A COMMON, OPEN SOURCE INTERFACE BETWEEN EARTH OBSERVATION DATA INFRASTRUCTURES AND FRONT-END APPLICATIONS

Deliverable 06
Version 1.0 from 2018/03/26

Two early prototype back-ends
## Change history

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Author(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2018/03/15</td>
<td>Gunnar Busch, EODC</td>
<td>First draft.</td>
</tr>
<tr>
<td>0.2</td>
<td>2018/03/19</td>
<td>openEO consortium</td>
<td>Revision of back-end driver descriptions.</td>
</tr>
<tr>
<td>0.3</td>
<td>2018/03/20</td>
<td>Alexander Jacob, EURAC</td>
<td>Updated introduction with programming language section and added graph to description of WCPS back-end driver. Typo correction throughout the document.</td>
</tr>
<tr>
<td>0.4</td>
<td>2018/03/22</td>
<td>Miha Kadunc, Sinergise</td>
<td>Reviewed the deliverable and revised the section on Sentinel Hub back-end</td>
</tr>
<tr>
<td>1.0</td>
<td>2018/03/26</td>
<td>Matthias Schramm, TU Wien</td>
<td>Last Review and creation of final version</td>
</tr>
</tbody>
</table>

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

Number of pages: **24**

---

### 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

© openEO Consortium, 2018  
This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.
Table of Contents

1 Executive Summary 6

2 Introduction 6

3 Back-End Driver Demonstrators 8
   3.1 GeoPySpark VITO 8
      3.1.1 Back-End Description 8
      3.1.2 Current Capabilities 9
   3.2 GRASS GIS Mundialis 9
      3.2.1 Back-End Description 9
      3.2.2 Current Capabilities 9
      3.2.3 Example POST Request 10
      3.2.4 Visualisation 12
   3.3 OpenShift EODC 13
      3.3.1 Back-End Description 13
      3.3.2 Current Capabilities 13
      3.3.3 Example POST Request 14
      3.3.4 Visualisation 15
   3.4 R Back-End WWU 16
      3.4.1 Back-End Description 16
      3.4.2 Current Capabilities 16
      3.4.3 Example POST Request 17
   3.5 Sentinel Hub Sinergise 18
      3.5.1 Back-End Description 19
      3.5.2 Current Capabilities 19
      3.5.3 Example POST Request 20
      3.5.4 Multi-temporal Processing Script Generated by the Back-end 20
   3.6 WCPS EURAC 21
      3.6.1 Back-End Description 21
      3.6.2 Current Capabilities 21
      3.6.3 Example POST Request 22
      3.6.4 Visualisation 23

4 Outlook 23

5 References 23
## List of Figures

3.1 openEO GRASS GIS driver test suite (openEO wrapper test) . . . . . . . . . . . . 12  
3.2 Resulting NDVI raster layer as GeoTiff . . . . . . . . . . . . . . . . . . . . . . . . 12  
3.3 openEO service endpoints on OpenShift . . . . . . . . . . . . . . . . . . . . . . . . 15  
3.4 Running and completed jobs submitted on /jobs . . . . . . . . . . . . . . . . . . . . 15  
3.5 Result log of a finished data extraction job . . . . . . . . . . . . . . . . . . . . . . . 16  
3.6 Overview of the WCPS driver in the context of a high level backend inside the  
    Eurac Research Sentinel Alpine Observatory . . . . . . . . . . . . . . . . . . . . . . . 23
List of Acronyms

API  Application Programming Interface
EO  Earth Observation
EODC  Earth Observation Data Centre
EURAC  Eurac Research
GIS  Geo Information System
GRASS  Geographic Resources Analysis Support System
HTTP  Hypertext Transfer Protocol
NDVI  Normalized Difference Vegetation Index
NFS  Network File System
OGC  Open Geospatial Consortium
openEO  Name of the project for which this report was written
PoC  Proof of Concept
POSIX  Portable Operating System Interface
REST  Representational State Transfer
WCPS  Web Coverage Processing Service
WCS  Web Coverage Service
WWU  Westphalian Wilhelms-University
1 Executive Summary

This document is intended to provide an overview of the achievements of the openEO deliverable "D4.2 Two early prototype back-end drivers (file-based, SciDB)". The deliverable is a demonstrator and one of four deliverables (D3.1, D4.1, D4.2 and D5.1) in month 6 of the openEO project that together result in the milestone “M2 Proof of Concept (PoC)". The overall aim of the PoC is to provide a first prototypical implementation of the openEO interface, which includes a python client, a first prototype of the Core API and two back-end prototypes.

2 Introduction

The first six months of the openEO project were efficiently used to develop back office drivers in parallel, that implement the openEO core API specification and provide API based access to available data and operations. Two weeks of intense collaboration in Belgium and Germany and bi-weekly video conferences, together with the Core API (WP3) and Client (WP5) developers, allowed close collaboration, coordination of common problems and finding common solutions. As already discussed in the project kick off meeting, the exact variation of implemented back-end drivers for D4.2, that were defined as one file-based and one SciDB driver, varied. The consortium decided to replace the development of a SciDB driver with a WCPS/rasdaman driver, given that none of the back-office project partners are currently using SciDB. While the implemented drivers diversified, the amount of currently available prototypical back-end driver implementations widely exceeds the proposed two back-end drivers.

The openEO proposal defines three different kinds of back-end categories that are intended to implement the openEO core API specification: File-based, mid-level and high-level back- offices. File-based back-ends provide access to raw data via POSIX, NFS or Object Storage. Early prototype drivers for file-based back-ends were implemented for Sentinel Hub (Sinergise) and OpenShift (EODC). Mid-level API back office drivers provide an interface between software systems that can utilise file-based data processing. Currently available prototypical mid-level drivers were implemented for GeoTrellis (VITO) and GRASS GIS (Mundialis). Furthermore, drivers for high-level back offices, that provide access to data and processes through API calls (e.g. as defined by OGC standards) were implemented for WCPS (EURAC) on top of rasdaman. Moreover, a back-end driver in R was developed by WWU to provide a lightweight and simplistic version of an openEO compliant back-end driver for local development and testing.

Apart from the three general categories of back-end types also a number of different programming languages have been used for implementing the drivers, ranging from Python over R and JavaScript to Java. These implementations provide a good starting point for any developer aiming to develop drivers in various back-ends and software development environments.

The following sections will give a short overview of available back-end driver prototypes, the current status and the technical specifications of the individual drivers. Interfaces that are currently implemented by the back-ends with respect to the version 0.0.2 of the openEO core API are listed for each driver. A complete overview of specified routes, that are included in the openEO Core API v0.0.2 is given below. Further details on the Core API can be found at https://open-eo.github.io/openeo-api/.
D06: Prototype back-ends

For each of the back-end driver prototypes, each subsection contains a specification of the respective capabilities. The following list gives a general description of the structure of these subsections:

Capabilities

- **API information**: `/capabilities`
- **Output formats**: `/capabilities/output Formats`
- **Web service types**: `/capabilities/services`

EO Data Discovery

- **Data discovery**: `/data`
- **Product information**: `/data/<product_id>`

Process Discovery

- **Process discovery**: `/processes`
- **Process information**: `/processes/<process_id>`

User Defined Functions (UDF)

- **UDF runtimes**: `/udf_runtimes`
- **UDF process description**: `/udf_runtimes/<lang>/udf_type`

User Content

- **Jobs submitted by user**: `/users/<user_id>/jobs`
- **Services submitted by user**: `/users/<user_id>/services`
- **Credits available to user**: `/users/<user_id>/credits`
- **Submit, retrieve process graphs of user**: `/users/<user_id>/process_graphs`
- **Manage process graph of user**: `/users/<user_id>/process_graphs/<process_graph_id>`
- **List user-uploaded files**: `/users/<user_id>/files`
- **Manage user-uploaded files**: `/users/<user_id>/files/<path>`

Authentication

- **Authentication of user**: `/auth/login`
- **Register new user**: `/auth/register`

Job Management

- **Submit new job**: `/jobs`
D06: Prototype back-ends

- **Synchronous execution of job**: `/execute`
- **Manage submitted job**: `/jobs/<job_id>`
- **Subscribe to job updates**: `/jobs/<job_id>/subscribe`
- **Running job in batch mode**: `/jobs/<job_id>/queue`
- **Pause a job**: `/jobs/<job_id>/pause`
- **Cancel a job**: `/jobs/<job_id>/cancel`
- **Download links to job results**: `/jobs/<job_id>/download`

**Service Management**

- **Publish a new service**: `/services`
- **Manage services**: `/services/<service_id>`

Instructions for installing and using the back-end drivers, e.g., for local development or testing, are included in the GitHub repositories of the individual driver. References to the GitHub repositories are given as this document is not intended to provide specific instructions on how to implement and run the drivers. These instructions are available in the “README” files of the GitHub repositories. Some back-ends provide a publicly accessible URL to their openEO API implementation that can be used to execute HTTP request examples. URLs to the public endpoints, if available and example HTTP-requests, are given in the technical specifications and may be used to demonstrate the back-office prototypes. Specific API routes may be secured, and the user needs to request a token to access these routes. For the purpose of sending requests on secured routes and for any other questions concerning a specific back-end one may contact the back-office provider or the openEO project or work package coordinators.

### 3 Back-End Driver Demonstrators

#### 3.1 GeoPySpark VITO

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
</tr>
<tr>
<td><strong>Language</strong></td>
</tr>
<tr>
<td><strong>GitHub Link</strong></td>
</tr>
<tr>
<td><strong>Public Endpoint</strong></td>
</tr>
</tbody>
</table>

#### 3.1.1 Back-End Description

GeoTrellis is an open source Scala library that facilitates fast operations on raster data using Scala and Apache Spark. It enables distributed computing on large data sets using map algebra techniques and supports vector, raster and point cloud data. GeoTrellis provides RESTful
endpoints and raster processing at web speeds (sub-second or less) [1]. The driver implements a direct (non-REST) version of the openEO client API on top of GeoPySpark. The back-end has been tested with Spark on a Yarn cluster and with Apache Accumulo as the tile storage back-end for Geotrellis. The openEO driver could, with minor modifications, be adapted to work with other Geotrellis back-ends, such as S3.

### 3.1.2 Current Capabilities

- **Data discovery**: /data
- **Product information**: /data/<product_id>
- **Synchronous execution of job**: /execute
- **Download links to job results**: /jobs/<job_id>/download
- **Services**: TMS (Tile Map Service)
- **Filters**: Bands, Date Range, Bounding Box
- **Operations**: NDVI, Min Time, Max Time

### 3.2 GRASS GIS Mundialis

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
</tr>
<tr>
<td>Language</td>
</tr>
<tr>
<td>GitHub Link</td>
</tr>
<tr>
<td>Public Endpoint</td>
</tr>
</tbody>
</table>

#### 3.2.1 Back-End Description

The GRASS GIS driver implements the openEO Core API interface for the GRASS GIS as a Service (Actinia Core; available from https://github.com/mundialis/actinia_core) - a software solution for parallel, large scale geodata processing. It is a highly scalable REST interface to process geodata with the GRASS GIS in a distributed environment. GRASS GIS is a free and open source software package providing geospatial processing engines in a single integrated environment for raster, vector, and 3d-voxel processing as well as image processing capabilities [2, 3].

#### 3.2.2 Current Capabilities

- **API information**: /capabilities
- **Data discovery**: /data
- **Product information**: /data/<product_id>
D06: Prototype back-ends

- **Process discovery**: /processes
- **Process information**: /processes/<process_id>
- **UDF runtimes**: /udf_runtimes
- **UDF process description**: /udf_runtimes/<lang>/udf_type
- **Submit new job**: /jobs
- **Manage submitted job**: /jobs/<job_id>
- **Filters**: Bands, Date Range, Bounding Box
- **Operations**: NDVI, Min Time, Max Time, Zonal Statistics, Raster Exporter
- **UDFs**: Reduce Time

### 3.2.3 Example POST Request

```json
{
    "process_graph": {
        "process_id": "min_time",
        "args": {
            "collections": [{
                "process_id": "NDVI",
                "args": {
                    "collections": [
                        {
                            "process_id": "filter_daterange",
                            "args": {
                                "collections": [
                                    {
                                        "process_id": "filter_bbox",
                                        "args": {
                                            "collections": [{"product_id": "LL.sentinel2A_openeo_subset.strds.S2A_B04"}],
                                            "bottom": 38.9,
                                            "left": -4.8,
                                            "right": -4.6,
                                            "top": 39.1,
                                            "ewres": 0.0001,
                                            "nsres": 0.0001,
                                            "srs": "EPSG:4326"
                                        }
                                    }
                                ]
                            },
                            "from": "2017-04-12 11:17:08",
                            "to": "2017-09-04 11:18:26"
                        }
                    ]
                }
            }
        }
    }
}
```
{
  "process_id": "filter_daterange",
  "args": {
    "collections": [{
      "process_id": "filter_bbox",
      "args": {
        "collections": [{
          "product_id": "LL.sentinel2A_openeo_subset.strds.S2A_B08"},
          "bottom": 38.9,
          "left": -4.8,
          "right": -4.6,
          "top": 39.1,
          "ewres": 0.0001,
          "nsres": 0.0001,
          "srs": "EPSG:4326"
        }
      ],
      "from": "2017-04-12 11:17:08",
      "to": "2017-09-04 11:18:26"
    }
  ],
  "red": "S2A_B04",
  "nir": "S2A_B08"
}
3.2.4 Visualisation

Figure 3.1: openEO GRASS GIS driver test suite (openEO wrapper test)

Figure 3.2: Resulting NDVI raster layer as GeoTiff
3.3 OpenShift EODC

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
</tr>
<tr>
<td>Language</td>
</tr>
<tr>
<td>GitHub Link</td>
</tr>
<tr>
<td>Public Endpoint</td>
</tr>
</tbody>
</table>

### 3.3.1 Back-End Description

OpenShift Origin is an open source and community driven Platform-as-a-Service (PaaS) by RedHat that is based on Kubernetes and Docker containers to provide cluster and container life-cycle management. OpenShift includes Kubernetes for container orchestration and offers multiple extensive layers on top of it to realise user management, container building, integrated networking and persistent storage [4]. The OpenShift cluster at EODC is hosted in an integrated environment that allows direct access to the file-based storage of EODC and enables to perform operations on the data by executing single Docker containers that have mounted persistent storage. Web services based on Flask provide the endpoints to the openEO API.

### 3.3.2 Current Capabilities

- **API information**: /capabilities
- **Output formats**: /capabilities/output_formats
- **Data discovery**: /data
- **Product information**: /data/<product_id>
- **Process discovery**: /processes
- **Process information**: /processes/<process_id>
- **Authentication of user**: /auth/login
- **Submit new job**: /jobs
- **Running job in batch mode**: /jobs/<job_id>/queue
- **Manage submitted job**: /jobs/<job_id>
- **Download links to job results**: /jobs/<job_id>/download
- **Filters**: Bands, Date Range, Bounding Box
- **Operations**: NDVI, Min Time, Sentinel-2 Extraction
3.3.3 Example POST Request

```json
{
    "process_graph": {
        "process_id": "min_time",
        "args": {
            "imagery": {
                "process_id": "NDVI",
                "args": {
                    "imagery": {
                        "process_id": "filter_daterange",
                        "args": {
                            "imagery": {
                                "process_id": "filter_bbox",
                                "args": {
                                    "imagery": {
                                        "product_id": "s2a_prd_msil1c",
                                        "left": 652000,
                                        "right": 672000,
                                        "top": 5161000,
                                        "bottom": 5181000,
                                        "srs": "EPSG:32632"
                                    }
                                },
                                "from": "2017-01-01",
                                "to": "2017-01-08"
                            }
                        }
                    }
                }
            }
        }
    }
}
```
3.3.4 Visualisation

Figure 3.3: openEO service endpoints on OpenShift

Figure 3.4: Running and completed jobs submitted on /jobs
3.4 R Back-End WWU

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
</tr>
<tr>
<td>Language</td>
</tr>
<tr>
<td>GitHub Link</td>
</tr>
<tr>
<td>Public Endpoint</td>
</tr>
</tbody>
</table>

3.4.1 Back-End Description

The R back-end is a reference implementation for the openEO core API as a proof-of-concept written in R, utilising the `plumber` package as a lightweight web server. The web server is not final and in terms of security aspects not optimised. The goal of this package is to provide a simplistic version of local server back-end compliant with openEO [5].

3.4.2 Current Capabilities

- **API information**: /capabilities
- **Output formats**: /capabilities/output_formats
• Data discovery: /data
• Product information: /data/<product_id>
• Process discovery: /processes
• Process information: /processes/<process_id>
• Jobs submitted by user: /users/<user_id>/jobs
• Submit, retrieve process graphs of user: /users/<user_id>/process_graphs
• Manage process graph of user: /users/<user_id>/process_graphs/<process_graph_id>
• List user-uploaded files: /users/<user_id>/files
• Manage user-uploaded files: /users/<user_id>/files/<path>
• Authentication of user: /auth/login
• Submit new job: /jobs
• Download links to job results: /jobs/<job_id>/download
• Synchronous execution of job: /execute

3.4.3 Example POST Request

```
{
  "process_graph": {
    "process_id": "find_min",
    "args": {
      "imagery": {
        "process_id": "calculate_ndvi",
        "args": {
          "imagery": {
            "process_id": "filter_daterange",
            "args": {
              "imagery": {
                "product_id": "sentinel2_subset"
              },
              "from": "2017-04-01",
              "to": "2017-05-01"
            }
          },
          "nir": 8,
          "red": 4
        }
      }
    }
  }
}
```
D06: Prototype back-ends

```json
{
  "process_graph": {
    "process_id": "zonal_statistics",
    "args": {
      "imagery": {
        "process_id": "filter_daterange",
        "args": {
          "imagery": {
            "process_id": "filter_bands",
            "args": {
              "imagery": {
                "product_id": "sentinel2_subset"
              },
              "bands": "8"
            }
          },
          "from": "2017-04-01",
          "to": "2017-07-01"
        }
      },
      "regions": "/users/me/files/polygons.geojson",
      "func": "median"
    }
  },
  "output": {
    "format": "GPKG"
  }
}
```

### 3.5 Sentinel Hub Sinergise

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform / Framework</strong></td>
</tr>
<tr>
<td><strong>Programming Language</strong></td>
</tr>
<tr>
<td><strong>GitHub Link</strong></td>
</tr>
<tr>
<td><strong>Public Endpoint</strong></td>
</tr>
</tbody>
</table>
3.5.1 Back-End Description

Sentinel Hub (https://www.sentinel-hub.com/) is a service-oriented platform that aims to facilitate processing and exploitation of satellite imagery by providing efficient access to several Earth observation data sets via easy-to-integrate web services.

The main features of the system are: automated archiving process, ensuring that data are always synchronised with official sources; full resolution access over web services, with automatic re-projection, mosaicking, ortho-rectification and other standard processing; user-defined scripts to define spectral band expressions and other remote sensing products; multi-temporal processing for on-the-fly computation of user-defined algorithms on time-series; service for statistical analysis of a time-series over an area or point of choice; and APIs for advanced feature integration.

Sentinel Hub works with original EO data, avoiding intensive pre-processing and additional storage for processed tiles. Some of the data accessible through Sentinel Hub are stored as AWS Public Datasets\(^1\) (Sentinel-1, Sentinel-2, Landsat 8, MODIS) and some are hosted on CloudFerro EO Cloud\(^2\) (Sentinel-1, Sentinel-2, Sentinel-3, ENVISAT, ESA’s archive of Landsat 5, 7 and 8) [6].

3.5.2 Current Capabilities

- **API information:** /capabilities
- **Web service types:** /capabilities/services
- **Data discovery:** /data
- **Product information:** /data/<product_id>
- **Process discovery:** /processes
- **Process information:** /processes/<process_id>
- **Jobs submitted by user:** /users/<user_id>/jobs
- **Services submitted by user:** /users/<user_id>/services
- **Submit new job:** /jobs
- **Publish a new service:** /services
- **Manage services:** /services/<service_id>
- **Filters:** Date Range
- **Operations:** NDVI, Min Time, Max Time

---

\(^1\)https://aws.amazon.com/earth/
\(^2\)https://finder.eocloud.eu/
3.5.3 Example POST Request

```json
{
  "process_graph": {
    "process_id": "min_time",
    "args": {
      "imagery": {
        "process_id": "NDVI",
        "args": {
          "imagery": {
            "process_id": "filter_daterange",
            "args": {
              "imagery": {
                "product_id": "Sentinel2A-L1C"
              },
              "from": "2017-01-01",
              "to": "2017-01-31"
            }
          },
          "red": "B04",
          "nir": "B08"
        }
      }
    }
  }
}
```

3.5.4 Multi-temporal Processing Script Generated by the Back-end

```javascript
const findMinIndex = arr => {
  let min = Infinity;
  let minIdx = -1;
  for (var i = 0; i < arr.length; i++) {
    let cur = arr[i];
    if (cur < min) {
      min = cur;
      minIdx = i;
    }
  }
  return minIdx;
};
const findMin = arr => arr[findMinIndex(arr)];
const index = (a, b) => (a - b) / (a + b);
const dateRangeFilter = (from, to) => {
  return (scene => from < scene.date.getTime() && scene.date.getTime() < to);
};
function setup(dss) {
  setInputComponents([dss.B04, dss.B08]);
  setOutputComponentCount(1);
}```
D06: Prototype back-ends

```javascript
function filterScenes(scenes, inputMetadata) {
    scenes = scenes.filter(
        dateRangeFilter(Date.parse('2017-01-01'), Date.parse('2017-01-31'))
    );
    return scenes;
}

function eval(samples, scenes) {
    samples = samples.map(s => index(s.B08, s.B04));
    samples = [findMin(samples)];
    return samples;
}
```

### 3.6 WCPS EURAC

<table>
<thead>
<tr>
<th>Technical Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Platform</strong></td>
</tr>
<tr>
<td><strong>Language</strong></td>
</tr>
<tr>
<td><strong>GitHub Link</strong></td>
</tr>
<tr>
<td><strong>Public Endpoint</strong></td>
</tr>
</tbody>
</table>

#### 3.6.1 Back-End Description

WCS (Web Coverage Service) is an interface standard by the OGC (Open Geospatial Consortium) that supports the retrieval of geospatial data as coverages representing space and time-varying phenomena. The coverage data of a WCS is returned in forms that are useful for client-side rendering, as input into scientific models, and for other clients [7]. WCPS (Web Coverage Processing Service) extends this standard by defining a language that is capable of processing and retrieval of multi-dimensional geospatial coverages that represent images, sensor observations or statistical data. The WCPS language is independent from any particular request and response encoding, as no concrete request/response protocol is specified by the standard. For setting up a WCPS instance, therefore, a separate, additional specification establishing the concrete protocol is required. This allows embedding of WCPS into different target service frameworks [8]. The openEO WCPS driver is built on top of rasdaman (“raster data manager”), an Array Database that allows the storing and querying of multi-dimensional arrays. Rasdaman can process arrays residing in file system directories as well as in databases. The developed driver itself however is operational with all WCPS compliant web services.

#### 3.6.2 Current Capabilities

- **API information**: /capabilities
- **Output formats**: /capabilities/output_formats
- **Data discovery**: /data
- **Product information**: /data/<product_id>

---

openEO
Grant agreement No 776242
D06: Prototype back-ends

- **Process discovery**: /processes
- **Process information**: /processes/<process_id>
- **Authentication of user**: /auth/login
- **Submit new job**: /jobs
- **Synchronous execution of job**: /execute
- **Manage submitted job**: /jobs/<job_id>
- **Running job in batch mode**: /jobs/<job_id>/queue
- **Download links to job results**: /jobs/<job_id>/download
- **Filters**: Bands, Date Range, Bounding Box
- **Operations**: NDVI, Min Time, Max Time

### 3.6.3 Example POST Request

```
{
    "process_graph": {
        "process_id": "max_time",
        "args": {
            "imagery": {
                "process_id": "NDVI",
                "args": {
                    "imagery": {
                        "process_id": "filter_daterange",
                        "args": {
                            "imagery": {
                                "process_id": "filter_bbox",
                                "args": {
                                    "imagery": {
                                        "product_id": "S2_L2A_T32TPS_20M"
                                    },
                                    "left": 652000,
                                    "right": 672000,
                                    "top": 5161000,
                                    "bottom": 5181000,
                                    "srs": "EPSG:32632"
                                }
                            },
                            "from": "2017-01-01",
                            "to": "2017-01-31"
                        }
                    }
                }
            }
        }
    }
}
```
3.6.4 Visualisation

Figure 3.6: Overview of the WCPS driver in the context of a high level backend inside the Eurac Research Sentinel Alpine Observatory

4 Outlook

The Proof of Concept demonstrates the feasibility of openEO core functionality. These features, that are crucial for the success of the project, were implemented on the basis of investigations in standardising interfaces connecting the back-ends with the clients. These developed driver implementations serve as a basis for further extending the current features in accordance with the Core API specification. Major functionality like user management and advanced job processing and monitoring are yet to be implemented. Therefore, from month 7, the back-end providers will work towards the implementation of the first full API integration, which will be available in month 20. The proof of concept itself, together with the user requirements meeting and the hackathon in month 9 are serving as continued feedback and validation streams to improve openEO and assure that the realisation meets the user and back-end requirements.

5 References


