D3.3 End user documentation for batch processing system

Status:	Under EC Review	**Planned due date:**	31/03/2022
Version:	1.0	**Submission date:**	31/03/2022
Lead Participant:	VITO	**Lead Author:**	Hande Erdem
Related WP:	WP3	**Document Ref:**	D3.3
Dissemination Level:	Public (PU)		
Document Link:	https://doi.org/10.5281/zenodo.6403077		

Deliverable Abstract

This document aims to provide information to the end users on the C-SCALE Batch Processing system, which is deployed and running on the C-SCALE Compute Federation.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017529.

COPYRIGHT NOTICE

C-SCALE receives funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 101017529.

DELIVERY SLIP

Date	Name	Partner/Activity
Lead Author:	Hande Erdem (HE)	VITO
Contributors:	Jeroen Dries (JD)	VITO
Moderated by:	Charis Chatzikyriakou (CC)	EODC
Reviewed by:	Sebastian Luna-Valero (SLV)	EGI Foundation
	Zacarias Benta (ZB)	INCD
Approved by:	C-SCALE Activity Management Board (AMB): Christian Briese (CB), Diego Scardaci (DS), Charis Chatzikyriakou (CC), Zdeněk Šustr (ZS), Enol Fernández (EF), Björn Backeberg (BB), Eleonora Testa (ET)	EODC, CESNET, EGI Foundation, Deltares

DOCUMENT LOG

Issue	Date	Comment	Author(s)
V0.1	14/03/2022	First draft for review	HE, JD
V0.2	23/03/2022	Addressed the comments from the reviewers	HE, SLV, ZB
V0.3	28/03/2022	Created second draft	HE
V0.4	31/03/2022	Addressed the comments from the AMB	HE, CC
V1.0	31/03/2022	Final version approved by AMB	C-SCALE AMB

Table of Contents

List of Images

List of Acronyms

Acronym	Description
AAI	Authentication and Authorisation Infrastructure
API	Application Programming Interface
CO	Collaborative Organisation
C-SCALE	Copernicus - eoSC AnaLytics Engine
DIAS	Data and Information Access Services
EO	Earth Observation
EODC	Earth Observation Data Centre (C-SCALE provider)
EOSC	European Open Science Cloud
GPU	Graphics Processing Unit
GRNET	Greek Research and Technology Network (C-SCALE provider)
GUI	Graphical User Interface
HPC	High Performance Computing
HTC	High Throughput Computing
HTDP	High Throughput Data Processing
IaaS	Infrastructure as a Service
INCD-NCG	Infrastructura Nacional de Computação Distribuída-National Computing Grid (C-SCALE provider)
IDE	Integrated Development Environment
OLA	Operational Level Agreement
PBS	Portable Batch System
PaaS	Platform as a Service
REST	Representational State Transfer
SLA	Service Level Agreement
SRAM	SURF Research Access Management
STAC	SpatioTemporal Asset Catalog
TB	Terabyte
TRL	Technology Readiness Level

VA	Virtual Access
VM	Virtual Machine
WP	Work Package
M2M	Machine to Machine

Executive Summary

The C-SCALE (Copernicus – eoSC AnaLytics Engine) project aims to federate existing European Earth Observation (EO) service providers, cloud resources and computing centres to empower the European EO research community to more easily discover, access, process, analyse and share Copernicus data, tools, resources and services. The C-SCALE data and compute federation will ensure interoperability between distributed data catalogues, computational tooling, and infrastructure, and will thereby increase the service offer of the EOSC Portal providing state-of-the-art research enabling services to its users.

This document aims to provide information to the end users on the C-SCALE Batch Processing system, which is deployed and running on the C-SCALE Compute Federation. In the later stages of the project, the information intended for end users will be available as an online resource so that it can be updated, maintained, and accessed in an easier and more user friendly manner.

After the introduction, Section 2 of the document focuses on the openEO[1], which is offered as the standard batch processing environment for the C-SCALE project. In addition, Section 3 provides a brief overview of the Interactive Analysis Environments – also known as Jupyter Notebooks – that can be used in combination with the batch processing system within the C-SCALE Compute Federation.

[1] https://openeo.cloud/

1 Introduction

C-SCALE Batch Processing and Interactive Analysis Environments provide users with necessary tools to process Earth Observation (EO) data. The default batch processing environment in C-SCALE is openEO. openEO offers a unified API and standard functions as well as data cubes to make it easy to work with EO data. Using openEO, end users can take advantage of the out of the box functions and data sets. openEO can be used within Jupyter Notebooks, within an Integrated Development Environment (IDE) through openEO client libraries (e.g., Python client), for M2M integration through REST API or directly through the openEO Web Editor (also known as openEO Platform Editor).

Jupyter Notebooks provide users with a web environment, on which they can directly work with the satellite data using their preferred programming language. They can import the necessary libraries to work with EO data and can take advantage of APIs like openEO to use out-of-the-box functions and distributed processing capabilities. Moreover, Jupyter Notebooks allow users to combine code with other media types (text, images, interactive maps and fields), which makes it suitable for knowledge sharing and classroom situations.

The information provided to the end user is based on the state of the system when this document is written. Users are encouraged to find and access to the most up-to-date documentation on the C-SCALE Documentation Portal: https://wiki.c-scale.eu/C-SCALE.

2 Batch Processing Environment

Earth Observation research often involves processing huge amounts of data, therefore processing and generating value on it is challenging and presents a high entry barrier for the value creators. With C-SCALE Batch Processing Environment we are offering openEO to simplify this aspect.

The data in an openEO service is exposed as a 'data cube', irrespective of how the data is stored internally. As a result, users of C-SCALE Batch Processing Environment will no longer need to deal with individual files, formats, and EO product catalogues, therefore making the user's life easier.

openEO is easy to use, while hiding the complex technical details of distributed cloud processing of C-SCALE Compute Federation. Users can trace how results are generated, so that their work is reproducible. Well-documented interfaces make it easy to integrate openEO in any application. Users can share their processes and algorithms as a service using openEO.

openEO offers a number of predefined functions, which the backend has to support (e.g., computing an NDVI from spectral bands). This ensures that users can easily use different C-SCALE service providers or can easily compare results.

Next to the predefined functions, openEO also offers user-defined functions. They allow the C-SCALE users to reuse the wealth of algorithms that already exist, for instance, those that are available as open source in the Python and R communities. User-defined functions are sent to the C-SCALE Batch Processing backend in the form of a simple script. For instance, a Python script that loads a Machine Learning model to classify pixels. Then, the function is executed on the backend. This capability and its ease of use is truly unique for openEO, and it will increase the uptake of cloud-based processing even more.

The openEO specification, which is utilised by C-SCALE Compute Federation, is built by an open community of various research institutions and companies. Concurrently, multiple open source client and backend implementations are developed to ensure that the standard is validated in realistic scenarios. This approach makes the specification sustainable, as it can continue to evolve freely as long as there are interested parties.

The open source nature of the implementations ensures that results are reproducible and traceable. The implementation of the various processing functions can be verified and improved independently, which is important in a world where information derived from EO data is used more and more to drive policies and decisions.

2.1 openEO Backends in C-SCALE Compute Federation

Following are the openEO Backends that are currently available in C-SCALE:

- EODC
- CreoDIAS
- Terrascope/VITO
- INCD

More backends are being explored and added as the project progresses. This information will be available on the C-SCALE Documentation portal.

2.2 openEO End User Documentation

The User Manual contains all essential information for the C-SCALE user to make full use of the openEO platform. This manual is published and maintained on the openEO website, so users can easily find required descriptions on the used tools.

The manual consists of specific parts:

- General info on openEO usage
- Specific info on connecting to the openEO platform
- Code examples

We target researchers working in Python in this manual. The platform also supports JavaScript, and an R client exists, but this is not considered the main target audience for this manual.

2.2.1 Getting Started with C-SCALE Batch Processing Environment

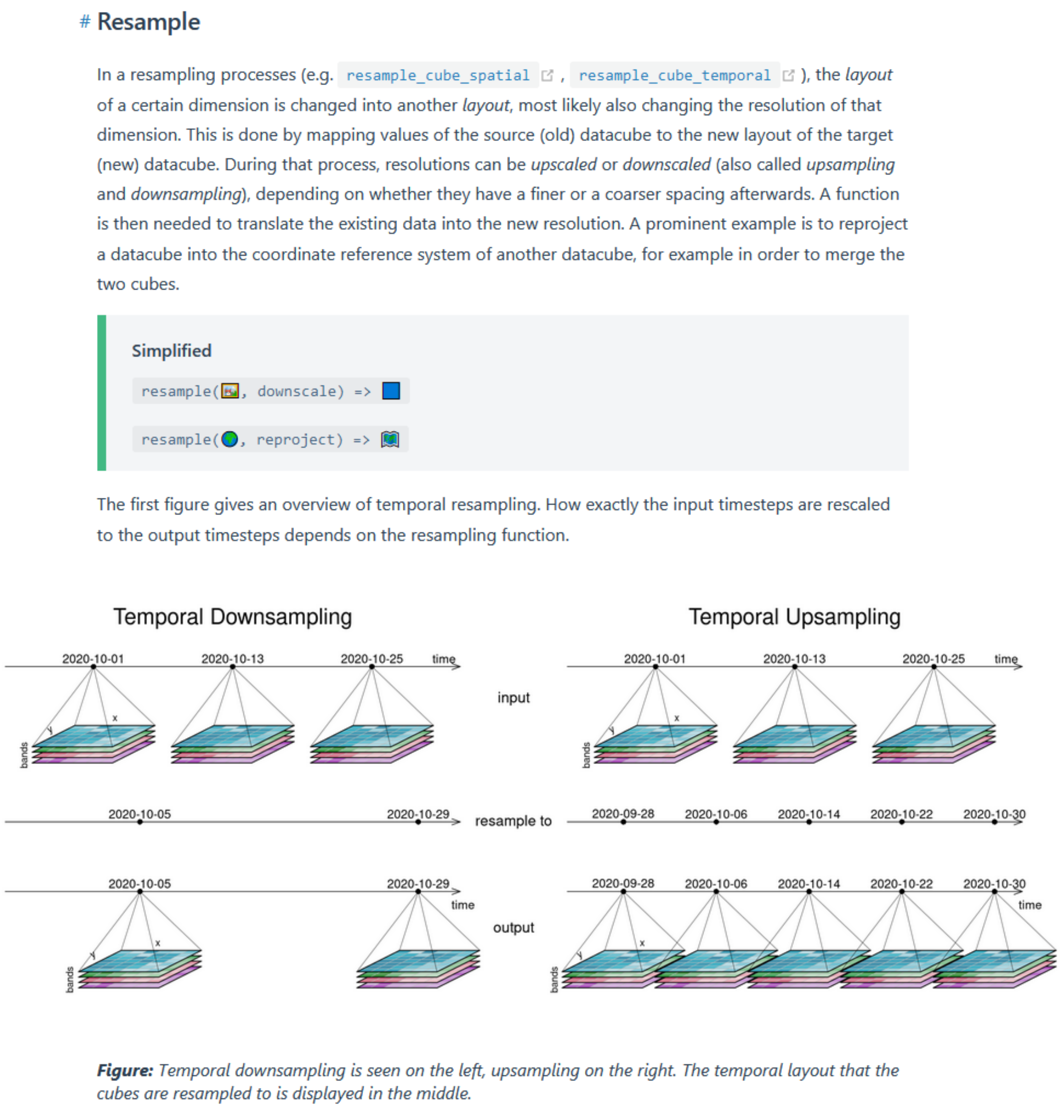
Below is the list of web pages to start with all the different tools inside openEO:

- Data Cubes (https://openeo.org/documentation/1.0/datacubes.html)
  - What are Datacubes?
  - Dimensions
  - Processes on Datacubes
- openEO Web Editor (https://docs.openeo.cloud/getting-started/editor/)
  - Explanation of the Web Editor (also known as Platform Editor)
- Java Script Client (https://openeo.org/documentation/1.0/javascript/)
  - Installation
  - Connection
  - Collection
  - Processes
- Python Client (https://openeo.org/documentation/1.0/python/)
  - Installation
  - Processes
- “R” Client (https://openeo.org/documentation/1.0/r/)
  - Installation
  - Authentication

2.2.2 Data Cube Documentation

The Data Cubes Description Web page covers the definition of the Data Cube, the dimensions, and processes. A more detailed description can be found online under the following domain - https://openeo.org/documentation/1.0/datacubes.html#what-are-datacubes

Figure 1 describes the data cube resampling documentation.



Resample

In a resampling processes (e.g. `resample_cube_spatial`, `resample_cube_temporal`), the *layout* of a certain dimension is changed into another *layout*, most likely also changing the resolution of that dimension. This is done by mapping values of the source (old) datacube to the new layout of the target (new) datacube. During that process, resolutions can be *upscaled* or *downscaled* (also called *upsampling* and *downsampling*), depending on whether they have a finer or a coarser spacing afterwards. A function is then needed to translate the existing data into the new resolution. A prominent example is to reproject a datacube into the coordinate reference system of another datacube, for example in order to merge the two cubes.

Simplified

`resample(🖼, downscale) => 🟦`

`resample(🌐, reproject) => 🗺`

The first figure gives an overview of temporal resampling. How exactly the input timesteps are rescaled to the output timesteps depends on the resampling function.

Figure: *Temporal downsampling is seen on the left, upsampling on the right. The temporal layout that the cubes are resampled to is displayed in the middle.*

Figure 1: Data cube resampling documentation

2.2.3 Web Editor Documentation

The openEO Web Editor (also called the Platform Editor) within C-SCALE Batch Processing Environment is a browser-based Graphical User Interface (GUI) for openEO. It allows to use the openEO services without any coding experience. You can explore the service offerings such as data collections and processes, but also create and run custom processes on our infrastructure and then visualize the results. Result visualization is still a bit limited, but all other features of the platform are supported.

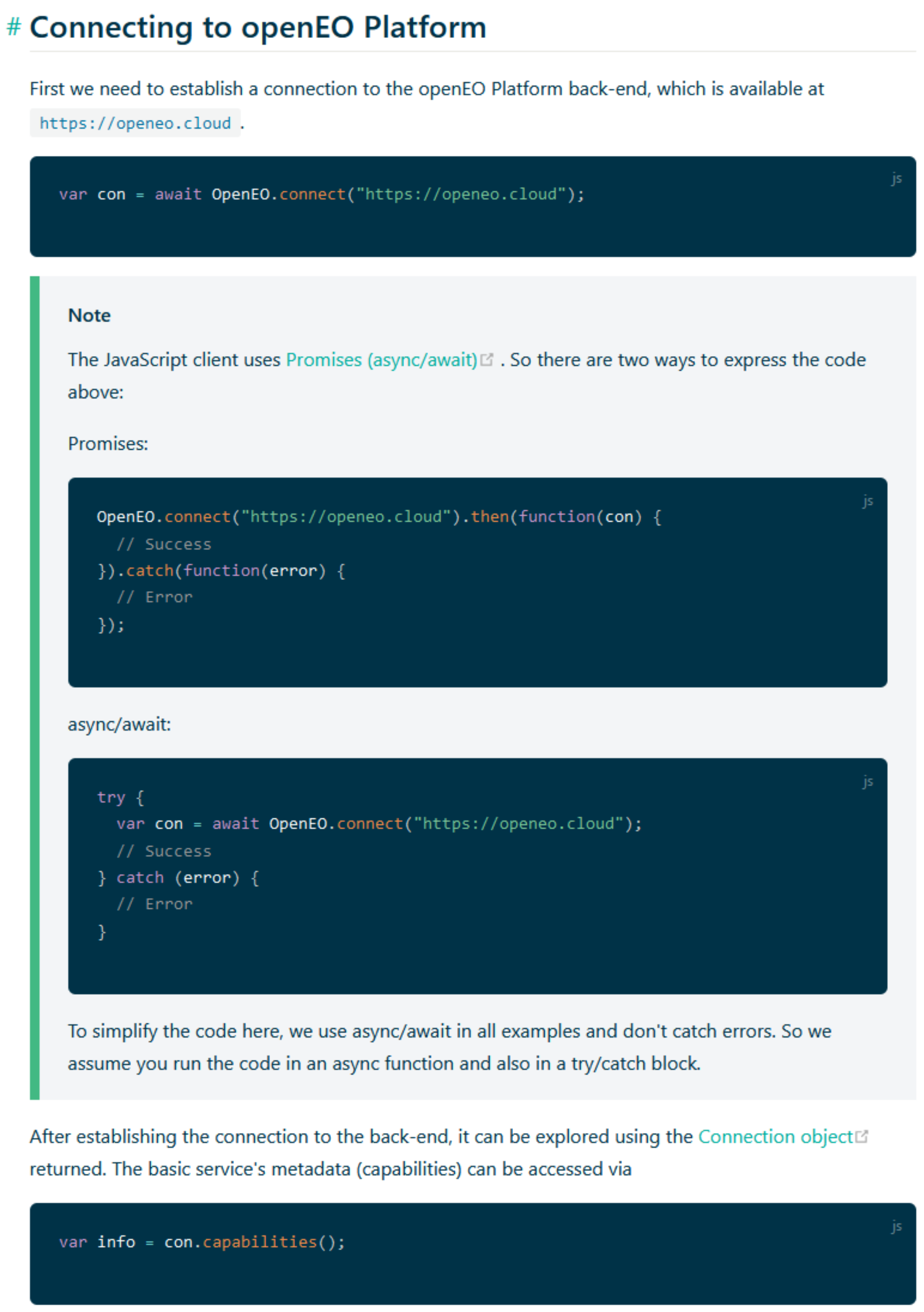
2.2.4 JavaScript Client Documentation

The openEO JavaScript Client can be used in all modern browsers (excludes Internet Explorer) and all maintained Node.js versions (>= 10.x). It can also be used for mobile app development with the Ionic Framework for example.

The easiest way to try out the client is using one of the examples. Alternatively, you can create an HTML. Afterwards you can load the library. Depending on whether you are directly working in Node.js or are just using a Node.js build tool, the import can be different. Please inform yourself which import is suited for your project.

When the installation is successfully finished, you can now connect to openEO compliant back-ends.

If you have trouble installing the client, feel encouraged to leave an issue at the GitHub project.



Connecting to openEO Platform

First we need to establish a connection to the openEO Platform back-end, which is available at `https://openeo.cloud` .

```js
var con = await OpenEO.connect("https://openeo.cloud");
```

Note

The JavaScript client uses Promises (async/await) . So there are two ways to express the code above:

Promises:

```js
OpenEO.connect("https://openeo.cloud").then(function(con) {
  // Success
}).catch(function(error) {
  // Error
});
```

async/await:

```js
try {
  var con = await OpenEO.connect("https://openeo.cloud");
  // Success
} catch (error) {
  // Error
}
```

To simplify the code here, we use async/await in all examples and don't catch errors. So we assume you run the code in an async function and also in a try/catch block.

After establishing the connection to the back-end, it can be explored using the Connection object returned. The basic service's metadata (capabilities) can be accessed via

```js
var info = con.capabilities();
```

2.2.5 Python Client Documentation

2.2.5.1 Installation

The openEO Python client library, used by C-SCALE Batch Processing Environment, is available on PyPI and can easily be installed with a tool like `pip`, for example:

```
pip install openeo
```

It's recommended to work in a virtual environment of some kind (`venv`, `conda`, etc.), containing Python 3.6 or higher.

For more details, alternative installation procedures or troubleshooting tips: see the official `openeo` package installation documentation.

2.2.5.2 Exploring a back-end

If you do not know an openEO back-end that you want to connect to yet, you can have a look at the list of available openEO Backends in C-SCALE Compute Federation, to find all known back-ends with information on their capabilities.

For this tutorial we will use the openEO instance of Terrascope, which is available at `https://openeo.vito.be`. Note that the code snippets in this guide work the same way for the other back-ends available in C-SCALE. Just the collection identifier and band names might differ.

First, we need to establish a connection to the back-end.

```
import openeo

connection = openeo.connect("https://openeo.vito.be")
```

The `Connection` object is your central gateway to - list data collections, available processes, file formats and other capabilities of the back-end - start building your openEO algorithm from the desired data on the back-end - execute and monitor (batch) jobs on the back-end - etc.

2.2.5.2.1 Collections

The EO data available at a back-end is organised in so-called collections. For example, a back-end might provide fundamental satellite collections like "Sentinel-1" or "Sentinel-2", or preprocessed collections like "NDVI". Collections are used as input data for your openEO jobs. More information on how openEO "collections" relate to terminology used in other systems can be found in (the openEO glossary).

Let's list all available collections on the back-end, using `list_collections`:

```
print(connection.list_collections())
```

which returns a list of collection metadata dictionaries, e.g., something like:

```
[{'id': 'AGERA5', 'title': 'ECMWF AGERA5 meteo dataset', 'description': 'Daily surface me
teorolociga datal ...', ...},
 {'id': 'SENTINEL2_L2A_SENTINELHUB', 'title': 'Sentinel-2 top of canopy', ...},
 {'id': 'SENTINEL1_GRD', ...},
 ...]
```

This listing includes basic metadata for each collection. If necessary, a more detailed metadata listing for a given collection can be obtained with `describe_collection`.

Programmatically listing collections is just a very simple usage example of the Python client. In reality, you probably want to look up or inspect available collections in a web based overview such as the openEO Hub.

2.2.5.2.2 Processes

Processes in openEO are operations that can be applied on EO data (e.g., calculate the mean of an array, or mask out observations outside a given polygon). The output of one process can be used as the input of another process, and by doing so, multiple processes can be connected that way in a larger "process graph": a new (user-defined) processes that implements a certain algorithm can be implemented following this logic. Please check the openEO glossary for more details on pre-defined, user-defined processes and process graphs.

Let's list the (pre-defined) processes available on the back-end with `list_processes`:

```
print(connection.list_processes())
```

which returns a list of Python dictionaries describing the processes (including expected arguments and return type), e.g.:

```
[{'id': 'absolute', 'summary': 'Absolute value', 'description': 'Computes the absolute va
lue of a real number `x`, which is th...',
 {'id': 'mean', 'summary': 'Arithmetic mean(average)', ...}
 ...]
```

Like with collections, instead of programmatic exploration you'll probably prefer a web-based overview such as the openEO Hub for back-end specific process descriptions or browse the reference specifications of openEO processes.

2.2.5.3 Authentication

In the code snippets above we did not need to log in since we just queried publicly available back-end information. However, to run non-trivial processing queries one has to authenticate so that permissions, resource usage, etc. can be managed properly.

Even though openEO supports Basic Authentication, within C-SCALE federation, only OpenID Connect Authentication method is allowed.

A detailed description of why and how to use the authentication methods is on the official documentation.

2.2.5.3.1 Basic Authentication

The Basic Authentication method is a common way of authenticating HTTP requests given username and password.

The preferred authentication method is OpenID Connect due to better security mechanisms implemented in the OpenID Connect protocol. If possible, use OpenID Connect instead of HTTP Basic authentication.

The following code snippet shows how to log in via Basic Authentication:

```
print("Authenticate with Basic authentication")
connection.authenticate_basic("username", "password")
```

After successfully calling the `authenticate_basic` method, you are logged into the back-end with your account.

This means that every call that comes after that via the connection variable is executed by your user account.

2.2.5.3.2 OpenID Connect Authentication

The OIDC (OpenID Connect) Authentication can be used to authenticate via an external service given a client ID. For C-SCALE Batch Processing Environment, this means authenticating through EGI Check-in service.

The following code snippet shows how to log in via OIDC authentication:

```
print("Authenticate with OIDC authentication")
connection.authenticate_OIDC("egi")
```

Calling this method opens your system web browser, with which you can authenticate yourself on the back-end authentication system. After that the website will give you the instructions to go back to the python client, where your connection has logged your account in. This means that every call that comes after that via the connection variable is executed by your user account.

2.2.5.4 Working with Datacubes

Now that we know how to discover the capabilities of the back-end and how to authenticate, let's do some real work and process some EO data in a batch job. We'll build the desired algorithm by working on so-called "Datacubes", which is the central concept in openEO to represent EO data, as discussed in great detail here.

2.2.5.4.1 Creating a Datacube

The first step is loading the desired slice of a data collection with `Connection.load_collection`:

```
datacube = connection.load_collection(
  "SENTINEL1_GRD",
  spatial_extent={"west": 16.06, "south": 48.06, "east": 16.65, "north": 48.35},
  temporal_extent=["2017-03-01", "2017-04-01"],
  bands=["VV", "VH"]
)
```

This results in a `Datacube` object containing the "SENTINEL1_GRD" data restricted to the given spatial extent, the given temporal extend and the given bands .

You can also filter the datacube step by step or at a later stage by using the following filter methods:

```
datacube = datacube.filter_bbox(west=16.06, south=48.06, east=16.65, north=48.35)
datacube = datacube.filter_temporal(start_date="2017-03-01", end_date="2017-04-01")
datacube = datacube.filter_bands(["VV", "VH"])
```

Still, it is recommended to always use the filters directly in load_collection to avoid loading too much data upfront.

2.2.5.4.2 Applying processes

By applying an openEO process on a datacube, we create a new datacube object that represents the manipulated data. The standard way to do this with the Python client is to call the appropriate `Datacube` object method. The most common or popular openEO processes have a dedicated `Datacube` method (e.g., `mask`, `aggregate_spatial`, `filter_bbox`, …). Other processes without a dedicated method can still be applied in a generic way. On top of that, there are also some convenience methods that implement openEO processes in a compact, Pythonic interface. For example, the `min_time` method implements a `reduce_dimension` process along the temporal dimension, using the `min` process as reducer function:

```
datacube = datacube.min_time()
```

This creates a new datacube (we overwrite the existing variable), where the time dimension is eliminated and for each pixel, we just have the minimum value of the corresponding timeseries in the original datacube.

See the Python client `Datacube` API for a more complete listing of methods that implement openEO processes.

openEO processes that are not supported by a dedicated `Datacube` method can be applied in a generic way with the `process` method, e.g.:

```
datacube = datacube.process(
    process_id="ndvi",
    arguments={
        "data": datacube,
        "nir": "B8",
        "red": "B4"}
)
```

This applies the `ndvi` process to the datacube with the arguments of “data”, “nir” and “red” (This example assumes a datacube with bands B8 and B4).

2.2.5.4.3 Defining output format

After applying all processes, you want to execute, we need to tell the back-end to export the datacube, for example as GeoTiff:

```
result = datacube.save_result("GTiff")
```

2.2.5.5 Execution

It’s important to note that all the datacube processes we applied up to this point are not actually executed yet, neither locally nor remotely on the back-end. We just built an abstract representation of the algorithm (input data and processing chain), encapsulated in a local `Datacube` object (e.g., the `result` variable above). To trigger an actual execution (on the C-SCALE back-end) we have to explicitly send this representation to the back-end.

openEO defines several processing modes, but for this introduction we’ll focus on batch jobs, which is a good default choice.

2.2.5.5.1 Batch job execution

The `result` datacube object we built above describes the desired input collections, processing steps and output format. We can now just send this description to the C-SCALE back-end to create a batch job with the `send_job` method like this:

```
# Creating a new job at the back-end by sending the datacube information.
job = result.send_job()
```

The batch job, which is referenced by the returned job object, is just created at the back-end, it is not started yet. To start the job and let your Python script wait until the job has finished then download it automatically, you can use the `start_and_wait` method.

```
# Starts the job and waits until it finished to download the result.
job.start_and_wait()
job.get_results().download_files("output")
```

When everything completes successfully, the processing result will be downloaded as a GeoTIFF file in a folder “output”.

The official openEO Python Client documentation has more information on batch job basics or more detailed batch job (result) management.

2.2.5.6 Full Example

In this chapter we will show a full example of an EO use case using the openEO Python client and the Terrascope back-end within C-SCALE Compute Federation.

We want to produce a monthly RGB composite of Sentinel-1 backscatter data over the area of Vienna, Austria for three months in 2017. This can be used for classification and crop monitoring.

In the following code example, we use inline code comments to describe what we are doing.

```
import openeo


# First, we connect to the back-end and authenticate ourselves via OIDC authentication.
con = openeo.connect("https://openeo.vito.be").authenticate_oidc("egi")

# Now that we are connected, we can initialize our datacube object with the area around Vienna
# and the time range of interest using Sentinel 1 data.
datacube = con.load_collection("SENTINEL1_GRD",
                      spatial_extent={"west": 16.06, "south": 48.06, "east": 16.65, "north": 48.35},
                      temporal_extent=["2017-03-01", "2017-06-01"],
                      bands=["VV"])

# Since we are creating a monthly RGB composite, we need three (R, G and B) separated time ranges.
# Therefore, we split the datacube into three datacubes by filtering temporal for
# March, April and May.
march = datacube.filter_temporal("2017-03-01", "2017-04-01")
april = datacube.filter_temporal("2017-04-01", "2017-05-01")
may = datacube.filter_temporal("2017-05-01", "2017-06-01")

# Now that we split it into the correct time range, we have to aggregate the timeseries values into
# a single image. Therefore, we make use of the Python Client function `mean_time`, which reduces th
e # time dimension, by taking for every timeseries the mean value.

mean_march = march.mean_time()
mean_april = april.mean_time()
mean_may = may.mean_time()

# Now the three images will be combined into the temporal composite.
# Before merging them into one datacube, we need to rename the bands of the images, because
# otherwise, they would be overwritten in the merging process.
# Therefore, we rename the bands of the datacubes using the `rename_labels` process to "R", "G" and
# "B". After that we merge them into the "RGB" datacube, which has now three bands ("R","G" and "B")

R_band = mean_march.rename_labels(dimension="bands", target=["R"])
G_band = mean_april.rename_labels(dimension="bands", target=["G"])
B_band = mean_may.rename_labels(dimension="bands", target=["B"])

RG = R_band.merge_cubes(G_band)
RGB = RG.merge_cubes(B_band)


# Last but not least, we add the process to save the result of the processing. There we define that
# the result should be a GeoTiff file.
# We also set, which band should be used for "red", "green" and "blue" colour in the options.
# With the last process we have finished the datacube definition and can create and start the job at
# the back-end.

job = RGB.execute_batch(out_format="GTiff")
results = job.get_results()
results.get_metadata()
```

Now the resulting GTiff file of the RGB backscatter composite can be retrieved through the link that is provided in the result metadata.

2.2.5.7 User Defined Functions

If your use case cannot be carried out with the openEO's pre-defined functions, you can create a User Defined Function (UDF). Therefore, you can create a Python function that will be executed at the back-end and functions as a process in your process graph. Detailed information about Python UDFs can be found in the official documentation as well as examples in the Python client repository.

2.2.5.8 Additional Information

Additional information and resources about the openEO Python Client Library:

Example scripts

Example Jupyter Notebooks

Official openEO Python Client Library Documentation

Repository on GitHub

2.2.6 OpenEO API Documentation

The most up-to-date openEO REST API documentation can be found on the openeo.org website.

3 Interactive Analysis Environment

3.1 Jupyter Notebooks

Notebooks is a browser-based tool for interactive analysis of data using C-SCALE storage and compute services. Notebooks is based on the JupyterHub technology.

This service can combine text, mathematics, computations and their rich media output using Jupyter technology, and can scale to multiple servers and users with the Cloud Compute service.

A basic instance of the Notebooks is available to users as an open service. Any researcher can access to this automatically to write and play notebooks on limited capacity cloud servers.

C-SCALE also offers customised Notebooks service to scientific communities. Such customised instances can be hosted on special hardware (for example with fat nodes and GPUs), can offer special libraries, data import/export and user authentication system.

EGI, EODC, CREODIAS and Terrascope/VITO are the C-SCALE providers that offer Jupyter Notebook services, and the related documentation can be found at:

- EGI: https://notebooks.egi.eu/hub/welcome
- EODC: https://marketplace.eosc-portal.eu/services/eodc-jupyterhub-for-global-copernicus-data?q=EODC+JupyterHub+for+global+Copernicus+data
- CREODIAS: https://creodias.eu/creodias-jupyter-hub
- Terrascope/VITO: https://notebooks.terrascope.be/hub/login

4 Conclusions

C-SCALE Batch Processing and Interactive Analysis Environments aim to provide users with a modern and simplified approach to process Earth Observation (EO) data by leveraging the strengths of the C-SCALE Compute Federation. For this purpose, openEO and Jupyter Notebooks are offered by several providers. This document is meant to give an overview of the available functionality to the end users. It is expected that, during the lifecycle of the project, new functionality or new providers will be available, rendering this document obsolete. Therefore, we aim to provide an in-depth and up-to-date online documentation through the C-SCALE Documentation page and through dedicated documentation channels of the relevant components, i.e., openEO.