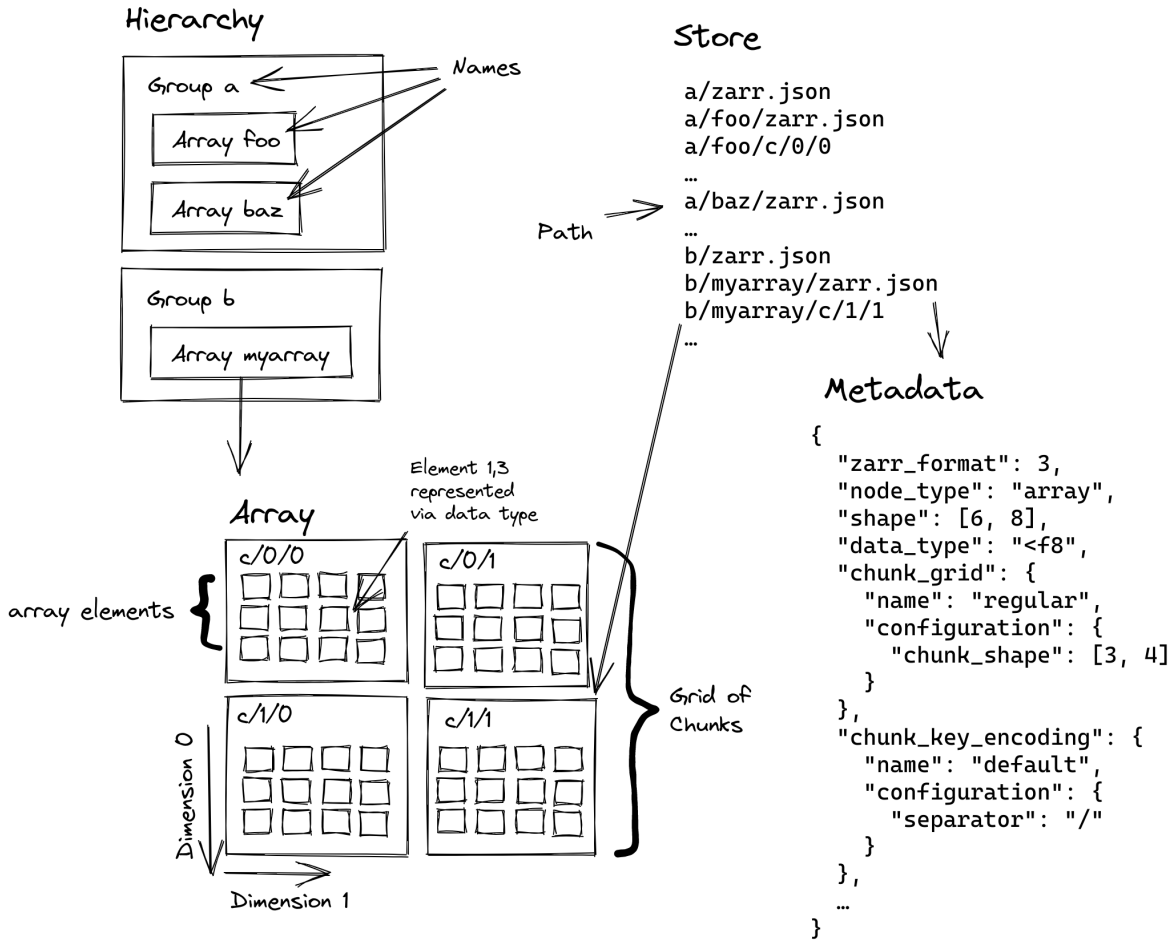


Figure 3.2: Zarr's hierarchical organization showing stores, groups, arrays, and chunks

3.3.3 When to use COG versus Zarr?

The table below compares some features of COG and Zarr:

Feature	Zarr	COG
Structure	Multi-file chunks	Single file
Access	Parallel	Sequential
Compression	Differently per-chunk	Whole-file
Scales	Multi-scale in single file	Separate, pre-generated lower-resolution files

Based on the structure and capabilities for each format, **COGs** are used when:

- You work with two-dimensional raster data (like satellite images or elevation models)
- You need to easily visualise or access specific geographic areas without loading the entire dataset.
- Interoperability with existing GIS software is important, as COG is a widely adopted standard.

On the other hand, **Zarr** is more often used when:

- You deal with large, multi-dimensional datasets that might be updated or modified.
- You are performing complex analyses that involve accessing different parts of the data in parallel.
- An efficient handling of different resolutions or variables within a single dataset is required.

> **Note**
>
> Zarr vs COG: Want to learn more about the differences and similarities of COG and Zarr? Then we recommend the following blog post by Julia Signell and Jarrett Keifer from Element84, where they discuss "Is Zarr the new COG?"

3.4 Conclusion

In this section, we explored the fundamental concepts of **cloud-optimised geospatial formats**. By understanding the core characteristics of these formats and by looking at specific examples like **Zarr**, you now have a solid foundation for appreciating how these innovations are making geospatial data more accessible, efficient, and powerful in the cloud.

3.5 What's next?

Now that we have an idea of the available cloud-optimised formats for satellite imagery and the reason why we need to optimise traditional formats for the cloud, in the next section, we will explore the EOPF data products that are being re-processed as part of the EOPF Zarr Sample Service.

4 Overview of EOPF Zarr Products

4.0.1 Introduction

In the previous section, we introduced the **Earth Observation Processing Framework** (EOPF) initiative and explored the advantages of cloud-optimised formats like Zarr. Now, it is time to discover which data products will be available and where you can access them.

4.0.2 What we will learn

- What Sentinel data will be available as EOPF Zarr products?
- A sneak peek into where you can access EOPF Zarr products

4.1 Available EOPF Zarr products

Re-engineered EOPF Zarr products are available for exploration via the EOPF Sentinel Zarr Sample Service STAC Catalog. Data from Sentinel-1, Sentinel-2 and Sentinel-3 missions are being reprocessed and made available.

> **!** Important
>
> The re-processing from the Sentinel missions is an ongoing activity as part of the EOPF Sentinel Zarr Sample Service. This page and our tutorials will continuously be updated as soon as new data products are available.

An overview of the datasets that are being re-engineered for different processing levels is given below.

4.1.1 Sentinel-1

Sentinel-1 is a radar imaging mission that is composed of a constellation of two polar-orbiting satellites providing continuous all-weather, day and night imagery.

Product	Instrument	Description	Available at
Level-1 GRD	Ground Range Detected	The Sentinel-1 Level-1 GDR products consist of focused SAR data that has been detected, multi-looked and projected to ground range using the Earth ellipsoid model WGS84.	this link
Level-1 SLC	Single Look Complex (	The Sentinel-1 Level-1 SLC products consist of focused SAR data, geo-referenced using orbit and attitude data from the satellite, and provided in slant-range geometry.	this link
Level-2 OCN	Ocean	The Sentinel-1 Level-2 OCN products for wind, wave and currents applications may contain the following geophysical components derived from the SAR data: Ocean Wind field (OWI), Ocean Swell spectra (OSW), Surface Radial Velocity (RVL).	this link

4.1.2 Sentinel-2

Sentinel-2 acquires optical imagery at high spatial resolution (10m to 60m) over land and coastal waters. The mission supports applications such as agricultural monitoring, emergency

management, land cover classifications, and water quality.

Product	Instrument	Description	Available at
Level-1C	Multi-Spectral Instrument	The Sentinel-2 Level-1C product is composed of 110x110 km2 tiles (ortho-images in UTM/WGS84 projection). Earth is subdivided into a predefined set of tiles, defined in UTM/WGS84 projection and using a 100 km step.	this link
Level-2A	Multi-Spectral Instrument	The Sentinel-2 Level-2A Collection 1 product provides orthorectified Surface Reflectance (Bottom-Of-Atmosphere: BOA), with sub-pixel multispectral and multitemporal registration accuracy.	this link

4.1.3 Sentinel-3

Sentinel-3 is a mission that regularly measures our Earth's oceans, land, rivers, lakes, ice on land, sea ice, and the atmosphere. Its goal is to keep track of and help us understand how these large parts of our planet change over long periods.

4.1.3.1 Ocean and Land Colour Instrument

Product	Product	Description	Available at
Level-1 EFR	Earth Full Resolution	Provides TOA radiances at full resolution for each pixel in the instrument grid, each view and each OLCI channel, plus annotation data associated with OLCI pixels.	this link
Level-1 ERR	Earth Reduced Resolution	The Sentinel-3 OLCI L1 ERR product provides TOA radiances at reduced resolution for each pixel in the instrument grid, each view and each OLCI channel, plus annotation data associated with OLCI pixels.	this link
Level-2 LFR	Land Full Resolution	The Sentinel-3 OLCI L2 LFR product provides land and atmospheric geophysical parameters computed for full resolution.	this link
Level-2 LRR	Land Reduced Resolution	The Sentinel-3 OLCI L2 LRR product provides land and atmospheric geophysical parameters computed for reduced resolution.	this link

4.1.3.2 Sea and Land Surface Temperature Radiometer

Product	Data	Description	Available at
Level-1 RBT	Radiance Brightness Temperature	The Sentinel-3 SLSTR Level-1B RBT product provides radiances and brightness temperatures for each pixel in a regular image grid for each view and SLSTR channel.	this link
Level-2 LST	LST: Land Surface Temperature	The Sentinel-3 SLSTR Level-2 LST product provides land surface temperature.	this link

4.2 Conclusion

In this section, we provide an overview of the available **Sentinel Missions** that will be re-processed and made available as EOPF Zarr products.

4.3 What's next?

In the following chapter, we will dive deep into `EOPF Zarr` format and understand why it can provide an optimal and efficient application.

Part II

About EOPF Zarr

5 Overview of the EOPF Zarr format

5.0.1 Introduction

In our journey to understand **cloud-optimised Earth Observation** (EO) data, we have frequently mentioned the **Zarr** format. Now, we will take a closer look and truly understand what **Zarr** is and why it is such a game-changer for large datasets like those from ESA's Sentinel missions. This chapter will break down the essential building blocks of **Zarr**, explaining how it organises data to make it incredibly efficient for cloud-based analysis.

5.0.2 What we will learn

- What is Zarr?
- What are the main components of Zarr?
- How is Zarr organised?

5.1 What Is Zarr?

Zarr is an open-source, cloud-native protocol for storing multi-dimensional arrays. It is specifically designed to work well with cloud storage and larger-scale computing systems and can be seen as a cloud-native alternative to older formats like HDF5 or NetCDF.

Key advantage to traditional formats is that the Zarr specification stores large multi-dimensional arrays in **chunks**, which are smaller pieces of the larger array. Chunks can be accessed individually, or multiple chunks can be read and written in parallel, making data access highly **efficient**.

Zarr works across different storage systems, including local file systems, cloud object storage, as well as distributed file systems, offering a greater **flexibility** compared to traditional file formats.

In addition, Zarr embeds **metadata** directly alongside the data. This makes Zarr **self-descriptive**, as each data array contains descriptive information about itself, such as data type, dimensions or additional attributes.

> **Note**
>
> Pro tip: Learn more about Zarr in the official Zarr Documentation and the Zarr V3 storage specification

5.2 Components of Zarr

Zarr is organised in a **human-readable**, **hierarchical** structure using simple JSON metadata files and is composed of `groups and stores`, `chunks` and `metadata`:

5.2.1 Groups and Stores

Groups and **stores** are concepts that allow Zarr to differentiate between (i) where the data is stored (**stores**) and (ii) how it is organised (**groups**). A **group** is a container for logically organising the data, similar to folders in a file system. A **store** defines where the data is stored; it can be, e.g. a bucket in the cloud or a directory on a disk.

5.2.2 Chunks

Zarr divides arrays into smaller, independent pieces (**chunks**). Through chunking, it is possible to retrieve and process specific areas without loading the complete dataset. Its organisation into chunks is the main reason for Zarr's high performance. Chunks are saved as binary files inside a `/c` directory and are further organised through nested folder paths based on their index, e.g. `c/0/0/0` for the chunk position `[0,0.0]`.

5.2.3 Metadata

Zarr uses descriptive **metadata** to describe the individual arrays but also the full hierarchy of the dataset. Metadata is stored in `zarr.json` files and is available on the array, group and store levels. This structured metadata approach makes Zarr datasets **self-descriptive** and easy to navigate.

The graphic below shows an overview of all relevant Zarr components.

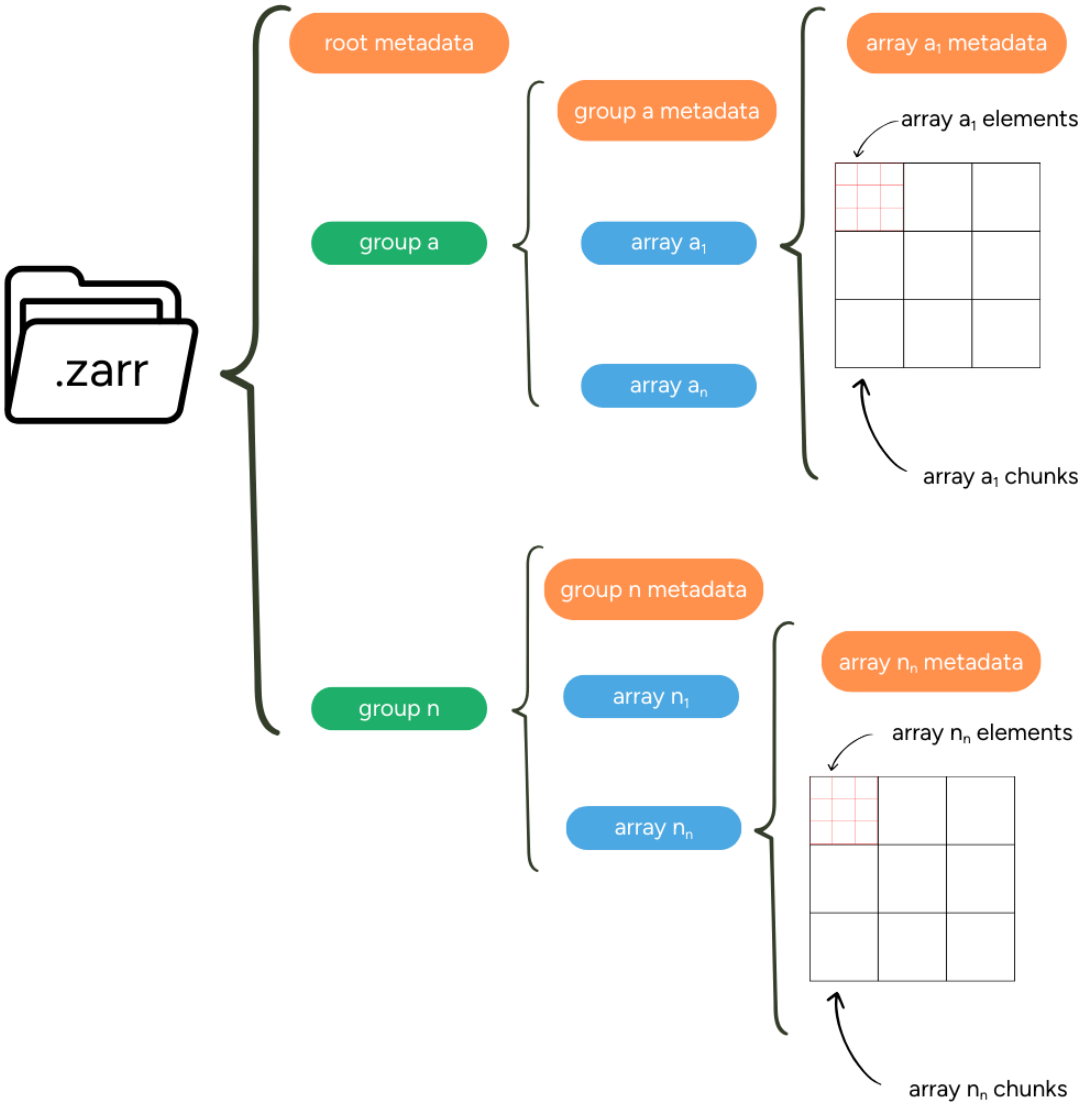


Figure 5.1: Zarr conceptual structure and overview of Zarr components

5.3 EOPF Zarr Structure

The ESA EOPF defines `Zarr` as the encoding format for the EOPF Sentinel Zarr Samples Service. The Zarr encoding is well aligned with ESA's objective of enhancing the accessibility of Sentinel data by modernising the previous `.SAFE` encoding into a flexible, cloud-native

structure. The cloud-native nature of `zarr` is expected to broaden the applications of the Sentinel data within the geospatial community while maintaining data quality and established algorithms.

EOPF Zarr products consist of four main groups:

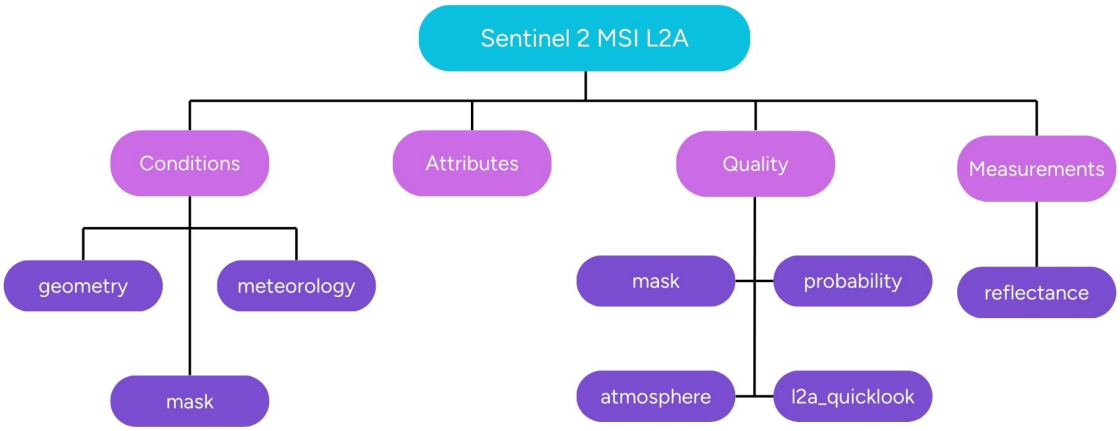
Group	Description
Attributes	STAC format metadata for the `Zarr` element
Measurements	Main retrieved variables
Conditions	Measurement context (geometric angles, meteorological/instrumental data)
Quality	Flags and quality information for measurement filtering

5.3.1 EOPF Zarr product example - Sentinel-2 L2A

Let us imagine a Sentinel-2 L2A tile. The tile has dimensions of approximately 10,980 by 10,980 pixels, and includes 12 spectral bands (B01 to B12, excluding B10) at different resolutions, plus additional data arrays such as a **Scene Classification Map** (SCL) and **Atmospheric Optical Thickness** (AOT).

For efficient handling, the data is divided into 1,024 by 1,024-pixel chunks. This chunking strategy allows for optimal performance when reading specific spatial regions of interest.

The figure below gives a graphical overview of how an EOPF Zarr Sentinel-2 L2A product file is organised.



The table below provides a more detailed outline of what content is available in the different groups.

Group		Content
Attributes		Processing history metadata Chunking configuration Global metadata (acquisition time, sensing time, etc.) Product-specific metadata
Measurements	**10m resolution (r10)**	B02 (Blue, 490nm) B03 (Green, 560nm) B04 (Red, 665nm) B08 (NIR, 842nm)
	20m resolution (r20)	B05 (Red Edge 1, 705nm) B06 (Red Edge 2, 740nm) B07 (Red Edge 3, 783nm) B8A (Narrow NIR, 865nm) B11 (SWIR 1, 1610nm) B12 (SWIR 2, 2190nm)
	60m resolution (r60)	B01 (Coastal aerosol, 443nm) <br?B09 (Water vapour, 945nm)

Group	Content
Conditions	Sun angles (zenith, azimuth) Viewing angles Mean solar irradiance Atmospheric parameters such as (i) Aerosol Optical Thickness (AOT), (ii) Water Vapour (WV) and (iii) Cloud and snow probability
Quality	Scene Classification Layer (SCL) Quality flags for each band Detector footprint Defective pixels masks

> **Note**
>
> Zarr Deep Dive: Dive deeper into the benefits of Zarr in a blog post by Lindsey Nield from the Earthmover team: Fundamentals: What is Zarr? A Cloud-Native Format for Tensor Data.

5.4 Conclusion

The EOPF Zarr structure allows for efficient access to individual bands or specific spatial regions without loading the entire dataset, making it ideal for large-scale geospatial analysis. It further ensures all relevant metadata is co-located with the data, enhancing data discoverability and usability.

5.5 What's next?

Now that you have a theoretical grasp of the **Zarr** format, the next section Discover EOPF Zarr - Sentinel-2 L2A will provide a first hands-on experience opening an EOPF Zarr product. We will transition to our first **Jupyter Notebook** where you will directly interact with a Zarr store.

6 Discover EOPF Zarr - Sentinel-2 L2A

Launch this notebook in JupyterLab

6.0.1 Introduction

This tutorial introduces you to the structure of an **EOPF Zarr** product sample for **Sentinel-2 L2A** data. We will demonstrate how to access and open a `.zarr` product sample with `xarray`, how to visualise the `zarr` encoding structure, explore embedded information, and retrieve relevant metadata for further processing.

6.0.2 What we will learn

- How to open a `.zarr` file using `xarray`?
- The general structure of a Sentinel-2 L-2A item
- How to access metadata that describes the `.zarr` encoding?

6.0.3 Prerequisites

This tutorial uses a re-processed sample dataset from the EOPF Sentinel Zarr Samples Service STAC API that is available for direct access here.

The selected `zarr` product is a Sentinel-2 L2A tile from the 10th of June 2025: * File name: `S2C_MSIL2A_20250610T103641_N0511_R008_T32UMD_20250610T132001.zarr`.).

6.0.3.1 Import libraries

```
import os
import xarray as xr
```

6.0.3.2 Helper functions

6.0.3.2.1 `print_gen_structure`

This function helps us to retrieve and visualise the names for each of the stored groups inside a `.zarr` product. As an output, it will print a general overview of elements inside the `zarr`.

```
def print_gen_structure(node, indent=""):
    print(f"{indent}{node.name}")     #allows us access each node
    for child_name, child_node in node.children.items(): #loops inside the selected nodes to
        print_gen_structure(child_node, indent + "  ") # prints the name of the selected node
```

6.1 Open a Zarr Store

In a first step, we use the function `open_datatree()` from the `xarray` library to open a `.zarr` store as a `DataTree`. Inside, we need to define the following key word arguments:

- `filename_or_obj`: path leading to a `zarr` store
- `engine`: `'eopf-zarr'`, designed for the EOPF `zarr` by ESA.
- `op_mode`: extension by the `xarray-eopf` development for allowing an analysis or native mode. For more information visit the xarray-eopf documentation.
- `chunks`: loads the data with dask using the engine's preferred chunk size, generally identical to the format's chunk size

The final print of the `DataTree` object is commented out, as the display can be quite extensive, showing the entire content within the `.zarr`. An alternative is to apply a helper function that only displays the higher level structure as shown in the next code cell.

```
url = 'https://objects.eodc.eu/e05ab01a9d56408d82ac32d69a5aae2a:202506-s02msil2a/10/products/
s2l2a_zarr_sample= xr.open_datatree(url,
    engine="eopf-zarr", # storage format
    op_mode="native", # no analysis mode
    chunks={}, # allows to open the default chunking
)
```

If we apply the helper function `print_gen_structure` on the root of the `DataTree` object, we will get a listing of the tree-like structure of the object. We can see all Zarr groups, such as `measurements`, `quality` and `conditions`, their sub-groups and content.

```
print("Zarr Sentinel 2 L2A Structure")
print_gen_structure(s2l2a_zarr_sample.root)
print("-" * 30)
```

```
Zarr Sentinel 2 L2A Structure
None
  conditions
    geometry
    mask
      detector_footprint
        r10m
        r20m
        r60m
      l1c_classification
        r60m
      l2a_classification
        r20m
        r60m
    meteorology
      cams
      ecmwf
  measurements
    reflectance
      r10m
      r20m
      r60m
  quality
    atmosphere
      r10m
      r20m
      r60m
    l2a_quicklook
      r10m
      r20m
      r60m
    mask
      r10m
      r20m
      r60m
    probability
      r20m
------------------------------
```

6.2 Extract information from Zarr groups

In a next step, we can explore the content of individually contained `.zarr` groups. By specifying the name of the group and subgroup and adding it into square brackets, we can extract the content of the relevant group. Let us for example extract the content of the subgroup `reflectance` under `measurements`.

As a result, it is visible that there are three subgroups of the parent node `measurements/reflectance`: `r10`, `r20` and `r60`, which are the `DataArrays` with the three different resolutions of the Sentinel-2 L2A data.

The `xarray.DataTree` structure allows the exploration of additional group-related metadata and information. For example, we can find the `chunksize` of each array and the coordinates.

```
# Retrieving the reflectance groups:
# s2l2a_zarr_sample["measurements/reflectance"] # Run it yourself for an inteactive overview
```

6.3 Extract Zarr metadata on different levels

Through `s2l2a_zarr_sample.attrs[]` we are able to visualise both the `stac_discovery` and `other_metadata` included in the `zarr` store. For the properties inside `stac_discovery` for example we can get the parameters included:

```
# STAC metadata style:
print(list(s2l2a_zarr_sample.attrs["stac_discovery"].keys()))
```

```
['assets', 'bbox', 'geometry', 'id', 'links', 'properties', 'stac_extensions', 'stac_version
```

We are also, able to retrieve specific information by diving deep into the `stac_discovery` metadata, such as:

```
print('Date of Item Creation: ', s2l2a_zarr_sample.attrs['stac_discovery']['properties']['cr
print('Item Bounding Box    : ', s2l2a_zarr_sample.attrs['stac_discovery']['bbox'])
print('Item ESPG            : ', s2l2a_zarr_sample.attrs['stac_discovery']['properties']['pr
print('Sentinel Platform    : ', s2l2a_zarr_sample.attrs['stac_discovery']['properties']['pl
print('Item Processing Level: ', s2l2a_zarr_sample.attrs['stac_discovery']['properties']['pr
```

```
Date of Item Creation:  2025-06-10T13:20:01+00:00
Item Bounding Box    :  [9.146276872400831, 52.25344953517325, 7.500940412097549, 53.2495367
Item ESPG            :  32632
Sentinel Platform    :  sentinel-2c
Item Processing Level:  L2A
```

And from `other_metadata`, we are able to retrieve the information specific to the instrument variables.

```
# Complementing metadata:
print(list(s2l2a_zarr_sample.attrs["other_metadata"].keys()))
```

```
['AOT_retrieval_model', 'L0_ancillary_data_quality', 'L0_ephemeris_data_quality', 'NUC_table_
```

6.4 Now it is your turn

As we are able to retrieve several items from the EOPF Sentinel Zarr Samples Service STAC API, let us try the following: ### Task Go to the Sentinel-2 Level-2A collection and: - Choose an item of interest. - Replicate the workflow and explore the item's metadata. When was it retrieved? - What are the dimensions? - What is the detailed location of the item?

6.5 Conclusion

This tutorial provides an initial understanding of the `zarr` structure for a Sentinel-2 L2A product sample. By using the `xarray` library, we can effectively navigate and inspect the different components within the `zarr` format, including its metadata and array organisation.

6.6 What's next?

Now that you've been introduced to the `.zarr` encoding format, learned its core concepts, and understood the basics of how to explore it, you are prepared for the next step. In the following chapter we will introduce you to **STAC** and the **EOPF Zarr STAC Catalog**. As we go along, we are more and more transition from theory to practice, providing you with hands-on tutorials working with EOPF `.zarr` products.

Part III

EOPF and STAC

7 Introduction to STAC

7.0.1 Introduction

Welcome to the chapter on EOPF and STAC. In the following section, we will introduce you to the **S**patio-**T**emporal **A**sset **C**atalog (STAC). We will explain its fundamental principles and, most importantly, we will explore its structure and core components. Understanding the fundamentals of STAC is key in order to be able to effectively discover and access data from STAC catalogs.

7.0.2 What we will learn

- What STAC is and why it is important?
- Navigate through the STAC ecosystem, and
- Understand the main components of STAC

7.1 About STAC

The **S**patio-**T**emporal **A**sset **C**atalog (STAC) is a standardised way to catalog and describe geospatial (raster) data. STAC makes it easier to discover, access, and work with geospatial data, in particular satellite data, as it provides a **common language for describing spatial and temporal characteristics** of the data. This common language improves interoperability between different data providers and software tools.

The main goal of STAC is to allow data providers to share their data easily, making it universal for users to understand the where, when, how, and what of the collected data.

STAC uses **JSON** (JavaScript Object Notation) to structure the metadata of geo-referenced datasets. JSON makes it machine-readable. Through its design, STAC is simple and extensible in its design as it is based on a network of JSON files.

STAC has evolved into a well-recognised community standard. The key benefit supporting its wide adoption is that one can use the same code and API to access data from different data repositories.

7.2 The STAC ecosystem

STAC has evolved into a vast ecosystem offering various resources and tools for accessing, managing, and building STAC catalogs. Below is a non-exclusive list of tools and plug-ins that will help to explore the STAC ecosystem:

Category	Tool/Plugin	Description	Language
STAC Tools	STAC Browser	A user-friendly web interface for visually exploring and interacting with various STAC catalogs.	Web interface
	STAC Server	A reference implementation for serving STAC catalogs and collections.	Python
STAC libraries and plug-ins	STAC Validator	A tool for programmatically validating STAC Catalogs, Collections, and Items to ensure compliance with the STAC specification.	Python
	PySTAC	A Python library for reading, writing, and validating STAC objects, facilitating the creation and manipulation of STAC data.	Python
	pystac-client	A Python library that provides a convenient and powerful interface for searching and accessing STAC data from STAC API servers.	Python
	rstac	An R package that provides functionalities for interacting with STAC APIs and working with STAC objects within the R environment.	R
	STAC.jl	A Julia package designed for working with STAC, enabling users to interact with STAC catalogs and process geospatial data.	Julia
	STACCube.jl	A Julia package that facilitates the creation and management of STAC-compliant data cubes from various geospatial datasets.	Julia

7.3 STAC components

Now, let us start exploring the structure of STAC. STAC consists of four main components: (i) `Catalog`, (ii) `Collection`, (iii) `Item` and (iv) `Asset`. See the figure below for the principal organisation of the STAC components.

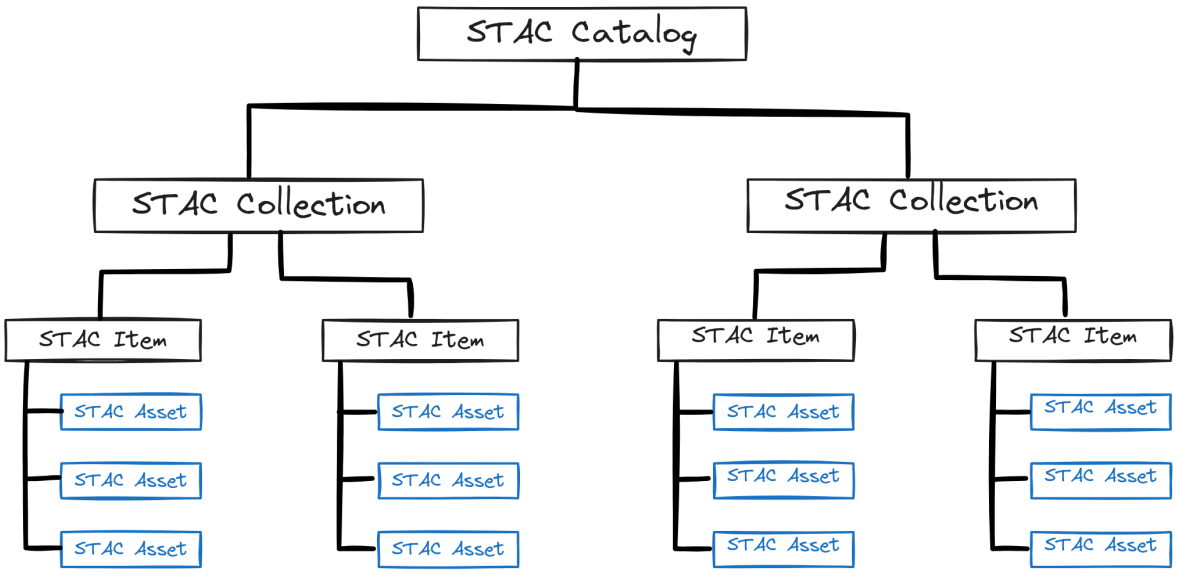


Figure 7.1: STAC structure

Let us now explore in more detail the individual components:

7.3.1 Catalog

A `Catalog` serves as the initial entry point of a STAC. A catalog is a very simple construct; it simply provides links to `Collections` or `Items`. The closest analogue is a folder on your computer. A `Catalog` can be a *folder* for `Items`, but it can also be a `folder` for `Collections` or other `Catalogs`. When searching for specific data, you first establish a connection to a valid STAC catalog.

7.3.2 Collection

Collections are containers that support the grouping of `Items`. The `Collection` entity shares most fields with the `Catalog` entity but has several additional fields, such as license, extent (spatial and temporal), providers, keywords and summaries. Every `Item` in a `Collection` links back to its `Collection`. `Collection` is often used to provide additional structure in a STAC catalog.

> **Note**
>
> But when to use a `Collection` versus a `Catalog`? A `Collection` generally consists of a set of assets that share the same properties and share higher-level metadata. For example, data from the same satellite sensor or constellation would typically be in one `Collection`.
> `Catalogs`, in turn, are used to split overly large `Collections` into groups and to group collections into a catalog of Collections (e.g. as an entry point for navigation to several Collections).
> It is recommended to use `Collections` for what you want users to find and `Catalogs` for structuring and grouping `Collections`.

7.3.3 Item

An `Item` is the fundamental element of STAC and typically represents a single scene at one place and time. It is a `GeoJSON` supplemented with additional metadata, which serves as an index to `Assets`.

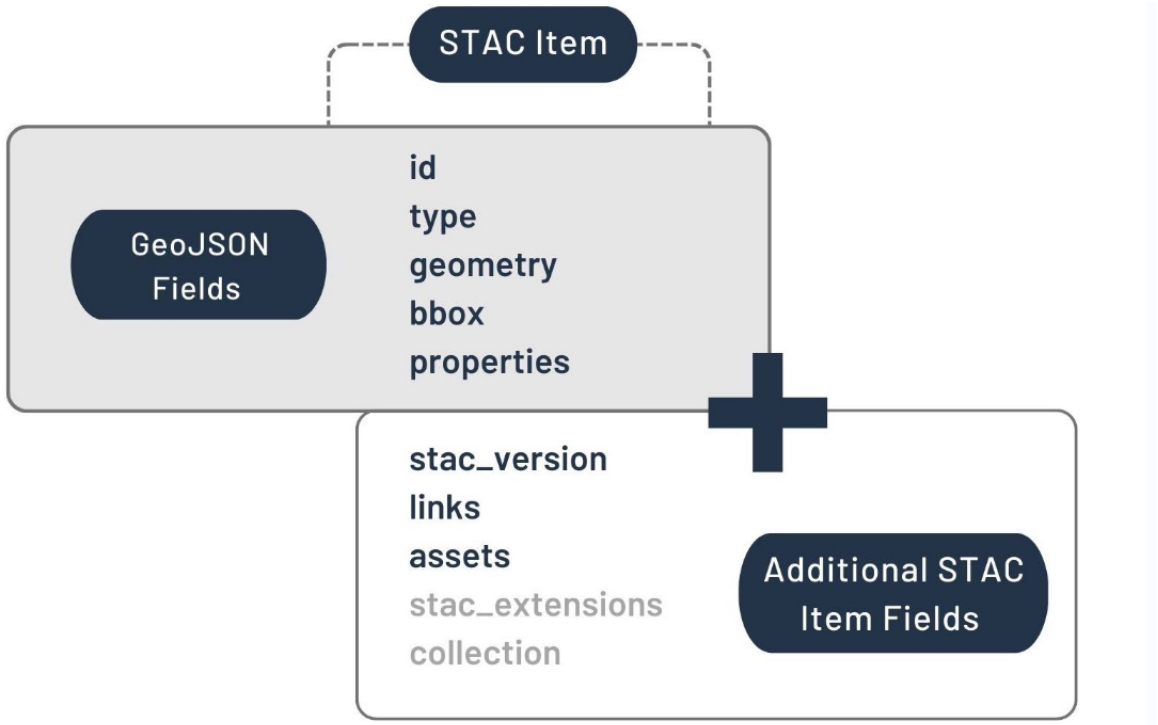


Figure 7.2: Item entity

7.3.4 Asset

An `Asset` is the smallest element inside a STAC and represents the individual data file that is linked in a STAC `Item`.

7.3.5 Analogy: Organising a drinks menu as a STAC

To better understand the relation of STAC components, let us imagine a **Drinks** menu as a STAC. How would you structure *Drinks* as a STAC?

Let us start with the *Drinks* category itself. The menu is analogous to a STAC `Catalog`, as it serves as the top-level entry point, providing an overview of all beverages available.

Within this Drinks **catalog**, we can group the *Drinks* further in There are hot and cold beverages, caffeinated and non-caffeinated drinks. These categories represent **Collections** in STAC. For our analogy, let us say the menu is divided into two main collections:

- **Caffeinated Drinks Collection**: This section groups all beverages that contain caffeine.
- **Non-Caffeinated Drinks Collection**: This section groups all beverages that do not contain caffeine.

Each of these collections contains specific drinks, which are analogous to STAC **Items**. Drink `Items` could be, e.g. Juices or Milks. Both represent again a group of specific juices and milks, which are analogous to `Assets` in STAC. For the Drink `Items defined, their`Assets‘ might include:

Item	Assets
Milks	Oat Milk Regular Milk
Juices	Apple Juice Organge Juice …

The STAC structure allows us to easily navigate a vast amount of data, just as a well-organised menu helps a customer quickly find their desired drink.

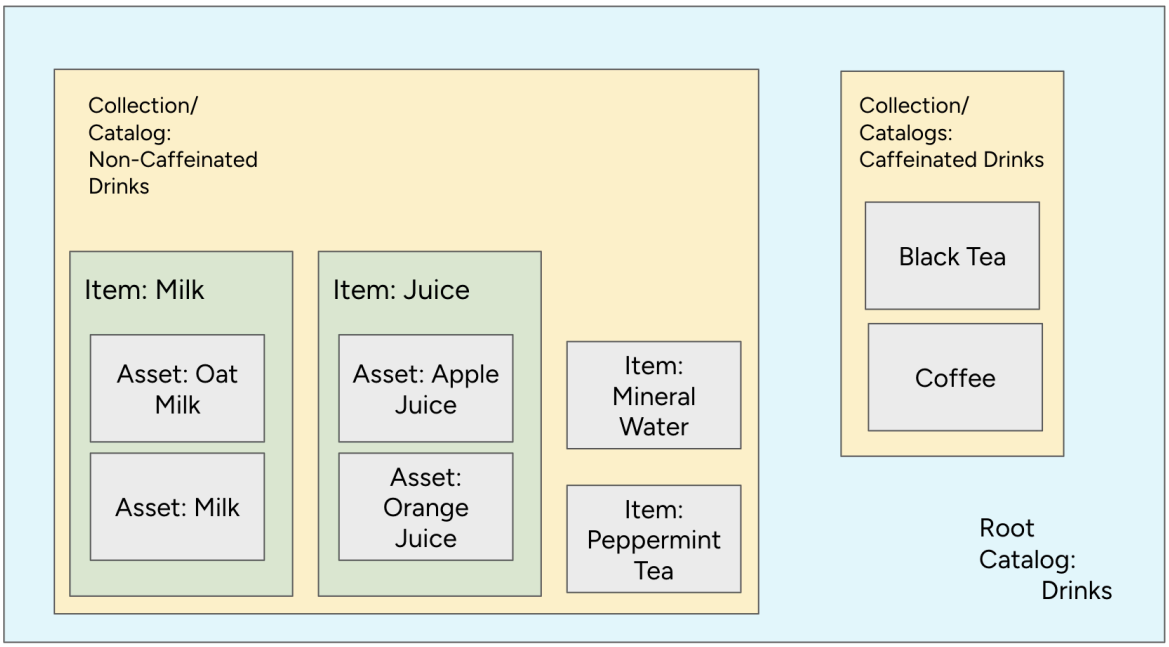


Figure 7.3: Drinks Menu as a STAC analogy

7.4 Conclusion

In this section, you got an introduction to the **Spatio-Temporal Asset Catalog** (STAC) and learned what STAC is and explored the main components of a STAC. Understanding the distinction between `Catalog`, `Collection`, `Items` and `Assets` is important to effectively navigating through STAC APIs.

7.5 What's next?

In the following section, we will explore the web interface of the EOPF Sentinel Zarr Samples Service STAC Catalog.

8 Explore the web interface of the EOPF Zarr STAC Catalog

8.0.1 Introduction

This section introduces you to the EOPF Sentinel Zarr Samples Service STAC Catalog, which offers access to the re-engineered Sentinel-1, Sentinel-2 and Sentinel-3 data products. We will guide you through its web interface, inspect the various levels of STAC components, and demonstrate how to access the underlying Sentinel Zarr data.

8.0.2 What we will learn

- How does the STAC browser interface work
- Explore the available Collections within the EOPF Sentinel Zarr Samples Service STAC Catalog
- How to obtain access to EOPF Sentinel Zarr products from the EOPF Sentinel Zarr Samples Service STAC Catalog

8.1 Our Starting Point

The first step is to access the main homepage of the EOPF Sentinel Zarr Samples Service STAC Catalog. The landing page offers you a comprehensive overview of the available data collections. This serves as our **initial entry** point into the catalog.

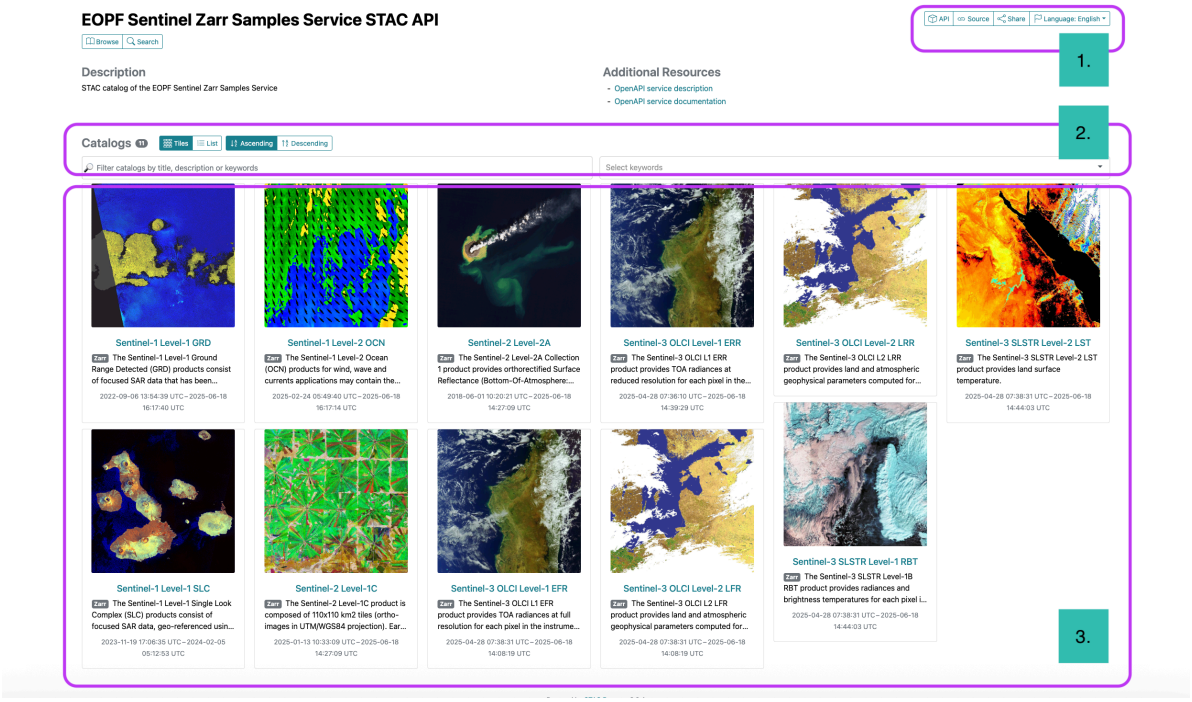


Figure 8.1: Home page

Three main areas can be identified: (i) the **API and URL** section, (ii) **Search bar** and (iii) **Collections Display**.

As outlined in detail in the book section The EOPF Available Datasets, the catalog displays the 11 distinct collections available from the Sentinel-1, Sentinel-2, and Sentinel-3 missions. The user interface provides an intuitive way to browse through all of these collections. It is possible to filter them by specific criteria or select them manually, allowing precise control over the displayed data.

8.2 Exploring Sentinel Collections

Let us now select one of the 11 `Collection` available, e.g. let us select the Sentinel 2-Level-2A collection. Once you have selected the Collection, you will be directed to the interface of a specific `Collection`.

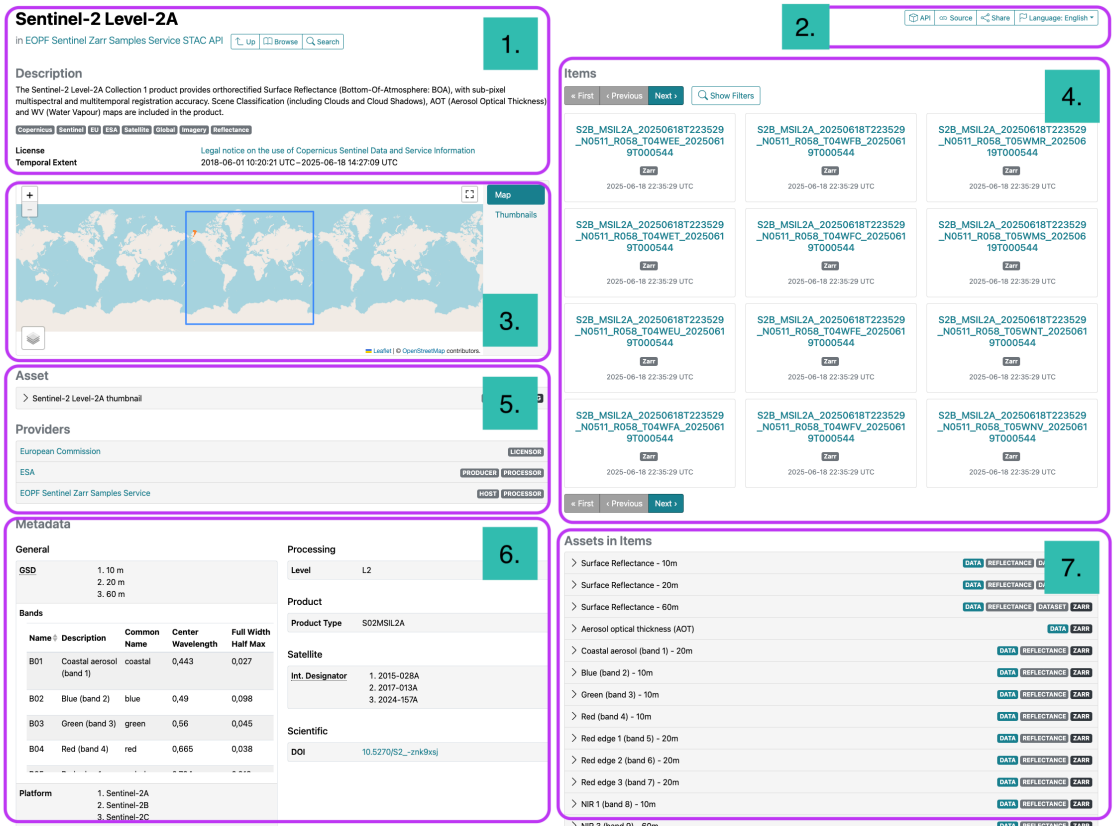


Figure 8.2: Web interface of the Sentinel-2 Level-2A Collection of the EOPF Sentinel Zarr Samples Service STAC Catalog

The interface can be divided into five principal sections:

1. **Description**: The name and a brief introduction to the available collection. This includes crucial details such as the **temporal extent** (the period over which the data was acquired)
2. **API and URL**: for further application.
3. **Spatial Extent**: The geographical area of the displayed items is shown on the map.
4. **Available Items**: On the right-hand side of the page, a list of links for the items contained within the collection corresponds to the selected collection.
5. **Collection Metadata**: A general overview of the Collection's metadata, its providers, instruments, and the corresponding DOIs for research.
6. **Metadata**: Specific information about the bands and instruments, such as backscatter or reflectance information.
7. **Assets in Items**: The Assets structure is available for the Items available in the Collection.

8.2.1 Filtering Collections

Any selected `Collection` (in this case Sentinel-2 Level-2A) allows us to filter the items of our interest by **temporal** and **spatial** extent. We can access this tab by clicking on **Show Filters** on the right side under the **Items** section.

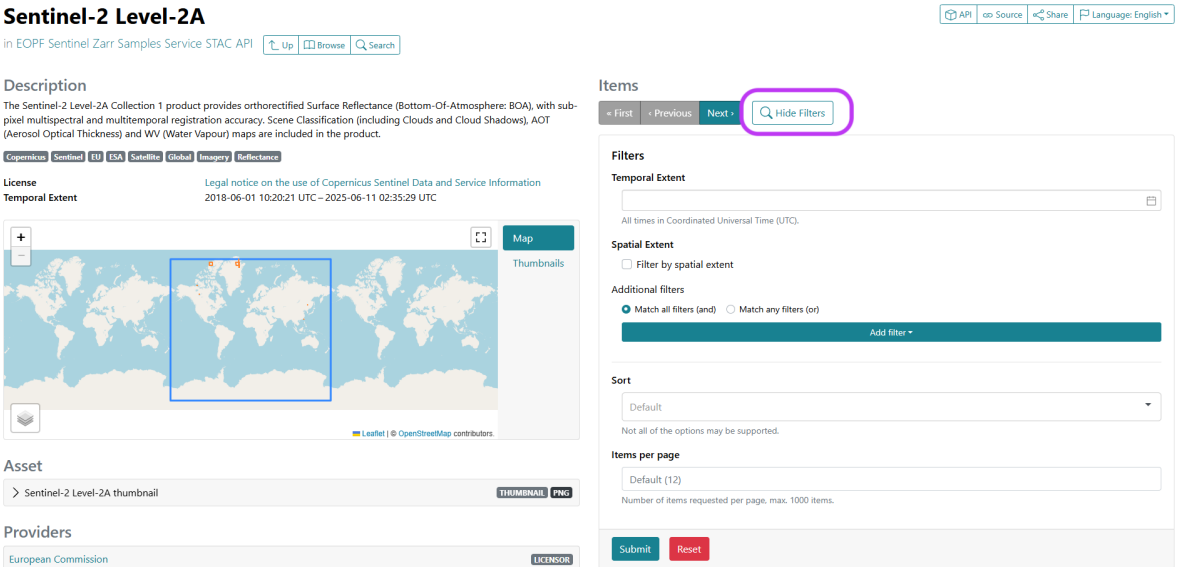


Figure 8.3: Overview of interface that opens when clicking in *Show Filters*

The interface that opens allows us to select on the calendar a specific period we are interested in. This is particularly useful when we are interested in a temporal analysis. Let us search over the available items captured between May 1st and May 5th 2024.

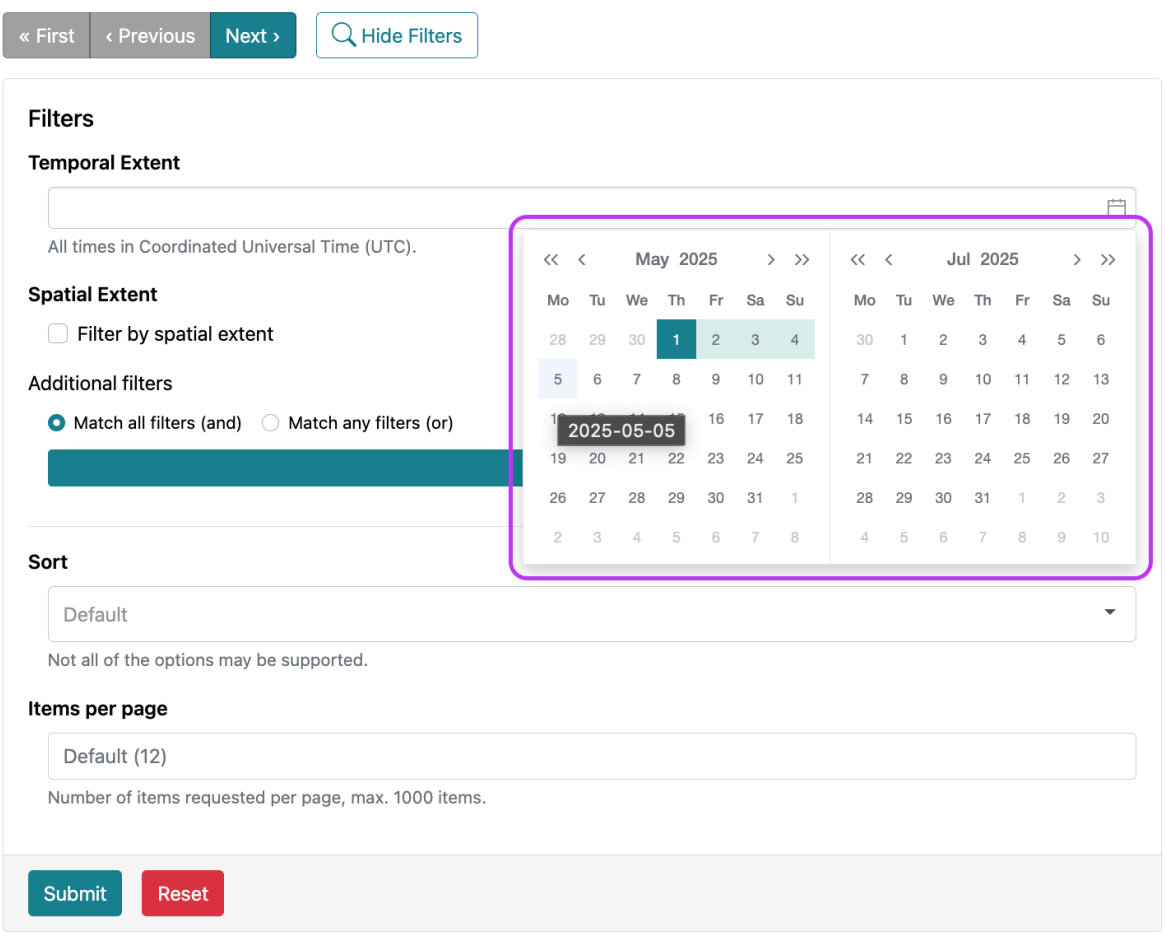


Figure 8.4: Filtering over time

Additionally, we can select a location we are interested in by checking the **Filter by spatial extent** box. This allows us to refine our search over an area of interest. Once you tick the box, a map activates and allows us to draw a bounding box that we can drag and drop. Let us select Europe as our area of interest.

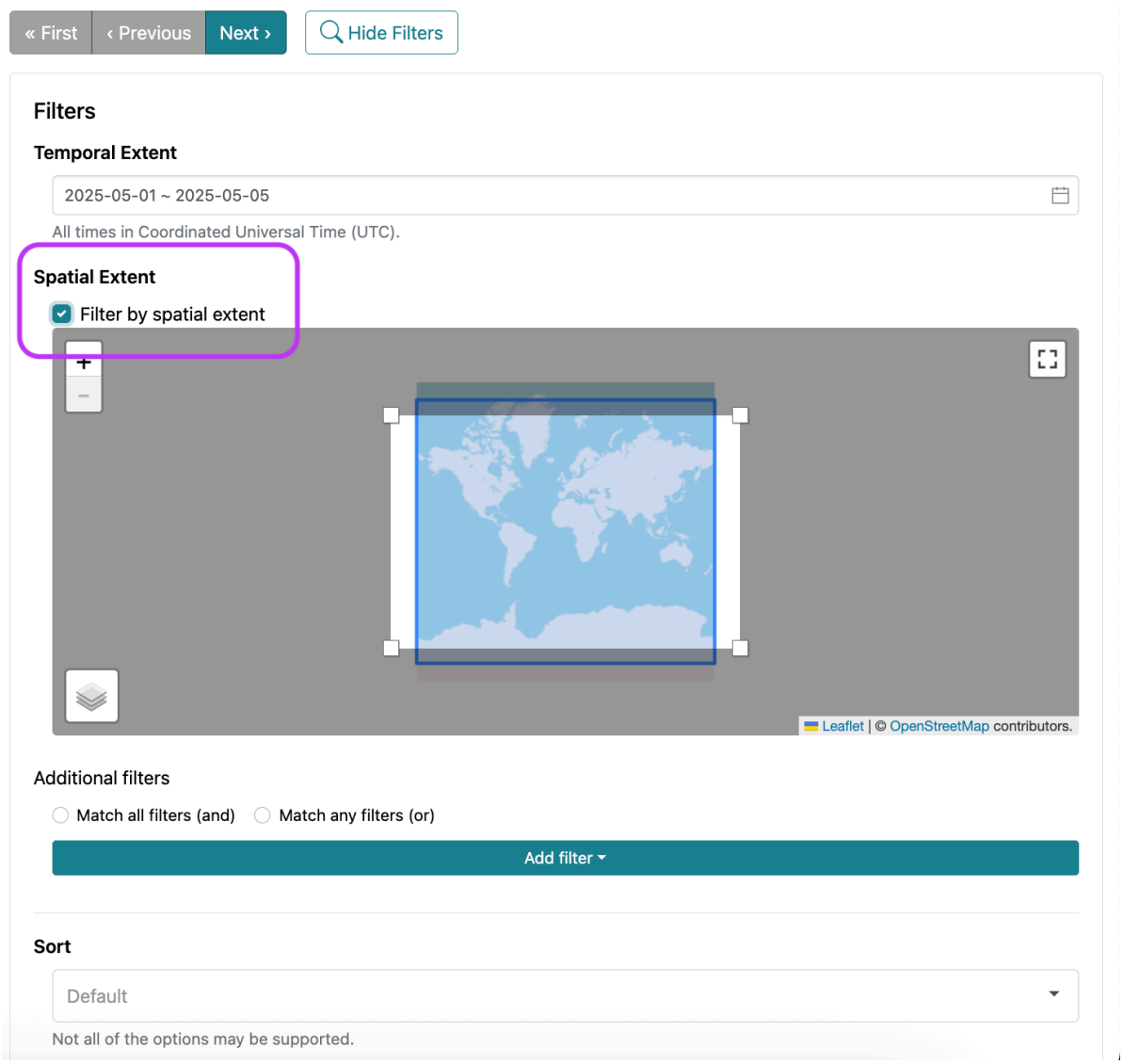


Figure 8.5: Filtering over spatial extent

Once we select the desired period and area via the filters, we can sort the items that match our search by ID, Date and Time, or select the number of resulting items we are interested in per page. In this case, we select 2, so the overview is digestible. Then, we click **Submit**.

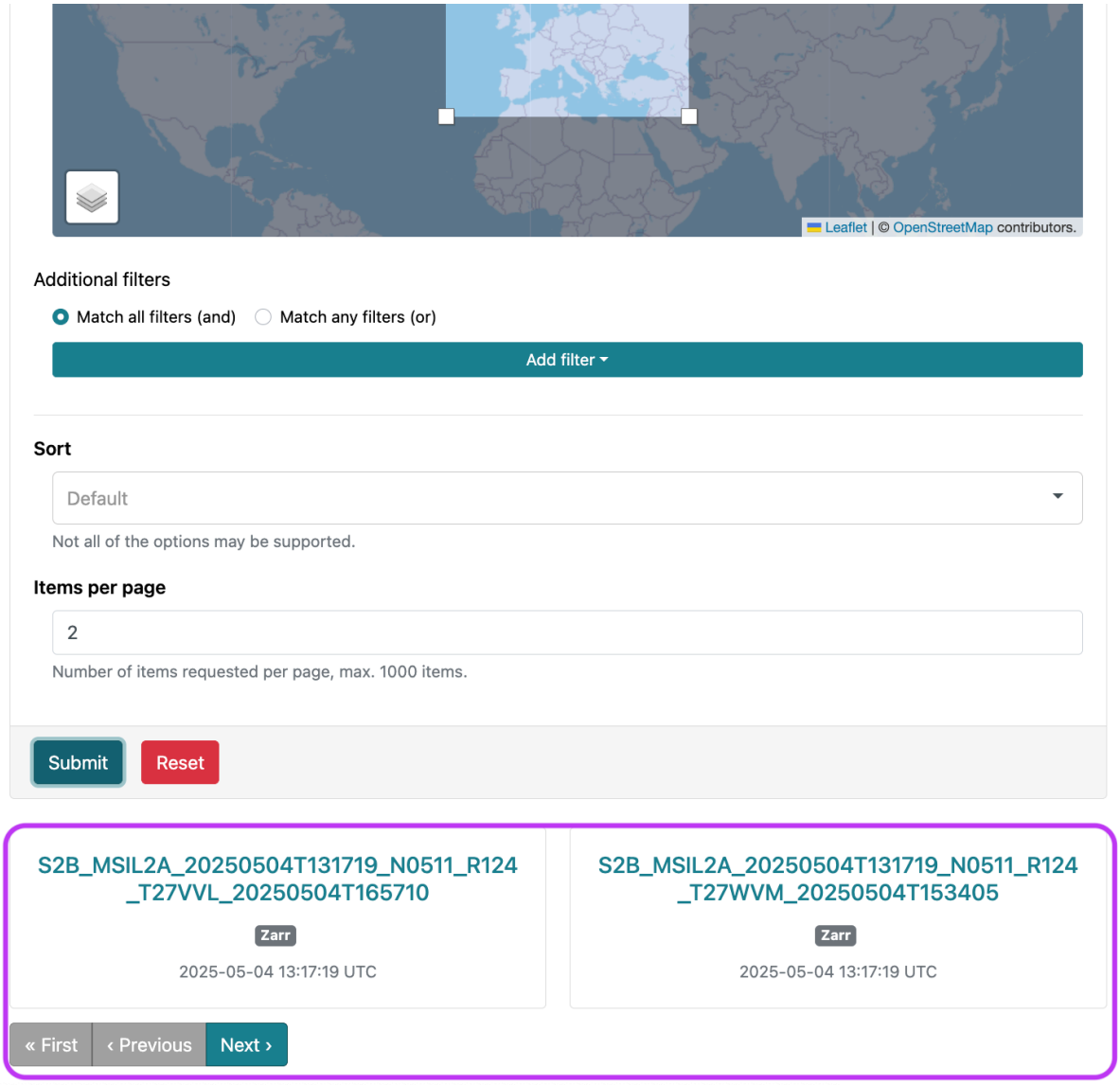


Figure 8.6: Resulting Items

Under the window, we can now see that two items appear. For example: `S2B_MSIL2A_20250504T131719_N0511_R`

We can select any of the resulting items, and this will enable us to access an `Item` inside the `Collection`.

8.3 Exploring Items

When a specific `Item` sparks our interest, clicking on it will bring us to a detailed overview page for that selected `Item`.

This `Item` interface is composed of:

1. **Description**: The name of the `Item`. Depending on the mission and collection it belongs to, the composition of the name changes.
2. **Spatial Extent**: The geographical footprint of the selected `Item` is shown on the map.
3. **Collection Metadata**: A general overview of the Collection's metadata, its providers, instruments, resolution, grid and the corresponding DOIs for research.
4. **API and URL**: Allows the entry point for further retrieval of the individual assets.
5. **Assets menu**: lists all the `Assets` that are part of the selected `Item`.

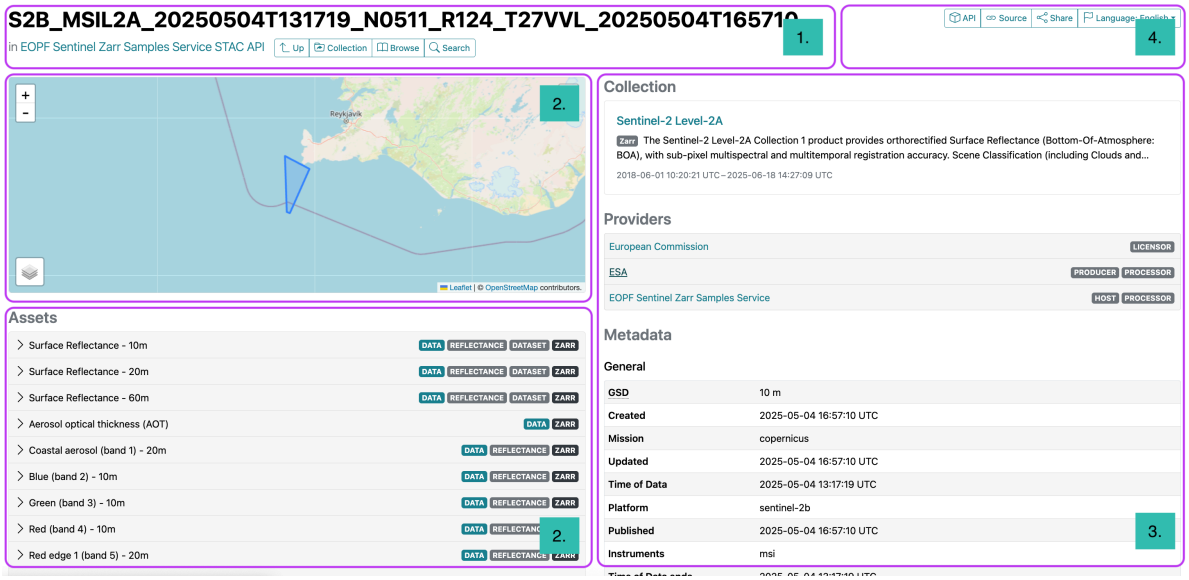


Figure 8.7: The item selected inside the Sentinel-2 Level-2A Collection

> **!** Important
>
> At this level of the STAC structure, we are already diving deep into the STAC levels, and have explored the `Catalog`, `Collection` and `Item` components described in the Introduction to STAC section.

8.4 Assets

Inside the selected `Item`, we will get an overview of the available `Assets` that belong to the `Item`. By expanding the **dropdown menu** for any `Asset` of interest, you can access its specific metadata and, most importantly, the actual data.

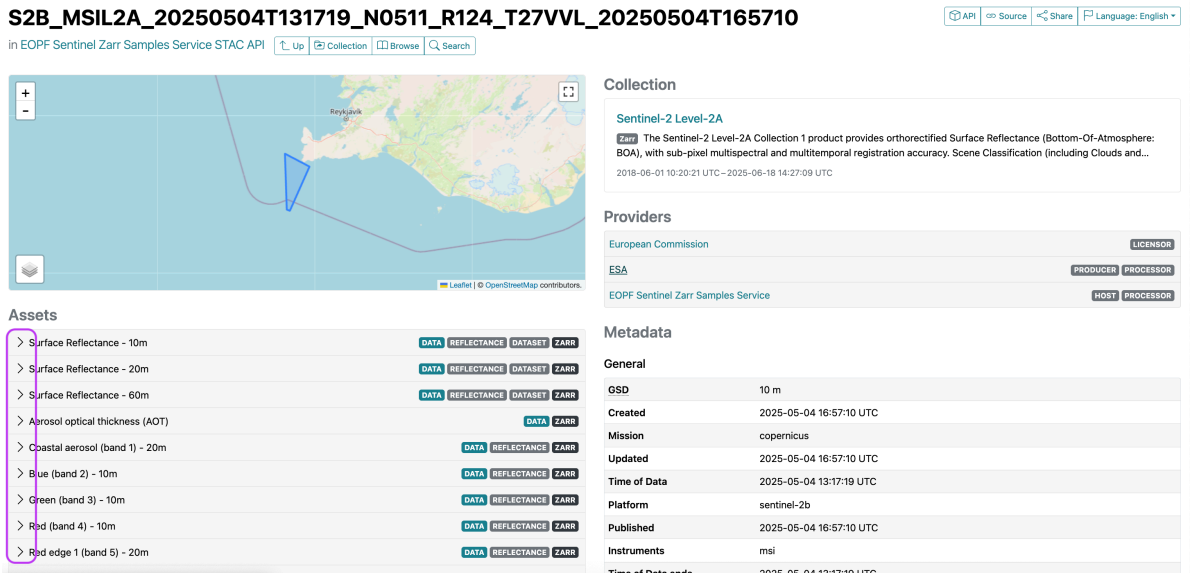


Figure 8.8: Overview of the web interface on Asset level

In the case of the **Sentinel-2 Level-2A** collection, each asset corresponds to one of the 13 spectral bands available. The array information provides details about the structure and content of the data, hinting at the actual values contained within the asset. It also contains the chunking grid specification and all the crucial metadata necessary to structure and process the data through a wide range of geospatial methodologies.

8.4.1 Accessing Assets

For direct data access of `Assetz` and therefore the actual data, there are two options provided via the web interface: * **Download**: It is possible to download only one asset separately in `.zarr` format by clicking the *Download* option associated with an Asset, and * **Copy URL**: We can also retrieve the unique URL of the specified `Asset`, which allows us to directly integrate the link into programmatic workflows, without the need to download the `Asset`. We can get it by simply clicking on the *Copy URL* option under the `Asset` of our interest.

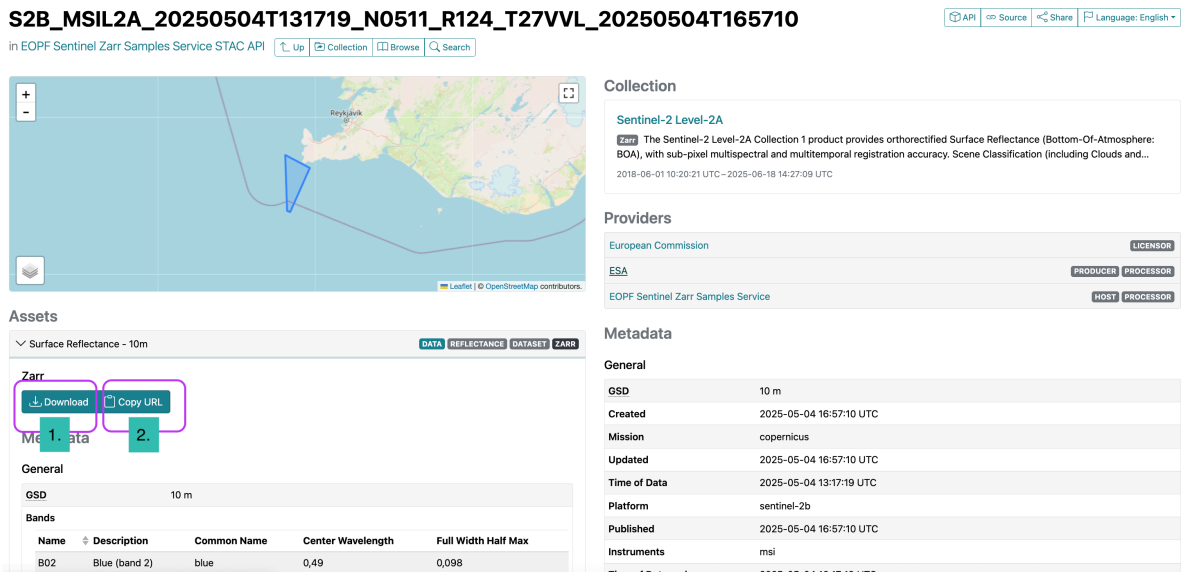


Figure 8.9: Data access options via the web interface

8.5 Now it is your turn

Your task now is to explore the web interface of the EOPF Sentinel Zarr Samples Service STAC Catalog on your own and explore other `Collections` than the one we showcased.

8.5.1 Task 1: Discover Sentinel-1 GRD Data

Navigate to the Sentinel-1 Level-1 GRD Collection.

- How many items can you find for the most recent two years available in the catalogue?
- Try filtering by different periods and observe how the results change. How many items are available for September 2023?

8.5.2 Task 2: Mapping Your Interests

Explore the interactive map within the Sentinel-1 Level-1 GRD and the Sentinel-2 Level-2A collections.

- Can you identify and list the names of at least three distinct geographical areas where a significant number of items are available? (Hint: Look for clusters of the displayed items!)

8.5.3 Task 3: Unpacking an Asset

Select an item from any collection that looks interesting to you. Click on it to view its details. Then, expand one of the assets.

- What kind of array information is provided?
- Who is the data provider?
- How would this information be useful if you were to process this data?

8.6 Conclusion

This section walked you through the web interface of the EOPF Sentinel Zarr Samples Service STAC Catalog. We have demonstrated how to navigate its interface, from the initial overview of the available `Collections` to the detailed inspection of specific `Items` and `Assets`. By understanding the structure and components of a STAC catalog, we are able to efficiently access re-engineered EOPF Zarr assets.

8.7 What's next?

In the next section, we will explore how to programmatically connect to and search through the EOPF Sentinel Zarr Samples Service STAC API with the help of the `pystac` and the `pystac-client` Python libraries.

9 Access the EOPF Zarr STAC API with Python

Launch this notebook in JupyterLab

9.0.1 Introduction

In this section, we will dive into the programmatic access of EOPF Zarr Collections available in the EOPF Sentinel Zarr Sample Service STAC Catalog. We will introduce Python libraries that enable us to effectively access and search through STAC catalogs.

9.0.2 What we will learn

- How to **programmatically browse** through available collections available via the EOPF Zarr STAC Catalog
- Understanding **collection metadata** in user-friendly terms
- **Searching for specific data** with help of the `pystac` and `pystac-client` libraries.

9.0.3 Prerequisites

For this tutorial, we will make use of the pystac and pystac_client Python libraries that facilitate the programmatic access and efficient search of a STAC Catalog.

9.0.3.1 Import libraries

```
import requests
from typing import List, Optional, cast
from pystac import Collection, MediaType
from pystac_client import Client, CollectionClient
from datetime import datetime
```

9.0.3.2 Helper functions

9.0.3.2.1 `list_found_elements`

As we are expecting to visualise several elements that will be stored in lists, we define a function that will allow us retrieve item `id`'s and collections `id`'s for further retrieval.

```
def list_found_elements(search_result):
    id = []
    coll = []
    for item in search_result.items(): #retrieves the result inside the catalogue.
        id.append(item.id)
        coll.append(item.collection_id)
    return id , coll
```

9.1 Establish a connection to the EOPF Zarr STAC Catalog

Our first step is to establish a connection to the EOPF Sentinel Zarr Sample Service STAC Catalog. For this, you need the Catalog's base URL, which you can find on the web interface under the **API & URL** tab. By clicking on **Source**, you will get the address of the STAC metadata file - which is available here.

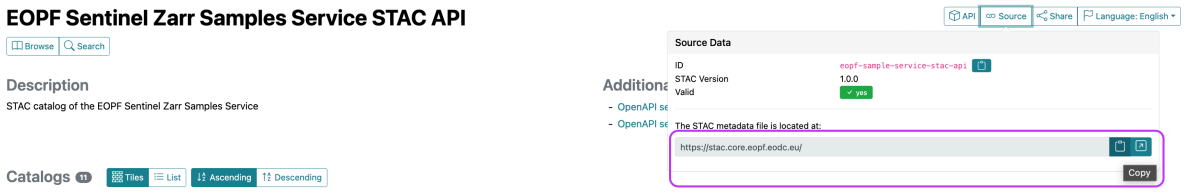


Figure 9.1: EOPF API url for connection

Copy paste the URL: `https://stac.core.eopf.eodc.eu/`.

With the `Client.open()` function, we can create the access to the starting point of the Catalog by providing the specific url. If the connection was successful, you will see the description of the STAC catalog and additional information.

```
eopf_stac_api_root_endpoint = "https://stac.core.eopf.eodc.eu/" #root starting point
eopf_catalog = Client.open(url=eopf_stac_api_root_endpoint) # calls the selected url
eopf_catalog
```

```
<Client id=eopf-sample-service-stac-api>
```

Congratulations. We successfully connected to the EOPF Zarr STAC Catalog, and we can now start exploring its content.

9.2 Explore available collections

Once a connection established, the next logical step is to get an overview of all the collections the STAC catalog offers. We can do this with the function `get_all_collections()`. The result is a list, which we can loop through to print the relevant collection IDs.

Please note: Since the EOPF Zarr STAC Catalog is still in active development, we need to test whether a collection is valid, otherwise you might get an error message. The code below is testing for validity and for one collection, it throws an error.

You see, that so far, we can browse through 10 available collections

```
try:
    for collection in eopf_catalog.get_all_collections():
        print(collection.id)

except Exception:
    print(
        "* [https://github.com/EOPF-Sample-Service/eopf-stac/issues/18 appears to not be reso
    )
```

```
sentinel-2-l2a
sentinel-3-slstr-l1-rbt
sentinel-3-olci-l2-lfr
sentinel-2-l1c
sentinel-3-slstr-l2-lst
sentinel-1-l1-slc
sentinel-3-olci-l1-efr
sentinel-3-olci-l1-err
sentinel-1-l2-ocn
sentinel-1-l1-grd
* [https://github.com/EOPF-Sample-Service/eopf-stac/issues/18 appears to not be resolved]
```

In a next step, we can select one `collection` and retrieve certain metadata that allow us to get more information about the selected collection, such as keywords, the ID and useful links for resources.

```
S2l2a_coll = eopf_catalog.get_collection('sentinel-2-l2a')
print('Keywords:        ',S2l2a_coll.keywords)
print('Catalog ID:      ',S2l2a_coll.id)
print('Available Links: ',S2l2a_coll.links)
```

```
Keywords:         ['Copernicus', 'Sentinel', 'EU', 'ESA', 'Satellite', 'Global', 'Imagery',
Catalog ID:       sentinel-2-l2a
Available Links:  [<Link rel=items target=https://stac.core.eopf.eodc.eu/collections/sentinel
```

9.3 Searching inside the EOPF STAC API

With the `.search()` function of the `pystac-client` library, we can search inside a STAC catalog we established a connection with. We can filter based on a series of parameters to tailor the search for available data for a specific time period and geographic bounding box.

9.3.1 Filter for temporal extent

Let us search on the `datetime` parameter. For this, we specify the `datetime` argument for a time period we are interested in, e.g. from 1 May 2020 to 31 May 2023. In addition, we also specify the `collection` parameter indicating that we only want to search for the Sentinel-2 L2A collection.

We apply the helper function `list_found_elements` which constructs a list from the search result. If we check the length of the final list, we can see that for the specified time period, 196 items were found.

```
time_frame = eopf_catalog.search(  #searching the catalog
    collections='sentinel-2-l2a',
    datetime="2020-05-01T00:00:00Z/2023-05-31T23:59:59.999999Z")  # the interval we are inte

# we apply the helper function `list_found_elements`
time_items=list_found_elements(time_frame)
print(time_frame)

print("Search Results:")
print('Total Items Found for Sentinel-2 L-2A between May 1, 2020, and May 31, 2023:  ',len(t
```

```
<pystac_client.item_search.ItemSearch object at 0x762f127a3130>
Search Results:
Total Items Found for Sentinel-2 L-2A between May 1, 2020, and May 31, 2023:   196
```

9.3.2 Filter for spatial extent

Now, let us filter based on a specific area of interest. We can use the `bbox` argument, which is composed by providing the top-left and bottom-right corner coordinates. It is similar to drawing the extent in the interactive map of the EOPF browser interface.

For example, we defined a bounding box of the outskirts of Innsbruck, Austria. We then again apply the helper function `list_found_elements` and see that for the defined area, only 39 items are available.

```
bbox_search =  eopf_catalog.search(  #searching the catalog
    collections='sentinel-2-l2a',
    bbox=(
        11.124756, 47.311058, #top left
        11.459839, 47.463624  #bottom-right
        )
)

innsbruck_sets=list_found_elements(bbox_search) #we apply our constructed function that store

#Results
print("Search Result:")
print('Total Items Found:  ',len(innsbruck_sets[0]))
```

```
Search Result:
Total Items Found:   50
```

9.3.3 Combined filtering: Collection + temporal extent + spatial extent

As a usual workflow, we often look for datasets within an AOI and a specific period of time. The `search()` function allows us also to combine the `collection`, `bbox` and `datetime` arguments in one search request.

Let us now search for Items available for the AOI around Innsbruck within the previously defined timeframe for the **Sentinel-2 Level-2A** collection. As a result, we get 27 Items that are available for our selection.

```
innsbruck_s2 = eopf_catalog.search(
    collections= 'sentinel-2-l2a', # interest Collection,
    bbox=(11.124756, 47.311058, # AOI extent
          11.459839,47.463624),
    datetime='2020-05-01T00:00:00Z/2025-05-31T23:59:59.999999Z' # interest period
```

```
)

combined_ins =list_found_elements(innsbruck_s2)

print("Search Results:")
print('Total Items Found for Sentinel-2 L-2A over Innsbruck:  ',len(combined_ins[0]))
```

```
Search Results:
Total Items Found for Sentinel-2 L-2A over Innsbruck:   27
```

Let us now repeat a combine search for a different collection. Let us define a new AOI for the coastal area of Rostock, Germany and let us search over the **Sentinel-3 SLSTR-L2** collection for the same time period as above. As a result, 14 Items are available for the specified search.

```
rostock_s3 = eopf_catalog.search(
    bbox=(11.766357,53.994566, # AOI extent
          12.332153,54.265086),
    collections= ['sentinel-3-slstr-l2-lst'], # interest Collection
    datetime='2020-05-01T00:00:00Z/2025-05-31T23:59:59.999999Z' # interest period
)

combined_ros=list_found_elements(rostock_s3)

print("Search Results:")
print('Total Items Found for Sentinel-3 SLSTR-L2 over Rostock Coast:  ',len(combined_ros[0]))
```

```
Search Results:
Total Items Found for Sentinel-3 SLSTR-L2 over Rostock Coast:   18
```

9.3.4 Retrieve Asset URLs for accessing the data

So far, we have made a search among the STAC catalog and browsed over the general metadata of the collections. To access the actual EOPF Zarr `Items`, we need to get their storage location in the cloud.

The relevant information we can find inside the `.items` argument by the `.get_assets()` function. Inside, it allows us to specify the `.MediaType` we are interested in. In our example, we want to obtain the location of the `.zarr` file.

Let us retrieve the `url` of the 27 available items over Innsbruck. The resulting URL we can then use to directly access an asset in our workflow.

```
assets_loc=[] # a list with the ids of the items we are interested in
for x in range(len(combined_ins[0])): # We retrieve only the first asset in the Innsbruck lis
    assets_loc.append(S2l2a_coll # we set into the Sentinel-2 L-2A collection
                      .get_item(combined_ins[0][x])  # We only get the Innsbruck filtered ite
                      .get_assets(media_type=MediaType.ZARR)) # we obtain the .zarr location

first_item = assets_loc[0]   # we select the first item from our list

print("Search Results:")
print('URL for accessing',combined_ins[0][0],'item:  ',first_item['product']) # assets_loc[0]
```

```
Search Results:
URL for accessing S2B_MSIL2A_20250530T101559_N0511_R065_T32TPT_20250530T130924 item:   <Asset
```

9.3.5 Retrieve Item metadata

Finally, once you selected an `Item`, you can also explore the relevant metadata on `Item` level. For example with the `keys()` function, you can retrieve the available assets of the selected Item.

```
print('Available Assets: ', list(first_item.keys()))
```

```
Available Assets:  ['SR_10m', 'SR_20m', 'SR_60m', 'AOT_10m', 'B01_20m', 'B02_10m', 'B03_10m'
```

9.4 Now it is your turn

The following expercises will help you master the STAC API and understand how to find the data you need.

9.4.1 Task 1: Explore Your Own Area of Interest

- Go to http://bboxfinder.com/ and select an area of interest (AOI) (e.g. your hometown, a research site, etc.)
- Copy the bounding box coordinates of your area of interest
- Change the provided code above to search for data over your AOI

9.4.2 Task 2: Temporal Analysis

- Compare data availability across different years for the **Sentinel-2 L-2A Collection**.
- Search for items in year 2022
- Repeat the search for year 2024

9.4.3 Task 3: Explore the SAR Mission and combine multiple criteria

- Do the same for a different `Collection` for example the **Sentinel-1 Level-1 GRD**, e.g. you can use the ID `sentinel-1-l1-grd`
- How many assets are availanble for the year 2024?

9.5 Conclusion

This tutorial has provided a clear and practical introduction on how you can programmatically access and search through EOPF Sentinel Zarr Sample Service STAC API.

9.6 What's next?

In the following section, we will explore how to retrieve an Item of our interest, based on several parameters and load the actual data array as `xarray`.

10 From STAC to Data: Accessing EOPF Zarr with xarray

Launch this notebook in JupyterLab

10.0.1 Introduction

In this tutorial we will demonstrate how to access `EOPF Zarr` products directly from the EOPF Sentinel Zarr Sample Service STAC Catalog.

10.0.2 What we will learn

- How to open cloud-optimised datasets from the EOPF Zarr STAC Catalog with `xarray`
- Examine loaded datasets
- Perform simple data analyses with the loaded data

10.0.3 Prerequisites

This tutorial requires the `xarray-eopf` extension for data manipulation. To find out more about the library, access the documentation.

It is advised that you go through the previous section, as it gives you an introduction on how to programmatically access a STAC catalog.

10.0.3.1 Import libraries

```
import requests
import numpy as np
import matplotlib.pyplot as plt
from typing import List, Optional, cast
from pystac import Collection, MediaType
from pystac_client import Client, CollectionClient
```

```
from datetime import datetime
import xarray as xr
```

10.0.3.2 Helper functions

10.0.3.2.1 `list_found_elements`

As we are expecting to visualise several elements that will be stored in lists, we define a function that will allow us retrieve item `id`'s and collections `id`'s for further retrieval.

```
def list_found_elements(search_result):
    id = []
    coll = []
    for item in search_result.items(): #retrieves the result inside the catalogue.
        id.append(item.id)
        coll.append(item.collection_id)
    return id , coll
```

10.1 Establish a connection to the EOPF Zarr STAC Catalog

Our first step is to a connection to the EOPF Zarr STAC Catalog. This involves defining the `url` of the STAC endpoint. See the previous section for a more detailed explanation how to retrieve the end point `url`.

Through the `Client.open()` function, we can establish the connection to the EOPF Zarr STAC catalog by providing the specific `url`.

```
max_description_length = 100

eopf_stac_api_root_endpoint = "https://stac.core.eopf.eodc.eu/" #root starting point
eopf_catalog = Client.open(url=eopf_stac_api_root_endpoint)
# eopf_catalog  #print to have an interative visualisation
```

10.2 Filtering for items of interest

For this tutorial, we will focus on the Sentinel-2 L2A Collection. The EOPF STAC Catalog corresponding `id` is: `sentinel-2-l2a`.

As we are interested in retrieving and exploring an Item from the collection, we will focus again over the Innsbruck area we have defined in the previous tutorial.

```
innsbruck_s2 = eopf_catalog.search( # searching in the Catalog
    collections= 'sentinel-2-l2a', # interest Collection,
    bbox=(11.124756, 47.311058, # AOI extent
          11.459839,47.463624),
    datetime='2020-05-01T00:00:00Z/2025-05-31T23:59:59.999999Z' # interest period
)

combined_ins =list_found_elements(innsbruck_s2)

print("Search Results:")
print('Total Items Found for Sentinel-2 L-2A over Innsbruck:  ',len(combined_ins[0]))
```

```
Search Results:
Total Items Found for Sentinel-2 L-2A over Innsbruck:   27
```

Let us now select the first Item in the list of 27 Items.

```
first_item_id=combined_ins[0][0]
print(first_item_id)
```

```
S2B_MSIL2A_20250530T101559_N0511_R065_T32TPT_20250530T130924
```

In a next step, we retrieve the `url` of the cloud location for the specific item. We will need the `url` to access and load the selected Item with the help of `xarray`.

```
c_sentinel2 = eopf_catalog.get_collection('sentinel-2-l2a')
#Choosing the first item available to be opened:
item= c_sentinel2.get_item(id=first_item_id)
item_assets = item.get_assets(media_type=MediaType.ZARR)

cloud_storage = item_assets['product'].href

print('Item cloud storage URL for retrieval:',cloud_storage)
```

```
Item cloud storage URL for retrieval: https://objects.eodc.eu:443/e05ab01a9d56408d82ac32d69a5
```

10.3 Examining Dataset Structure

In the following step, we open the cloud-optimised Zarr dataset using `xarray.open_datatree` supported by the `xarray-eopf extension`.

The subsequent loop then prints out all the available groups within the opened `DataTree`, providing a comprehensive overview of the hierarchical structure of the EOPF Zarr products.

```
dt = xr.open_datatree(
    cloud_storage,        # the cloud storage url from the Item we are interested in
    engine="eopf-zarr",   # xarray-eopf defined engine
    op_mode="native",     # visualisation mode
    chunks={})            # default eopf chunking size

for dt_group in sorted(dt.groups):
    print("DataTree group {group_name}".format(group_name=dt_group)) # getting the available
```

```
DataTree group /
DataTree group /conditions
DataTree group /conditions/geometry
DataTree group /conditions/mask
DataTree group /conditions/mask/detector_footprint
DataTree group /conditions/mask/detector_footprint/r10m
DataTree group /conditions/mask/detector_footprint/r20m
DataTree group /conditions/mask/detector_footprint/r60m
DataTree group /conditions/mask/l1c_classification
DataTree group /conditions/mask/l1c_classification/r60m
DataTree group /conditions/mask/l2a_classification
DataTree group /conditions/mask/l2a_classification/r20m
DataTree group /conditions/mask/l2a_classification/r60m
DataTree group /conditions/meteorology
DataTree group /conditions/meteorology/cams
DataTree group /conditions/meteorology/ecmwf
DataTree group /measurements
DataTree group /measurements/reflectance
DataTree group /measurements/reflectance/r10m
DataTree group /measurements/reflectance/r20m
DataTree group /measurements/reflectance/r60m
DataTree group /quality
DataTree group /quality/atmosphere
DataTree group /quality/atmosphere/r10m
DataTree group /quality/atmosphere/r20m
```

```
DataTree group /quality/atmosphere/r60m
DataTree group /quality/l2a_quicklook
DataTree group /quality/l2a_quicklook/r10m
DataTree group /quality/l2a_quicklook/r20m
DataTree group /quality/l2a_quicklook/r60m
DataTree group /quality/mask
DataTree group /quality/mask/r10m
DataTree group /quality/mask/r20m
DataTree group /quality/mask/r60m
DataTree group /quality/probability
DataTree group /quality/probability/r20m
```

10.4 Root Dataset Metadata

We specifically look for groups containing data variables under `/measurements/reflectance/r20m` (which corresponds to Sentinel-2 bands at 20m resolution). The output provides key information about the selected group, including its dimensions, available data variables (the different spectral bands), and coordinates.

```
# Get /measurements/reflectance/r20m group
groups = list(dt.groups)
interesting_groups = [
    group for group in groups if group.startswith('/measurements/reflectance/r20m')
    and dt[group].ds.data_vars
]
print(f"\n  Searching for groups with data variables in '/measurements/reflectance/r20m'...")
```

```
  Searching for groups with data variables in '/measurements/reflectance/r20m'...
```

```
if interesting_groups:
    sample_group = interesting_groups[0]
    group_ds = dt[sample_group].ds

    print(f"Group '{sample_group}' Information")
    print("=" * 50)
    print(f"Dimensions: {dict(group_ds.dims)}")
    print(f"Data Variables: {list(group_ds.data_vars.keys())}")
    print(f"Coordinates: {list(group_ds.coords.keys())}")
```

```
else:
    print("No groups with data variables found in the first 5 groups.")
```

```
Group '/measurements/reflectance/r20m' Information
================================================
Dimensions: {'y': 5490, 'x': 5490}
Data Variables: ['b01', 'b02', 'b03', 'b04', 'b05', 'b06', 'b07', 'b11', 'b12', 'b8a']
Coordinates: ['x', 'y']
```

```
/tmp/ipykernel_307671/1451209294.py:7: FutureWarning: The return type of `Dataset.dims` will
  print(f"Dimensions: {dict(group_ds.dims)}")
```

In a next step, we inspect the attributes of the root dataset within the `DataTree`. Attributes often contain important high-level metadata about the entire product, such as processing details, STAC discovery information, and more. We print the first few attributes to get an idea of the available metadata.

```
# Examine the root dataset
root_dataset = dt.ds

print("Root Dataset Metadata")

if root_dataset.attrs:
    print(f"\nAttributes (first 3):")
    for key, value in list(root_dataset.attrs.items())[:3]:
        print(f"   {key}: {str(value)[:80]}{'...' if len(str(value)) > 80 else ''}")
```

```
Root Dataset Metadata

Attributes (first 3):
   other_metadata: {'AOT_retrieval_model': 'SEN2COR_DDV', 'L0_ancillary_data_quality': '4',
   stac_discovery: {'assets': {'analytic': {'eo:bands': [{'center_wavelength': 0.4423, 'comm
```

10.5 Visualising the RGB quicklook composite

EOPF Zarr Assets include a quick-look RGB composite, which we now want to open and visuliase. We open the Zarr dataset again, but this time, we specifically target the `quality/l2a_quicklook/r20m group` and its variables.

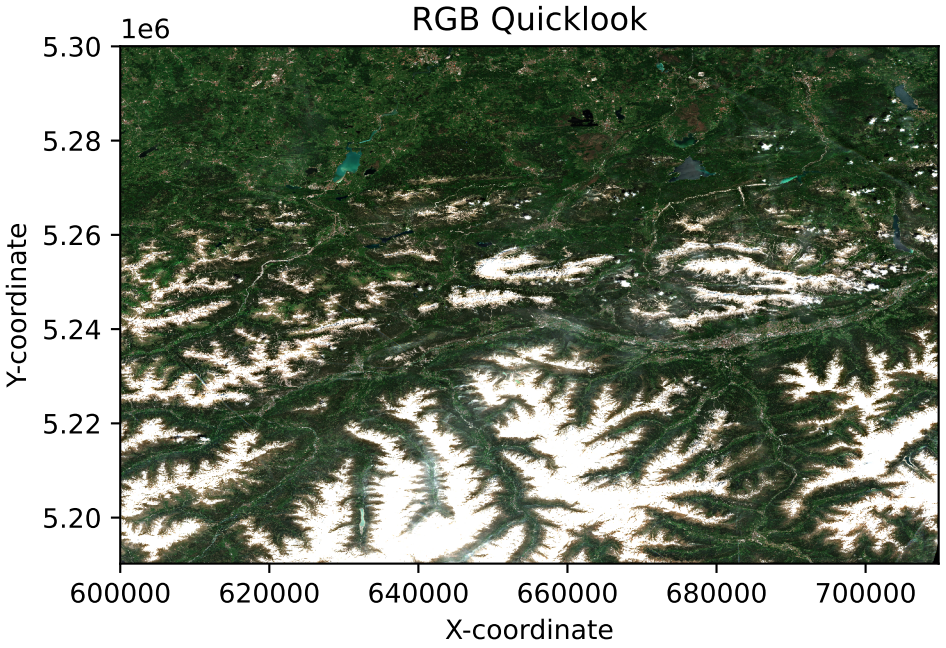
This group typically contains a true colour (RGB) quick-look composite, which is a readily viewable representation of the satellite image.

We use `xr.open_dataset()` and specify the following set of arguments in order to load the quick-look.

```
## Visualising the RGB quicklook composite:
ds = xr.open_dataset(
    cloud_storage,        # the cloud storage url from the Item we are interested in
    engine="eopf-zarr",   # xarray-eopf defined engine
    op_mode="native",     # visualisation mode
    chunks={},            # default eopf chunking size
    group_sep="/",
    variables="quality/l2a_quicklook/r20m/*",
)
```

As soon as we loaded it, we can create a simple plot with `imshow()` to see the quick-look.

```
ds["quality/l2a_quicklook/r20m/tci"].plot.imshow()
plt.title('RGB Quicklook')
plt.xlabel('X-coordinate')
plt.ylabel('Y-coordinate')
plt.grid(False) # Turn off grid for image plots
plt.axis('tight') # Ensure axes fit the data tightly
```



10.6 Simple Data Analysis: Calculating NDVI

Let us now do a simple analysis with the data from the EOPF Zarr STAC Catalog. Let us calculate the Normalized Difference Vegetation Index (NDVI).

First, we open the `.zarr` dataset specifically for the **red** (B04) and **Near-Infrared** (B08) bands, which are crucial for the calculation of the NDVI. We also specify `resolution=20` to ensure we are working with the 20-meter resolution bands.

```
red_nir = xr.open_dataset(
    cloud_storage,
    engine="eopf-zarr",
    chunks={},
    spline_orders=0,
    variables=['b04', 'b08'],
    resolution= 60,
)
```

In a next step, we cast the red (B04) and Near-Infrared (B08) bands to floating-point numbers. This is important for accurate mathematical operations, which we will conduct in the next cell.

```
red_f = red_nir.b04.astype(float)
nir_f = red_nir.b08.astype(float)
```

Now, we perform the initial steps for **NDVI** calculation: - `sum_bands`: Calculates the sum of the Near-Infrared and Red bands. - `diff_bands`: Calculates the difference between the Near-Infrared and Red bands.

```
sum_bands = nir_f + red_f
diff_bands = nir_f - red_f
ndvi = diff_bands / sum_bands
```

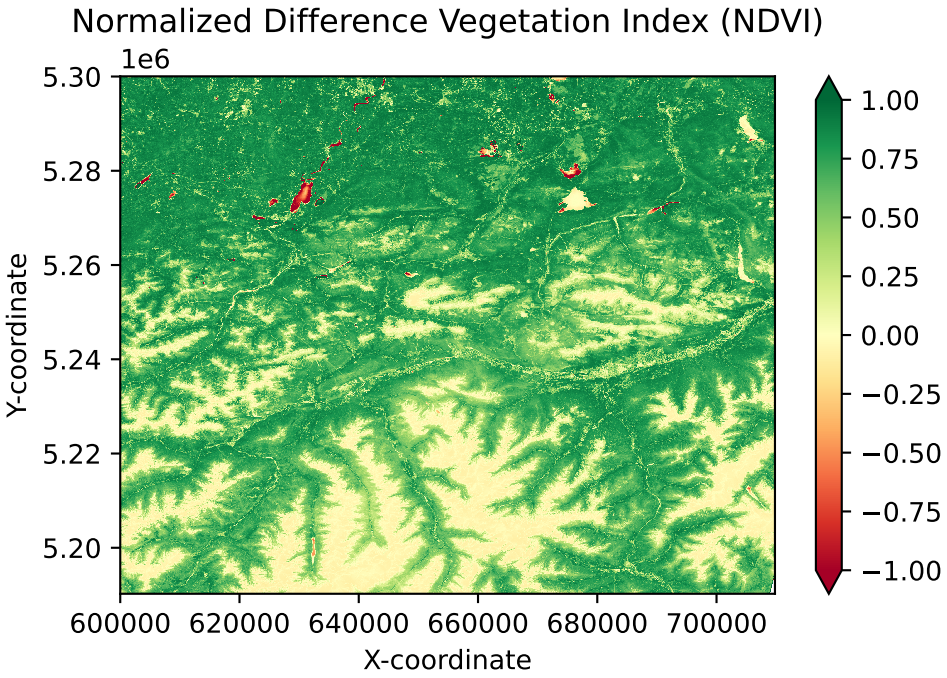
To prevent division by zero errors in areas where both red and NIR bands might be zero (e.g., water bodies or clouds), this line replaces any **NaN** values resulting from division by zero with the 0 value. This ensures a clean and robust NDVI product.

```
ndvi = ndvi.where(sum_bands != 0, 0)
```

In a final step, we can visualise the calculated NDVI.

```
ndvi.plot(cmap='RdYlGn', vmin=-1, vmax=1)
plt.title('Normalized Difference Vegetation Index (NDVI)')
plt.xlabel('X-coordinate')
plt.ylabel('Y-coordinate')
plt.grid(False) # Turn off grid for image plots
plt.axis('tight') # Ensure axes fit the data tightly

# Display the plot
plt.show()
```



10.7 Now it is your turn

With the foundations learned so far, you are now equipped to access products from the EOPF Zarr STAC catalog. These are your tasks:

10.7.1 Task 1: Explore five additional Sentinel-2 Items for Innsbruck

Replicate the RGB quick-look and have an overview of the spatial changes.

10.7.2 Task 2: Calculate NDVI

Replicate the NDVI calculation for the additional Innsbruck items.

10.7.3 Task 3: Applying more advanced analysis techniques

The EOPF STAC Catalog offers a wealth of data beyond Sentinel-2. Replicate the search and data access for data from other collections.

10.8 Conclusion

In this section we established a connection to the EOPF Sentinel Zarr Sample Service STAC Catalog and directly accessed an EOPF Zarr item with `xarray`. In the tutorial you are guided through the process of opening hierarchical EOPF Zarr products using `xarray`'s `DataTree`, a library designed for accessing complex hierarchical data structures.

10.9 What's next?

This online resource is under active development. So stay tuned for regular updates.

Part IV

[COMING SOON] Tools to work with EOPF Zarr

Part V

[COMING SOON] EOPF Zarr in Action

11 Glossary

Here we introduce some helpful terms that are mentioned throughout the EOPF 101.

Acronym	Definition
EOPF	Earth Observation Processing Framework
CDSE	Copernicus Data Space Ecosystem
CMP	Core Python Modules
HEALPix	Hierarchical Equal Area isoLatitude Pixelation
GDR	Ground Range Detected
SLC	Single Look Complex
NRB	Normalized Radar Backscatter
SAFE	Standard Archive Format for Europe
STAC	Spatio Temporal Asset Catalog
COG	Cloud Optimised GeoTIFF
Zarr	Cloud-optimised version for `netCDF` and `HDF5` formats, specifically designed for storing and accessing large n-dimensional arrays

12 References

The table below provides a categorized overview of key resources used during the development of the EOPF 101 book.

Category	Resource Title	Description & Context	Link
Sentinel Mission & Product Specifications	Sentinel-1 Level 1 Product Specification	Detailed specifications for Sentinel-1 Level 1 products.	Available here
	Sentinel-2 - MSI - Level 2A Products	Auxiliary and data product specifications for Sentinel-2 MSI Level 2A products.	Available here
	Sentinel-3 – Documentation (via CDSE)	Comprehensive documentation for Sentinel-3 data products and mission.	Available here
EOPF Platform & Modules	Supported Products Formats — EOPF - Core Python Modules	Guide to the product formats supported by the Earth Observation Processing Framework's (EOPF) Core Python Modules.	Available here

Category	Resource Title	Description & Context	Link
	EOPF Sentinel Zarr Samples Service STAC Catalog	A STAC (Spatio Temporal Asset Catalog) for accessing Sentinel Zarr samples hosted by EOPF.	Available here
	EOPF Sentinel Zarr Samples	Direct access to Sentinel data samples stored in Zarr format within the EOPF ecosystem.	Available here
Data Formats & Standards	Zarr Documentation	Official documentation for the Zarr format, a cloud-optimised standard for n-dimensional arrays.	Available here
	Introduction to the Zarr Format | Copernicus Marine Help Center	An introductory guide to the Zarr format, provided by the Copernicus Marine Help Center.	Available here
	Zarr + STAC	Article discussing the integration and benefits of combining Zarr data with STAC catalogs.	Available here
	Is Zarr the new COG?	An insightful discussion comparing Zarr with Cloud Optimised GeoTIFF (COG) for cloud-native geospatial data.	Available here
	About STAC	General information and principles behind the Spatio Temporal Asset Catalog (STAC) specification.	Available here

Category	Resource Title	Description & Context	Link
Cloud-Native Geospatial Initiatives	Cloud-Native Geospatial Forum (CNG)	The official website for the Cloud-Native Geospatial Forum, promoting cloud-native approaches in geospatial.	Available here