OGC Earth Observation Applications Pilot

Spacebel Engineering Report

Publication Date: 2020-10-22

Approval Date: 2020-09-23

Submission Date: 2020-08-25

Reference number of this document: OGC 20-034

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

Category: OGC Public Engineering Report

Editor: Christophe Noël

Title: OGC Earth Observation Applications Pilot: Spacebel Engineering Report

OGC Public Engineering Report

Table of Contents

Chapter 1. Subject

This Engineering Report (ER) describes the achievements of Spacebel as a Platform Provider in the OGC Earth Observation Applications (EO Apps) Pilot and the lessons learned from the project.

Chapter 2. Executive Summary

2.1. Objectives

The objective of the project is to build a “real world” environment of the Earth Observation (EO) Exploitation Platform architecture which has been developed over the last two years as part of various OGC Innovation Program initiatives.

The first phase of the project invited Application Developers to join a requirements definition workshop: developers were informed of platform capabilities, then asked to express their requirements in terms of data discovery, data loading, data processing and result delivery. The definition of use cases directed the implementation and evaluation of the second phase: EO platform operators (including Spacebel) implemented the necessary user infrastructure to enable application developers to achieve the goals of their use cases.

As stated in the Call for Participation (CFP) of the EO Apps Pilot, the Testbed-13 reports are considered relevant here because they help to understand the design decisions in context, but the documents are superseded by Testbed-14 reports which describe the target architecture.

The architecture is essentially based on two main components:

- Execution Management Service (EMS): provides a RESTful interface (defined using OpenAPI) to register applications and build workflows from registered applications. The EMS selects the appropriate Application Deployment and Execution Service (ADES) platform to execute the processes based on the runtime input parameters (close to the data).
- Application Deployment and Execution Platform: allows to deploy, discover, and execute applications or to perform quoting requests.

2.2. Proposed EO Platform

As a Platform Provider, Spacebel proposed the Walloon EO Regions! [http://www.eoregions.com] Platform based on the specified "Geospatial Exploitation Platform" implementation. The platform’s different components and containerized applicative service chains (Docker) are running on a Kubernetes cluster on a public cloud infrastructure (Google Cloud Platform).

Application Developers selected by OGC are able to deploy their applications via application packages on the platform and execute the applications through the required specified interfaces (EMS, ADES etc.). Spacebel would support these application developers to achieve this result.

The EO Regions! [http://www.eoregions.com] Platform was originally funded by the Walloon region and is being operated by Spacebel. EO Regions! [http://www.eoregions.com] is the Walloon hub of the European EUGENIUS network and is recognized as a Copernicus Relay [https://www.copernicus.eu/sites/default/files/CopCSO_Factsheet_1.0_2020.pdf] for Wallonia by the European Commission for the promotion and support to users of Copernicus data. The main user facing Web Portal is available at http://www.eoregions.com/ and provides more background information about the platform and its objectives.

2.3. Document contributor contact points

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

Contacts

Name	Organization	Role
Christophe Noël	Spacebel s.a.	Editor

2.4. 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. References

The following normative documents are referenced in this document.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: OGC 14-065r2, OGC Web Processing Service 2.0.2 Interface Standard Corrigendum 2, 2018 [https://portal.opengeospatial.org/files/14-065r2]
- Commonwl.org: Common Workflow Language Specifications, v1.0.2 [https://www.commonwl.org/v1.0/]
- OGC: OGC 13-026r8, OGC OpenSearch Extension for Earth Observation 1.0, 2016 [https://portal.opengeospatial.org/files/13-026r8]
- OGC: OGC 10-032r8, OGC OpenSearch Geo and Time Extensions 1.0.0, 2014 [https://portal.opengeospatial.org/files/?artifact_id=56866]
- OGC: OGC 10-157r4, OGC Earth Observation Metadata profile of Observations & Measurements 1.1, 2014 [https://docs.opengeospatial.org/is/10-157r4/10-157r4.html]

Chapter 4. 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.

- **Container**

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

- **OpenAPI Document**

  A document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification (OpenAPI [https://docs.ogc.org/per/19-020r1.html#OAI])

- **OpenSearch**

  Draft specification for web search syndication, originating from Amazon's A9 project and given a corresponding interface binding by the OASIS Search Web Services working group.

- **Service interface**

  Shared boundary between an automated system or human being and another automated system or human being

- **Workflow**

  Automation of a process, in whole or part, during which electronic documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (source ISO 12651-2:2014)

4.1. Abbreviated terms

- ADES Application Deployment and Execution Service
- AOI Area Of Interest
- AP Application Package
- BPEL Business Process Execution Language
- CFP Call For Participation
- CWL Common Workflow Language
- DWG Domain Working Group
- EMS Execution Management Service
- EO Earth Observation
- EP Exploitation Platform
- ER Engineering Report
- ESA European Space Agency
- GUI Graphical User Interface
- JSON JavaScript Object Notation

- MEP Mission Exploitation Platform
- OWC OWS Context
- REST REpresentational State Transfer
- 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
- WPS-T Transactional Web Processing Service

Chapter 5. Overview

Section 6 introduces the project context, the initial requirements and baseline.

Section 7 discusses the Earth Observation Exploitation Platform aspects.

Section 8 presents the lessons learned during the project as a Platform Provider.

Section 9 describes the Technology Integration Experiments (TIE) achieved by Spacebel and Application Providers.

Section 10 provides the conclusions and recommendations for future work.

Chapter 6. Initial Context

The section provides an overview of the project requirements and the tentative baseline agreed between the Platform and Application providers.

6.1. Initial requirements

The CFP detailed a set of evaluation criteria for ranking the proposed platform of the applicants. Additionally, some starting points for improving the architecture were also identified. An overview of the main topics to be explored during the pilot is provided below:

- Discovery of applications, of input EO data and analysis results.
- Provision of data, typically using the OGC Web Coverage Service (WCS) standard as an agnostic interface to data.
- ADES and Application Package (AP) Constraints.
- Workflow language, in particular reevaluating the Common Workflow Language (CWL).
- Multiple outputs.
- Trust, Deployment and Security.
- Developer support in particular through the access to the execution logs.
- Billing and Quoting Model
- Communication Message Exchange (JSON rather than XML).
- Asynchronous Communication such as a web hook callback.
- Execution and Deployment on a (Kubernetes) cluster.

Participants shall not explore all these topics during the pilot, but experiences matching some aspects shall be captured as reflected in this engineering report.

6.2. Tentative Baseline

During the Kick Off meeting, participants & staff tried to find an agreement for a common baseline of features supported by the various platform.

As no presentation really covered a detailed description of the EMS and ADES components, those aspects have been unfortunately agreed with a very limited knowledge of the actual architecture.

6.2.1. Data Discovery

Data Discovery can be performed through either OpenSearch (two-steps) queries or OGC WMS/WCS API.

Despite that the data discovery aspects are not yet properly covered in the Testbed architecture, Testbed-14 already introduced the OpenSearch API for selecting the EO data inputs submitted to the process.

The Spacebel platform already includes an OpenSearch Catalogue, and detailed product metadata can be requested in the OGC O&M EO profile format (OGC 10-157r4). The Web Map Service (WMS) and WCS are not supported.

6.2.2. Data Retrieval

Data can be retrieved by an internal stage-in (performed by the ADES), through WMS/WCS, or using Object Storage.

The Spacebel platform stage-in requires mounting local data resources to the Kubernetes cluster node. In addition to the nominal stage-in, it is not perfectly clear if the proposed alternatives offer direct (network) access from the application, or if they suggest the support of those protocols for provisioning local files (as for the stage-in).

The Spacebel platform fetches all remote HTTP and FTP URLs and makes them available through the local file system to the processes. As discussed during OGC Testbed-14, the application descriptor should typically describe if the process supports natively HTTP or Object Storage access.

6.2.3. User Interface (EMS/ADES)

The agreed ADES/EMS interface is based on the draft OGC API – Processes specification, with REST JSON bindings, and Web Processing Service (WPS) standard.

The Spacebel platform required to be aligned with the work performed during OGC Testbed-14. However, the OGC repository hosting the mentioned OpenAPI specification does not exist anymore, so Spacebel shared its own version (OGC-ADES-Hackaton [https://app.swaggerhub.com/apis/Spacebel.be/WPS/TESTBED.HACKATON]) reviewed during the 2019 OGC API Hackathon that was held in London [1].

The baseline also recommended to explore the handling of asynchronous callback. Because most application providers focus on the platform's user interface, the priority was not given to this aspect during the project.

6.2.4. Workflows

The agreed Application Package is the CWL profile defined in Testbed-14. Additionally, bulk (parallel) processing might be investigated.

Based on the Application Providers feedback, Spacebel essentially worked on the parallelization approach, as described further in the document.

6.3. Tentative Grouping

During the Kick Off meeting, Spacebel discussed with Application Providers and tried to identify potential candidates to collaborate with. The following participants agreed to work on the EORegions! platform:

- Computer Research Institute of Montreal (CRIM) acting as an Application Provider, but also reproducing a multi-platform workflow execution as demonstrated during the OGC Testbed-14 project

- European Union Satellite Centre (SatCen) with the Coherence application.
- Pixalytics with the Mosaicing application.
- German Aerospace Center (DLR) with the Urban Indicee application.

Afterwards, ESA also decided to participate acting as an Application Provider with the Mosaicing application.

Chapter 7. Earth Observation Exploitation Platform

OGC EO Apps Pilot project invited Earth Observation platform operators to implement the OGC Earth Observation Applications Pilot architecture as it has been defined in previous IP initiatives.

Spacebel acts in this pilot as a platform provider with the *EORegions!* platform.

7.1. Architecture Technical Aspects

OGC Testbed activities in Testbed-13, Testbed-14, and Testbed-15 have contributed to the definition of an EO platform architecture that allows the deployment and execution of applications close to the physical location of the source data. The goal is to minimize data transfer between data repositories and application processes.

The main Engineering Reports describing the architecture are listed below:

- OGC 18-049r1 [https://docs.ogc.org/per/18-049r1.html] - Testbed-14: Application Package Engineering Report [2]
- OGC 18-050r1 [https://docs.ogc.org/per/18-050r1.html] - Testbed-14: ADES & EMS Results and Best Practices Engineering Report [3]

The figure below shows an overview of the Testbed-14 architecture showing the main components (EMS and ADES).

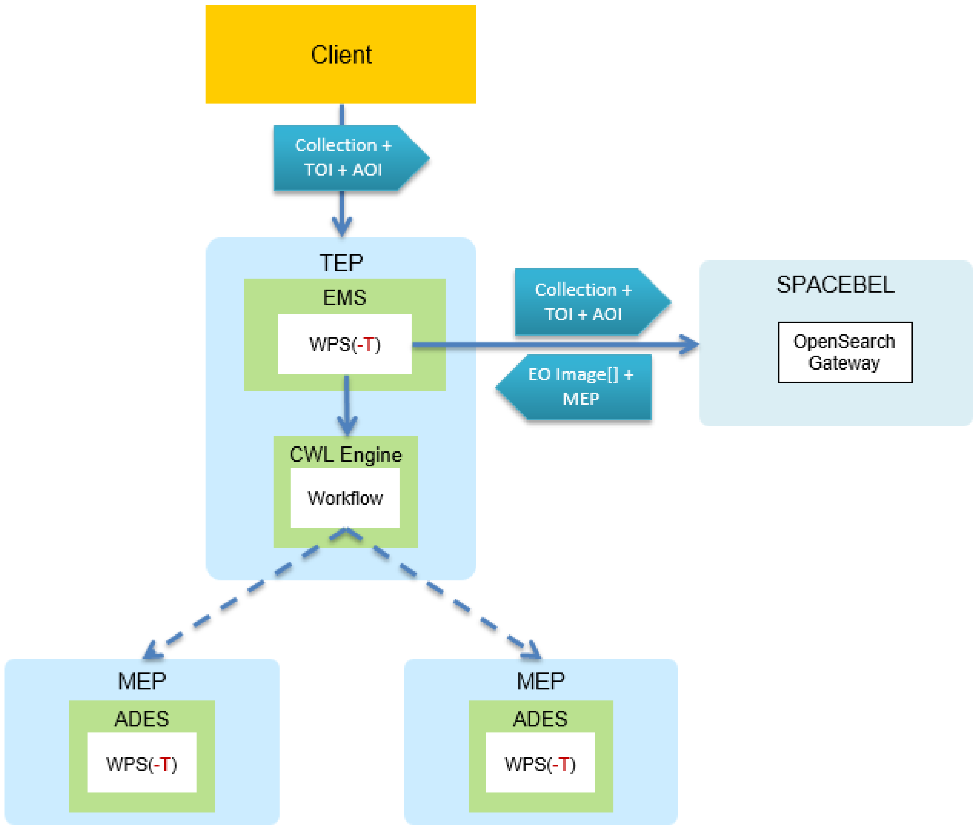


Figure 1. Testbed 14 Architecture Overview

7.1.1. Application Deployment and Execution Service

Applications developed by application providers are deployed using the Application Deployment and Execution Service (ADES). **OGC Web Processing Service 2.0** is the standard interface used to expose all ADES operations (application discovery and execution management). Some operations rely on an OGC WPS extension:

- Transactional extension (register and unregister processes);
- Quoting extension (quote an execution before submission);
- Billing extension (provide payment information);
- Visibility extension (authorization mechanism superseded by policy enforcement point architecture).

Web Processing Service (WPS) Specification

OGC WPS (version 2.0) provides a standard remote interface to encapsulate any processing application. WPS is a front-end interface and the implementation typically delegates the execution of application to a back-end server.

Concretely, a WPS server might support many architectures including the following examples:

- Java program executed by the backend Java Virtual Machine (JVM);
- Python library submitted to the configured interpreter;
- Docker container distributed to a Kubernetes cluster node;
- Business Process Execution Language (BPEL) workflow managed by a remote workflow engine;

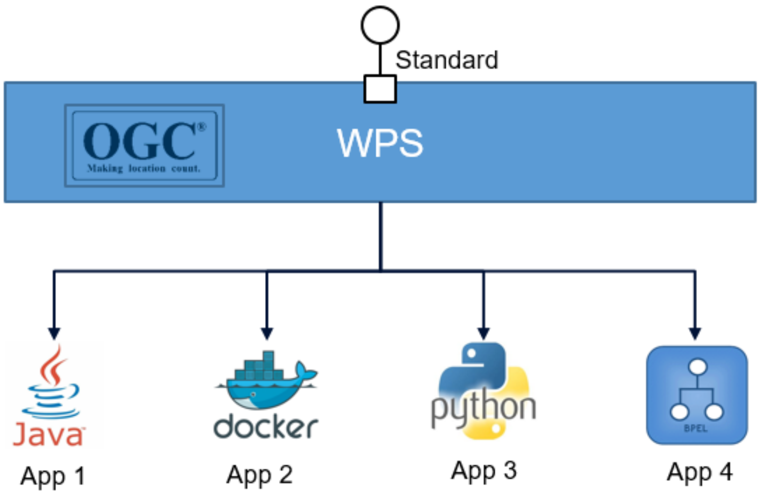


Figure 2. WPS Possible Backends

Discovery and Application Descriptor

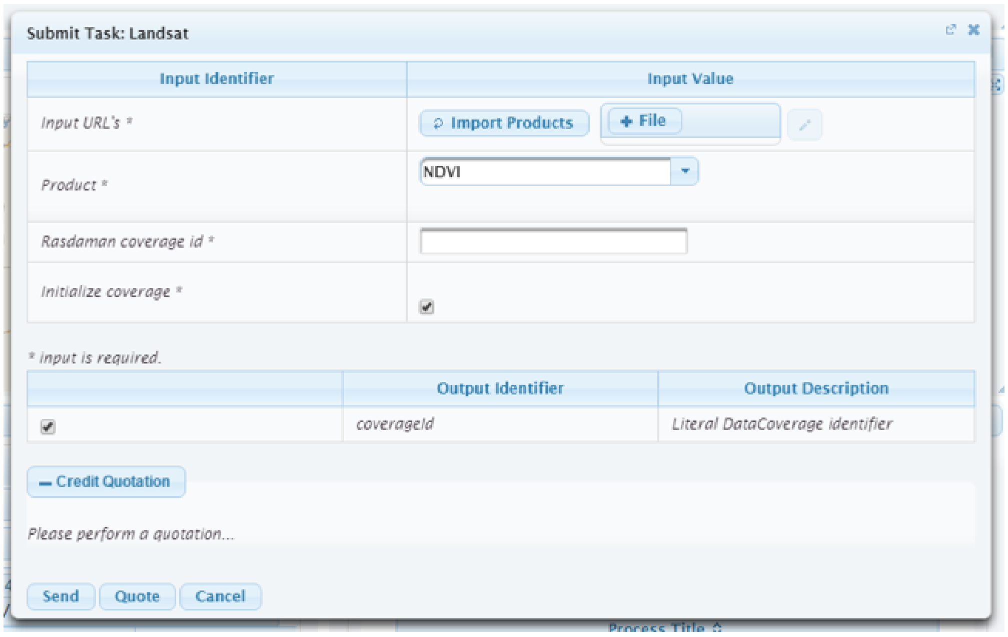
The WPS defines a well-known stable interface contract (i.e. an API) but intends to support a wide range of geospatial applications offering different functions and various inputs & outputs.

In order to discover the registered applications, the WPS provides the operations *GetCapabilities* and *DescribeProcess*. Processes are modeled in a **Process Description**, i.e. an application descriptor capturing all details **needed by the client to interact and execute** the application.

Concretely, the Process Description helps to generate a customized user interface for composing the execution request, integrate adapted tools (EO data browser), enforce constraints and to potentially automate the executions. The descriptor provides various relevant information such as the following properties:

- Name, description, cardinality of inputs;
- Input types (literal, file, EO product);
- Input constraints (temporal, spatial, etc.) and relations (e.g. same relative orbit);
- Product catalogue or database endpoints;
- Billing related information;
- Output data and metadata types;
- Etc.

Figure 3 illustrates an example of the view generated from an application descriptor in a WPS user interface. The integration of a tailored selection of products is shown for the first input field.


Submit Task: Landsat

Input Identifier	Input Value
*Input URL's **	Import Products · + File
*Product **	NDVI
*Rasdaman coverage id **	
*Initialize coverage **	✓

** input is required.*

	Output Identifier	Output Description
✓	*coverageId*	*Literal DataCoverage identifier*

Please perform a quotation...

Figure 3. WPS User Interface

Deployment and Application Package

Since the early days of WPS 1.0, the need for an operation for registering new application emerged. In 2008, the first draft of a *transactional* extension was submitted for harmonizing the deployment of processes. The operations *deploy* and *undeploy* have been then updated and improved continuously as illustrated on figure below.

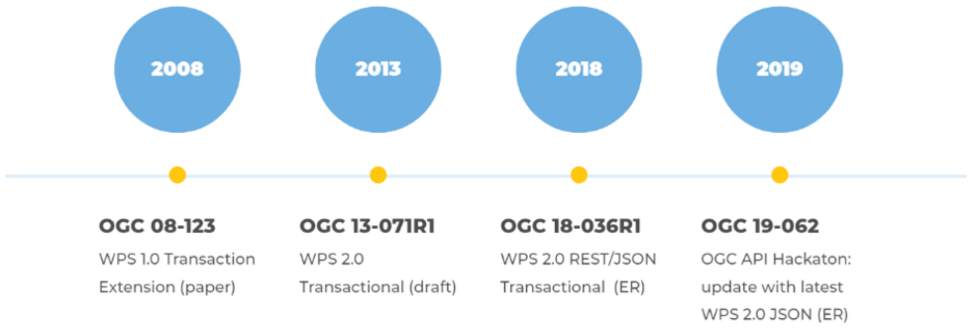

2008
2013
2018
2019
OGC 08-123
WPS 1.0 Transaction Extension (paper)
OGC 13-071R1
WPS 2.0 Transactional (draft)
OGC 18-036R1
WPS 2.0 REST/JSON Transactional (ER)
OGC 19-062
OGC API Hackaton: update with latest WPS 2.0 JSON (ER)

Figure 4. WPS Transactional Specification History

As described in the OGC 18-036r1 WPS-T [https://docs.ogc.org/per/18-036r1.html] Engineering Report, the deploy request essentially consists of the **list of execution units** (files, libraries) composing the application [4]. For each kind of supported application (BPEL workflow, Java library, CWL package), the exact format of items should be defined and is identified using the **Application Package** reference.

From the provided bundle of files composing the application, the WPS needs to generate the corresponding application descriptor described above, excepted if the document is provided.

As illustrated in Figure 5 below, WPS might expose inputs diverging from the backend process. For example, the WPS could expose some Catalogue search parameters, while the search results (EO data) are actually consumed by the process.

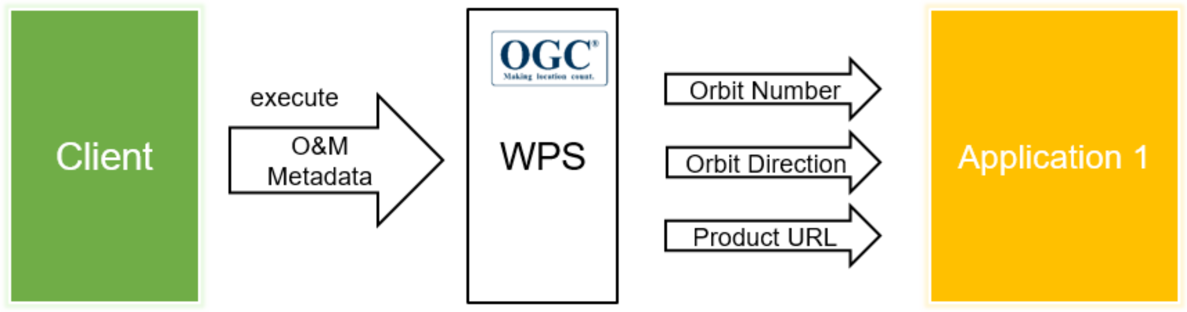


Figure 5. WPS Discrepancy between Internal and External Interfaces

7.1.2. Execution Management Service

An EMS is defined as the Thematic Exploitation Platform (TEP) processing frontend ahead of multiple platform (Mission Exploitation Platforms).

The end-users of the TEP register their applications through the EMS, and the target platform used to execute a process is selected at runtime based on the location of the data to be processed. The EMS also provides support for processing chains: workflows can be deployed as a CWL (Common Workflow Language) document. The Business Process Model and Notation (BPMN) alternative has also been explored in Testbed-14 [https://docs.ogc.org/per/18-085.html] [5].

The current EMS interface also implements the WPS standard, and thus the component acts like a proxy aggregator.

7.1.3. Supported Application Packages

The application is registered in a package providing the execution unit and optionally the application descriptor. Testbed architecture currently defines the following Application Package profiles:

- Docker container
- CWL Workflow

Examples are provided in OGC Testbed-14: Application Package Engineering Report [2].

Docker Container Application Package

The Docker Application Package is defined as follows:

- Process Description (with the reference of the CWL);
- CWL Descriptor based on the *CommandLineTool* class and properties *itemSeparator*, *position*, *prefix*, *separate* and *glob*;
- Execution Unit: Docker Image reference.

CWL Workflow Application Package

The CWL Workflow Application Package is defined as follows:

- Process Description;
- Execution Unit: CWL workflow describing the ADES calls using the application identifier.

7.2. Spacebel Platform Overview

As a "Platform Provider" (WP2), Spacebel is offering the "Geospatial Exploitation Platform" (GEP) as a facility to be used for the Pilot, and its main instantiation called “EO Regions!”.

7.2.1. EO Regions! Context

The EO Regions! [http://www.eoregions.com] Platform was originally funded by the Walloon region and is being operated by Spacebel. EO Regions! [http://www.eoregions.com] is the Walloon hub of the European EUGENIUS network and is recognised as Copernicus Relay for Wallonia by the European Commission for the promotion and support to users of Copernicus data. The main user facing Web Portal is available at http://www.eoregions.com/ and provides more background information about the platform and its objectives.

The Walloon regional government has recently confirmed its industrial policy and related strategy for exploiting EO data in its Déclaration de Politique Régionale (DPR) 2019-2024 [https://www.wallonie.be/sites/default/files/2019-09/declaration_politique_regionale_2019-2024.pdf] , and the intention to implement the Walloon part of the Luxembourg/Walloon Collaborative Ground Segment (CGS). EO Regions! [http://www.eoregions.com] and its applications (currently hosted on Google Cloud Platform) will be hosted on the CGS (expected to be located in Transinne-Belgium) when it becomes available.

The EO Regions! [http://www.eoregions.com] platform mostly relies on a set of components known internal as "Geospatial Exploitation Platform" and described in section below.

7.2.2. GEP components

The main components of the EO Regions! [http://www.eoregions.com] Platform are described below:

- The ADES Processing Service handles all aspects pertaining to the deployment and execution of EO applications on a cloud environment as designed by OGC. The component implements the Web Processing Service Transactional specification (WPS-T) with REST/XML bindings and relies on a Kubernetes cluster for the execution of Docker containers.
- The ADES Scheduler component automates repeated executions triggered by a set of events. For example, any processes can be executed with a set of defined parameters (e.g. a given area of interest) either on regular basis (daily, weekly, etc.), or based on the availability of new source tile.
- The Ingestion Data Manager essentially registers and manages the geospatial data generated by the processes by copying (if needed) the new data resources into the File Store and registering metadata into the OpenSearch Catalogue. The Data Manager is able to discover the Landsat, Sentinel-2 and NEXRAD data available on the Google Cloud Platform and can also fetch remote data shared using a set of protocols (HTTP, FTP, etc.).
- The Catalogue registers metadata of all products available locally and makes them discoverable using an OpenSearch interface (OGC 13-026r8). The resources may be input data (e.g. from a satellite sensor) or data generated by the applications. The metadata includes the references with the protocols available for the given resources.
- The Data Resources File Store holds the EO resources and generated products in a hybrid set of logical storages. The supported storage systems are NFS, Object Storage (virtually abstracted as a file storage using FUSE) and HDFS.
- The Data Access Service provides standardized access Data Resources File Storage (additionally to the POSIX File System protocol). The Data Access Service currently only supports HTTP.

7.3. Application Developer Perspective

The following sections cover the development and execution aspects from a user point of view.

7.3.1. Application Development

The nominal scenario for developing an application that can be registered and executed on an EO platform consists of the following steps:

1. Implement a compatible application
2. Build a Docker Image
3. Register the Docker Image
4. Describe the Application

Write the application

The assumptions below must be satisfied for ensuring compatibility with the EO platform:

- Executable can be started from a UNIX command (e.g. a bash script).
- Inputs (i.e. parameters or EO products locations) can be provided to the app either using environment variable or through UNIX command arguments.
- Output files should must be written either in the working directory when starting the application, or in the location indicated in a configurable environment variable.

The example below is a very simple bash script which receives as input a file (indicated in the `$WPS_INPUT_INPUT1` environment variable) and writes a compressed archive of the input in the specified output location (received in `$WPS_OUTPUT_OUTPUT1`).

```
#!/bin/bash
set -x # display environment variables (for information only)
echo "Starting this dummy processing chain - Hello World !"
echo "Display input directory location"
echo $WPS_INPUT_INPUT1
echo "As a simple demo, let's archive the input directory into a zip file"
tar -czvf "$WPS_OUTPUT_OUTPUT1" ${WPS_INPUT_INPUT1//,/ }
echo "Chain is finished, thank you"
```

Build the Docker Image

A major benefit of Docker containers is that the entire environment is embedded in the application package. Indeed, the environment is described in a formal description file named a Dockerfile. The procedure for building containers using the very popular Docker technology is explained at https://docs.docker.com/engine/reference/builder/.

In the example below, the container is based on a java:8-jre image, to which the bash script shown above is added, and finally the script is declared as the container entrypoint (using CMD command).

```
FROM java:8-jre
USER root
COPY copyscript.sh /
# Depending on the docker host umask , the following line is optional so please keep
it
RUN  chmod a+x /copyscript.sh
ENTRYPOINT["/copyscript.sh"]
```

In order to build a docker image locally (which requires a Docker engine installed), the following command is used:

```
docker build -t my-container-name
```

Register the Docker Image

A free and easy way to share the Docker image is to use the Docker Hub, i.e. the official Docker repository. In order to register the Docker container, a Docker Hub account is needed. The following command authenticates the user:

```
docker login --username=yourhubusername --email=youremail@company.com
```

Docker images are then pushed using the following command (e.g. we called our container “copy-processing-chain”):

```
docker tag copy-processing-chain yourhubusername/copy-processing-chain:latest
docker push yourhubusername/copy-processing-chain:latest
```

Note that the Spacebel GEP platform provides a private registry for hosting the developed applications.

Application Package and Application Descriptor

The Application Package provides the execution units composing the application and all information needed by the platform to interact with the application. In the scope of a containerized application, this includes:

- The Docker Image Reference
- The application command-line interface (how inputs can be submitted, and where outputs can be collected)
- An optional Application Descriptor for advanced users (further described in the report).

Note that the Spacebel platform provides some UI tools that help the developer generate the Application Package items.

7.3.2. Deployment and Execution

Access

When accessing the Application Provider console of the Spacebel EO Regions! [http://www.eoregions.com] platform, the sign in page is displayed. The user needs to authenticate through the OpenAM Single Sign On facility.

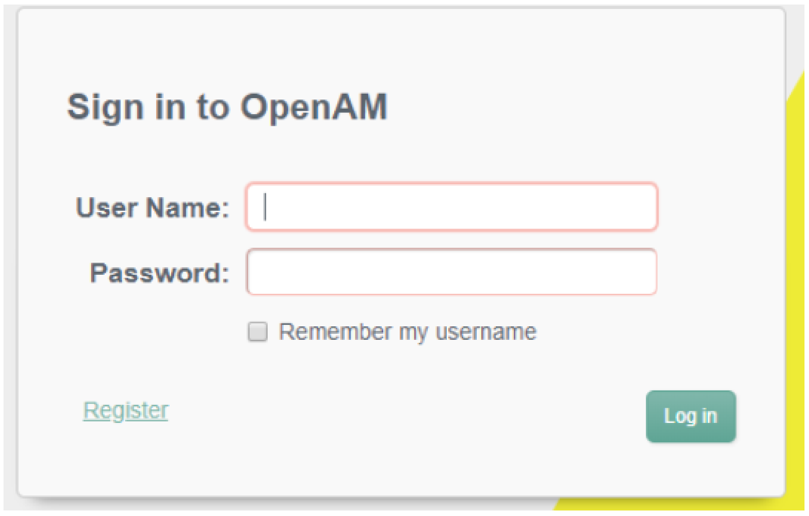
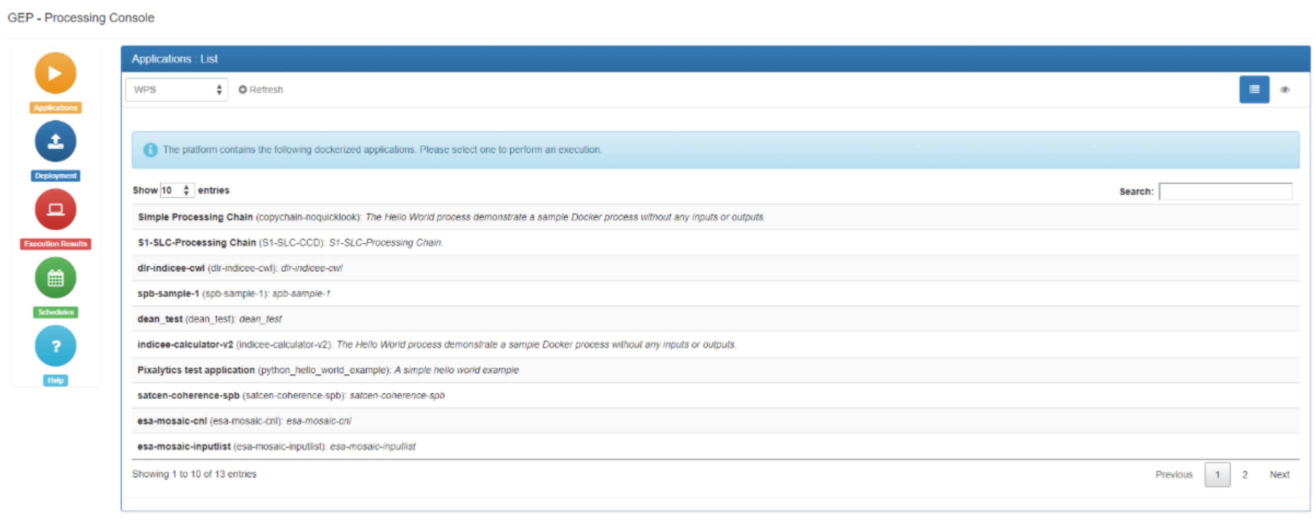


Figure 6. OpenAM Sign In

The initial displayed view is the Applications tab.

Applications

The Applications tab displays the list of available processes.



The platform contains the following dockerized applications. Please select one to perform an execution.

Show 10 entries

Simple Processing Chain (copychain-noquicklook): *The Hello World process demonstrate a sample Docker process without any inputs or outputs.*

S1-SLC-Processing Chain (S1-SLC-CCD): *S1-SLC-Processing Chain.*

dlr-indicee-cwl (dlr-indicee-cwl): *dlr-indicee-cwl*

spb-sample-1 (spb-sample-1): *spb-sample-1*

dean_test (dean_test): *dean_test*

indicee-calculator-v2 (indicee-calculator-v2): *The Hello World process demonstrate a sample Docker process without any inputs or outputs.*

Pixalytics test application (python_hello_world_example): *A simple hello world example*

satcen-coherence-spb (satcen-coherence-spb): *satcen-coherence-spb*

esa-mosaic-cnl (esa-mosaic-cnl): *esa-mosaic-cnl*

esa-mosaic-inputlist (esa-mosaic-inputlist): *esa-mosaic-inputlist*

Showing 1 to 10 of 13 entries

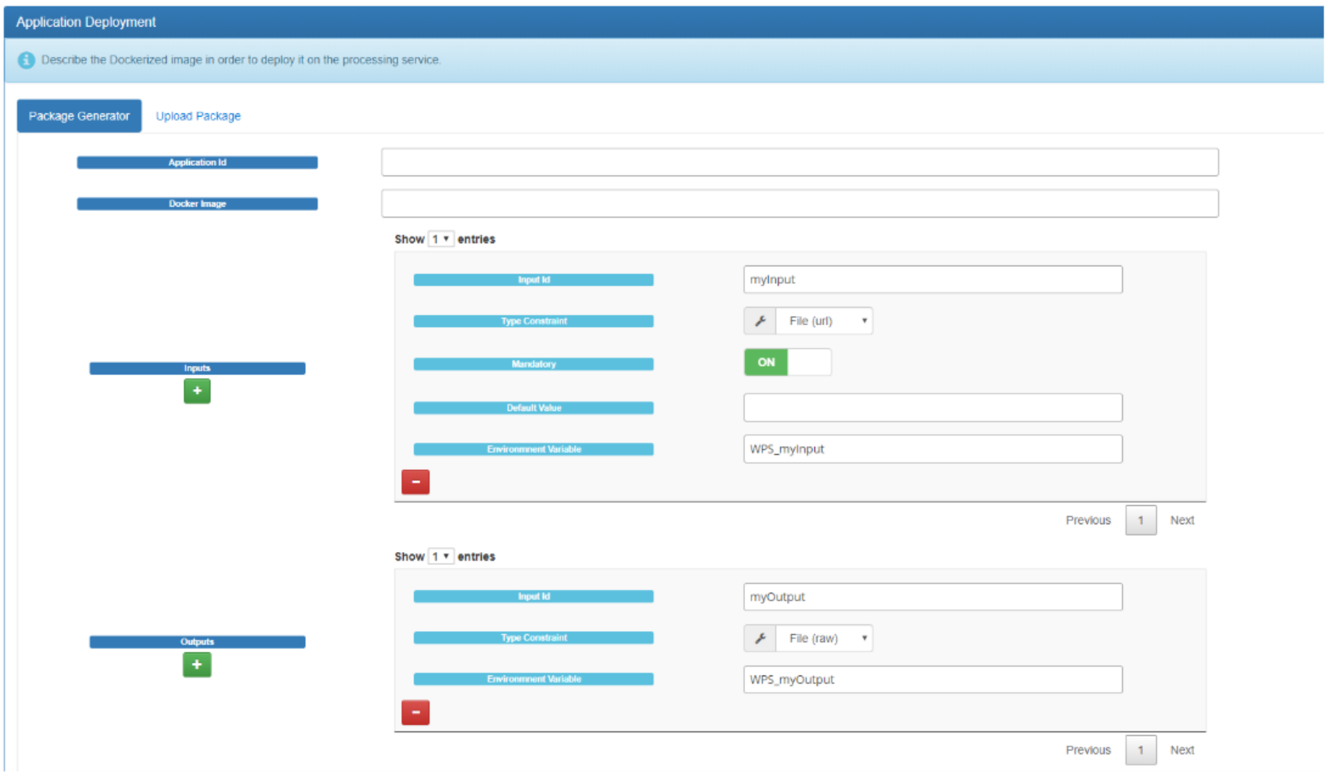
Figure 7. Spacebel Platform - Applications List

Clicking on a specific process displays an execution form for submitting an execution.

Deployment

The deployment tab allows the user to register a new application. From the tab Upload Package, the user can provide the CWL or the Process Description (both supported) of the application.

The Wizard tab helps the user to build automatically the Application Package if the Application Descriptor is not available.


Application Deployment

Describe the Dockerized image in order to deploy it on the processing service.

Package Generator | Upload Package

Application Id	
Docker Image	

Inputs

Show 1 entries

Input Id	myInput
Type Constraint	File (url)
Mandatory	ON
Default Value	
Environment Variable	WPS_myInput

Previous 1 Next

Outputs

Show 1 entries

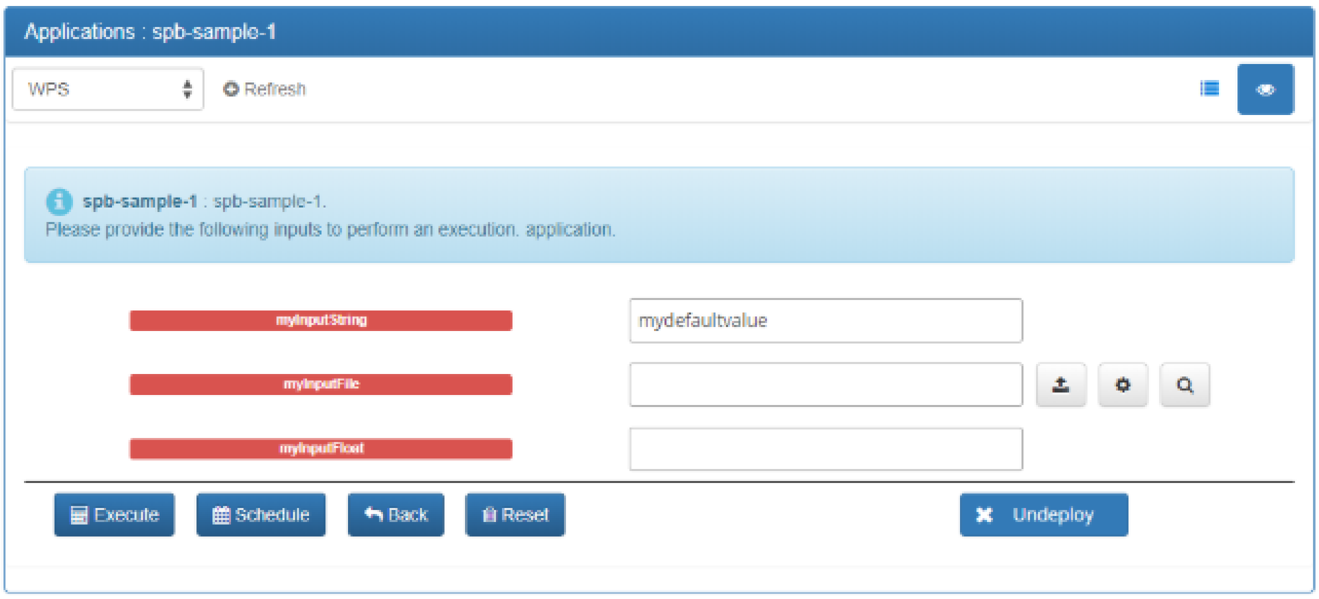
Input Id	myOutput
Type Constraint	File (raw)
Environment Variable	WPS_myOutput

Previous 1 Next

Figure 8. Spacebel Platform - Deployment Wizard

Execution

The Execution manager displays a generated form including the fields declared in the Application Package. The execution form ensures all constraints and integrates all the relevant tools depending on the discovered content of the Process Description.


Applications : spb-sample-1

WPS | Refresh

spb-sample-1 : spb-sample-1.
Please provide the following inputs to perform an execution. application.

myInputString	mydefaultvalue
myInputFile	
myInputFloat	

Execute | Schedule | Back | Reset | Undeploy

Figure 9. Spacebel Platform - Execution Manager

The fields that are declared as File or Directory (Complex Data) can be either:

- Local computer resources: uploaded from the upload button;
- Remote resources: provided by an HTTP reference (which will be staged in by the platform);
- Platform EO resources: provided by an NFS reference found by browsing the catalogue.

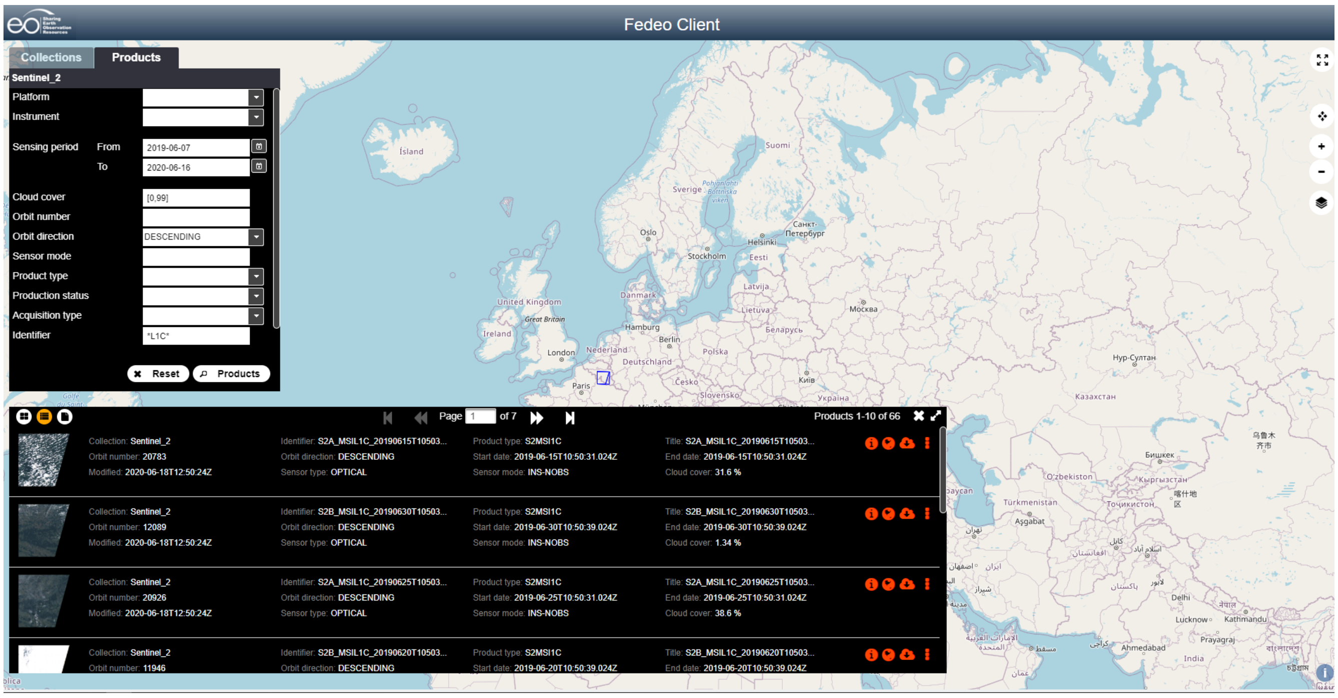
Once the input values have been set, the user needs to click on the Execute button. The Execution Results view can be selected to monitor the running execution.

Catalogue Search

From the Execution Management view, the "magnifier" button opens a dedicated browser tab that allows to search for input EO products.

The first step allows the user to select the collection of interest: either native Copernicus products, or products generated through the execution of other processes.

The second step allows to search for products by providing the required filters (orbit number, time period, tile identifier, etc.) or drawing an area of interest.



Sentinel_2

Platform		
Instrument		
Sensing period	From	2019-06-07
	To	2020-06-16
Cloud cover		[0,99]
Orbit number		
Orbit direction		DESCENDING
Sensor mode		
Product type		
Production status		
Acquisition type		
Identifier		*L1C*

Collection: Sentinel_2	Identifier: S2A_MSIL1C_20190615T10503...	Product type: S2MSI1C	Title: S2A_MSIL1C_20190615T10503...
Orbit number: 20783	Orbit direction: DESCENDING	Start date: 2019-06-15T10:50:31.024Z	End date: 2019-06-15T10:50:31.024Z
Modified: 2020-06-18T12:50:24Z	Sensor type: OPTICAL	Sensor mode: INS-NOBS	Cloud cover: 31.6 %
Collection: Sentinel_2	Identifier: S2B_MSIL1C_20190630T10503...	Product type: S2MSI1C	Title: S2B_MSIL1C_20190630T10503...
Orbit number: 12089	Orbit direction: DESCENDING	Start date: 2019-06-30T10:50:39.024Z	End date: 2019-06-30T10:50:39.024Z
Modified: 2020-06-18T12:50:24Z	Sensor type: OPTICAL	Sensor mode: INS-NOBS	Cloud cover: 1.34 %
Collection: Sentinel_2	Identifier: S2A_MSIL1C_20190625T10503...	Product type: S2MSI1C	Title: S2A_MSIL1C_20190625T10503...
Orbit number: 20926	Orbit direction: DESCENDING	Start date: 2019-06-25T10:50:31.024Z	End date: 2019-06-25T10:50:31.024Z
Modified: 2020-06-18T12:50:24Z	Sensor type: OPTICAL	Sensor mode: INS-NOBS	Cloud cover: 38.6 %
Collection: Sentinel_2	Identifier: S2B_MSIL1C_20190620T10503...	Product type: S2MSI1C	Title: S2B_MSIL1C_20190620T10503...
Orbit number: 11946	Orbit direction: DESCENDING	Start date: 2019-06-20T10:50:39.024Z	End date: 2019-06-20T10:50:39.024Z

Figure 10. Spacebel Platform - EO Catalogue Search

Execution Monitoring and Results

In the Execution Results tab, the list of completed and running executions is displayed indicating the general status.

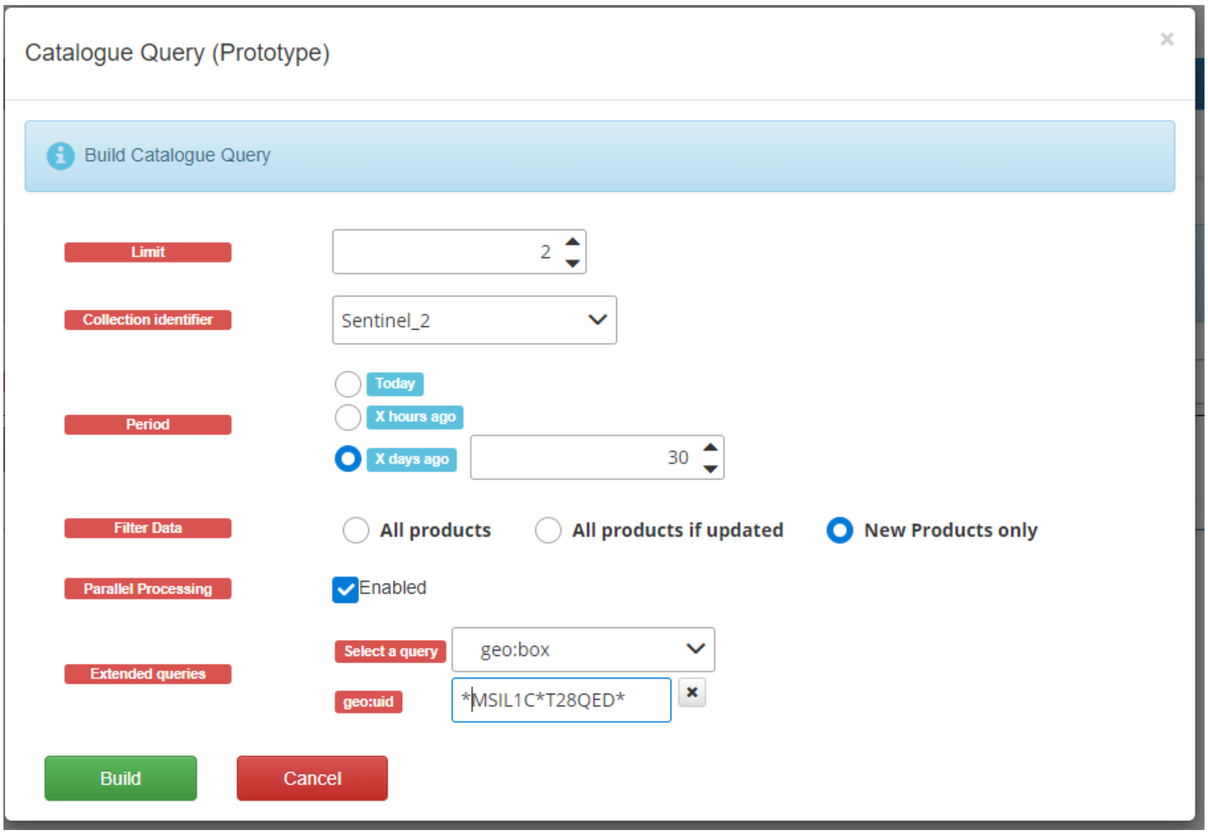
Clicking on an entry displays the details about the execution. The view allows to download the result or display the Catalogue product entry. The Logs button allows to see the standard out/error generated during the execution.

OpenSearch for Inputs

The platform allows users to express the EO inputs as an OpenSearch Query. The following parameters can be set:

- Limit: maximum number of products
- Collection
- Period: the time period restriction for the data

- Filter data: an option that ensures products are processed only once
- Parallel Processing: split multiple products in parallel executions (fan out)
- Extended Queries: other open search parameters (e.g. tile)



Catalogue Query (Prototype)

Build Catalogue Query

Limit	2
Collection identifier	Sentinel_2
Period	Today [✓ X days ago] 30
Filter Data	All products / All products if updated / [✓ New Products only]
Parallel Processing	[✓ Enabled]
Extended queries	Select a query: geo:box; geo:uid: *MSIL1C*T28QED*

Figure 11. Spacebel Platform - OpenSearch Query as Input

Automated Processing

The EO Regions! [http://www.eoregions.com] platform includes a specific Scheduler component which allows to trigger automated execution of processes. The Scheduler can be used to start an execution when a new EO product (matching a specific query) is registered in the Catalogue.

The functionality is currently only configured by the platform administrator but was used during the technology integration experiments.

Chapter 8. Lessons Learned as a Platform Provider

The feedback received during the pilot highlighted various aspects that may be refined to consolidate the Testbed architecture.

Application Providers essentially requested to reduce the complexity of required technical skills. On the other hand, Expert users would be interested to control advanced functionalities such expressing complex inputs inter-dependencies.

These points are elaborated further below.

8.1. EMS Ambiguity

Since the original definition in Testbed-14, the Execution Management Service wears several hats that may confuse the platform users. Indeed, the components endorse the following various roles:

- Interact with multiple Mission Exploitation Platforms (MEP).
- Frontend of the Thematic Exploitation Platform (TEP) processing framework.
- Register the developed Applications and Workflows.
- Select (at runtime) the target platform (based on data location) then deploy the process, execution and handle the stage in/out of data.
- Expose EO inputs (usually files) through a Catalogue query (i.e. collection, period, region parameters) in order to search for products that are then submitted to the process.
- Execute processing chains (workflows) with the behaviors mentioned above.

Even the workflow coding conventions are ambiguous: indeed, the Processing Chains are expressed using Common Workflow Language but the **EMS CWL workflows cannot be executed by a CWL engine**. As detailed below, the workflow Application Package is not compliant with the CWL assumptions:

- CWL engines execute the chain steps locally (local docker engine);
- EMS target ADES of the steps must be computed at runtime (based on a Catalogue search response);
- EMS steps must be actually executed by an ADES client (not contained in the EMS workflow packaged);
- EMS Stage in/out mechanism between remote platforms is not handled by CWL engines.

The discrepancy between the EMS Workflow Application Package (deployed) and the actual executed CWL workflow is illustrated on the figure below.

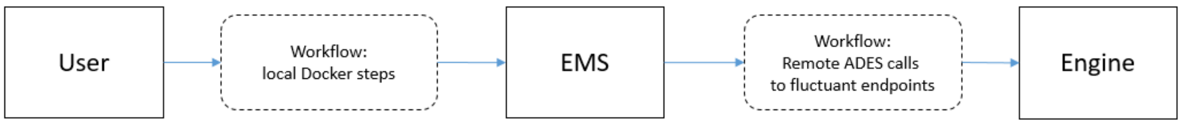


Figure 12. Developed EMS Workflows versus Executed EMS workflows

In conclusion, Application Providers looking for a **simple (single-platform) workflow support** could be interested in an ADES with a CWL backend (as recommended in a further section). The EMS should be considered essentially for handling the multiple platforms aspects.

However, the main disadvantage of the EMS is that it obscures and imposes policies governing the product selection, the platform selection and the stage in/out operations. The **EMS** would **benefit from being tied to the Jupyter Notebook** facilities currently being explored in the Testbed-16 [https://www.ogc.org/projects/initiatives/t-16] project.

The example below shows pseudo-code illustrating how a developer would easily control the processing chain flow in a Jupyter Notebook using a user-friendly platform API:

Jupyter Notebook Workflow Example

```
results = catalogue["federated"].search("Sentinel-2","2019","2020","Belgium");
ndviResult = platform[results.getPlatformId()].processing["ndvi"].execute(results);
maja = platform[results.getPlatformId()].processing["maja"].execute(ndviresults);
```

8.2. Improvement of the Docker Application Package

Testbed-13 defined a simple Docker Application Profile containing only the Process Description. Testbed-14 introduced a few CWL properties to expand the means for describing the command-line interface of applications.

Improving Testbed-13 limitations can preferably be achieved through the Kubernetes API, which is more intuitive, simpler and more popular than CWL. Indeed, the Application Providers only expressed interest in the command-line interface that **simply enables declaration of the command and arguments** of the Docker Image:

```
command: /usr/bin/python3 myapp.py
args: --i $(INPUT1) --j $(INPUT2)
```

If the Process Description is not provided, the document can be automatically generated by the ADES and thus reduces the Application Package to **only two pieces of information:**

- Docker Image Reference
- Base command and arguments expression

Furthermore, additional conventions could be explored to express the inputs cardinality, potentially item separators or input formats. The example below shows a possible syntax for an optional PNG input and an array of TIFF images with a separator:

```
command:/usr/bin/python3 myapp.py
args: --i $(INPUT1)[0,1,"*.png"] --j $INPUT2[1,8,"*.tif"]
outputs: "/outputs/*.tif"
```

Such a profile would ensure a short, simple and flexible means of describing a Docker application and still enable expert users to provide additional advanced functionalities (typically embedded in the Process Description).

8.3. Docker Advanced Concerns

Some aspects tied to the containers have been discussed during the pilot and are briefly described below.

8.3.1. Docker Build Automation

Docker is a widely used technology that provides OS-level virtualization and has been introduced since Testbed-13. The application can be packaged as standard containers that embed all environment and software dependencies for running applications written in any language, and a number of tools to manage the execution of the containers.

The main complexity of deploying Docker containers is that the runtime environment and dependencies must be prepared by Application Providers.

Additionally to the Docker Application Package, the ADES might support the build of Docker images for a set of formats proposed to novices and non-experts users such Java, Python, Go, etc.

For each language or backend environment, the ADES defines an Application Package profile (e.g. in Java, the library main class typically must copy a given interface). From the registered source package, the ADES then automatically generates the application descriptor, builds the binaries, and then prepares the Docker environment.

As all possible packages are actually packaged as Docker images, the support of additional languages or backends **only requires effort** for implementing **build pipelines** of the specific language: the execution remains a container managed process.

8.3.2. Docker Registry Authentication

The access control and confidentiality of Docker images has been discussed as an important aspect of the containers management. As a matter of fact, the architecture has yet only been prototyped in testbeds with Docker images being publicly available on the Docker Hub. To this end, Spacebel has provided access to a private platform repository during the project.

However, the **automation of the Docker build** on the ADES side (explained in above section) could also be envisioned to **solve the access control aspects** of containers: instead of providing the Docker image reference, the application provider would supply the software source and *Dockerfile* in order to let the ADES build the container and register the image in the platform registry. This would also **solve the container access control issues related to federated platforms** as the involved ADES would trust and share their mutual Docker registries.

8.3.3. Docker Image updated

When registering a given Docker container image, the ADES has no control of the remote reference of the Docker image. Concretely, the Application Provider could update the docker image and break a working process by introducing bugs in the updated image.

Still with the same approach of building the image on the ADES side, providing the source and *Dockerfile* would **avoid any uncontrolled changed** and enforce the Application Provider to request an update using the dedicated and explicit operation.

8.3.4. Docker Security

Some concerns have been discussed about the security aspects related to Docker containers. Although some isolation can be enforced on containers (e.g. through non-privileged Docker mode, closed Kubernetes environment, advanced write access control), executing the containers as the **root** user is generally seen as a **bad practice**.

Again using the same approach of generating the Docker image on the ADES side, the possible security breaches might be detected by inspecting the *Dockerfile* and fixed before building the container.

8.4. Kubernetes Cluster Support

Although Earth Observation platforms are typically installed in a Cloud infrastructure, the majority of implementations in OGC Testbeds have limited the execution on a single node machine. This considerably restricts the system scalability and computing possibilities.

Kubernetes has become the de facto standard for deploying containerized applications in private and public cloud environments. The CFP strongly encouraged prototyping and testing the execution and deployment of the applications in a Kubernetes cluster.

In this pilot, Spacebel implemented the whole system on top of a Kubernetes cluster, including both the system components (ADES, Catalogue) and the processing jobs. Spacebel also prototyped the split of supplied inputs to execute the atomic task in parallel. The **automated scalability** of the cluster nodes allows to extend the infrastructure by provisioning additional machines when multiple processes are requested, thus allowing to execute simultaneously a very high number of concurrent jobs.

Note that the Docker Application Package format was limited to the few CWL properties defined in OGC Testbed-14 because no CWL engine provides Kubernetes support.

8.5. CWL Pros and Cons

During Testbed-14, certain parts of the CWL specification were adopted to extend the possibilities to express the command-line syntax of Docker containers, **not** to execute the process on a **CWL engine backend**. Participants decided to support only the CWL properties related to the command-line syntax. Indeed, supporting the entire specification raises the following issues:

- With the emergence of **Kubernetes as the de facto standard** for container orchestration (in particular since 2017, as Docker announced the integration of Kubernetes in its Enterprise Edition), it is not understood why an EO app container would be described using the little-known CWL alternative instead of using the Kubernetes resource model. Indeed, Kubernetes Pod definition (https://kubernetes.io/docs/concepts/workloads/pods/) supports detailed Cloud-related aspects such as communication between containers, resource sharing, health check, init containers, job patterns, parallel executions for jobs, and Kubernetes Argo provides support for workflows (DAG or step-based) and pipelines with optimized execution on distributed environment (https://argoproj.github.io/). We recommend adopting Kubernetes properties for describing the container interfaces in the Application Package.

- CWL specification is huge and **exceeding the command-line syntax** with a number of specific functions and capabilities, representing a very considerable effort for the implementation.
- Existing CWL engines are generally limited to a local single node execution (e.g. Airflow supports Kubernetes but not when using CWL language, CWLRunner requires implementing an extension). The few existing solutions (calrissian, REANA) did not meet our expectations.
- Adopted Docker technology enables a wide variety of solutions and backends (Kubernetes, Swarm, CWL, Argo, homemade scheduler, etc.) clearly restricted by the majority of available free CWL engines.
- The CWL object model (Semantic Annotations for Linked Avro Data) can hardly be expressed in OpenAPI (and JSON schema).
- Featuring a subset of the CWL language might confuse users and implies expressing clearly the capabilities (e.g. conformance classes).

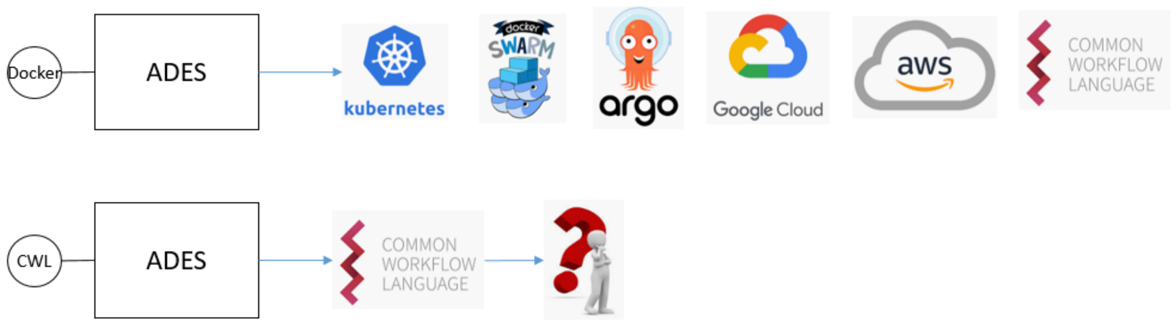


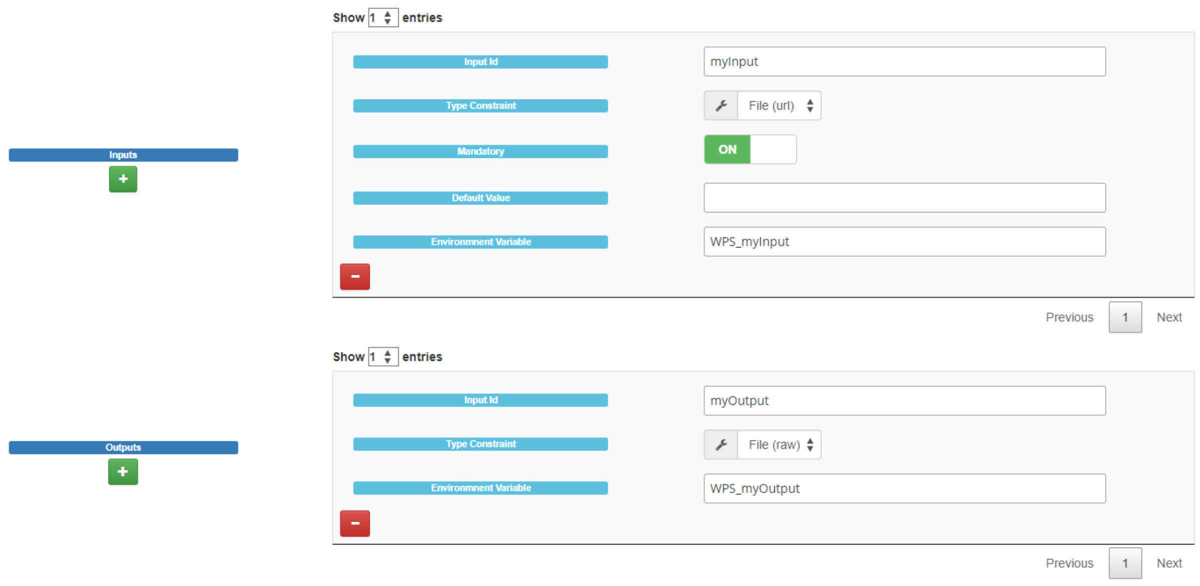
Figure 13. Docker versus CWL Application Package

Moreover, CWL addresses specific Docker backend aspects (how the ADES interact with containers) but should not be confounded with the WPS Process Description, which describes the external interface (with WPS clients) of any backend and embeds geospatial related information (though WPS actually has limited support for such capabilities).

On the other hand, CWL provides a wide set of advanced functionalities: multiple supported formats (JSON, YAML) and structures (arrays, objects), environment preparation using JavaScript functions, variables and expressions for defining inputs and constraints, settings of network aspects, etc.). The various functionalities raised interest from many Participants. However, Application Providers also noticed some deficiencies:

- High complexity and considerable effort for learning the basics.
- Developers are still required to provide information not supported by CWL (i.e. in the WPS Process Description or equivalent):

    ◦ Input constraints (temporal, spatial, format, sensor, orbit, cloud coverage, etc.)
    ◦ Input cardinality
    ◦ Potential metadata output and format
- Some prefer a simpler and more intuitive facet for describing a Dockerized application.
- Developers familiar with WPS are more confident with the Process Description for defining the interface as already used to discover the process of any possible Application Package (BPEL workflow, Python, Docker, etc.). Moreover, the Application Package generation can be assisted using generic WPS-T client tools (as illustrated below).


Show 1 entries

Inputs

Input Id	myInput
Type Constraint	File (url)
Mandatory	ON
Default Value	
Environmnent Variable	WPS_myInput

Previous 1 Next

Show 1 entries

Outputs

Input Id	myOutput
Type Constraint	File (raw)
Environmnent Variable	WPS_myOutput

Previous 1 Next

Figure 14. Generic Process Description Generator

In conclusion, the CWL **makes sense in the set** of supported Application Packages as an **alternative to BPEL**, to target the **Expert** users. However, the support of Kubernetes Jobs (or Workflows) seems more relevant as being the de facto standard for container orchestration.

On the other side, a **basic** and extendable **Docker Application Package** inspired from the Testbed-13 implementation (and Testbed-14 alternatives from CubeWerx and Spacebel) would **benefit both** the Application Providers through simpler conventions to learn and the Platform Provider through a more flexible backend implementation. A recommendation of the Application Package based on Kubernetes properties is detailed in a further section.

8.6. Process Descriptions

As already mentioned, the Process Description is a useful resource for expert Application Providers, because it allows for provision of additional information (e.g. geospatial metadata) that can be exploited by a client application.

8.6.1. Optional Process Description

The Process Description is also generic as used with any specific Application Packages (e.g Docker, BPEL, Java, etc.). Therefore, defining properties (e.g. hardware requirements) in the Process Description benefits all implementations and projects.

Therefore, Spacebel advises, for any Application Package definition, to allow the Application Provider to supply the Process Description, and thus provide advanced information about the process.

8.6.2. Applications Requirements

As very often in Processing, the applications generally require a minimal hardware configuration for one of the following resources:

- CPU
- RAM
- Storage

When using the CWL language, the *ResourceRequirement* property can be used to express such requirements.

```
ResourceRequirement:
  ramMin: 16000
```

However, Spacebel recommends not relying on the specific CWL language, while the hardware requirement is a general concern that should be handled thus by the OGC WPS specification and for any kind of Application Package. In particular, we suggest to define a set of new properties in the Process Description based on the Kubernetes properties *requests* and *limits* as documented on Managing resources for Containers page [https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/].

Furthermore, the network policies might be restricted and controlled by the ADES component. Therefore, it might also be useful to list the possible external URLs (e.g. remote catalogue or data repository) required by the application in the Process Description. More generally, all properties supported by Kubernetes (resource sharing, health check, affinity, security context, priority, etc.) could be potentially explored if relevant.

8.6.3. Expressing Inputs and Output Constraints

As already mentioned, the **Process Description** is the document that allows a client application to discover and interact with a specific application. The information provided by the *Process Description* may include details on the supported inputs formats, the constraints, some useful services or endpoints for collecting the inputs, etc.

Inputs & outputs constraints was a major topic of the CFP, and was confirmed as a main concern from the Application Providers. Moreover, the users looked at how they might **express some correlations between some inputs**, such as '12 days older than input 1', or "same relative orbit".

In the architecture, the EO data products can be searched using the OpenSearch standard extended with the OGC OpenSearch Extension for Earth Observation specification (OGC 13-026r8) and OpenSearch Geo and Time Extensions (OGC 10-032r8). Therefore, the properties defined by the **OpenSearch specification** qualify perfectly as **standard for expressing the inputs constraints** in the Process Description: geo:box, geo:geometry, geo:relation, time:start, time:end, time:relation, eo:orbitNumber, eo:acquisitionType, eo:cloudCover, eo:snowCover, etc.

Furthermore, future work could explore how those properties may define possible values, ranges or expression from other inputs. The following example define some input constraints as being 5 days older than *input1*, between 0 and 60 percent of cloud coverage, and with a geometry intersecting input 1:

```
{ key: "time:start",
  value: "(input1.time:start - PT5D)" },
{ key: "eo:cloudCover",
  value: "[0,60]" },
{ key: "geo:geometry",
  value: "input1.geometry"},
{ key: "geo:relation",
    value: "intersects"},
```

Finally, expressing the constraints like this would also clearly **facilitate the automation of executions**. During our technology integration experiment with the SatCen Coherence application, we missed the means for expressing the discussed constraints (e.g. same relative orbit and previous sensing period): our scheduler component may automatically compute the constraints and enforce the criteria when selecting the products.

8.7. WPS EO Inputs

Popular APIs or models for handling process already exist (e.g. Kubernetes). Thus, in this context, the main purpose of the OGC API – Processes candidate standard is to provide a processing interface for handling **geospatial** aspects, like the *BoundingBox* input type defined by WPS 1.0 which has been poorly used. In this sense, the WPS clearly lacks built-in functionalities for EO related data.

In particular, Spacebel recommends that research is conducted into the **introduction of the *EOData* type** in addition to the *LiteralData* and *ComplexData* types (themselves requiring some revision). Spacebel also suggests association of the following behaviors with this new 'EO data' type:

- The *Input Description* of *EOData* allows to declare constraints (allowed values or range) using any property of the OpenSearch EO extensions (geo:geometry, eo:relativeOrbit, etc.)
- Such input can be submitted in an execution request as either :
  - The remote or local URL
  - The metadata, encoded as OGC Observations and Measurements (O&M) Earth Observation Products (EOP), including the location
  - OpenSearch query parameters

Defining advanced constraints allows Application Providers to **accurately describe the requirements and correlations between the inputs** in order to complete executions successfully. Some inputs might even be automatically computed from others.

The possibility to provide the metadata would confer many possibilities to the ADES such as generating the output metadata or performing actions based on advanced criteria.

The opportunity to specify inputs using an OpenSearch query would delegate the OpenSearch search operation to the ADES if the platform integrates an EO products catalogue.

In addition, other predefined types of data might be proposed: the **bounding box** definition could be extended to allow a **set of CRS and formats** for expressing an area of interest in Well Known Text (WKT), Geography Markup Language (GML), Polyline, or GeoJSON. The translation to the application specific format could be delegated to the WPS implementation.

Finally, as mentioned in section above, the EO inputs and outputs constraints could be described using OpenSearch extensions.

8.8. Process Outputs Metadata

The CFP suggested to explore *discovery aspects* such as publication and discoverability of results. The Spacebel ADES implementation allows to collect metadata output and store it in the platform Catalogue (in OGC O&M EOP [OGC 10-157r4] format). Although the application providers were satisfied to generate EOP metadata, we recommend to support alternative approaches.

A first solution that has been introduced in the platform during the project is the support of a simple property file for each product output to indicate the useful metadata information about the generated products. An initial set of properties is supported to ease the creation of the metadata built by the application itself.

```
IDENTIFIER=name of my product
FILES=list of product files (required if multiple products are generated)
GEOMETRY=the coordinates (polygon) of the data
BEGIN_POSITION=start time of the product (e.g when the source image capture is
started)
END_POSITION=end time of the product
TIME_POSITION=source image time
PROCESSOR_NAME=typically the process identifier
CREATION_DATE=creation time of the product (automatically filled if not provided)
ACQUISITION_TYPE=for example nominal
PRODUCT_TYPE=type
STATUS=status
PROCESSING_CENTER=platform (e.G. Spacebel GEP, automatically filled)
PROCESSING_DATE=processing time (automatically filled)
PROCESSING_METHOD=method
PROCESSING_PROCESSOR_NAME=typicall WPs process identifier (automaticalled filled)
PROCESSOR_VERSION=1.0
NATIVE_PRODUCT_FORMAT=original format (e.g. S2)
PROCESSING_MODE=nominal
```

Spacebel also recommends exploration in future projects of how to automatically build the output metadata: indeed, we believe that in most usual scenarios, relevant metadata of output results can be deduced:

- From runtime parameters such as the processing time, the EO platform executing the analysis;
- From the Application Descriptor which indicates the function name, the description, the data provider, and potentially any application related information that may be useful.
- From the collection of inputs which already detail the input sensing period, location, raw format, etc.

Tools such as GDAL might be also used in order to generate a quicklook or even perform a format conversion to a standardized output format for the published results.

8.9. Process Inputs and Outputs Interoperability

Application Providers in favor of reading raw data (e.g. Sentinel-2) currently use a simple but arbitrary convention to find the data items (e.g. SAFE directory is the direct descendant of the provided location).

Spacebel recommends (as suggested in ESA EO Platform Master System design (https://eoepca.github.io/master-system-design/published/v1.0/) to integrate the OGC Coverage (WCS) and Feature (WFS) Services in order to provide an format-agnostic advanced data access mechanism. Concretely, the inputs can be expressed using a WCS request that includes the desired bands or layers of the selected products, and let the ADES request the appropriate format (e.g. GeoTIFF) supported by the application (and reported in the *Process Description*).

An alternative based on an input/output manifest (in STAC [https://stacspec.org] format) has been proposed. We believe this introduces additional conventions that would be unnecessary if data access services were used.

Chapter 9. Technology Integration Experiments (TIEs)

9.1. DLR Urban Indicee TIE (TIE-6100)

9.1.1. Overview

- Identifier: **TIE-6100**
- Status: Success

This technology integration experiment consisted of the parallel execution of the DLR Urban Indicee process, and the automated triggering of the Temporal Aggregation process from available Urban Indicee products. The source products were Sentinel-2 scenes retrieved from an OpenSearch Query (based on the 28QED tile during the experiment).

The diagram below provides an overview of the integration experiment:

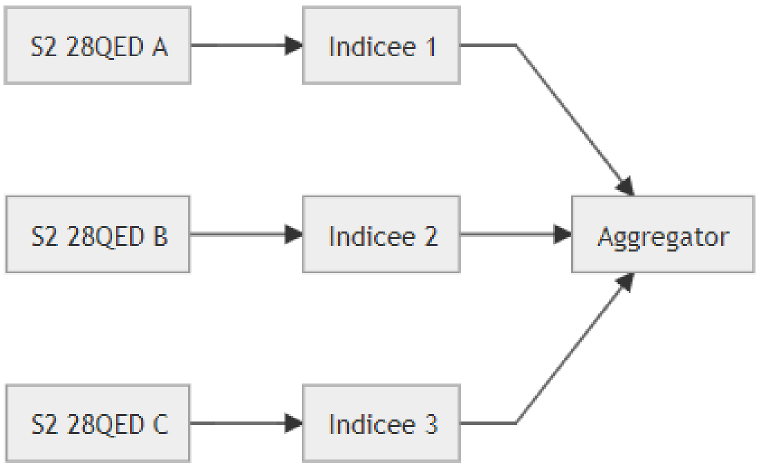


Figure 15. Indicee TIE Diagram

9.1.2. Execution Procedure

The parallel execution of *Urban Indicee* is started manually using the Scheduler component of the Spacebel platform. The query includes:

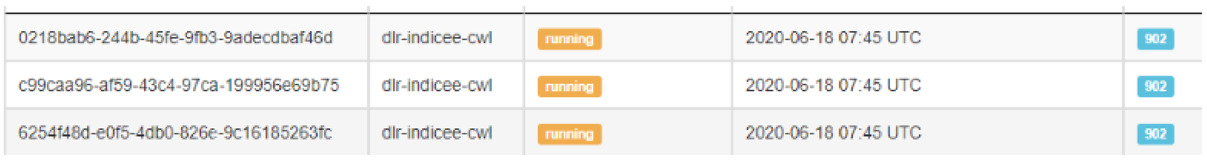
- Process Identifier: Indicee
- Scene input:
  - OpenSearch Query on S2 for Tile 28QED
  - Split of results into parallel executions
- Schedule: immediate

The automated execution of *Indicee Aggregator* is configured using the Scheduler component of the Spacebel platform. The query includes:

- Process Identifier: indicee-aggregator
- Indicees input array:
    - OpenSearch Query of indicee products (limited to max 10 last products)
- Schedule: on new indicee product

9.1.3. Results

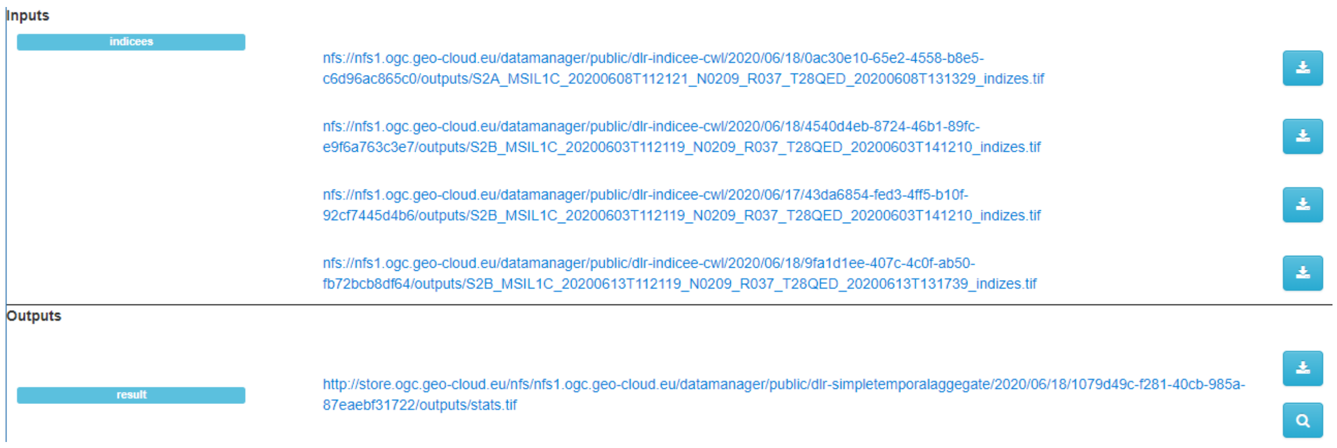
The Indicee executions were started in parallel as shown on the figure below:



0218bab6-244b-45fe-9fb3-9adecdbaf46d	dlr-indicee-cwl	running	2020-06-18 07:45 UTC	902
c99caa96-af59-43c4-97ca-199956e69b75	dlr-indicee-cwl	running	2020-06-18 07:45 UTC	902
6254f48d-e0f5-4db0-826e-9c16185263fc	dlr-indicee-cwl	running	2020-06-18 07:45 UTC	902

Figure 16. Indicee Execution - Part 1

The *Indicee Aggregator* execution was then automatically triggered as soon as *Indicee* output data and metadata were available and successfully completed. The screen capture below shows the aggregator received 10 source Indicee products:



Inputs

indicees

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_indizes.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_indizes.tif

nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/dlr-indicee-cwl/2020/06/17/43da6854-fed3-4ff5-b10f-92cf7445d4b6/outputs/S2B_MSIL1C_20200603T112119_N0209_R037_T28QED_20200603T141210_indizes.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_20200613T131739_indizes.tif

Outputs

result

http://store.ogc.geo-cloud.eu/nfs/nfs1.ogc.geo-cloud.eu/datamanager/public/dlr-simpletemporalaggegate/2020/06/18/1079d49c-f281-40cb-985a-87eaebf31722/outputs/stats.tif

Figure 17. Indicee Execution - Part 2

9.1.4. Lessons Learned

This experiment essentially showed the power of a scalable cluster (Kubernetes) for processing a set of tasks.

9.1.5. Application Package

The CWL below essentially provides the Docker image reference (*dockerPull*), the UNIX command (*baseCommand*), the various command arguments (describing the command-line format), and the output file pattern (*glob*).

```
cwlVersion: v1.0
class: CommandLineTool
id: dlr-indicee-cwl
hints:
  DockerRequirement:
    dockerPull: registry.gitlab.com/ogceo/urban-app:simple-ndvi-cwl-0.02
baseCommand: /usr/local/bin/simpleSentinel2NDVIIndex.sh
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
```

9.2. SatCen Coherence TIE (TIE-2001)

9.2.1. Overview

- Identifier: **TIE 2001**
- Status: Success

The SatCen experiment executed, in parallel, a coherence analysis from 3 pairs of Sentinel-1 products. It then automatically triggers the multi-coherence process from the two generated coherence outputs. The diagram below provides an overview of the integration experiment:

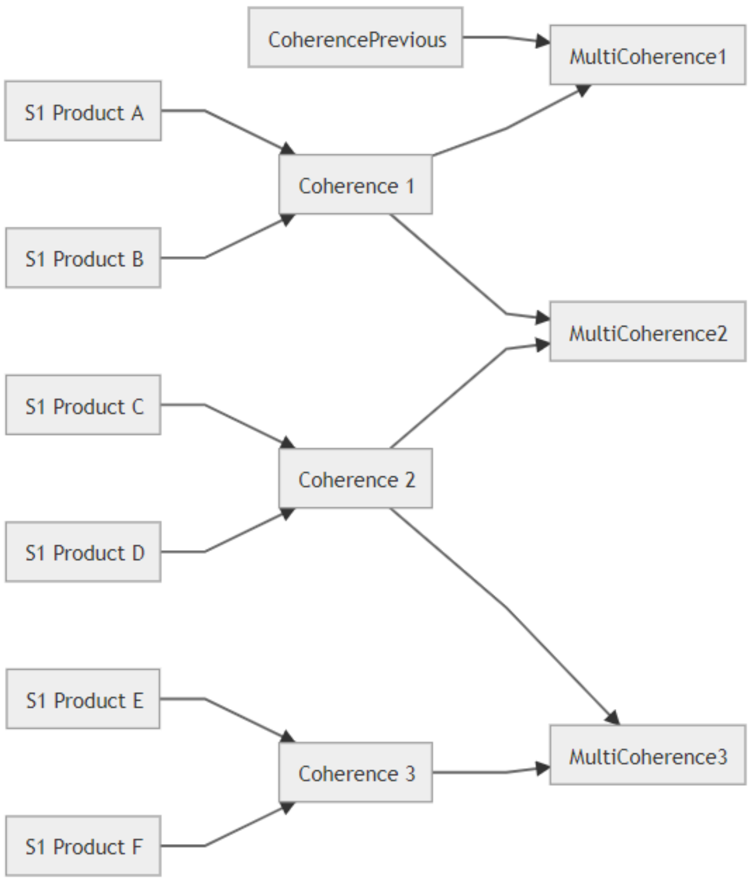


Figure 18. Coherence TIE Diagram

9.2.2. Execution Procedure

The parallel execution of *Coherence* is started manually using the Scheduler component of the Spacebel platform. The query includes:

- Process Identifier: Coherence
- Input_files:
  - OpenSearch Query on S1 orbit 37 in Belgium (Liège) region (limited to 6 results)
  - Split of the results grouped by 2 items
- aoi_wkt: Liège region geometry
- Schedule: immediate

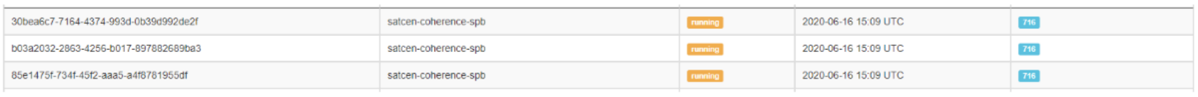
The automated execution of *multi-coherence* is configured using the Scheduler component of the Spacebel platform. The query includes:

- Process Identifier: coherence
- input1:
  - OpenSearch Query on coherence product (limited to last month)
- input2:
  - OpenSearch Query on coherence product selecting previous product
- aoi_wkt: Liège region geometry

- Schedule: on new coherence product

9.2.3. Results

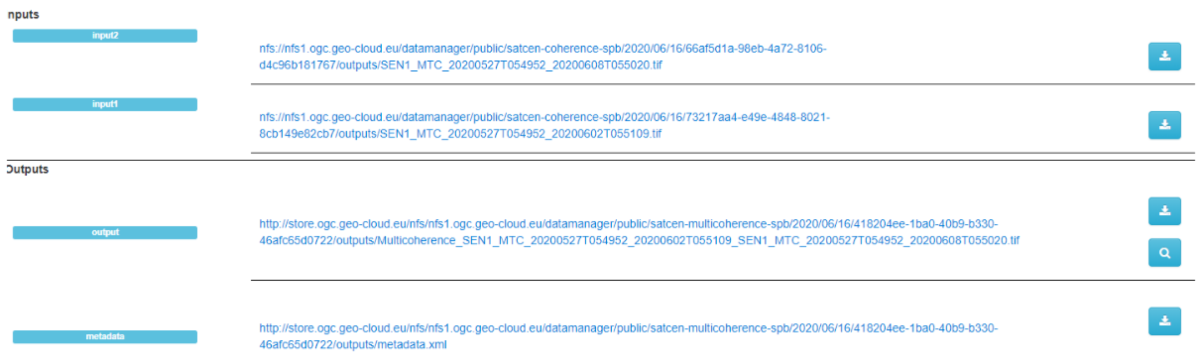
The coherence executions are started in parallel as shown on the figure below:



30bea6c7-7164-4374-993d-0b39d992de2f	satcen-coherence-spb	running	2020-06-16 15:09 UTC	716
b03a2032-2863-4256-b017-897882689ba3	satcen-coherence-spb	running	2020-06-16 15:09 UTC	716
85e1475f-734f-45f2-aaa5-a4f8781955df	satcen-coherence-spb	running	2020-06-16 15:09 UTC	716

Figure 19. Coherence TIE Execution - Part A

The multi-coherence execution is then automatically triggered as soon as a coherence output is available and successfully completed:



nputs

input2	nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/satcen-coherence-spb/2020/06/16/66af5d1a-98eb-4a72-8106-d4c96b181767/outputs/SEN1_MTC_20200527T054952_20200608T055020.tif
input1	nfs://nfs1.ogc.geo-cloud.eu/datamanager/public/satcen-coherence-spb/2020/06/16/73217aa4-e49e-4848-8021-8cb149e82cb7/outputs/SEN1_MTC_20200527T054952_20200602T055109.tif

Outputs

output	http://store.ogc.geo-cloud.eu/nfs/nfs1.ogc.geo-cloud.eu/datamanager/public/satcen-multicoherence-spb/2020/06/16/418204ee-1ba0-40b9-b330-46afc65d0722/outputs/Multicoherence_SEN1_MTC_20200527T054952_20200602T055109_SEN1_MTC_20200527T054952_20200608T055020.tif
metadata	http://store.ogc.geo-cloud.eu/nfs/nfs1.ogc.geo-cloud.eu/datamanager/public/satcen-multicoherence-spb/2020/06/16/418204ee-1ba0-40b9-b330-46afc65d0722/outputs/metadata.xml

Figure 20. Coherence TIE Execution - Part B

Unfortunately, the second input submitted to the multi-coherence process by the scheduler component is not the previous coherence product. Indeed, no sorting option is supported when configuring the automated execution.

9.2.4. Lessons Learned

When trying to automate the execution of the **coherence process**, a gap of current ADES component is clearly highlighted because of the following expectations:

- *input_files* requires data products from the same relative orbit and sharing the same spatial coverage;
- *aoi_wkt* expects to receive the coverage coordinates of the submitted data inputs.

ADES (in particular the Process Description document) does not define any property for expressing such advanced constraints (spatial, temporal, others). As already mentioned, OpenSearch properties would enable declaring allowed values as well as mandatory search fields for a set of inputs (e.g. in this example: the relative orbit and geometry).

Going further, if the EO product metadata (e.g. O&M EOP) was provided as input to the ADES, the value of some inputs (such *aoi_wkt*) could be expressed and computed at runtime from the input metadata, e.g. : `aoi_wkt=input_files.geometry`

9.2.5. SatCen CWL Packages

The CWL below still embeds the same information but points to a different Docker image, and includes 2 inputs (*inputfiles* and _aoi_wkt).

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: s1_coherence_cd
hints:
  DockerRequirement:
    dockerPull: obarrilero/s1coherence:1.0
id: satcen-coherence-spb
label: Satcen S1 Coherence Process (SPB version)
inputs:
  input_files:
    inputBinding:
      position: 1
      prefix: --input_files
    type:
      items:
      - File
      - Directory
      type: array
  aoi_wkt:
    inputBinding:
      position: 2
      prefix: --aoi_wkt
    type: string?
outputs:
  output:
    outputBinding:
      glob: '*.tif'
    type: File
  metadata:
    outputBinding:
      glob: 'metadata.xml'
    type: File
```

Multi Coherence CWL:

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: s1_multicoherence
hints:
  DockerRequirement:
    dockerPull: obarrilero/s1coherence:1.1
id: satcen-multicoherence-spb
label: Satcen S1 Multicoherence Process (SPB version)
doc: Generate byte RGB with coherences at two different dates.
inputs:
  input1:
    inputBinding:
      position: 1
      prefix: --input1
    type: File
  input2:
    inputBinding:
      position: 2
      prefix: --input2
    type: File
outputs:
  output:
    outputBinding:
      glob: '*.tif'
    type: File
  metadata:
    outputBinding:
      glob: 'metadata.xml'
    type: File
```

9.3. Pixalytics Mosaicing TIE (TIE-3100)

9.3.1. Overview

- Identifier: **TIE 3100**
- Status: Success

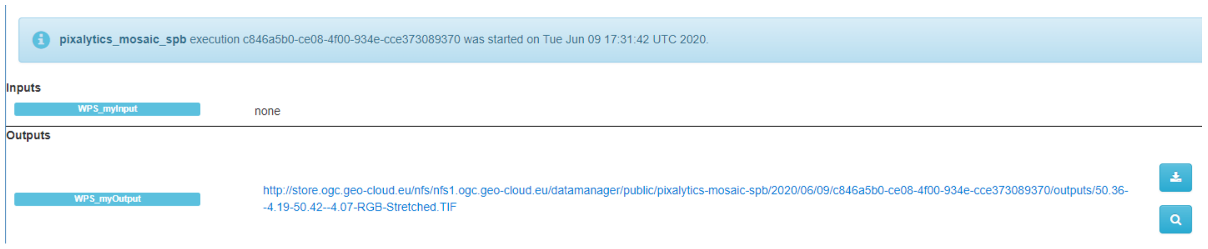
Pixalytics's application executes a Mosaic process from EO data downloaded from a remote repository.

9.3.2. Execution Procedure

The input selection in this experiment is hardcoded in the application container. Therefore, the application is simply executed using the execute button of the platform user interface.

9.3.3. Results

The *Mosaic* process generates a single GeoTIFF output as illustrated on figure below.


pixalytics_mosaic_spb execution c846a5b0-ce08-4f00-934e-cce373089370 was started on Tue Jun 09 17:31:42 UTC 2020.

Inputs

WPS_myInput none

Outputs

WPS_myOutput http://store.ogc.geo-cloud.eu/nfs/nfs1.ogc.geo-cloud.eu/datamanager/public/pixalytics-mosaic-spb/2020/06/09/c846a5b0-ce08-4f00-934e-cce373089370/outputs/50.36--4.19-50.42--4.07-RGB-Stretched.TIF

Figure 21. Mosaic TIE Execution

9.3.4. Lessons Learned

From a provider point of view, the *Mosaic* process has the particularity to fetch remote data. Pixalytics explored multiple approaches which highlight valuable improvement such as supporting remote storage system support (S3, FTP, etc.) and handling authentication to possibly mount and access remote repositories.

During the project, Pixalytics also expressed the need to register the Docker image in a private Docker registry. To this end, Spacebel provided access to such a Docker registry.

9.3.5. Pixalytics CWL Package

The CWL of the *Mosaic* process is provided below:

```
cwlVersion: v1.0
class: CommandLineTool
id: pixalytics_mosaic_spb
label: Pixalytics API application
hints:
  DockerRequirement:
    dockerPull: eu.gcr.io/spb-gep-ogc/pixalytics_mosaic_remote:latest
inputs:
  myInput:
    label: Atmospheric correction flag
    default: none
    inputBinding:
      eo:envvar: WPS_myInput
    type: String
outputs:
  WPS_myOutput:
    type: File
    outputBinding:
        glob: '50.36--4.19-50.42--4.07-RGB-Stretched.TIF'
    eo:envvar: WPS_myOutput
$namespaces:
  eo: http://www.opengis.net/eo/cwl/extension
```

9.4. ESA Mosaic TIE (TIE-4001)

9.4.1. Overview

- Identifier: **TIE 4001**
- Status: Success

ESA’s application executes a Mosaic process from S3 EO data.

9.4.2. Execution Procedure

The input selection is performed manually by browsing the Catalogue S3 products. The execution is started using the platform User Interface.

9.4.3. Results

The *Mosaic* process generates a GeoTIFF image as output, a PNG quicklook and a a property file.

9.4.4. Lessons Learned

The experiment required a new EO data collection (Sentinel-3) which has been added to the ingestion system.

Moreover, ESA tried to rapidly be able to deploy and execute the process which showed that a simpler deployment profile would be welcome. Indeed, in this experiment, the CWL does not really bring any added-value and is too complex.

9.4.5. ESA Mosaic CWL Package

The CWL of the *Mosaic* process is provided below:

```
cwlVersion: v1.0
class: CommandLineTool
id: esa-mosaic-spb
baseCommand: python3.6 /reading_folder_S3.py
label: ESA lst mosaic multiple inputs (spb)
hints:
  DockerRequirement:
    dockerPull: josemanueldelgadoblasco/lst_mosaic:v1.2_multiple_inputfiles
inputs:
  inputlist:
    label: S3 input products
    inputBinding:
      position: 1
    type:
      items:
      - File
      - Directory
      type: array
outputs:
  mosaic:
    outputBinding:
      glob: 'S3_*.tif'
    type: File
  quicklook:
    outputBinding:
      glob: 'S3_*.png'
    type: File
  properties:
    outputBinding:
      glob: 'S3_*.properties'
    type: File
```

Chapter 10. Conclusions and Recommendations

Application Providers essentially requested to reduce the complexity of the needed technical skills. On the other hand, Expert users are usually interested in controlling advanced functionalities such expressing complex inputs inter-dependencies.

Spacebel proposed a set of solutions and ideas to consolidate the EO platform architecture in this sense:

- Review the **EMS** component to mitigate the ambiguous aspects essentially by exploring a **Jupyter based approach** for handling the federated platform aspects (platform selection, data transfers, etc.).
- Solve a set of concerns related to Docker (image access control, security enforcement, version control) by executing the **Docker build pipeline on the ADES** side.
- The support of the de facto standard **Kubernetes for** defining the **Application Packages** (Jobs and Workflows) would be more relevant than the little-known CWL specification. Kubernetes covers Cloud-related aspects such as communication between containers, resource sharing, health check, init containers, job patterns, parallel executions for jobs, and Kubernetes Argo provides support for workflows (DAG or step-based) and pipelines with optimized execution on distributed environments. We recommend adopting Kubernetes properties for describing the container interfaces.
- Refine the (Testbed-14) **Docker Application Package to a simple** and limited set of mandatory items (as sketched in this report): novice developer only provides the Docker image and the command properties (based on Kubernetes properties), while expert users can also provide the Process Description with advanced information (e.g. EO constraints on specific inputs).
- Bank on the OGC WPS **Process Description** (common to all kinds of applications) to work on the needed extensions such as expressing application inputs constraints and correlations (e.g. based on OpenSearch EO, Geo and Time extensions), hardware requirements, etc.
- Investigate how **automation** (e.g. generation of the process description, building metadata for outputs) can simplify the Application Providers efforts by enabling default behaviors.

Although the EO platform architecture is typically running in a Cloud infrastructure, the current implementations were restricted to a single node. In this pilot, the Spacebel platform supports a Kubernetes cluster, which ensures scalability, availability, fault-tolerance, intelligent scheduling, automated deployment and the various other benefits of this very popular technology.

Moreover, Spacebel demonstrated 4 successful Technology Integration Experiments with Application Providers (SatCen, DLR, Pixalytics, ESA), collecting very positive feedback from those users. In particular, the user-friendly platform GUI and simplification of the deployment allowed Developers to start development on the intuitive platform.

Finally, the Web Processing Service should be improved to handle more geospatial aspects (e.g. EO typical inputs) and thus better catering for its original purpose.

Appendix A: Revision History

Table 1. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
2020-07-22	C. Noël	1	all	initial release
2020-09-29	G. Hobona	1	all	Final staff review

Appendix B: Bibliography

[1] Hobona, G.: OGC Hackathon 2019 Engineering Report. OGC 19-062,Open Geospatial Consortium, https://docs.ogc.org/per/19-062.html (2019).

[2] Sacramento, P.: OGC Testbed-14: Application Package Engineering Report. OGC 18-049,Open Geospatial Consortium, https://docs.ogc.org/per/18-049r1.html (2018).

[3] Sacramento, P.: OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report. OGC 18-050,Open Geospatial Consortium, https://docs.ogc.org/per/18-050r1.html (2018).

[4] Pross, B., Noel, C.: OGC Testbed-14: WPS-T Engineering Report. OGC 18-036r1,Open Geospatial Consortium, https://docs.ogc.org/per/18-036r1.html (2019).

[5] Meek, S.: OGC Testbed-14: BPMN Workflow Engineering Report. OGC 18-085,Open Geospatial Consortium, https://docs.ogc.org/per/18-085.html (2019).