OGC Earth Observation Applications Pilot

EOX-Sinergise-DLR-UVT-Terrasigna Engineering Report

Publication Date: 2020-10-22

Approval Date: 2020-09-23

Submission Date: 2020-08-26

Reference number of this document: OGC 20-043

Reference URL for this document: http://www.opengis.net/doc/PER/EOAppsPilot-EOX

Category: OGC Public Engineering Report

Editors: Stefan Achtsnit, Joachim Ungar, and Stephan Meißl (EOX [https://eox.at]), Anja Vrecko and Grega Milčinski (Sinergise [https://sinergise.com]), Torsten Heinen, Julian Zeidler, and Jonas Eberle (DLR [https://www.dlr.de/eoc/en/]), Marian Neagul (UVT [https://www.uvt.ro/en/]), Adrian Stoica and Vasile Craciunescu (Terrasigna [http://www.terrasigna.com])

Title: OGC Earth Observation Applications Pilot: EOX-Sinergise-DLR-UVT-Terrasigna Engineering Report

OGC Public Engineering Report

COPYRIGHT

WARNING

LICENSE AGREEMENT

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Table of Contents

Chapter 1. Subject

This Engineering Report documents findings, achievements, and learnings gained through activities during the OGC Earth Observation (EO) Applications Pilot by the EOX team (EOX, DLR, UVT, Sinergise, and Terrasigna). Both perspectives, from application developer's as well as from platform provider's view, are represented here.

Chapter 2. Executive Summary

Running algorithms on a cloud platform close to the source data and trying to abstract platform specifics for interoperability are common paradigms in the world of large-scale processing. They are explicitly relevant for the Earth Observation space with the huge EO data archives and the manifold application scenarios in different domains.

OGC EO Applications Packages declare a contract between

- the application developer and
- the platform provider

to get a specific application deployed and executed on a target platform, with the platform not just being responsible for providing the compute resources and the execution runtime but also for managing all data handling.

From an application developer's perspective the main goal was to package EO applications in the most interoperable way to get the applications running on different platforms. Focus was put to test different kinds of application packages and to document needed modifications and possible solution paths.

On the platform side the necessary functionality was established to support these different execution scenarios. The various implementation changes and challenges were highlighted and put also in context to generally needed platform capabilities, like multi-tenancy or the tracking of resource consumption.

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Stefan Achtsnit	EOX [https://eox.at]	Editor
Joachim Ungar	EOX [https://eox.at]	Editor
Stephan Meißl	EOX [https://eox.at]	Editor
Anja Vrecko	Sinergise [https://sinergise.com]	Editor
Grega Milčinski	Sinergise [https://sinergise.com]	Editor
Torsten Heinen	DLR [https://www.dlr.de/eoc/en/]	Editor
Julian Zeidler	DLR [https://www.dlr.de/eoc/en/]	Editor

Name	Organization	Role
Jonas Eberle	DLR [https://www.dlr.de/eoc/en/]	Editor
Marian Neagul	West University of Timisoara (UVT) [https://www.uvt.ro/en/]	Editor
Adrian Stoica	Terrasigna [http://www.terrasigna.com]	Editor
Vasile Craciunescu	Terrasigna [http://www.terrasigna.com]	Editor

2.2. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **Coherence**

  The coherence refers to the amplitude of the complex correlation coefficient between two Synthetic Aperture Radar (SAR) images. It is a measure of the similarity of two SAR images.

- **Container**

  A standardized unit of software (Docker [https://www.docker.com/resources/what-container]).

- **SNAP Graph**

  A set of operators that define a processing chain in SNAP. SNAP [https://step.esa.int/main/toolboxes/snap/], the Sentinel Application Platform, is a common architecture for all Sentinel Toolboxes.

3.1. Abbreviated terms

- ADES Application Deployment and Execution Service
- AI Artificial Intelligence
- AOI Area Of Interest
- AP Application Package
- AWS Amazon Web Services
- BPEL Business Process Execution Language
- CFP Call For Participation
- CWL Common Workflow Language
- DEM Digital Elevation Model
- DWG Domain Working Group
- EMS Execution Management Service
- EO Earth Observation
- EP Exploitation Platform
- ER Engineering Report
- ESA European Space Agency
- FUSE Filesystem in Userspace
- GCP Google Cloud Platform
- GDAL Geospatial Data Abstraction Library
- GUI Graphical User Interface
- JSON JavaScript Object Notation

- MEP Mission Exploitation Platform
- OGC Open Geospatial Consortium
- OWC OWS Context
- REST REpresentational State Transfer
- S1 Sentinel-1
- S2 Sentinel-2
- SAR Synthetic Aperture Radar
- SLC Single Look Complex
- SNAP SeNtinel Application Platform
- STAC Spatio-Temporal Asset Catalog
- TEP Thematic Exploitation Platform
- TIE Technology Integration Experiments
- TOI Time Of Interest
- UI User Interface
- URI Uniform Resource Identifier
- URL Uniform Resource Locator
- VM Virtual Machine
- WKT Well-Known Text
- WCS Web Coverage Service
- WFS Web Feature Service
- WPS Web Processing Service
- WPST Web Processing Service Transactional
- XML eXtensible Markup Language

Chapter 4. Overview

Section 5 introduces the implemented Application Packages and their deployments tested in the pilot.

Section 6 presents the solution for the exploitation layer developed in this pilot.

Section 7 provides a summary of the main findings from the view of a platform provider.

Section 8 provides a summary of the main findings from the view of a application developer.

Section 9 draws some conclusions and recommendations from the work performed in the pilot.

Chapter 5. Earth Observation Application Packages

The team developed and demonstrated two Application Packages, an agriculture app (Agri App) as well as an urban app (Urban App). The two Application Packages are described in detail in the following sections.

5.1. Agri App

The Agri App aims to support decision making on eligibility of subsidies paid to farmers in European Union (EU) Member States. National Paying Agencies (PAs) in these states perform such eligibility checks according to the European Common Agricultural Policy (CAP). Currently, the implementation of this policy undergoes significant changes leading towards intensified utilization of Earth Observation data and cloud infrastructures for management and information extraction.

Overall, 78 PAs in Europe dispose close to 60 billion Euro of subsidies payments annually. Presenting tools “that work” to this stakeholder sphere is thus strongly impacting on Europe’s economy.

Subsidy claims require certain agricultural practices to be monitored and verified: Presence/absence of mowing in grassland; the occurrence of ploughing during a specific seasonal time window; or the rapid growth of vegetative cover during certain time period. Indicators include the detection of fallow periods (i.e., absence of activities) or the burning of stubbles following a harvest. Detection of such indicators is performed by EO data analyses and involves parcel-level averaged time series profiles of vegetation indexes, spectral band values, signal variability metrics, Synthetic Aperture Radar (SAR) backscatter or coherence time series. The high-cadence of Copernicus Sentinel-1 and Sentinel-2 coverages of every agricultural parcel in Europe is the enabling element of the described CAP area monitoring.

5.1.1. Overview

Based on the field data which are gathered in the national/sub-national Land Parcel Identification System (LPIS) farmers submit their Geospatial Aid Application (GSAA). Crucial to accurate eligibility checks are high-quality field boundaries/geometries and consistent farmer’s declarations about applied land management processes, all with respect to their country and crop/claim type specific reference periods. LPIS and GSAA data are to be managed as auxiliary data together with the EO data.

The Agri App is a prototype showcasing using Sentinel-2 data to conduct land usage classification with the help of a pre-trained machine learning model developed by EOX.

5.1.2. Inputs

The Agri App works on individual parcels per job so the parcel geometry has to be provided.

Further, the machine learning model requires a time stack of Sentinel-2 Level 2A data from 2018-04-01 to 2018-10-14 including the bands B02, B03, B04, B05, B06, B07, B08, B8A, B11, and B12 in

order being able to classify.

The input can either be provided as a spatiotemporal subset of the Sentinel-2 archive limited to the TOI mentioned above as well as the parcel bounding box. This subset has to be described with Spatio-Temporal Asset Catalog (STAC) [https://stacspec.org].

Alternatively — and solely for debugging purposes — the app will automatically fetch the data from the Sentinel-2 archive available on Amazon Web Services (AWS).

5.1.3. Processing

The Agri App runs a Python script which handles the following processing steps:

- data fetching (via STAC or AWS)
- data preparation
- apply classification model
- write output

5.1.4. Outputs

The Agri App writes a GeoJSON file containing the original parcel geometry as well as the model output parameters as properties.

5.1.5. Application Package

The application consists of a docker image [https://registry.gitlab.eox.at/maps/masterclassification:ogc_demo] which has its ENTRYPOINT set to the classification script inside as well as a CWL file [https://github.com/eurodatacube/ogc-eo-apps-pilot] describing the inputs and outputs. The Dockerfile together with the script used in the ENTRYPOINT as well as the CWL file are provided below for reference.

Dockerfile

```
# use builder to build python wheels #
######################################
FROM registry.gitlab.eox.at/maps/docker-base/mapchete:0.8 as builder
MAINTAINER Joachim Ungar

ENV GODALE_VERSION 0.2

ENV BUILD_DIR /usr/local
ENV MASTERCLASSIFICATION_DIR $BUILD_DIR/src/masterclassification
ENV WHEEL_DIR /usr/local/wheels

RUN apt-get update \
    && apt-get install --yes --no-install-recommends build-essential gcc git \
    && rm -rf /var/lib/apt/lists/*

RUN mkdir -p $MASTERCLASSIFICATION_DIR
```

```
# get dependencies before checking out source code to speed up container build
COPY requirements.txt $MASTERCLASSIFICATION_DIR/
# GDALs python bindings must be built separately with specific options, so lets filter
out this one from requirements.txt
RUN sed -n '/gdal/!p' $MASTERCLASSIFICATION_DIR/requirements.txt >
$MASTERCLASSIFICATION_DIR/requirements_filtered.txt
RUN cat $MASTERCLASSIFICATION_DIR/requirements_filtered.txt
RUN pip install cython \
    && pip wheel \
        git+http://gitlab+deploy-token-
7:3AFUNqdLiKayR9Ang9Gx@gitlab.eox.at/maps/godale.git@${GODALE_VERSION} \
        psutil \
        -r $MASTERCLASSIFICATION_DIR/requirements_filtered.txt \
        --wheel-dir $WHEEL_DIR \
        --no-deps \
    && pip wheel \
        lxml \
        pystac \
        pytz \
        --wheel-dir $WHEEL_DIR \
        --no-deps \
    && pip uninstall --yes cython

# build image using pre-built libraries and wheels #
#####################################################
FROM registry.gitlab.eox.at/maps/docker-base/mapchete:0.8 as runner
MAINTAINER Joachim Ungar

ENV AWS_REQUEST_PAYER requester
ENV C_FORCE_ROOT "yes"
ENV CURL_CA_BUNDLE=/etc/ssl/certs/ca-certificates.crt
ENV GML_SKIP_CORRUPTED_FEATURES YES
ENV MP_SATELLITE_REMOTE_TIMEOUT 30
ENV BUILD_DIR /usr/local
ENV MASTERCLASSIFICATION_DIR $BUILD_DIR/src/masterclassification
ENV WHEEL_DIR /usr/local/wheels

RUN mkdir /mnt/input && mkdir /mnt/output

# get and install wheels from builder
COPY --from=builder $WHEEL_DIR $WHEEL_DIR
RUN pip install \
        $WHEEL_DIR/godale*.whl \
        $WHEEL_DIR/psutil*.whl \
        $WHEEL_DIR/mapchete_satellite*.whl \
        $WHEEL_DIR/*.whl \
    && rm $WHEEL_DIR/*

# copy masterclassification source code and install
COPY . $MASTERCLASSIFICATION_DIR
```

```
RUN pip install -e $MASTERCLASSIFICATION_DIR

WORKDIR $MASTERCLASSIFICATION_DIR
ENTRYPOINT ["python3",
"/usr/local/src/masterclassification/scripts/classify_parcel.py"]
```

classify_parcel.py

```
#!/usr/bin/env python3

import click
import logging
from mapchete.log import set_log_level
import os
import shapely.wkt

from masterclassification.mapchete.classify import (
    classify,
    combine_tiles,
    get_stack,
    prepare_data,
    result_to_geojson
)

formatter = logging.Formatter('%(asctime)s %(levelname)s %(name)s %(message)s')
stream_handler = logging.StreamHandler()
stream_handler.setFormatter(formatter)
logger = logging.getLogger(__name__)
logger.addHandler(stream_handler)

SCRIPT_DIR = os.path.dirname(os.path.realpath(__file__))
DEFAULT_MODEL = os.path.join(
    SCRIPT_DIR,
    "../test/testdata/model/7c2ece43627b43cc8e8cec9b0c6f5bfc.zip"
)


def _set_debug_log_level(ctx, param, debug):
    if debug:
        set_log_level(logging.DEBUG)
        stream_handler.setLevel(logging.DEBUG)
        logger.setLevel(logging.DEBUG)
    return debug


@click.command()
@click.argument("input_wkt")
@click.argument("output_geojson")
@click.option(
    "--input-stac",
    type=click.Path(exists=True),
```

```
    help="STAC catalog with input data."
)
@click.option(
    "--model-path",
    type=click.Path(exists=True),
    default=DEFAULT_MODEL,
    help="Trained model to use for classification. (default:
{})".format(DEFAULT_MODEL)
)
@click.option(
    "--read-threads",
    type=click.INT,
    default=4,
    help="Number of threads to read data in parallel. (default: 4)"
)
@click.option(
    "--debug", "-d",
    is_flag=True,
    callback=_set_debug_log_level,
    help="Activate debug logging."
)
def classify_parcel(
    input_wkt,
    output_geojson,
    input_stac=None,
    model_path=None,
    read_threads=None,
    debug=False
):
    """Classify parcel using model from 2018."""
    try:
        geom = shapely.wkt.loads(input_wkt)
    except Exception as e:
        raise TypeError("invalid WKT geometry given: %s" % e)

    if input_stac and os.path.isdir(input_stac):
        input_stac = os.path.join(input_stac, "catalog.json")

    output = classify(
        stack=prepare_data(
            combine_tiles(
                get_stack(
                    geom,
                    provider=input_stac or "AWS",
                    read_threads=read_threads
                ),
            ),
            geom
        ),
        model_path=model_path,
        config_path=model_path
```

```
    )

    logger.debug(output)
    result_to_geojson(result=output, geometry=geom, geojson_path=output_geojson)

if __name__ == '__main__':
    classify_parcel()
```

IndiceeContainer.cwl

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.0
class: CommandLineTool
id: ogc-eo-app-pilot
inputs:
  wkt_geometry:
    inputBinding:
      position: 1
    type: string
  output_geojson:
    inputBinding:
      position: 2
    type: string
  staged_stac:
    inputBinding:
      position: 3
      prefix: --input-stac
    type: Directory?
outputs:
  result:
    outputBinding:
      glob: $(inputs.output_geojson)
    type: File
hints:
  DockerRequirement:
    dockerPull: registry.gitlab.eox.at/maps/masterclassification:ogc_demo
```

The Agri App Application Package can be tested locally using the `cwl-runner` tool (https://pypi.org/project/cwltool) using an input YAML file like the one below.

test_job_stac.yml

```
staged_stac:
  class: Directory
  path: "staged_stac/"
wkt_geometry: "POLYGON ((15.86310262770707 48.66955927947117, 15.86311855755744
48.66970326261455, 15.86312980743018 48.66977823837983, 15.86522000713624
48.66952195190192, 15.86519883338782 48.66919844543806, 15.86436099165035
48.66926323953117, 15.86307025032472 48.66937224210698, 15.86310262770707
48.66955927947117))"
output_geojson: parcel.geojson
```

The command to test the app looks like:

```
cwl-runner eo_agri_app.cwl testdata/test_job_stac.yml
```

5.2. Urban App

5.2.1. Overview

The Urban App provides a multi band intermediate data product (5 temporal statistics per input band (minimum, maximum, mean, standard deviation, mean slope) + Nr. inputs + Nr. valid inputs) temporally aggregating different indexes (dependent on the input data and user selection) of all selected individual scenes. This dataset removes any dependency on acquisition dates and provides a worldwide homogeneous basis - without any cloud artifacts or missing data - on which to base further analysis, like the production of the World Settlement footprint, global classifications, or machine learning analysis. Experiments with Landsat have shown that calculating a 3-year time frame results in enough usable data even in cloudy areas.

The application is based on the TimeScan processing framework, which consists of three steps (see Figure 1). To simplify the workflow and the dependencies on external data within this pilot, the first two steps – Data2TimeSeries (`Data2TimeS`) and TimeSeries2Stats (`TimeS2Stats`) – have been provided as two application packages. The first Data2TimeSeries is a scene-based processing step to calculate different indexes, such as the Normalized Difference Vegetation Index (NDVI). The TimeSeries2Stats afterwards calculates a temporal aggregation from the whole available time-series of index data. The application is split into two parts to allow for platforms to first parallelize the scene-based processing (`fan-out` processing pattern) and then run the second aggregation step once all first steps have been completed.

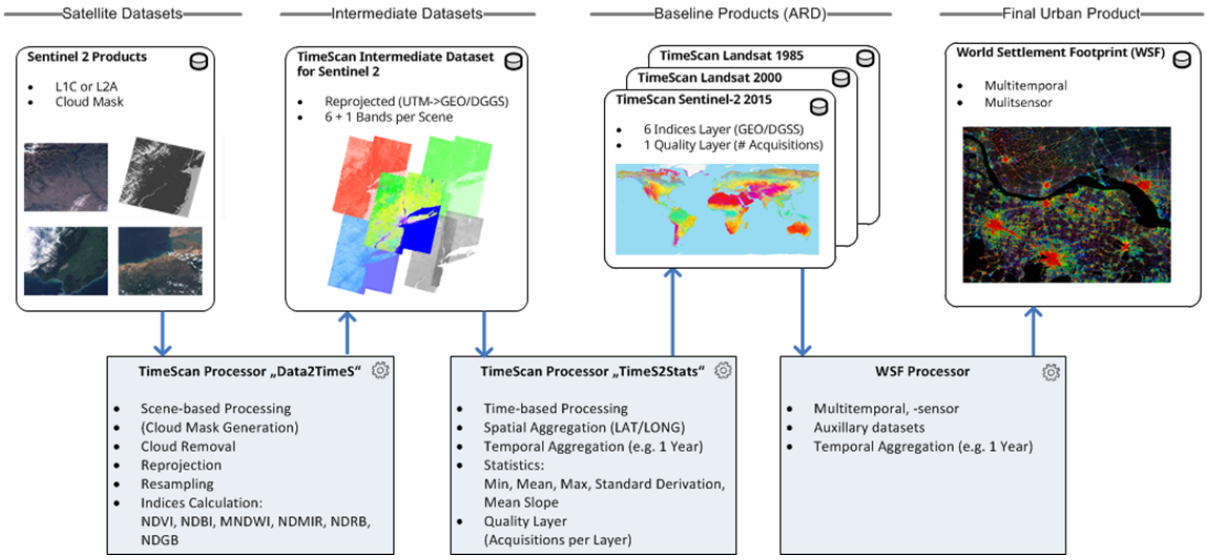

Figure 1. TimeScan processing workflow (© DLR)

5.2.2. Inputs

Step 1: Index calculation

The input of the index calculation step is a single Sentinel-2 scene. Within the runtime environment of the application the scene needs to be available as a folder. Thus, platform providers need to stage in the input data to be available locally (see input examples below).

Example input of Spacebel platform:

```
nfs://nfs1.ogc.geo-
cloud.eu/datamanager/public/Sentinel_2/2019/06/30/S2B_MSIL1C_20190630T105039_N0207_R05
1_T31UFR_20190630T125411/_value/S2B_MSIL1C_20190630T105039_N0207_R051_T31UFR_20190630T
125411.SAFE
```

Example input of Terradue platform:

```
https://catalog.terradue.com/sentinel2/search?format=json&uid=S2B_MSIL1C_20200618T0223
29_N0209_R103_T51NTH_20200618T035332
```

Step 2: Temporal aggregation

The input of this process is a list of scene-based indexes (e.g., NDVI) for the same area (e.g., Sentinel-2 tile), ideally processed with the index calculation step.

Example inputs of Spacebel platform:

```
nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/dlr-indicee-cwl/2020/06/18/0ac30e10-
65e2-4558-b8e5-
c6d96ac865c0/outputs/S2A_MSIL1C_20200608T112121_N0209_R037_T28QED_20200608T131329_indi
zes.tif
nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/dlr-indicee-cwl/2020/06/18/4540d4eb-
8724-46b1-89fc-
e9f6a763c3e7/outputs/S2B_MSIL1C_20200603T112119_N0209_R037_T28QED_20200603T141210_indi
zes.tif
nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/dlr-indicee-cwl/2020/06/18/9fa1d1ee-
407c-4c0f-ab50-
fb72bcb8df64/outputs/S2B_MSIL1C_20200613T112119_N0209_R037_T28QED_20200613T141210_indi
zes.tif
```

5.2.3. Processing

The Urban App is based on the internal TimeScan processing framework, which consists of multiple executables dealing with data download, data preparation, and data processing. Two executables have been used in this pilot for index calculation and temporal aggregation:

1. S2Indicee
This program calculates indexes for Sentinel-2 and Landsat TM/ETM+/OLI Level-2A data, which are selected during runtime - in this example the Normalized Difference Vegetation Index (`ndvi`).

Example command line request:

```
S2Indicee -t $inputFile -i ndvi -o $outputFile -writeSingleOutputImage
```

2. RasterDerivates
This program calculates the temporal maximum, minimum, mean, standard deviation, and median absolute deviation for every pixel and band of the input images. Moreover, the last two bands contain the number of valid observations for every pixel and the total number of observations per pixel.

Example command line request:

```
RasterDerivates  -ifl inputfilelist.txt -o $outputFile
```

5.2.4. Outputs

Step 1: Index calculation
The application produces a single GeoTIFF file with one band per selected index, which in this pilot is the NDVI. Additionally, an XML metadata file is provided as second output, which can be used by the platform provider to register the data into a metadata catalogue.

Step 2: Temporal aggregation
The application produces a single GeoTIFF file with multiple bands, which contains statistics of the

temporal aggregation (min, max, mean, standard deviation, median absolute deviation, valid observations, total number of observations). Additionally, an XML metadata file is provided as second output, which can be used by the platform provider to register the data into a metadata catalog.

5.2.5. Metadata

For each output file of the application described in the previous chapter, an XML metadata file based on the OGC Earth Observation Metadata profile of Observations & Measurements [https://docs.opengeospatial.org/is/10-157r4/10-157r4.html] specification is produced.

A template has been generated (see code block below) for both applications, which contains information specific for the output file, application, and platform:

- Output-specific information (generated based on the input file/s)
  - begin date
  - end date
  - bounding box
  - identifier
- Platform-specific information (needs to be provided by the platform to the application package)
  - creation date / processing date
  - processing center
- Application-specific information (provided by the application package):
  - processor name
  - processor version

Metadata template:

```
<?xml version="1.0" encoding="utf-8"?>
<eop:EarthObservation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xmlns:xlink="http://www.w3.org/1999/xlink"
                      xmlns:gml="http://www.opengis.net/gml/3.2"
                      xmlns:eop="http://www.opengis.net/eop/2.1"
                      xmlns:ssp="http://www.opengis.net/ssp/2.1"
                      xmlns:ows="http://www.opengis.net/ows/2.0"
                      xmlns:swe="http://www.opengis.net/swe/1.0"
                      xmlns:om="http://www.opengis.net/om/2.0"
                      xsi:schemaLocation="http://www.opengis.net/ssp/2.1
../xsd/ssp.xsd">
    <om:phenomenonTime>
        <gml:TimePeriod>
            <gml:beginPosition>$BeginDate</gml:beginPosition>
            <gml:endPosition>$EndDate</gml:endPosition>
        </gml:TimePeriod>
    </om:phenomenonTime>
```

```
    <om:resultTime>
        <gml:TimeInstant>
            <gml:timePosition>$BeginDate</gml:timePosition>
        </gml:TimeInstant>
    </om:resultTime>
    <om:procedure>
        <eop:processorName>yourProcessorName</eop:processorName>
    </om:procedure>
    <om:observedProperty xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"/>
    <om:featureOfInterest>
        <eop:Footprint>
            <eop:multiExtentOf>
                <gml:MultiSurface srsName="$EPSGCode">
                    <gml:surfaceMembers>
                        <gml:Polygon srsName="$EPSGCode">
                            <gml:exterior>
                                <gml:LinearRing>
                                    <gml:posList>$BoundingBox</gml:posList>
                                </gml:LinearRing>
                            </gml:exterior>
                        </gml:Polygon>
                    </gml:surfaceMembers>
                </gml:MultiSurface>
            </eop:multiExtentOf>
            <gml:locationName></gml:locationName>
        </eop:Footprint>
    </om:featureOfInterest>
    <eop:metaDataProperty>
        <eop:EarthObservationMetaData>
            <eop:identifier>$productID</eop:identifier>
            <eop:creationDate>$prodDate</eop:creationDate>
            <eop:parentIdentifier>None</eop:parentIdentifier>
            <eop:acquisitionType>NOMINAL</eop:acquisitionType>
            <eop:productType>$productType</eop:productType>
            <eop:status>ARCHIVED</eop:status>
            <eop:processing>
                <eop:ProcessingInformation>
                    <eop:processingCenter>$processingCenter</eop:processingCenter>
                    <eop:processingDate>$prodDate</eop:processingDate>
                    <eop:method>$method</eop:method>
                    <eop:processorName>$processorName</eop:processorName>
                    <eop:processorVersion>$processorVersion</eop:processorVersion>
                    <eop:nativeProductFormat>GeoTIFF</eop:nativeProductFormat>
                    <eop:processingMode>$processingMode</eop:processingMode>
                </eop:ProcessingInformation>
            </eop:processing>
        </eop:EarthObservationMetaData>
    </eop:metaDataProperty>
</eop:EarthObservation>
```

5.2.6. Application Package

Two application packages have been developed, one for index calculation and one for temporal aggregation. An application package consists of three parts:

- Executable
- Environment (e.g., Docker container)
- CWL description

Executable
The executable of the application package parses the arguments, which are passed by the CWL execution, defines the output file, and conducts the further processing (layerstacking, index calculation, metadata creation). For each of the application packages a script has been developed (`simpleSentinel2NDVIIndex.sh` and `simpleTemporalAggregate.sh`).

simpleSentinel2NDVIIndex.sh

```
#!/bin/bash

# Get arguments
while [ -n "$1" ]; do # while loop starts
    case "$1" in
    --folder)
        sentinel2Folder="$2"
        echo "--folder option passed, with value $sentinel2Folder"
        # check for subdirectory (relevant for SpaceBel platform)
        subFolder=$(ls -d $sentinel2Folder/*.SAFE)
        if [ $? -eq 0 ]; then
            sentinel2Folder=$subFolder
            echo "found .SAFE directory $sentinel2Folder"
        fi
        shift
        ;;
    esac
    shift
done

# Set output file
outputFile=$(basename ${sentinel2Folder/.SAFE/})"_indizes.tif"

# General processing
echo "Layerstacking and resampling Sentinel2"
create10mSentinel2vrt.sh $sentinel2Folder

inputVrt=$(ls *vrt)

echo "Starting Index Calculation"
S2Indicee -t $inputVrt -i ndvi -o $outputFile -writeSingleOutputImage -oExt ""

echo "Generating Metadata"
generateMetadata.sh ${inputVrt/vrt/xml} $inputVrt $sentinel2Folder
```

simpleTemporalAggregate.sh

```
#!/bin/bash

# Get arguments
while [ -n "$1" ]; do # while loop starts
        case "$1" in
        --inputs=*)
                echo ${1#*=} | cut -f 1- -d "," --output-delimiter=$'\n' >
inputfilelist.txt
                echo "--inputs option passed, with value ${1#*=}"
                shift
                ;;
        esac
        shift
done

# Set output file
outputFile=stats.tif

# General processing
echo "Starting temporal aggregation"
RasterDerivates -naIn -9999 -ifl inputfilelist.txt -o $outputFile

echo "Generating Metadata"
generateMetadata.sh ${outputFile/tif/xml} $outputFile
```

Environment
This Dockerfile describes the environment where the executable is available and executed. All dependencies (e.g., GDAL, processing tools) are already resolved within the parent Docker image. The CMD command will be overwritten from the CWL execution with the executable and the arguments described in the CWL.

```
FROM registry.example.com/repository/urban-app:rastertools-gdal-ubuntu

USER root
COPY scriptsCommon /usr/local/bin
COPY scriptsCWL /usr/local/bin

# Depending on the docker host umask , the following line is optional so please keep
it
RUN  chmod -R a+x /usr/local/bin/

ARG DLRProcessingCenter="Local"
ARG DLRprocessorName="DLR_Indicee"
ARG DLRProcessorVersion="0.1"

ENV DLRProcessingCenter=$DLRProcessingCenter DLRProcessorName=$DLRProcessorName
DLRProcessorVersion=$DLRProcessorVersion

CMD ["/usr/local/bin/simpleSentinel2NDVIIndex.sh"]
```

CWL

The CWL of the application packages consists of the `CommandLineTool` class, which can be provided either in the local environment or within a Docker environment (see `DockerRequirement` block). The executable described above is linked within the `baseCommand` property. The arguments of this executable are described within the `inputs` block.

Both, the executable and the inputs lead to the following command, which is executed within the Docker environment:

```
/usr/local/bin/simpleSentinel2NDVIIndex.sh  --folder  /var/lib/cwl/stg9e2e27a2-3b6d-46d3-9032-
13840a2c9b01/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T130159.SAFE
```

As the processing of the Urban App has been split into two steps (index calculation and temporal aggregation), two CWL files have been developed (see IndiceeContainer.cwl and TemporalAggregateContainer.cwl).

IndiceeContainer.cwl

```
baseCommand: /usr/local/bin/simpleSentinel2NDVIIndex.sh
class: CommandLineTool
id: node
inputs:
  scene:
    inputBinding:
      position: 1
      prefix: --folder
    type: Directory
outputs:
  result:
    outputBinding:
      glob: '*.tif'
    type: File
  metadata:
    outputBinding:
      glob: '*.xml'
    type: File
stderr: std.err
stdout: std.out
hints:
  DockerRequirement:
    dockerPull: registry.example.com/repository/urban-app:simple-ndvi-cwl
cwlVersion: v1.0
```

TemporalAggregateContainer.cwl

```
baseCommand: /usr/local/bin/simpleTemporalAggegate.sh
class: CommandLineTool
id: node
inputs:
  indicees:
    inputBinding:
      position: 1
      prefix: --inputs=
      itemSeparator: ","
      separate: false
    type:
      items:
      - File
      type: array
outputs:
  result:
    outputBinding:
      glob: 'stats.tif'
    type: File
  metadata:
    outputBinding:
      glob: 'stats.xml'
    type: File
stderr: std.err
stdout: std.out
hints:
  DockerRequirement:
    dockerPull: registry.example.com/repository/urban-app:simple-temporalaggregate-cwl
cwlVersion: v1.0
```

CWL Local execution

The CWL can be tested locally with the `cwl-runner` tool (https://pypi.org/project/cwltool). Besides the CWL file an input file is necessary to use this tool:

Example IndiceeContainer.yaml

```
scene: {
  class: Directory,
  path: /data/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T130159.SAFE
}
```

Example TemporalAggregateContainer.yaml

```
indicees:
- {class: File, path:
/data/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T130159_indizes.tif}
- {class: File, path:
/data/S2B_MSIL1C_20200416T101549_N0209_R065_T33UUP_20200416T123638_indizes.tif}
- {class: File, path:
/data/S2B_MSIL1C_20200426T101549_N0209_R065_T33UUP_20200426T131809_indizes.tif}
```

Example execution with `cwl-runner`

```
cwl-runner IndiceeContainer.cwl IndiceeContainer.yaml
```

CWL Docker environment

The `DockerRequirement` leads to an execution of the application within the Docker image provided. A `File` or `Directory` input type will be automatically staged (mounted) into the Docker environment. In this example, the following docker command is being executed from the `cwl-runner` tool:

```
docker \
    run \
    -i \
    --mount=type=bind,source=/tmp/wvtsil8u,target=/kCKIpt \
    --mount=type=bind,source=/tmp/771myq2s,target=/tmp \

--mount=type=bind,source=/data/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T1
30159.SAFE,target=/var/lib/cwl/stg9e2e27a2-3b6d-46d3-9032-
13840a2c9b01/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T130159.SAFE,readonl
y \
    --workdir=/kCKIpt \
    --read-only=true \
    --log-driver=none \
    --user=4020409:2222 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/kCKIpt \
    --cidfile=/tmp/xc7k_yyu/20200708121429-782865.cid \
    registry.example.com/repository/urban-app:simple-ndvi \
    /usr/local/bin/simpleSentinel2NDVIIndex.sh \
    --folder \
    /var/lib/cwl/stg9e2e27a2-3b6d-46d3-9032-
13840a2c9b01/S2B_MSIL1C_20200406T101559_N0209_R065_T33UUP_20200406T130159.SAFE >
/tmp/wvtsil8u/std.out 2> /tmp/wvtsil8u/std.err
```

5.3. Deployment

The application packages have been deployed in the following platforms:

- Terradue
- Spacebel
- EOxHub

The application packages described above could be deployed on Spacebel and EOxHub platforms with no further changes. For the Terradue platform, changes to the CWL document were necessary, as the `Workflow` base class is needed for the `fan-out` design pattern. Furthermore, metadata about the Web Processing Service (WPS) process has been added to the CWL description.

CWL for Terradue platform

```
$graph:
- baseCommand: /usr/local/bin/simpleSentinel2NDVIIndex.sh
  class: CommandLineTool
  hints:
    DockerRequirement:
      dockerPull: registry.gitlab.com/ogceo/urban-app:simple-ndvi-cwl-0.02
  id: node
  inputs:
    arg1:
      inputBinding:
        position: 1
        prefix: --folder
      type:
        Directory
  outputs:
    results:
      outputBinding:
        glob: .
      type: Any
- class: Workflow
  doc: DLR Sentinel-2 indexes
  id: dlr_s2_indexes
  inputs:
    scene:
      doc: Sentinel-2 L1C acquisitions
      label: Sentinel-2 L1C acquisitions
      type: Directory
  label: DLR Sentinel-2 indexes
  outputs:
  - id: wf_outputs
    outputSource:
    - first/results
    type:
      items: Directory
      type: array
  steps:
    first:
      in:
        arg1: scene
      out:
      - results
      run: '#node'
cwlVersion: v1.0
```

Chapter 6. Exploitation Layer

6.1. Overview

The exploitation platform EOxHub uses the experimental eWPS software implementing a WPS server to provide ADES functionality.

Some of the important characteristics of the eWPS software are:

- Execution of applications as docker containers inside of a Kubernetes cluster (containers are executed asynchronously as Docker jobs)
- Support for executing CWL based applications
- Support for WPS-T based ADES, both XML and JSON

The platform is aiming to take advantage of as many Kubernetes features as possible and to expose them. Particularly in the current implementation eWPS allows only asynchronous execution and reference based data transmission (applying only for `ComplexData`, `LiteralValues` are transmitted as expected).

As previously mentioned eWPS aims to support both, the legacy WPS2.0 XML specification and REST Bindings specified in Testbed 13.

6.2. Supported Deployment Profiles

eWPS supports the following deployment profiles

- Dockerized Applications: http://www.opengis.net/profiles/eoc/dockerizedApplication
- Workflow Applications: http://www.opengis.net/profiles/eoc/workflow

For both types of deployment profiles eWPS provides staging of input data. This is achieved by means of an init container responsible for retrieving the data and, depending on mime-type, deflating the data.

6.2.1. Dockerized Applications

In the listing dlr_IndiceeCalculator.xml the DLR IndiceeCalculator application is defined as a dockerized application (see `wps:DeploymentProfileName`). This example uses the XML based representation of the `DeployProcess` operation.

dlr_IndiceeCalculator.xml

```
<?xml version="1.0"?>
<wps:DeployProcess xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:ows=
"http://www.opengis.net/ows/2.0" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi
="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.opengis.net/wps/2.0 ../wps.xsd" service="WPS" version="2.0.0">
  <wps:ProcessDescription>
```

```
    <wps:ProcessOffering jobControlOptions="async-execute dismiss" outputTransmission
="value reference" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:ows=
"http://www.opengis.net/ows/2.0" xmlns:owc="http://www.opengis.net/owc/1.0" xmlns:eoc
="http://www.opengis.net/wps/2.0/profile/tb13/eoc" xmlns:xlink=
"http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
https://raw.githubusercontent.com/spacebel/common-xml/master/52n-ogc-
schema/src/main/resources/META-INF/xml/wps/t/wps.xsd http://www.opengis.net/ows/2.0
http://schemas.opengis.net/ows/2.0/owsAll.xsd">
      <wps:Process>
        <ows:Title>IndiceeCalculator</ows:Title>
        <ows:Abstract>The Hello World process demonstrate a sample Docker process
without any inputs or outputs.</ows:Abstract>
        <ows:Identifier>IndiceeCalculator</ows:Identifier>
        <ows:AdditionalParameters>
          <ows:AdditionalParameter>
            <ows:Name>ImageReference</ows:Name>
            <ows:Value>registry.gitlab.com/ogceo/urban-app:simple-ndvi-spacebel-
0.02</ows:Value>
          </ows:AdditionalParameter>
          <ows:AdditionalParameter>
            <ows:Name>ImageReference2</ows:Name>
            <ows:Value>registry.gitlab.com/ogceo/urban-app:simple-ndvi-spacebel-
0.02</ows:Value>
          </ows:AdditionalParameter>
        </ows:AdditionalParameters>
        <wps:Input minOccurs="0">
          <ows:Title>File to Upload</ows:Title>
          <ows:Abstract>Folder of single Sentinel-2 L1C scene</ows:Abstract>
          <ows:Identifier>INPUT1</ows:Identifier>
          <ows:AdditionalParameters>
            <ows:AdditionalParameter>
              <ows:Name>EnvironmentVariable</ows:Name>
              <ows:Value>WPS_INPUT_INPUT1</ows:Value>
            </ows:AdditionalParameter>
          </ows:AdditionalParameters>
          <wps:ComplexData>
            <wps:Format mimeType="application/gep" default="true"/>
          </wps:ComplexData>
        </wps:Input>
        <wps:Output>
          <ows:Title>Output text</ows:Title>
          <ows:Abstract>Output text</ows:Abstract>
          <ows:Identifier>OUTPUT1</ows:Identifier>
          <ows:AdditionalParameters>
            <ows:AdditionalParameter>
              <ows:Name>EnvironmentVariable</ows:Name>
              <ows:Value>WPS_OUTPUT_OUTPUT1</ows:Value>
            </ows:AdditionalParameter>
          </ows:AdditionalParameters>
          <wps:ComplexData>
```

```
            <wps:Format mimeType="application/gep" default="true"/>
          </wps:ComplexData>
        </wps:Output>
        <wps:Output>
          <ows:Title>Metadata</ows:Title>
          <ows:Abstract>Metadata</ows:Abstract>
          <ows:Identifier>METADATA</ows:Identifier>
          <ows:AdditionalParameters>
            <ows:AdditionalParameter>
              <ows:Name>EnvironmentVariable</ows:Name>
              <ows:Value>WPS_OUTPUT_METADATA1</ows:Value>
            </ows:AdditionalParameter>
          </ows:AdditionalParameters>
          <wps:ComplexData>
            <wps:Format mimeType="application/gep" default="true"/>
          </wps:ComplexData>
        </wps:Output>
      </wps:Process>
    </wps:ProcessOffering>
  </wps:ProcessDescription>
  <wps:DeploymentProfileName>
http://www.opengis.net/profiles/eoc/dockerizedApplication</wps:DeploymentProfileName>
</wps:DeployProcess>
```

Internally the `DeployProcess` operation does not have any side effect and also, in the current implementation, it does not check for the effective existence of the referenced docker container.

For the `dockerized application` deployment profile the effective execution maps all the input arguments as environment variables accessible by the docker container using the naming convention outlined in the standard.

In the case of `ComplexData` inputs the data is staged locally (for most `mime-types`), with its effective location mapped as an environment variable, following the same conventions as the input arguments. Data staging is further discussed in the Stage In data - Raw data vs STAC section.

Dockerized applications can also be deployed using the REST binding of eWPS using a JSON based `application package` description, as for example in eox_jupyter.json.

eox_jupyter.json

```
{
    "processDescription": {
        "ProcessOffering": {
            "process": {
                "id": "executeNotebook",
                "title": "execute-notebook",
                "description": "Headless execution of Jupyter notebook",
                "metadata": [
                    {
                        "href": "null"
                    }
```

```
                ],
                "version": "1.0.0",
                "jobControlOptions": [
                    "async-execute",
                    "dismiss"
                ],
                "outputTransmission": [
                    "reference"
                ],
                "inputs": [
                    {
                        "id": "Notebook",
                        "title": "Jupyter notebook to execute (path relative to HOME
folder)",
                        "description": "Notebook",
                        "minOccurs": 1,
                        "maxOccurs": 1,
                        "input": {
                            "formats": [
                                {
                                    "mimeType": "text/plain",
                                    "encoding": "raw",
                                    "default": "true"
                                }
                            ]
                        }
                    },
                    {
                        "id": "Parameters",
                        "title": "Parameters represented as base64 encoded yaml
dictionary (e.g. ZGF5czogMTQK)",
                        "description": "Parameters",
                        "minOccurs": 1,
                        "maxOccurs": "unbounded",
                        "input": {
                            "formats": [
                                {
                                    "mimeType": "text/plain",
                                    "encoding": "base64",
                                    "default": "true"
                                }
                            ]
                        }
                    }
                ],
                "outputs": [
                    {
                        "id": "Result",
                        "title": "Jupyter notebook after execution",
                        "description": "Result",
                        "output": {
```

```
                            "formats": [
                                {
                                    "mimeType": "text/plain",
                                    "encoding": "raw",
                                    "default": "true"
                                }
                            ]
                        }
                    }
                ]
            }
        }
    },
    "immediateDeployment": false,
    "deploymentProfileName":
"http://www.opengis.net/profiles/eoc/dockerizedApplication",
    "executionUnit": [{
        "Unit": "eurodatacube/jupyter-user-g:0.18.4"
    }]
}
```

6.2.2. Workflow Applications

Currently the exploitation platform supports the definition and execution of the following types of CWL based workflows:

- `CommandLineApplication`: This type of CWL applications are aimed to wrap simple command line applications. Usually this results in a single docker container execution.
- `Workflow`: This type of CWL applications is composed out of multiple command line applications or workflows and can be used to parallelize applications.

For both types of applications the exploitation platform uses the http://www.opengis.net/profiles/eoc/workflow deployment profile.

Similarly to the other implementations, eWPS automatically extracts WPS input data based on the CWL defined mapping (as much as possible CWL data types to WPS data types).

It is important to mention that eWPS allows both, embedding the CWL definition as an execution unit and also reference remotely defined CWL workflows.

The CWL support offered by eWPS is based on a modified version of the Calrissian project: https://gitlab.dev.info.uvt.ro/sage/calrissian. The calrissian project provides CWL execution on top of Kubernetes, running command line applications in isolated Kubernetes jobs.

Command Line Applications

A typical CWL based application is depicted in dlr_application_package_command_line_tool.json. This example is based on the DLR Simple Temporal Aggregate application.

The CWL definition is embedded in the application package as an individual `executionUnit`. eWPS

supports also the retrieval of remote CWL definitions. The referenced CWL definition is retrieved at deployment time and reused in the subsequent execute operations.

dlr_application_package_command_line_tool.json

```
{
   "immediateDeployment":"false",
   "executionUnit":[
      {
         "baseCommand":"/usr/local/bin/simpleTemporalAggegate.sh",
         "class":"CommandLineTool",
         "id":"node",
         "inputs":{
            "indicees":{
               "inputBinding":{
                  "position":1,
                  "prefix":"--inputs=",
                  "itemSeparator":",",
                  "separate":false
               },
               "type":"File[]"
            }
         },
         "outputs":{
            "result":{
               "outputBinding":{
                  "glob":"stats.tif"
               },
               "type":"File"
            }
         },
         "stderr":"std.err",
         "stdout":"std.out",
         "hints":{
            "DockerRequirement":{
               "dockerPull":"registry.gitlab.com/ogceo/urban-app:simple-
temporalaggregate-cwl-0.01"
            }
         },
         "cwlVersion":"v1.0"
      }
   ],
   "deploymentProfileName":"http://www.opengis.net/profiles/eoc/workflow"
}
```

Workflow Applications

Full-fledged workflow definitions are also partially supported by the eWPS backend but are limited by the WPS data exchange characteristics in general and the eWPS implementation in particular. The listing dlr_workflow.json provides an example of an application package taking advantage of

the CWL functionality offered by the Calrissian backend.

dlr_workflow.json

```
{
    "immediateDeployment":"false",
    "executionUnit":[
       {
            "class": "Workflow",
            "cwlVersion": "v1.0",
            "id": "dlrworkflow",
            "doc": "This is a mockup of the DLR workflow for Indicee and
TemporalAggregate apps.",
            "requirements": [
                {
                "class": "MultipleInputFeatureRequirement"
                }
            ],
            "inputs": {
                "image1": {
                "type": "Directory"
                },
                "image2": {
                "type": "Directory"
                }
            },
            "outputs": {
                "output": {
                "outputSource": "temporalaggregate/result",
                "type": "File"
                }
            },
            "steps": {
                "indicee-run-1": {
                "in": {
                    "scene": {
                    "source": "image1"
                    }
                },
                "out": [
                    "result"
                ],
                "run": "https://gitlab.dev.info.uvt.ro/sage/ogc-pilot-apps/-
/raw/master/DLR/IndiceeContainer/IndiceeContainer.cwl"
                },
                "indicee-run-2": {
                "in": {
                    "scene": {
                    "source": "image2"
                    }
                },
```

```
                "out": [
                    "result"
                ],
                "run": "https://gitlab.dev.info.uvt.ro/sage/ogc-pilot-apps/-
/raw/master/DLR/IndiceeContainer/IndiceeContainer.cwl"
                },
                "temporalaggregate": {
                "in": {
                    "indicees": {
                    "source": [
                        "indicee-run-1/result",
                        "indicee-run-2/result"
                    ]
                    }
                },
                "out": [
                    "result"
                ],
                "run": "https://gitlab.dev.info.uvt.ro/sage/ogc-pilot-apps/-
/raw/master/DLR/TemporalAggregateContainer/TemporalAggregateContainer.cwl"
                }
            }
        }
    ],
    "deploymentProfileName":"http://www.opengis.net/profiles/eoc/workflow"
 }
```

The main advantage of this approach is that the CWL Engine Backend can run the independent steps in parallel, supporting multiple types of fan-in/fan-out approaches.

Unfortunately, the implementation in eWPS does not handle all types of data exchange steps that might be involved but should fit most basic applications.

Another aspect lacking in eWPS is the built-in ability of mixing steps running in multiple exploitation platforms.

6.3. Stage In data - Raw data vs STAC

Data staging proved to be one of the more challenging aspects of the pilot as it is handled differently by each platform provider.

From this perspective eWPS provides runtime data staging if needed by applications. Typically, this involves HTTP based data retrieval and storing inside the container.

For particular use cases like STAC based referenced input eWPS provides the ability of fetching data referenced inside the catalog, providing in this way a more complete data staging. Unfortunately the way STAC is organized leads to some issues that are further discussed in the section Findings and Discussed Topics (Platform provider).

6.4. Exploitation Platform

EOxHub [https://hub.eox.at/] is a Kubernetes-based Earth Observation (EO) exploitation and processing platform, which has been extended to support the deployment and execution of processing jobs defined as OGC EO application packages. These additional capabilities have been enabled during the OGC EO Applications Pilot through evolving and integration of the eWPS processing solution, a WPS 2.0 server supporting WPS-T also used in previous OGC Testbeds and described in the previous sections above, which now

- supports application descriptions also via CWL, both Single-Step processes via CWL CommandLineTool as well as Multi-Step processes via CWL workflow,
- provides a generic data staging mechanism to shift EO data between the platform and the processor,
- integrates natively into Kubernetes, bridging between processing jobs and the offered platforms job functionality.

The EOxHub workspace [https://eurodatacube.com/marketplace/infra/edc_eoxhub_workspace] capabilities are deeply integrated with Euro Data Cube [https://eurodatacube.com/] managed service offerings for global EO data access, represented as data cubes.

When used in conjunction with EO application packages, EOxHub as an underlying platform handles the aspects and concerns for application package execution described in the following sections.

6.4.1. Domain specific aspect

Discovery and retrieval of EO data

Discovery and retrieval of EO data: direct data access to huge object storage archives (e.g. AWS, Mundi DIAS [https://mundiwebservices.com/]) as well as specific managed services (e.g. EDC Sentinel Hub [https://eurodatacube.com/marketplace/services/edc_sentinel_hub] for EO raster data).

EO platforms are specifically built for this use-case, they are physically located close to EO data sources, have dedicated data connectors bundled and offer specific managed services for EO data access.

The EOxHub platform supports direct data access to huge object storage archives like on AWS or on the Mundi DIAS and offers various managed services, like the EDC Sentinel Hub Service for global EO raster data access.

Reuse of output results

Reuse of output results: use EDC Sentinel Hub BYOC functionality to register intermediate or final processing results.

All generated processing artifacts should be properly annotated by the application package e.g. by including additional metadata files describing the processing results. The platform can leverage this information and make the outputs discoverable for subsequent processing.

EOxHub offers the so-called EDC Sentinel Hub “Bring your own data” (BYOD) functionality, a specific API for registration of additional EO raster data.

6.4.2. Cross cutting concerns

Multi-tenancy

Multi-tenancy: isolated workspaces to separate user details, processes, configuration, and processing artifacts.

EOxHub offers a hosted JupyterLab [https://jupyterlab.readthedocs.io] environment with persistent storage allowing the user to host own customized application packages to manage processes, configuration settings, and previous processing artifacts.

Resource quotas

Resource quotas: apply resource guarantees and constraints based on individual user resource subscriptions to match the individual processes demands with the available platform resources (like CPU or memory).

The eWPS processing solution was extended so that all processes forward necessary resource constraints as stated in the application package to the EOxHub runtime environment for execution, to establish resource guarantees as well as communicate resource limits to not exhaust the platform.

Operational metrics

Operational metrics: allowing the end-user to understand the runtime behavior of processes and the platform provider to do proper reporting and billing.

Having the required managed service subscriptions and resource needs defined in the application packages allows the platform to infer which operational metrics should be tracked.

EOxHub uses Elasticsearch and Prometheus to collect application information and operational metrics. The Kubernetes native integration of eWPS into Kubernetes allows to leverage these established mechanisms for large-scale processing.

Chapter 7. Findings and Discussed Topics (Platform provider)

7.1. Application Package (Environment variables, CWL, etc.)

The Application Package specification using CWL represents one positive evolution over the standard approach as it simplifies the developer experience and allows for a portable approach for application execution.

Besides its advantages, the CWL-based approach created some challenges:

- The CWL types do not map directly to WPS types: for mitigating this in the eWPS platform we introduced some heuristics for properly translating CWL types to the corresponding WPS types (also the other way around).
- The CWL reference toolchain does not directly support Kubernetes.
- The CWL reference toolchain has a hard restriction of directly invoking docker. This proves to be challenging due to the fact that some execution platforms (including ours) do not support `docker in docker`.

Overall, we consider the CWL approach promising and a step forward towards portability and ease of use.

7.2. Execution patterns (fan-in, fan-out)

The fan-in pattern describes a processing pattern where multiple time slices of data are transformed into a single output. Examples for these patterns are merging multiple Sentinel-2 scenes into one large mosaic or using a time stack of Sentinel-2 scenes to apply a classification algorithm as in the Agri App described above.

The challenge here is to describe the input data stack (i.e. a collection of Sentinel-2 products for the given AOI and TOI) in a standardized way so that the application developers can develop towards this standard and don't have to implement one connector for each platform's data archive.

Further, if a process requires a longer time series the number of input parameters would be quite large if every product maps to one input. The inputs list would even get larger if every single band of every single input would have to be mapped to an input parameter.

The usage of STAC could solve this problem. The application developer would need to handle a STAC catalog containing the required time stack. Within STAC the access to the single products, bands, cloud masks, and other metadata is standardized. This way it is unambiguous for the app developer where to find the required data instead of having to deal with slightly different OpenSearch implementations.

Support for full-fledged CWL Workflow application packages is of particular significance as it allows the implementation of the `fan-in` and `fan-out` execution patterns. The support is constrained

by the underlaying CWL implementation, in our case by the Calrissian engine.

These patterns have been demonstrated with the DLR application packages and are completely supported by the eWPS platform.

The handling of input/output date is transparently handled by Calrissian using Kubernetes native volumes (typically using NFS volume claims).

7.3. EO Data Inputs

As part of the EO Apps Pilot eWPS was extended with support for STAC, adding the ability of locally staging STAC catalogs.

The primary advantage is that the input specification is abstracted, facilitating application portability: applications do not have to support the multitude of conventions employed by the various execution platforms, they only need to be able to handle STAC catalogs, eventually retrieving remotely stored data.

For further supporting STAC usage eWPS added support for data staging specialized for STAC catalogs, particularly eWPS is able to handle all three catalog types:

- Self Contained
- Absolute Published
- Relative Published

It needs to be noted that staging all these types of catalogs proved itself challenging mostly due to the integration with the PySTAC library.

Another issue related to STAC and impacting operational situations is handling of non `self-contained` catalogs, particularly catalogs that reference non `HTTP` references, as for example `S3` resources. For this situations credential passing needs to be further investigated and potentially standardized.

7.4. Workflows

With the aforementioned eWPS support for both CWL base classes - `CommandLineTool` as well as `Workflow` - the platform is now able to execute single-step as well as multi-step processes (="workflows"). However, the workflow engine acts as a 'blackbox' without any interaction with the platform during execution. This means that EO Data Discovery only takes place at the start, with the included data staging mechanism as well as the Results registration with the Outputs Metadata handling. This occurs at the end of the workflow, with all the data management between the individual steps being handled internally by the workflow engine.

Chapter 8. Findings and Discussed Topics (Application developer)

8.1. Application Package

8.1.1. Environment variables + WPS description file vs. CWL

The initial test version of the Urban application package was based on a Docker container, environment variables as inputs and outputs as well as a WPS description file, which was the legacy system supported by the Spacebel platform and allowed us to quickly get started. Within the execution script (e.g., `simpleSentinel2NDVIIndex.sh`) the inputs and outputs are set through environment variables defined in the WPS description file.

Inputs and outputs set through environment variables

```
sentinel2Folder=$WPS_INPUT_INPUT1
outputFile=$WPS_OUTPUT_OUTPUT1
metadataFile=$WPS_OUTPUT_METADATA1
```

WPS description: IndiceeContainer_WPS.xml

```
<wps:ProcessOffering jobControlOptions="async-execute dismiss" outputTransmission=
"value reference" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:ows=
"http://www.opengis.net/ows/2.0" xmlns:owc="http://www.opengis.net/owc/1.0" xmlns:eoc
="http://www.opengis.net/wps/2.0/profile/tb13/eoc" xmlns:xlink=
"http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/2.0
https://raw.githubusercontent.com/spacebel/common-xml/master/52n-ogc-
schema/src/main/resources/META-INF/xml/wps/t/wps.xsd http://www.opengis.net/ows/2.0
http://schemas.opengis.net/ows/2.0/owsAll.xsd">
  <wps:Process>
    <ows:Title>IndiceeCalculator</ows:Title>
    <ows:Abstract>The Hello World process demonstrate a sample Docker process without
any inputs or outputs.</ows:Abstract>
    <ows:Identifier>IndiceeCalculator</ows:Identifier>
    <ows:AdditionalParameters>
      <ows:AdditionalParameter>
        <ows:Name>ImageReference</ows:Name>
        <ows:Value>registry.example.com/repository/urban-app:simple-ndvi-
spacebel</ows:Value>
      </ows:AdditionalParameter>
    </ows:AdditionalParameters>
    <wps:Input minOccurs="0">
      <ows:Title>File to Upload</ows:Title>
      <ows:Abstract>Folder of single Sentinel-2 L1C scene</ows:Abstract>
      <ows:Identifier>INPUT1</ows:Identifier>
      <ows:AdditionalParameters>
```

```
        <ows:AdditionalParameter>
          <ows:Name>EnvironmentVariable</ows:Name>
          <ows:Value>WPS_INPUT_INPUT1</ows:Value>
        </ows:AdditionalParameter>
      </ows:AdditionalParameters>
      <wps:ComplexData>
        <wps:Format mimeType="application/gep" default="true"/>
      </wps:ComplexData>
    </wps:Input>
    <wps:Output>
      <ows:Title>Output text</ows:Title>
      <ows:Abstract>Output text</ows:Abstract>
      <ows:Identifier>OUTPUT1</ows:Identifier>
      <ows:AdditionalParameters>
        <ows:AdditionalParameter>
          <ows:Name>EnvironmentVariable</ows:Name>
          <ows:Value>WPS_OUTPUT_OUTPUT1</ows:Value>
        </ows:AdditionalParameter>
      </ows:AdditionalParameters>
      <wps:ComplexData>
        <wps:Format mimeType="application/gep" default="true"/>
      </wps:ComplexData>
    </wps:Output>
    <wps:Output>
      <ows:Title>Metadata</ows:Title>
      <ows:Abstract>Metadata</ows:Abstract>
      <ows:Identifier>METADATA</ows:Identifier>
      <ows:AdditionalParameters>
        <ows:AdditionalParameter>
          <ows:Name>EnvironmentVariable</ows:Name>
          <ows:Value>WPS_OUTPUT_METADATA1</ows:Value>
        </ows:AdditionalParameter>
      </ows:AdditionalParameters>
      <wps:ComplexData>
        <wps:Format mimeType="application/gep" default="true"/>
      </wps:ComplexData>
    </wps:Output>
  </wps:Process>
</wps:ProcessOffering>
```

However, this approach is only supported by the Spacebel platform, Terradue supports only the CWL execution. To limit the dependencies of multiple platform providers and to simplify the creation of an application package, the approach with environment variables and WPS description was changed to CWL-only as described in Chapter "EO Application Packages". With this, inputs and outputs are described within the CWL and can then be used directly within the application. Environment variables definition in the WPS description as done before are not necessary anymore and parameters can be specified exactly how the application expects them.

8.1.2. CWL definition

Although the concept of the CWL is the same for each platform provider, not all base classes are supported by each platform provider. Whereas the `CommandLineTool` class is supported by EOX and Spacebel, Terradue only supports the `Workflow` class as the base class (the `CommandLineTool` can be used in subsequent steps). EOX supports both `Workflow` and `CommandLineTool`. Thus, the application packages of the Urban App described in Chapter 5 needed to be adjusted with the `Workflow` class for the execution on the Terradue platform (see Deployment section in Chapter 5).

In addition, Terradue makes use of labels to document names and descriptions for the WPS process definition. As this is not supported by other platforms, a clear agreement on supported CWL classes and items is necessary to further standardize the application package.

Conclusion: There need to be a common agreement on the CWL base class, whether this is a `CommandLineTool` or a `Workflow`, as well as on documentation labels.

8.2. Execution patterns

Different processing design patterns - provided by Terradue - have been discussed within the pilot. Distinguishing between `fan-out` and `fan-in` pattern was very helpful in terms of designing the CWL. The `fan-out` pattern allows scene-based processing, which can automatically be executed in parallel. The `fan-in` pattern can be used for temporal aggregation in the case of the Urban App.

Both patterns can be implemented with CWL. For the `fan-out` pattern, the `Workflow` base class needs to be used in conjunction with `ScatterFeatureRequirement` to automatically conduct the processing multiple times. The `fan-in` pattern can be implemented using an `array` input type. Both have been tested locally for the Urban App. For the Spacebel platform, the `fan-out` pattern was realized by platform-specific executions as no support of the `Workflow` base class is available.

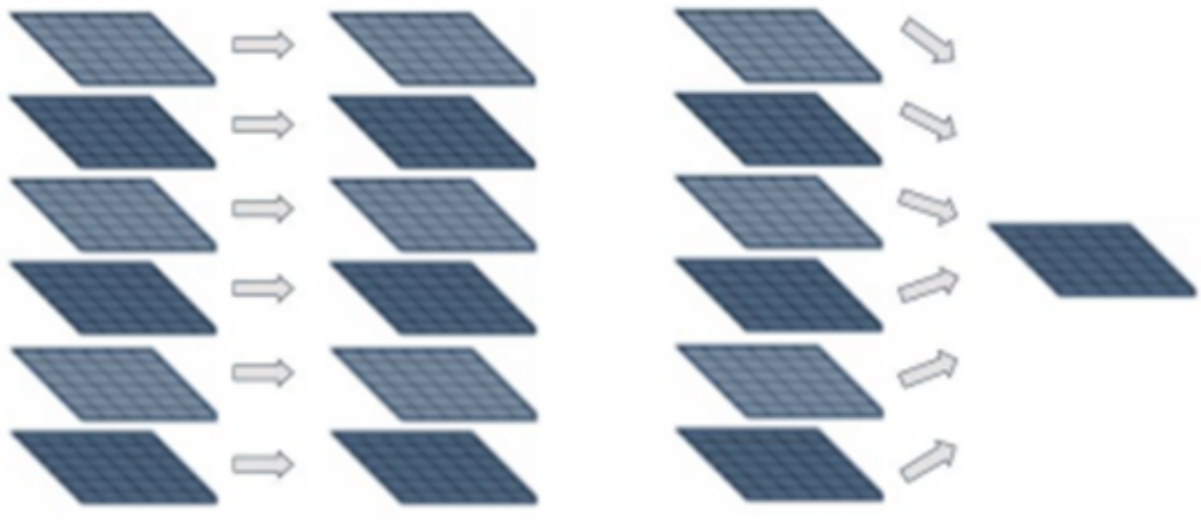

Figure 2. Processing design patterns: Fan-out (left) and fan-in (right)

8.3. EO Data Inputs

Depending on the platform, data can be provided with different services as well as in different data formats and data structures (e.g., local path, S3 bucket, OGC Web Coverage Service (OGC WCS),

HTTP URL, zipped or unzipped). Both, platform and application package, need to describe which kind of services and structures are supported. Platforms can also provide `staged-in` data in a format that is required by the application (e.g., automatically download data from OGC WCS and provide the dataset to be accessed as a local file).

Although the objective of our Urban App pilot proposal was to change from individual file handling to the use of OGC WCS as input, most of the platform providers have their data available as files either mounted into the file system or provided as an S3 bucket. This also enables the platform to discover and provide data as it is requested by the user. The application does not need to handle this, as it might also be handled differently on several platforms.

In addition, there were some discussions on whether it is possible to standardize the input of Earth Observation data, e.g., with the use of the STAC specification. This would allow not only to provide links to the geospatial files, but also to provide additional metadata, which can be used by the application without extracting the metadata on its own.

Finally, the assumption of the Urban application package is currently, that the input data is available as a local folder or file path. Further discussions are necessary for a standardized EO data input service, format, or structure.

8.4. EO Data Discovery

Discussions took place about whether the application or the platform needs to conduct data discovery requests. Although standardized specifications, such as OpenSearch, are available, there are often slight differences in the implementation in regard to providing specific information about Earth Observation data. With this, each application needs to know these differences and needs to comply with different implementations of discovery services when it is deployed on multiple platforms. Thus, data discovery has been kept out of our application packages and left to the cloud platform. Platforms need to provide tools for users to discover data and to use (e.g., stage in) this data for the processing of an application package (see previous **EO Data Inputs** section).

8.5. Outputs Metadata

Multiple output files of an application package need to be linked with multiple metadata files. Currently, there is no defined approach available to link a metadata file with a specific output file. The approach of using STAC was discussed, which can evolve to a solution (same as the EO data input standardization).

Further, platform-specific metadata values (e.g., name of the processing platform) need to be handed over into the application package, if the metadata needs to be created fully in the application. Environment variables can be used in conjunction with CWL for this, but names of environment variables need to be standardized and supported by platform providers.

8.6. Workflows

The support of workflows seems to be unclear so far and dependent on the platform provider. An automated workflow was set up and tested with Spacebel but only with platform-specific mechanisms. With the EOX platform a workflow based on CWL is supported and has been tested

manually. It is also unclear which role the ADES/EMS services play in terms of conducting workflows.

Chapter 9. Conclusions

9.1. Platform provider

Supporting a well-established declarative format like CWL to define application workflows gives platform providers the possibility to leverage a wide ecosystem of existing tools - available to be used by themselves to invoke and track workflow executions, but which can also be offered to application developers (e.g. to compose workflows visually) or to application consumers (e.g. to monitor workflow execution).

While direct usage of existing tools like `cwl-runner` enables the execution of single-step as well as multi-step processes in a straight-forward and conformant way, it only provides these capabilities in a blackbox style manner on top of the platform, conceptionally running on the ADES part. For the true orchestration of standalone processes, with individual data stage-in and stage-out mechanisms, possibly to be invoked on different ADES instances, a natively integrated CWL execution engine, understanding platform specifics, running on EMS and orchestrating calls on one or more ADES instances is needed.

9.2. Application developer

From the application developer's perspective the main technical outcome of the Urban app is the standardization of the application with CWL. This allows for the standardization of the processing within a Docker container and an executable together with the description of required and optional inputs and outputs. Furthermore, `fan-out` patterns and workflows can be described with CWL. Another benefit of CWL is the possibility to test the application package in a local environment with the `cwl-runner` tool.

Good discussions and exchanges between all participants were made during the pilot (e.g., CWL, output metadata, input data, processing design patterns), but more time is necessary to dive into additional topics more deeply, such as the standardization of input and output files as well as the parallelized execution of an application package.

The use of services for data access, such as the OGC Web Coverage Service, have not been investigated as most platforms provide their data for processing as files staged into the application environment. Thus, it is still open whether application packages should make use of web services for data access or rely on the data provided by each platform.

Another open topic is how the application package can register what kind of inputs/outputs formats and file access it can support (e.g. directly access S3, http(s), GDAL virtual file systems), so that platforms can choose the optimal match between platform and application and potentially skip staging in data or outright reject the package if no compatible format is available.

Finally, we got our application packages running on different platforms with the same Docker container, executable, input data, and output metadata, but slightly different CWL descriptions as the CWL base class supported by the platforms differ.

Recommendations
The following recommendations can be drawn:

- CWL demonstrated good capabilities to standardize an application package, but there needs to be a common agreement on the supported CWL items, such as the CWL base class.
- It has been shown within the pilot that no explicit process descriptions in XML are necessary as this information can be integrated in the CWL. This is preferable as it lowers the barrier of developing an application package and minimizes the number of files that have to be kept in sync on changes.
- Further discussions about the standardization of the format or structure of input and output files (e.g., through STAC as discussed in the pilot) are necessary.
- Tutorials are needed to describe in detail what is necessary for an application package and especially what is handled in detail by an ADES/EMS implementation. This is not easy to understand for users, who are not familiar with these specifications.

Appendix A: Revision History

Table 1. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
July 31, 2020	EOX team	1.0	all	initial version
August 5, 2020	I. Simonis	1.1	all	Editorial changes, full review
August 26, 2020	S. Meissl	1.2	all	Final changes based on full review comments
October 15, 2020	G. Hobona	1.3	all	Final staff review