A Common, Open Source Interface Between Earth Observation Data Infrastructures And Front-End Applications

Deliverable 07

Version 1.0 from 2018/03/27

Proof of Concept (Python)

Change history

Issue	Date	Author(s)	Description
0.1	2018/03/07	Jeroen Dries, VITO	First draft
0.2	2018/03/14	Dainius Masiliunas, WUR	First general review
0.3	2018/03/16	Jeroen Dries, VITO	Updated draft
0.4	2018/03/22	Dainius Masiliunas, WUR	Review
1.0	2018/03/27	Matthias Schramm, TU Wien	Final review and creation of final version

For any clarifications please contact openEO@list.tuwien.ac.at.

Number of pages: **12**

Disclaimer

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 776242. Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

Table of Contents

List of Acronyms

API Application Programming Interface

EO Earth Observation

NDVI Normalized Difference Vegetation Index

1 Executive Summary

This proof of concept is provided in the form of a Jupyter notebook [1]. This notebook demonstrates how a user can interact with an openEO back-end, using a Python interface. It shows this by writing an algorithm that first computes the NDVI parameter based on radiometric bands, and then generates a composite of these images.

The results of this computation are then retrieved as a GeoTiff and a time-series. By doing so, the core concepts of openEO, and the Python client API, are illustrated. A video explaining this notebook step-by-step can be found here: https://www.youtube.com/watch?v=qtIp9OC0qHY

The source code of the Python client API can be found in the GitHub repository: https://github.com/Open-EO/openeo-python-client

The full API documentation is published here: https://open-eo.github.io/openeo-python-client/

2 openEO Proof Of Concept: Compositing

This notebook explains and demonstrates the openEO client API, as well as some concepts of the core API. More information on the client API can be found in the documentation: https://open-eo.github.io/openeo-python-client/.

The openEO client API is distributed as a lightweight Python module. The dependencies of this module are limited to a set of well known modules such as numpy and pandas. This should allow it to run in different environments, and as part of larger workflows. To get started, we import openEO, and set up standard Python logging.

```
In [1]: import openeo
        import logging
        logging.basicConfig(level=logging.INFO)
```

To connect with an openEO back-end, we create a session. Each openEO back-end has a different endpoint, metadata and credentials. A session object contains this information, and is the starting point for subsequent calls.

```
In [2]: session = openeo.session("nobody", "http://openeo.vgt.vito.be")
```

Our first use case is to create a composite image by taking the maximum pixel value over a time-series of images. To do this, we first need to select input data. Each openEO endpoint exposes it's own list of image collections. For instance: http://openeo.vgt.vito.be/openeo/data.

Preferably, these layers and their descriptions and metadata can be browsed online, so a user can discover data that suits his needs. In this example, the collection id is 'S2_RADIOMETRY_V101', which corresponds to Sentinel 2 10M resolution bands over Belgium.

In the client code, the user can create an image collection quite easily. This is just an empty object on which further operations need to be defined:

```
In [3]: s2_radiometry = session.imagecollection("CGS_SENTINEL2_RADIOMETRY_V101")
        s2_radiometry

Out[3]: <openeo.rest.imagecollection.RestImageCollection at 0x7fe15c142438>
```

As the image collection can be quite large, a first step is usually to define a spatial and temporal subset on which we want to operate. This can be done by specifying a date range and a bounding box:

```
In [4]:
    timeseries = s2_radiometry\
    .date_range_filter("2017-10-14","2017-10-17")\
    .bbox_filter(left=761104,right=763281,bottom=6543830,top=6544655,srs="EPSG:3857")
    timeseries
```

```
Out[4]: <openeo.rest.imagecollection.RestImageCollection at 0x7fe0ea4ed128>
```

```
In [5]: bandFunction = lambda cells,nodata: (cells[3]-cells[2])/(cells[3]+cells[2])
    ndvi_timeseries = timeseries.apply_pixel([], bandFunction)
```

Now we're all set to compute the composite, the max_time function allows us to specify the function that needs to be applied, but does not yet compute a result:

```
In [6]: %time composite = ndvi_timeseries.max_time()
    composite
```

```
CPU times: user 11 µs, sys: 14 µs, total: 25 µs
Wall time: 38.4 µs
```

```
Out[6]: <openeo.rest.imagecollection.RestImageCollection at 0x7fe0ea4ed198>
```

Up to this point, the openEO back-end has not yet received a request for computation. We have only specified what is called a 'process graph' in openEO terms. We can do a few things with a process graph, let's start with downloading its result as a GeoTiff:

```
In [7]: %time composite.download("./openeo-ndvi-composite.geotiff","geotiff")
```

```
CPU times: user 184 ms, sys: 90 ms, total: 274 ms
Wall time: 14.9 s
```

The debug logging shows us that this call has sent our 'process graph' to: /openeo/execute. This particular composite took only 48 seconds to compute over a time-series of about 5 months worth of input data. This shows one of the key points of openEO: the algorithm gets distributed over processing resources close to the data, which can greatly speed up processing.

The next step will be visualising the result.

2.1 Plotting the result

Using rasterio, we can load and plot our downloaded file.

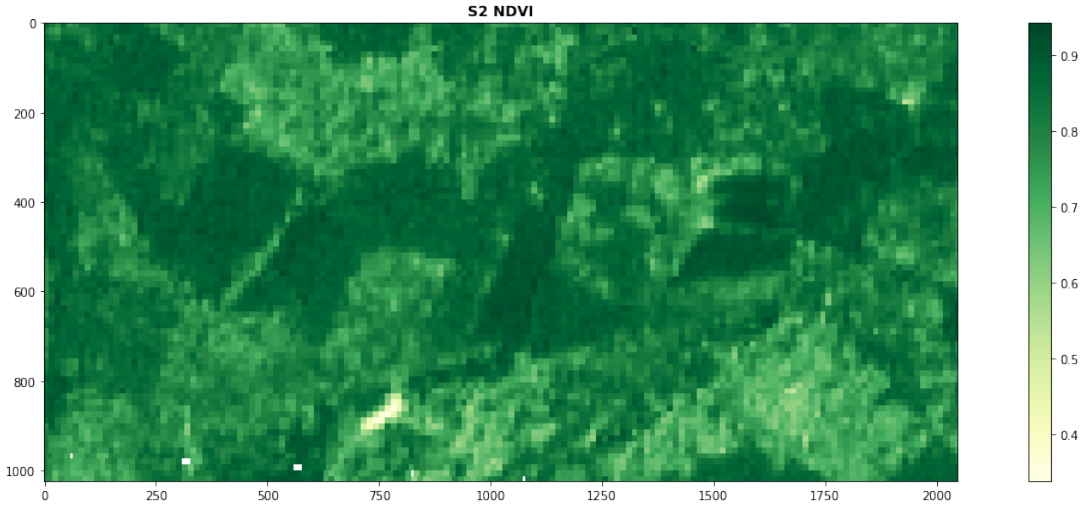
```
In [8]: import rasterio
    %matplotlib inline
```

```
In [9]: from rasterio.plot import show
    from matplotlib import pyplot


    composite_local = rasterio.open("./openeo-ndvi-composite.geotiff")
    ndvi_map = composite_local.read(1)
    composite_local.close()

    pyplot.set_cmap("YlGn")
```

```
    fig, (ndvi) = pyplot.subplots(1,1, figsize=(21,7))
    image = show(ndvi_map,ax=ndvi,title="S2 NDVI")
    fig.colorbar(image.images[0])
```

```
Out[9]: <matplotlib.colorbar.Colorbar at 0x7fe0d5a4dc88>
```

```
<Figure size 432x288 with 0 Axes>
```



2.2 Retrieving a time-series for a point

Instead of reducing our time-series of images into a composite, we can also request each value for a given geographical coordinate. For this call, we use the 'S2_FAPAR' image collection, which has a longer time-series available:

```
In [10]: %time point_timeseries = session.imagecollection("S2_FAPAR") \
     .bbox_filter(left=761104,right=763281,bottom=6543830,top=6544655,srs="EPSG:3857") \
     .timeseries(6.84638,50.56302,srs="EPSG:4326")
     point_timeseries.json()
```

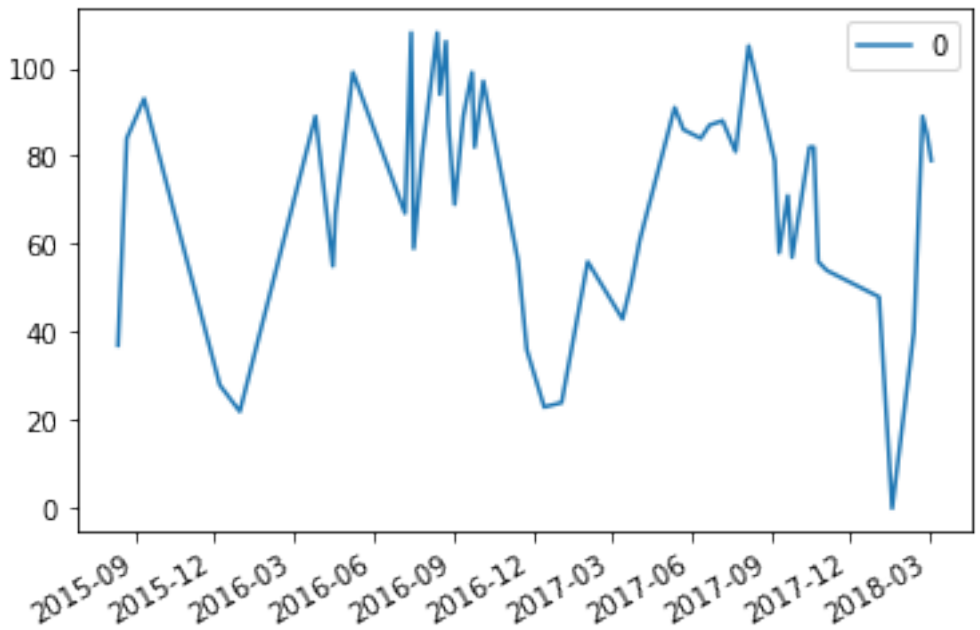
```
CPU times: user 13.7 ms, sys: 15.7 ms, total: 29.4 ms
Wall time: 14.3 s
```

```
Out[10]: {'2015-07-06T00:00:00+00:00': [nan],
      '2015-07-26T00:00:00+00:00': [nan],
      '2015-07-30T00:00:00+00:00': [nan],
      '2015-08-09T00:00:00+00:00': [nan],
      '2015-08-12T00:00:00+00:00': [37.0],
      '2015-08-19T00:00:00+00:00': [nan],
      '2015-08-22T00:00:00+00:00': [84.0],
      '2015-08-25T00:00:00+00:00': [nan],
        ...
      '2016-07-30T00:00:00+00:00': [nan],
      '2016-08-03T00:00:00+00:00': [nan],
      '2018-02-10T00:00:00+00:00': [nan],
      '2018-02-12T00:00:00+00:00': [40.0],
```

```
 '2018-02-15T00:00:00+00:00': [nan],
 '2018-02-17T00:00:00+00:00': [nan],
 '2018-02-20T00:00:00+00:00': [nan],
 '2018-02-22T00:00:00+00:00': [89.0],
 '2018-02-25T00:00:00+00:00': [nan],
 '2018-02-27T00:00:00+00:00': [85.0],
 '2018-03-02T00:00:00+00:00': [nan],
 '2018-03-04T00:00:00+00:00': [79.0],
 '2018-03-07T00:00:00+00:00': [nan],
 '2018-03-09T00:00:00+00:00': [nan],
 '2018-03-12T00:00:00+00:00': [nan],
 '2018-03-14T00:00:00+00:00': [nan]}
```

```
In [11]: %matplotlib inline
         import pandas as pd
         series_df = pd.DataFrame.from_dict(point_timeseries.json(), orient="index")
         series_df.index = pd.to_datetime(series_df.index)
         import seaborn as sns

         series_df.dropna().plot()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe0c78acfd0>
```



3 API docs

Python Client API for openEO back-ends. This client is a lightweight implementation with limited dependencies on other modules. The aim of openEO is to process remote sensing data on dedicated processing resources close to the source data.

This client allows users to communicate with openEO back-ends, in a way that feels familiar for Python programmers.

Basic example

```
import openeo
import logging

#enable logging in requests library
logging.basicConfig(level=logging.DEBUG)

#connect with EURAC back-end
session = openeo.connect("nobody", "http://saocompute.eurac.edu/openEO_WCPS_Driver")

#retrieve the list of available collections
collections = session.list_collections()
print(collections)

#create image collection
s2_fapar = session.imagecollection("S2_L2A_T32TPS_20M")

#specify workflow and download result as netcdf
s2_fapar \
    .date_range_filter("2016-01-01","2016-03-10") \
    .bbox_filter(left=652000,right=672000,top=5161000,bottom=5181000,srs="EPSG:32632
↪") \
    .max_time() \
    .download("/tmp/openeo-wcps.nc",format="NetCDF")
```

3.1 Main Interface

`openeo.session(`*userid=None*, *endpoint: str = 'https://openeo.org/openeo'*`)`

This method is the entry point to openEO. You typically create one session object in your script or application, per back-end, and re-use it for all calls to that back-end. If the back-end requires authentication, you should set your credentials.

Parameters `endpoint` (`str`) – The http url of an openEO endpoint.

Return type *openeo.sessions.Session*

`class openeo.sessions.Session`

A *Session* class represents a connection with an openEO service. It is your entry point to create new Image Collections.

`imagecollection(`*image_collection_id: str*`)` → openeo.imagecollection.ImageCollection

Retrieves an Image Collection object based on the id of a given layer. A list of available collections can be retrieved with *openeo.sessions.Session.list_collections()*.

Parameters `image_collection_id` (`str`) – The id of the image collection to retrieve.

Return type *openeo.imagecollection.ImageCollection*

`list_collections()` → dict
Retrieve all products available in the back-end. :return: a dict containing product information. The 'product_id' corresponds to an image collection id.

`class openeo.imagecollection.ImageCollection`
Class representing an Image Collection.

`aggregate_time(`*temporal_window*, *aggregationfunction*`)` → openeo.imagecollection.ImageCollection
Applies a windowed reduction to a time-series by applying a user defined function.

Parameters

- `temporal_window` – The time window to group by
- `aggregationfunction` – The function to apply to each time window. Takes a pandas time-series as input.

Returns An ImageCollection containing a result for each time window

`apply_pixel(`*bands: typing.List*, *bandfunction*`)` → openeo.imagecollection.ImageCollection
Apply a function to the given set of bands in this image collection.

This type applies a simple function to one pixel of the input image or image collection. The function gets the value of one pixel (including all bands) as input and produces a single scalar or tuple output. The result has the same schema as the input image (collection) but different bands. Examples include the computation of vegetation indexes or filtering cloudy pixels.

`bbox_filter(`*left: float*, *right: float*, *top: float*, *bottom: float*, *srs: str*`)` → openeo.imagecollection.ImageCollection
Specifies a bounding box to filter input image collections.

Parameters

- `left` (`float`) –
- `right` (`float`) –
- `top` (`float`) –
- `bottom` (`float`) –
- `srs` (`str`) –

Returns An image collection cropped to the specified bounding box.

`date_range_filter(`*start_date: typing.Union[str, datetime.date]*, *end_date: typing.Union[str, datetime.date]*`)` → openeo.imagecollection.ImageCollection
Specifies a date range filter to be applied on the ImageCollection

Parameters

- `start_date` – Start date of the filter, inclusive.

- `end_date` – End date of the filter, exclusive.

Returns An ImageCollection filtered by date.

`download`(*outputfile: str*, *bbox=''*, *time=''*, ***format_options*)
Extracts a binary raster from this image collection.

`max_time`() → openeo.imagecollection.ImageCollection
Finds the maximum value of time-series for all bands of the input dataset.

Returns An ImageCollection without a time dimension.

`min_time`() → openeo.imagecollection.ImageCollection
Finds the minimum value of time-series for all bands of the input dataset.

Returns An ImageCollection without a time dimension.

`reduce_time`(*aggregationfunction*) → openeo.imagecollection.ImageCollection
Applies a windowed reduction to a time-series by applying a user defined function.

Parameters `aggregationfunction` – The function to apply to each time window. Takes a pandas time-series as input.

Returns An ImageCollection without a time dimension

`send_job`() → openeo.job.Job
Sends the current process to the back-end, for batch processing.

Returns Job: A job object that can be used to query the processing status.

`tiled_viewing_service`() → typing.Dict
Returns metadata for a tiled viewing service that visualises this layer.

Returns A dictionary object containing the viewing service metadata, such as the connection 'url'.

`timeseries`(*x*, *y*, *srs='EPSG:4326'*) → typing.Dict
Extract a time-series for the given point location.

Return type `Dict`

Parameters

- `x` – The x coordinate of the point
- `y` – The y coordinate of the point
- `srs` (`str`) – The spatial reference system of the coordinates, by default this is 'EPSG:4326', where x=longitude and y=latitude.

Returns Dict: A timeseries

`class openeo.job.Job`(*job_id: str*)
Represents the result of creating a new Job out of a process graph. Jobs are stored in the back-end and can be executed directly (in batch), or evaluated lazily.

`download`(*outputfile: str*, *outputformat: str*)
Download the result as a raster.

4 References

[1] J. Dries, “Openeo proof of concept notebook.” [Online]. Available: https://github.com/Open-EO/openeo-python-client/blob/3d00f56f4365fa732bc6bbe8c8a877a4e412cddd/examples/notebooks/Compositing.ipynb