OGC Testbed-14

ADES & EMS Results and Best Practices Engineering Report

Table of Contents

Publication Date: 2019-02-08

Approval Date: 2018-12-13

Submission Date: 2018-11-21

Reference number of this document: OGC 18-050r1

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D009

Category: Public Engineering Report

Editor: Paulo Sacramento

Title: OGC Testbed-14: ADES & EMS Results and Best Practices Engineering Report

---

OGC Engineering Report

COPYRIGHT

WARNING

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements. However, the discussions in this document could very well lead to the definition of an OGC Standard.

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.

Chapter 1. Summary

This Engineering Report (ER) describes best practices and results gathered through the work performed in the Exploitation Platforms Earth Observation Clouds (EOC) Thread of OGC Testbed-14 concerning the Application Deployment and Execution Service (ADES) and the Execution Management Service (EMS). Both the ADES and EMS were identified by the European Space Agency (ESA), beforehand, as essential elements of a Thematic Exploitation Platform (TEP).

In the context of a generic Earth Observation Exploitation Platform ecosystem, populated by TEPs and Mission Exploitation Platforms (MEPs), which make use of cloud computing resources for Earth Observation data processing, ESA has established two fundamental building blocks within a TEP, with different functions, the ADES and the EMS. Users interact with a TEP using a Web Client, and the TEP contains a EMS and a ADES. The EMS includes most of the control logic, required for deploying and executing applications in different MEPs and TEPs, the chaining thereof, and the overall coherence of the execution chain (e.g. gathering all outputs and enabling their presentation to the user by a client sensibly). The ADES instead is responsible for the single application deployment and execution on a specific platform. Therefore, it is expected that there are ADES instances both in a TEP and in the individual MEPs.

The Testbed-14 Participants have experimented with different options for what concerns the functionality allocated to each of the two components, the information required by each of them and the interface requirements between them in order to produce a consistent chain, compliant with ESA's objectives (as the Sponsor). This report describes these experiments, providing their results and suggesting best practices on how the two services should be engineered in the Exploitation Platform context.

The OGC Web Processing Service (WPS) 2.0 standard is of particular relevance given that it is well-established in the OGC Web Service context, specifically that concerning processing, its interoperability value has been clearly demonstrated, and it therefore provides a useful mechanism for standardizing interfaces between components of heterogeneous provenance and implementation.

1.1. Requirements & Research Motivation

As introduced above, the ADES and EMS are two architectural blocks with different functions within TEPs/MEPs. They are required to support application deployment, execution and chaining, as well as the agreed authentication and authorization mechanisms. All interfaces, including the security ones, have to be standard so as to achieve the Sponsor's interoperability goal in the heterogeneous Exploitation Platform context.

Given that several options are possible for what concerns the functionality allocation and interface details, it is important to record those that were subject to experiment in Testbed-14, as well as the results of the experiments. Finally, it is important to recommend best practices in doing so.

1.2. Prior-After Comparison

In Testbed-13, a simpler architecture was considered, with just an Application Management Client

(AMC) and an Application Deployment and Execution Service (ADES) involved, i.e. no Execution Management Service (EMS). The interface between the AMC and the ADES was based on WPS, in its traditional XML encoding, with two alternative implementations having been pursued, one relying on standard WPS 2.0 and two dedicated - but regular - WPS processes for deploying and undeploying processes/applications in the ADES; and the other one relying on the so-called transactional extension of WPS, which has not been standardized but is described in a OGC discussion paper [1]. For more details, please refer to [2].

In Testbed-14, WPS is again the key OGC standard involved and there was consensus in using the transactional extension, both for the client<→EMS and for the EMS<→ADES interface. Furthermore, in line with one of the Sponsor's requirements, the Extensible Markup Language (XML) encoding was dropped in favor of one based on Representational State Transfer (REST)/JavaScript Object Notation (JSON) which is also not yet an OGC standard but a version of which was submitted for public review by the WPS Standards Working Group (SWG) during the timeline of the Testbed. In fact, the WPS 2.0 SWG and the Testbed-14 EOC Participants coordinated to align, to the maximum extent possible, the work and lessons learned in Testbed-14 with the evolving WPS 2.0 REST/JSON specification.

1.3. Recommendations for Future Work

The OGC WPS 2.0 standard was already a key part of the work done in Testbed-13 and continued to be so in Testbed-14. Indeed, given that one of the core aspects of Earth Observation Exploitation Platforms is the data processing, WPS is a natural fit. At the same time, not only in Testbed-13 but in general, variants and possibilities related to WPS, such as the Transactional extension (WPS-T) and the reliance on REST bindings and JSON encoding, are of increasing interest. The growing popularity of WPS-T and REST/JSON is because they bring the OGC, and WPS in particular, closer to the mainstream web and technological realms.

Given that a significant amount of effort in the EOC thread of Testbed-14 has been dedicated to experimenting with these WPS variants and possibilities, this work is expected to be of high interest to the WPS 2.0 SWG. The relevance of the Common Workflow Language (CWL) to the OGC should also be investigated. It could be particularly interesting for the Workflow DWG and could be considered as a potential future Community Standard.

Future work should be dedicated to consolidating the WPS REST API that has been proposed by the WPS SWG and contributed to by Testbed-14 EOC thread Participants, as well as initiating a standardization path for the transactional extension of WPS (WPS-T). A specific issue of interest (but also risk) resides in how to transform a message-based protocol such as WPS (with its traditional *GetCapabilities*, *DescribeProcess*, *Execute* operations) into a resource-based protocol (WPS REST), where resources such as *processes* and *jobs* are managed using the basic HTTP operations (*GET*, *POST*, *PUT*, etc.). A straightforward conversion (i.e. simply mapping the XML messages into REST/JSON) may result in APIs which are not intuitive and are semantically unclear. It is therefore important to dedicate time and effort to properly designing a resource-based version of a previously existing XML-based protocol. Another relevant and general issue in this discussion is how to transform/encode XML into JSON content. In coordination with the WPS 2.0 SWG, several smaller or one or two larger Change Requests can be prepared so as to start the standardization path for WPS-T and the WPS REST/JSON work.

Further areas of interest for future work are:

- the standardization of protocols and mechanisms for the ADES to fetch data required for application execution (users shall not need to change their applications due to specific data fetching requirements)
- more sophisticated and dynamic ways of associating data collections to execution platforms (the one used in the Testbed is completely static)
- the ability for systems involved in the flow to communicate between them, in an unambiguous and well-understood manner, lists of products to process and information on whether a catalogue search has been performed to determine them already (or if it is still required); see also next point
- an ADES can be used for single application executions but also for bulk processing (operating simply on an AOI and TOI, rather than receiving a product or list of predefined products to process). The bulk processing scenario has not been explored in Testbed-14 and could be in future ones
- the issue of application visibility and the level at which it is managed (Client, EMS, ADES), as well as the propagation/delegation of user credentials for authentication and authorization, if this is not simply considered an orthogonal discussion. In the work performed, it has been considered that MEPs are opaque and changing application visibility is only possible at EMS level.
- the ability to rely on authoritative *codespace* definitions
- the ability to rely on an authoritative list and/or well-understood mapping of *mimeTypes* (e.g. application/x-binary, application/octet-stream, plain/text) useful for the purposes of the EOC thread (to avoid ambiguity in understanding and usage - for example specify that *application/x-binary* is to be always used for images and *plain/text* for WKT). For EO Images in particular, a much more detailed classification would also be useful (to distinguish between different formats used for a given mission, e.g. Sentinel-2 hosted on AWS and IPT Poland, and formats from different missions, e.g. Sentinel-2 and Landsat-8)
- the asynchronous quotation model (with a *status* property that can be polled), particularly as part of the WPS REST/JSON API
- standardization or creation of Best Practices for platform identifiers (TEPs, MEPs such as "Sentinel-2 MEP @ XYZ" or "Proba-V MEP @ ABC") so they can be referred to univocally, maybe taking advantage of other standards (e.g. *skos:prefLabel* in Global Change Master Directory, http://gcmdservices.gsfc.nasa.gov/static/kms/platforms/platforms.rdf)
- WPS support for multiple outputs (see Chapter 10 for more information. The WPS SWG should look at making this more flexible)
- how to handle OGC Web Service interfaces providing process inputs/outputs

The work described in the ER makes the case for how OGC standards are helpful in such a densely populated ecosystem of very heterogeneous entities, technologies and implementations such as the Exploitation Platforms one. For OGC therefore, the work described in this ER is deemed extremely relevant in satisfying the needs of the European Space Agency as one of its Strategic Partners.

1.4. Document contributor contact points

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

Contacts

Name	Organization
Paulo Sacramento	Solenix
Patrick Jacques	Spacebel
Christophe Noel	Spacebel
Peter Vretanos	CubeWerx
Jerome Gasperi	Geomatys
Guilhem Legal	Geomatys
Tom Landry	CRIM
David Byrns	CRIM
Francis Charette-Migneault	CRIM

1.5. Acknowledgements

Besides the Tested-14 Participants, the following institutions have supported the activities of the thread:

- Deimos-Imaging
- VITO
- The FedEO Team at Spacebel
- ATOS CVC Romania

1.6. 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 2. References

The following normative documents are referenced in this document:

- OGC: OGC 06-121r9, OGC® Web Services Common Standard, 2010 [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 13-032r8, OGC OpenSearch Geo and Time Extensions 1.0.0, 2014 [https://portal.opengeospatial.org/files/?artifact_id=56866]

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.

3.1. Abbreviated terms

- ACL Access Control List
- ADES Application Deployment and Execution Service
- AOI Area Of Interest
- AP Application Package
- AWS Amazon Web Services
- CFP Call For Participation
- CORS Cross-Origin Resource Sharing
- CP CheckPoint
- CWL Common Workflow Language
- DACS Distributed Access Control System
- DWG Domain Working Group
- ECS Elastic Container Service
- EMS Execution Management Service
- EO Earth Observation
- EOC Earth Observation Clouds
- EP Exploitation Platform
- ER Engineering Report
- ESA European Space Agency
- FEDEO Federated Earth Observation
- GUI Graphical User Interface
- IaaS Infrastructure as a Service
- ICT Information and Communications Technology
- IdP Identity Provider
- IPT Innovative Platform Testbed (Poland)
- IT Information Technology
- JSON JavaScript Object Notation
- M2M Machine-2-Machine (interface)
- MEP Mission Exploitation Platform

- NRCan Natural Resources Canada
- OAS3 OpenAPI 3 Specification
- OSDD OpenSearch Description Document
- OWC OWS Context
- OWS OGC Web Services
- PaaS Platform as a Service
- PEP Policy Enforcement Point
- PFC Pacific Forestry Center
- QoS Quality of Service
- REST REpresentational State Transfer
- SNAP SeNtinel Application Platform
- SOAP Simple Object Access Protocol
- STS Security Token Service
- SWG Standards Working Group
- 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
- WFS Web Feature Service
- WPS Web Processing Service

Chapter 4. Overview

Section 5 briefly recaps the work done in Testbed-13 for what concerns the EMS and the ADES, as a technical introduction to the topics dealt with in this Engineering Report.

Section 6 discusses the sponsor requirements to which the identified solutions respond to.

Section 7 summarizes the final solution as implemented by Participants for what concerns the EMS, the ADES and the interface between them. For several details concerning the authentication, authorization, quotation and billing aspects, the reader is referred to the separate Testbed-14 Engineering Report, [3], which deals specifically with those.

Section 8 deals with aspects specific to the EMS, with contributions from each of the implementing Participants.

Section 9 deals with aspects specific to the ADES, with contributions from each of the implementing Participants.

Section 10 lists issues of interest and findings that the work performed by the Participants allowed to identify. These include accepted limitations and assumptions made for the purpose of the Testbed, that would have to be addressed in a more operational context, as well as particularly relevant trade-offs that were done to inform decisions.

Section 11 presents the TEP Client through screenshots which illustrate the main functionality. Even though the CFP did not require the client to be described in the ERs - this document focuses on the EMS and the ADES and the two other ERs focus on the Application Package and on the Authentication, Authorization, Billing and Quotation aspects - it was considered that some content on the client would also be useful.

Chapter 5. Recap Of Work Done In Testbed-13

The architecture established in the EOC thread of Testbed-13 by the Sponsor and embraced by the Participants consisted of two tiers: an Application Management Client (AMC) and an Application Deployment and Execution Service (ADES). Several implementations of these (3 each) were implemented so as to demonstrate interoperability and prove that different Participants, even in charge of implementing only one of the tiers, could successfully integrate their components using an OGC standard.

The interface between the AMC and the ADES was based on WPS, in its traditional XML encoding, with two alternative implementations having been pursued. One implementation relied on standard WPS 2.0 and two WPS processes for deploying and undeploying processes/applications in the ADES. The other implementation relied on the so-called transactional extension of WPS, which has not been standardized but is described in a OGC discussion paper [1].

The Testbed-13 work explored cloud computing service models of Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS). The figure below depicts the Testbed-13 architecture and flow, which allowed users (Jim and Marco) to interact with the two above-mentioned tiers of a TEP/PaaS (the AMC and the ADES) in order to deploy and execute Earth Observation (EO) applications running on two IaaS providers, namely Amazon Web Services (AWS) and IPT Poland. The figure includes the security (authentication and authorization) components (CP, IdP, STS), as well as the App/Docker Registry/Hub and the Central Geospatial Resource Catalogue. Due to lack of time, interactions with the security components were not implemented in Testbed-13. Containerization in Docker and registration of applications in the App/Docker Registry/Hub were pre-requisites for any applications that were deployed and executed. The Central Geospatial Resource Catalogue consisted of a deployment of Federated Earth Observation (FEDEO) gateway software, provided as in-kind contribution by the Sponsor.

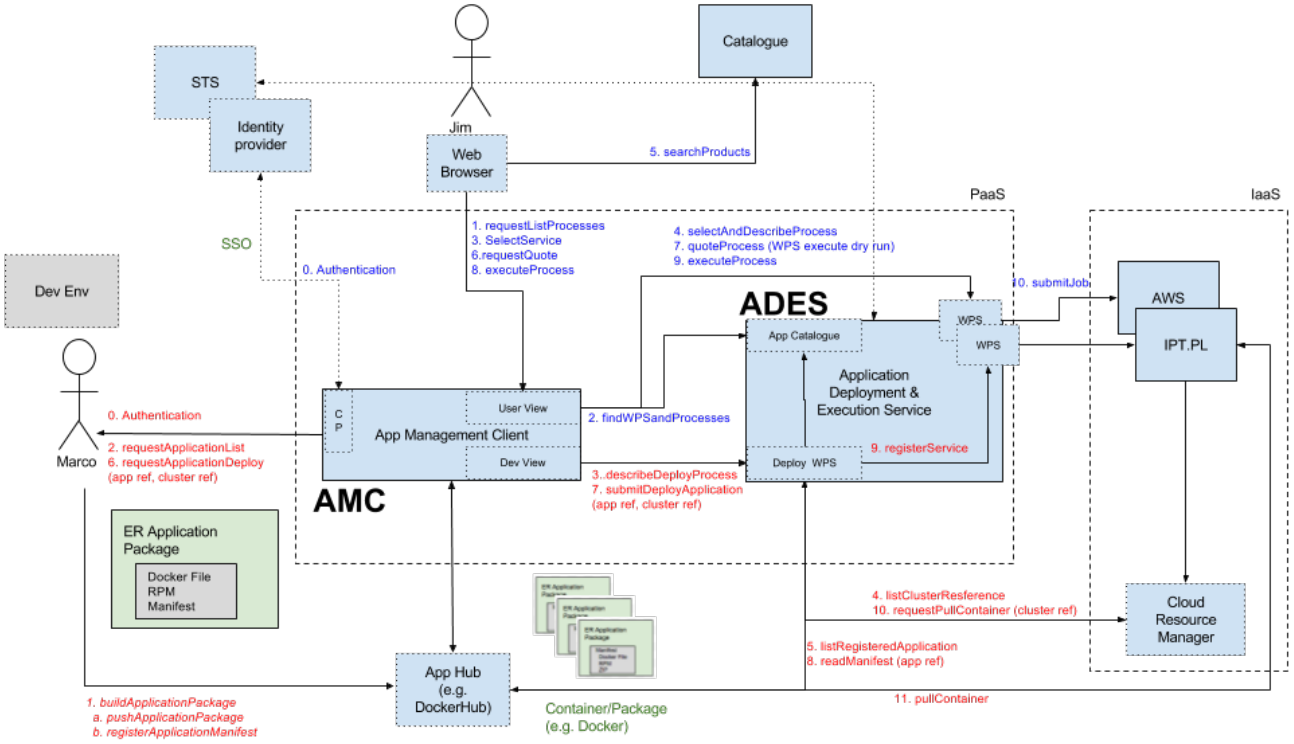

Figure 1. Testbed-13 Architecture and Flow

More details can be found in [2]. Additional findings on interoperability and portability of cloud computing architectures can be found in [4], also part of Testbed-13 EOC thread. The deliverables demonstrated data processing tools for deployment, management, and processing of EO data using an OGC WPS. As such, the proposed interfaces are coherent with the AMC, ADES and AP principles described here.

5.1. ADES in Testbed-13

The Application Deployment and Execution Service in Testbed-13 was a single server-side component and it was in charge of all aspects pertaining to the deployment and execution of EO applications on different cloud providers. It therefore shielded instances of the Application Management Client from all infrastructure details. In the canonical case, it was exposed as a WPS interface, with two pre-existing processes dedicated to the deployment and undeployment of applications. In the alternative implementation, it exposed a WPS-T interface in-line with [1], which adds two new operations to the basic WPS ones, namely DeployProcess and UndeployProcess.

The ability to register and unregister processes, just as any other WPS server, makes the ADES also a *defacto* repository and catalogue of processes that can be discovered using a GetCapabilities request and described in detail by issuing a DescribeProcess request for the process of interest. Thus, in Testbed-13, the ADES also performed the role of process/application catalogue, available for query by clients.

More details can be found in [2].

5.2. AMC in Testbed-13

The Application Management Client in Testbed-13 consisted of a WPS client implementing the WPS XML binding with support for the Simple Object Access Protocol (SOAP), and relying on ADES instances for all functionality. Briefly, the client allowed:

- a privileged user to register and unregister applications using a specified Application Package format (see [5]) and dedicated processes exposed by ADES instances
- to browse, select and use previously registered applications (using the WPS GetCapabilities operation)
- for applications requiring a catalogue search, to select a collection of interest, an Area-Of-Interest (AOI) and Time-Of-Interest (TOI) through a dynamically-built GUI (using the WPS DescribeProcess operation), perform queries and select a result
- execute applications, monitor execution conclusion and get results (using the WPS Execute, GetStatus and GetResult operations)

More details can be found in [2].

5.3. WPS in Testbed-13

As described above, the OGC WPS protocol was the basis for the interface between the several AMC and ADES instances in Testbed-13. It was used in its SOAP/XML bindings and encoding and, except for the alternative approach pursued by one of the Participants, which relied on the WPS-T extension described in [1], was the standard WPS 2.0 specification.

More details can be found in [2] and [4].

Chapter 6. Discussion of Sponsor Requirements

The figure below, taken from the Call For Participation (CFP), illustrates the basic architecture required by the Sponsor, consisting of the EMS and the ADES. The figure highlights that the same ADES software can be executed within a TEP and within a MEP. The Web-Client is depicted as an interface available to the TEP users (*Alice* and *Bob*) for accessing the TEP functionality, but it is actually a separate Testbed-14 deliverable.

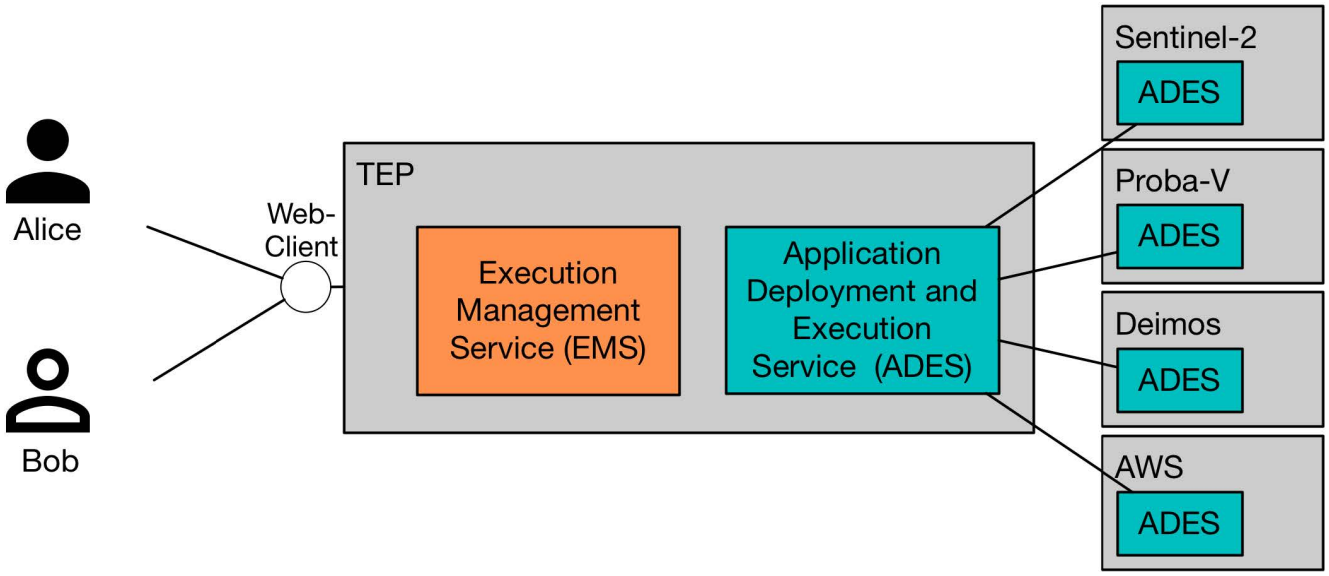

Figure 2. Testbed-14 Architecture from CFP

It must be highlighted that the diagram above has a misleading aspect concerning the ADES(in TEP)→ADES(in MEP) interface which is depicted. In fact, during the Testbed, it was agreed by all Participants, Sponsor and OGC that there should be no such interface and it is the EMS directly orchestrating and dispatching the application executions to the ADES in a MEP. The diagram presented in the next chapter contains the accurate flow.

The main requirements set out in the CFP can be enunciated as:

- ability to exercise a flow involving different mission and thematic exploitation platforms (MEPs and TEPs)
- support for two different types of user: an application developer (*Alice*) and an application consumer (*Bob*)
- ability to chain applications, where one uses the results of the other
- make use of standardized error handling and user messaging
- ensure users are authenticated and authorized to perform required operations
- allow users to manipulate application privacy settings for visibility, execution and update/removal

These requirements are exercised by implementing the following scenarios (refer to the CFP for complete details):

- Scenario 1: *Alice*, the application developer, loads a single application package in the TEP
- Scenario 2: *Alice* registers a basic chaining of applications on the TEP
- Scenario 3: *Bob* explores existing applications, selects the (chained) application registered by *Alice* in scenario 2 and uses it (selecting collections, AOI, TOI as necessary, waiting for the TEP EMS to orchestrate the several ADES instances in the MEPs, monitoring the execution and getting the results)

The following architectural constraints relevant to the EMS and ADES functionality were specified in the CFP:

- Each of the MEPs involved in the scenarios shall have an ADES in charge of application registration, deployment, execution, modification (by authorized users)
- A TEP may include both an EMS and an ADES or just an EMS (i.e. relying on ADES in MEPs)
- The EMS is in charge of dispatching the application execution to the relevant ADES/MEP and orchestrate the chaining
- All Machine-to-Machine (M2M) interfaces shall use RESTful architecture with JSON encoding

Further sponsor requirements and constraints related to billing, authentication and authorization are discussed and addressed in [3].

Chapter 7. Solution Overview

This chapter describes the finally adopted solution in three sections. The first section details the architecture, the second details the interfaces and the third explains the sequence of steps involved in the main implemented flows. The implementation experiences of several Participants for the EMS and ADES building blocks are described in the next chapters.

7.1. Architecture

The figure below depicts the architecture adopted by Testbed-14 Participants, with the main building blocks and interfaces, as well as some implementation notes that highlight important assumptions agreed between all stakeholders, including the Sponsor.

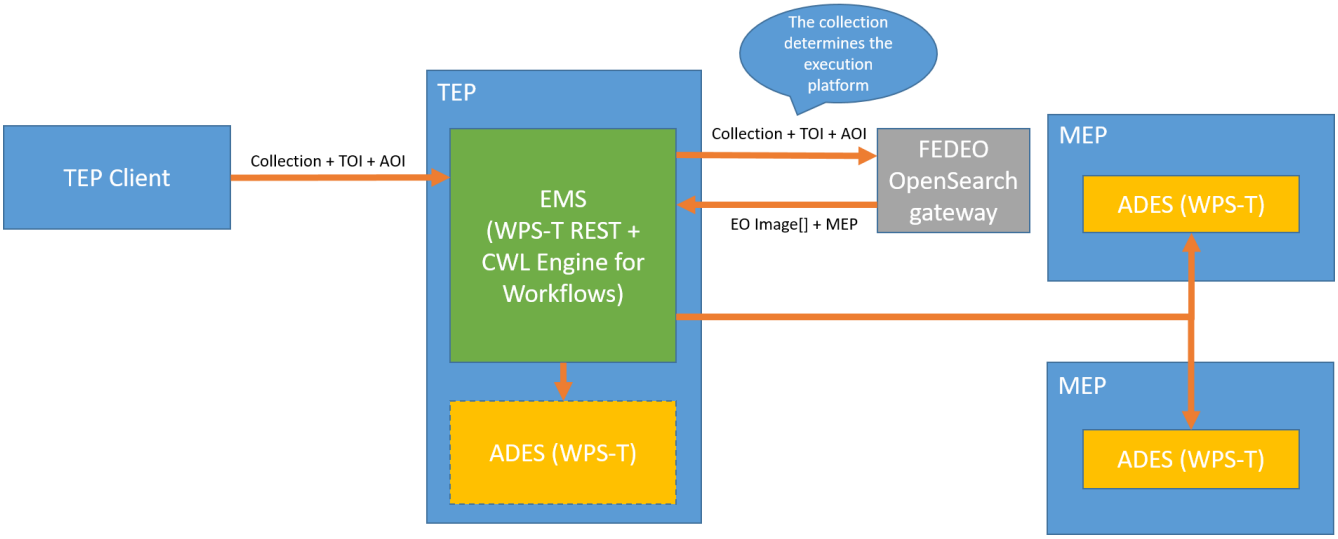

Figure 3. Final Testbed-14 EOC Thread Architecture

Three aspects in particular are worth noting as they represent changes with respects to the Sponsor requirements or issues left open in the CFP that have been herewith addressed:

- The TEP Client does not interface with the FEDEO OpenSearch gateway, it simply collects AOI, TOI and collection information from the user and sends these to the EMS as part of the execution request. The EMS then interfaces with the OpenSearch gateway in order to identify images of interest (a list) and the execution platform (see next point)
- The collection chosen by the user unequivocally identifies the execution platform (1-to-1 relationship), i.e. neither the Client nor the EMS need to implement any logic to determine it. The execution platform also translates to a MEP, i.e. the collection choice determines the MEP (see next point)
- For application execution, the EMS can rely both on ADES instances within the same TEP or outside ADES instances within MEPs. Both scenarios are in line with the Sponsor requirements, but only the latter has been exercised in the TIEs (the former is just a simpler case).

Particularly the first and second bullets above represent simplifications of the real scenario that would need to be addressed in a more operational context. For what concerns the first bullet, if instead it were the TEP Client or the ADES in charge of performing the search (or if this was a possibility), further work and logic would be required between the elements involved in the flow, to

communicate the list of products to be processed and whether such a search had been performed already or not.

7.2. Interfaces

7.2.1. WPS-T REST/JSON

The Testbed-14 EOC thread Participants cooperated in order to specify a WPS interface using a RESTful protocol, JSON encoding and the transactional extension of WPS (WPS-T) described in [1]. Given that during the timeframe of the Testbed there was ongoing work by the WPS SWG to generate a similar specification, except for the transactional aspect, the two parties coordinated to ensure alignment. At the end of the Testbed, a Change Request to the WPS standard, consisting of the agreed specification (and the numerous details involved) will be submitted to complete the alignment.

This interface is used both between the TEP Client and the EMS and between the EMS and the ADES.

The complete specification, defined using OpenAPI (originally known as Swagger), can be found in https://github.com/opengeospatial/D009-ADES_and_EMS_Results_and_Best_Practices_Engineering_Report/blob/master/code/ades_wpst.json . This machine-readable way of describing an interface details valid paths, supported HTTP methods for each path, expected parameters (including whether they are mandatory or not) and the possible responses (including the standard HTTP error codes).

The main resources and relevant details are listed in the table below.

Table 1. Resources of WPS-T REST interface

Resource/Path	HTTP Method	Purpose	Notes
/	GET	The landing page provides links to the API definition, the Conformance statements and the metadata about the processes offered by this API	
/processes	GET	Retrieve available processes	
	POST	Deploy a process	
/processes/{id}	GET	Retrieve a process description	
	DELETE	Undeploy a process	
/processes/{id}/jobs	GET	Retrieve the list of jobs for a process	
	POST	Execute a process	

Resource/Path	HTTP Method	Purpose	Notes
/processes/{id}/jobs/{jobID}	GET	Retrieve the status of a job	
	DELETE	Dismiss a job	
/processes/{id}/jobs/{jobID}/result	GET	Retrieve the result(s) of a job	
/processes/{id}/quotations	GET	Retrieve the list of quotation ids for a given process	See *Note 1*
	POST	Request a quotation for a given process	See *Note 1*
/processes/{id}/quotations/{quotationID}	GET	Retrieve quotation information	See *Note 1*
	POST	Execute a quoted process	See *Note 1*
/processes/{id}/visibility	GET	Retrieve the visibility status for a process	
	PUT	Change the visibility status for a process	
/quotations	GET	Retrieve the list of all quotation ids	See *Note 1*
/quotations/{quotationID}	GET	Retrieve quotation information	See *Note 1*
	POST	Execute a quoted process	See *Note 1*
/bills	GET	Retrieve the list of all bill identifiers	See *Note 1*
/bills/{billID}	GET	Retrieve bill information	See *Note 1*
/conformance	GET	list all requirements classes specified in the standard (WPS REST/JSON Binding Core) that the server conforms to	

Note 1: More information about the billing and quotation aspects is available in [3].

7.3. Sequence of Operations

This section describes in full detail the steps of the sequence of operations required for deploying and executing both applications and workflows:

1. Alice deploys a newly developed application.
2. Alice's sister discovers the new application and builds a workflow including that application.

3. Alice's sister deploys the workflow.
4. Bob discovers the new workflow and executes it.
5. The EMS performs the EO Image Discovery.
6. The EMS triggers the deployment on the ADES.
7. The ADES executes the application.

Steps 5, 6 and 7 pertain to steps internal to the EMS, required to execute a workflow or application.

Once again, the security, quoting and billing aspects are not covered here as they belong to the scope of [3].

The sequence of operations is shown in the diagram below.

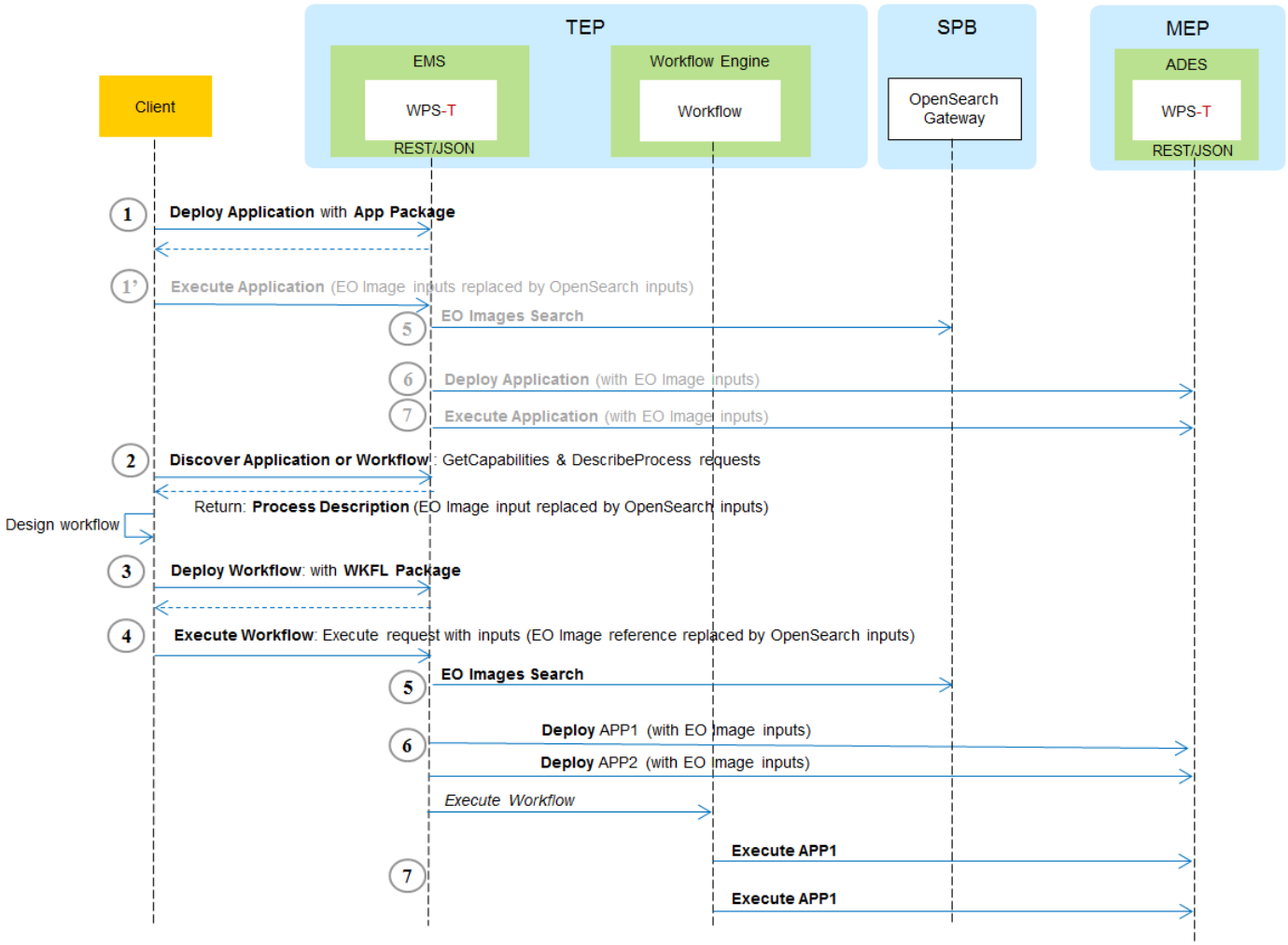

Figure 4. Sequence diagram for main Testbed-14 EOC Thread flows

The execution of a simple application (following step 1) is not covered explicitly because the steps (5, 6 and 7 in particular) are similar to the execution of a workflow, except that the EMS in that case is only a broker in front of the ADES.

For a more focused discussion of the Application Package format and its facets, the reader is referred to [6]. That document contains a brief introduction of CWL and an explanation of its role in the Testbed work.

7.3.1. Application deployment (Step 1)

Alice developed an application identified as 'MultiSensorNDVI'. She wants to make this application available to the users of the TEP.

Using the TEP Client, Alice performs a Deployment request (step 1). The WPS-T REST HTTP POST operation path is */processes* , and the request message includes the following items:

- The Process Description: a standard (WPS) description of the application interface used by the client. In particular, it includes a project-specific flag (EOImage) for Earth Observation Image inputs, and it also references a CWL file providing information on the Docker image of the application.
- The Execution Unit: the application binaries, packages and resources. In this case, this is a reference to the Docker image.
- The identifier of the Profile (in this case, a Dockerized application).

```
{
    "processDescription": {
        "process": {
            "id": "NDVIMultiSensor",
            "title": "NDVIMultiSensor",
            "owsContext": {
                "offering": {
                    "code": "http://www.opengis.net/eoc/applicationContext/cwl",
                    "content": {
                        "href": "https://some-host/CWL/NDVIMultiSensor.cwl"
                    }
                }
            },
            "abstract": "Normalized Difference Vegetation Index (NDVI) from an input list
of satellite images.",
            "keywords": ["NDVI"],
            "inputs": [
                {
                    "id": "files",
                    "title": "Input Image",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        },{
                            "mimeType": "application/x-hdf"
                        }
                    ],
                    "minOccurs": "1",
                    "maxOccurs": "unbounded",
                    "additionalParameters": [
                        {
                            "role":
```

```
"http://www.opengis.net/eoc/applicationContext/inputMetadata",
                            "parameters": [
                                {
                                    "name": "EOImage",
                                    "values": ["true"]
                                }
                            ]
                        }
                    ]
                }
            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "NDVI Images",
                    "formats": [
                        {
                            "mimeType": "image/tiff",
                            "default": true
                        }
                    ]
                }
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "reference"
        ]
    },
    "immediateDeployment": true,
    "executionUnit": [{
            "href": "docker.registry/ndvims:latest"
        }],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/dockerizedApplication"
}
```

It is important to note that, even though the Application Package above is for a simple application and not a workflow, CWL is still used. This is done so that, downstream, the ADES can map inputs/outputs to the Docker and run it properly. In fact, this solves an open issue from Testbed-13 (see the *stagein/stageout vs. environment variables* discussion in [2] and [7]). Therefore, even if it can be argued that this solution introduces a dependency on CWL even when only simple applications are involved, the Participants believe that this dependency is justified. Furthermore, this dependency only affects the EMS, not the ADES. ADES implementers can choose whether to rely on the CWL file (which is passed seamlessly by the EMS) or not (i.e. rely only on the WPS Process Description). If using CWL, the *CWL-runner* tool which is provided as part of the standard CWL toolset automatically manages the stagein/stageout operations.

Participants also highlighted that alternatives to CWL for describing a container interface are few, language-specific and are not exhaustive. CWL is a *defacto* standard language, providing the appropriate flexibility and simplicity for describing all cases appearing when defining a command-line interface. The risk of this dependency is therefore considered to be absolutely under control.

Following the request above, the TEP Client receives an acknowledgment of the successful deployment of the process as illustrated below.


```
{
  "processSummary": {
    "id": "MultiSensorNDVI",
    "title": "Multi Sensor NDVI",
    "abstract": "NDVI is calculated after the two bands values Near Infrared and red. It
is calculated by this formula : NDVI = (NIR-Red)/(NIR+Red)",
    "keywords": [
      "NDVI"
    ],
    "version": "1.0.0",
    "jobControlOptions": [
      "async-execute"
    ],
    "processDescriptionURL": "http://some.domain/wps/processes/MultiSensorNDVI"
  }
}
```


7.3.2. Application discovery and workflow design (Step 2)

Alice's sister is preparing a processing chain workflow. She first needs to discover the applications available on the TEP (step 2). The TEP Client can list a summary of the available processes.

The WPS-T REST HTTP GET operation path is */processes* and the response is illustrated below.

```
{
  "processes": [
    {
      "id": "NDVIMultiSensor",
      "title": "NDVIMultiSensor",
      "jobControlOptions": [
        "async-execute"
      ],
      "outputTransmission": [
        "reference"
      ],
      "processDescriptionURL": "http://185.52.193.7/wps-
proxy/processes/GeomatysNDVIMultiSensor"
    },
    {
      "id": "NDVIStacker",
[...]
    }
  ]
}
```


As explained in [6], the chosen language to describe workflows in the EOC thread of Testbed-14 is CWL. Therefore, for each application that Alice plans to include, the CWL file of the application needs to be retrieved by the client using a DescribeProcess operation. In this instance, the EMS on the TEP functions as a repository or catalogue of CWL files (pointed to by the corresponding WPS Process Descriptions).

The WPS-T REST HTTP GET operation path is */processes/{processId}* and the response illustrated below includes the CWL reference which was provided in the OWS Context element during deployment.

```
{
  "process": {
    "id": "NDVIMultiSensor",
    "title": "NDVIMultiSensor",
    "abstract": "Normalized Difference Vegetation Index (NDVI) from an input list of
satellite images.",
    "owsContext": {
      "offering": {
        "code": "http://www.opengis.net/eoc/applicationContext/cwl",
        "content": {
          "href": "https://some-host/CWL/NDVIMultiSensor.cwl"
        }
      }
    },
 [...]
  },
  "processVersion": "1.0.0",
  "jobControlOptions": [
    "async-execute"
  ],
  "outputTransmission": [
    "reference"
  ]
}
```


Note that the input and output description parts have been hidden from the example and will be covered in the workflow execution step (step 4).

Alice's sister can compose her CWL workflow using her preferred CWL workflow designer tool (e.g. Rabix Composer embedded by the TEP Client) and import the various applications' CWL files for building the workflow steps.

Before deploying the workflow generated by the design tool, the run property must be verified (can be enforced by the TEP Client) to ensure that:

- The CWL file path should be restricted to the file name.
- The CWL file name must be the process identifier (defined in the WPS Process Description).

```
  "steps":{
     "myOwnStep":{
        "run":"NDVIMultiSensor.cwl",
        "in":{
           "files":"myWorkflowInput"
        },
        "out":[
           "myOutput"
        ]
     },
   }
```

7.3.3. Workflow Deployment (Step 3)

Alice's sister composed the Multi Sensor NDVI Stack Generator processing chain. The chain performs a Multi Sensor NDVI processing on each of the 3 received EO Image inputs, then stacks the generated outputs, as illustrated on the diagram below.

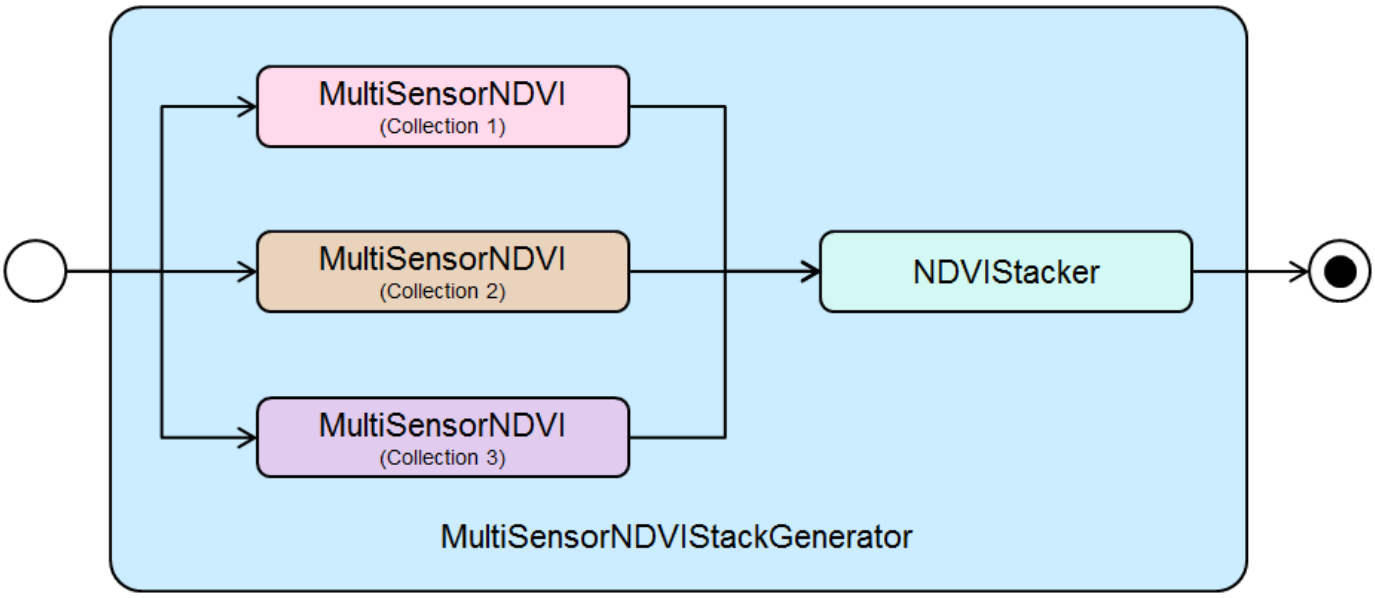

Figure 5. multisensorNDVIWorkflow

Alice's sister prepares the WPS Process Description (with the help of the TEP Client) for deploying the processing chain workflow, and uses the client to perform the deployment request. The WPS-T REST HTTP POST operation path is */processes* and the request is shown below.

```
{
    "processDescription": {
        "process": {
            "id": "MultiSensorNDVIStackGenerator",
            "title": "MultiSensorNDVIStackGenerator",
            "abstract": "",
            "keywords": [],
            "inputs": [
                {
```

```
                    "id": "image-collection1",
                    "title": "Input Image",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        }
                    ],
                    "minOccurs": 1,
                    "maxOccurs": "unbounded",
                    "additionalParameters": [
                        {
                            "role":
"http://www.opengis.net/eoc/applicationContext/inputMetadata",
                            "parameters": [
                                {
                                    "name": "EOImage",
                                    "values": [
                                        "true"
                                    ]
                                }
                            ]
                        }
                    ]
                },
                {
                    "id": "image-collection2",
                    [...]
                },
                {
                    "id": "image-collection3",
                    [...]
                }
            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "Stacked Image",
                    "formats": [
                        {
                            "mimeType": "image/tiff",
                            "default": true
                        }
                    ]
                }
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
```

```
        "outputTransmission": [
            "reference"
        ]
    },
    "executionUnit": [
        {
            "href": "https://some-host/CWL/MultiSensorStackGenerator.cwl"
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/workflow"
}
```

The client receives a similar deployment confirmation message as described earlier.

7.3.4. Workflow Execution (Step 4)

Bob tries to discover the applications and workflows available on the TEP in order to perform an execution (step 4). The TEP Client can list the available processes using the */processes* REST path as already mentioned earlier.

When a WPS Process Description is requested by the EMS client, the returned document differs from the one that was submitted by Alice. Indeed, the description of the EOImage input is replaced by fields required to perform a OpenSearch Catalogue query. The EMS is responsible for retrieving the EO Image references by performing a Catalogue search and passing the returned product URLs. Therefore, the Process Description returned by the EMS looks as illustrated below.

The WPS-T REST HTTP GET operation path is */processes/{processId}* and the response is displayed below. In that example, the inputs *image-collection1*, *image-collection2*, and *image-collection3* were replaced by the fields os_aoi, os_startDate, os_endDate, and 3 other inputs for the collection identifier.

```
{
  "process": {
    "id": "MultiSensorNDVIStackGenerator",
    "title": "MultiSensorNDVIStackGenerator",
    "abstract": "",
    "owsContext": {
      "offering": {
        "code": "http://www.opengis.net/eoc/applicationContext/cwl",
        "content": {
          "href": "https://some-host/multiSensorNDVIStacker.cwl"
        }
      }
    },
    "inputs": [
      {
        "id": "os_collectionId_image-collection1",
        [...]
      },
      {
        "id": "os_collectionId_image-collection2",
        [...]
      },
      {
        "id": "os_collectionId_image-collection3",
        [...]
      },
      {
        "id": "os_startDate",
        [...]
      },
      {
        "id": "os_endDate",
        [...]
      },
      {
        "id": "os_aoi"
        [...]
      }
    ],
 [...]
  }
}
```

Bob executes the workflow by submitting an *Execute* request. The WPS-T REST HTTP POST operation path is */processes/{processId}/jobs*.

```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "os_collectionId_image-collection1",
      "data": "EOP:IPT:Sentinel2"
    },
    {
      "id": "os_collectionId_image-collection2",
      "data": "urn:ogc:def:EOP:VITO:PROBAV_P_V001"
    },
    {
      "id": "os_collectionId_image-collection2",
      "data": "DE2_MS4_L1B"
    },
    {
      "id": "os_aoi",
      "data": "100.4,18.3,104.6,19.3"
    },
    {
      "id": "os_startDate",
      "data": "2018-01-30T00:00:00.000Z"
    },
    {
      "id": "os_endDate",
      "data": "2018-01-31T23:00:59.000Z"
    }
  ],
  "outputs": [
    {
      "id": "output",
      "transmissionMode": "reference"
    }
  ]
}
```


The execution is confirmed by an HTTP 201 response code, and includes the status document URL in the "Location" HTTP header.

The WPS-T REST HTTP GET operation path is */processes/*{processId}_/jobs/{jobId}_. The status shall be polled until the state is succeeded (or failed) as shown below.


```
{
"status":"succeeded",
"message":"Status of job 35efcdb8-7447-46bb-8338-2e706d1cfece",
"jobId":"35efcdb8-7447-46bb-8338-2e706d1cfece"
}
```

For retrieving the Result document, the WPS-T REST HTTP GET operation path is */processes/*{processId}_/jobs/{jobId}/result_. Typically, the result outputs are provided by reference to avoid retrieving files to the EMS between all the steps.

```
{
   "outputs":[
      {
         "mimeType":"image/tiff",
         "href":"http://some-host/WPS/xxxYYY",
         "id":"output"
      }
   ]
}
```

7.3.5. EMS Execution Steps

The approaches for performing a workflow or an application execution are specific to the EMS implementation. However, the implementation always includes the three steps detailed further.

OpenSearch Catalogue (Step 5)

As mentioned above, the EMS replaces the EO Image inputs with OpenSearch query fields (collection identifier, area of interest and time of interest). They are used by the EMS for searching the products URLs on the Gateway. The Gateway is based on the OGC OpenSearch Extension for Earth Observation specification (OGC 13-026r8) and OpenSearch Geo and Time Extensions (OGC 10-032r8).

In the context of Testbed-14, the OpenSearch Gateway developed by Spacebel is in front of the three MEP catalogues and is configured with the following collections:

- EOP:IPT:Sentinel2
- urn:ogc:def:EOP:VITO:PROBAV_P_V001
- PROBAV_S1-TOA_1KM_V001
- DE2_MS4_L1B

The first step for retrieving EO Image references is to search for the provided collection. The collection identifier is supplied in the WPS-T REST HTTP GET operation path *http://geo.spacebel.be/opensearch/request?uid={collectionId}*.

The operation returns the list of collection entries matching with this collection identifier. For each entry, it includes a reference to the OpenSearch document that describes the collection search engine in *atom:feed/atom:link[@rel='search'][@type='application/opensearchdescription+xml']*.

The EMS then retrieves the OpenSearch Description Document (OSDD) for the requested collection. In the context of Testbed-14, the first step can be skipped, because the OSDD reference may be retrieved using the WPS-T REST HTTP GET operation path *http://geo.spacebel.be/opensearch/description.xml?parentIdentifier={collectionId}*.

The Collection OSDD document defines in particular the search template for the product query request

in *atom:entry/atom:link[@rel='search'] with @type='application/atom+xml'*, as illustrated below.

```
<Url indexOffset="1" pageOffset="1" rel="results" template=
"http://geo.spacebel.be/opensearch/request?httpAccept=application%2Fatom%2Bxml&amp;parent
Identifier=EOP:IPT:Sentinel2&amp;query={searchTerms?}&amp;startDate={time:start?}&amp;end
Date={time:end?}&amp;geometry={geo:geometry?}&amp;platform={eo:platform?}&amp;orbitNumber
={eo:orbitNumber?}&amp;frame={eo:frame?}&amp;sensorMode={eo:sensorMode?}&amp;swathIdentif
ier={eo:swathIdentifier?}&amp;orbitDirection={eo:orbitDirection?}&amp;antennaLookDirectio
n={eo:antennaLookDirection?}&amp;polarisationChannels={eo:polarisationChannels?}&amp;proc
essingLevel={eo:processingLevel?}&amp;maximumRecords={count?}&amp;uid={geo:uid?}&amp;name
={geo:name?}&amp;lat={geo:lat?}&amp;lon={geo:lon?}&amp;radius={geo:radius?}&amp;recordSch
ema={sru:recordSchema?}&amp;bbox={geo:box?}&amp;startRecord={startIndex?}&amp;strict=true
" type="application/atom+xml">
```

The EMS builds the product search request path from the template by setting null values for unused fields and by setting the following parameter values:

- geo:box : AOI
- time:start : TOI start date
- time:end : TOI end date

The WPS-T REST HTTP GET operation path may look similar to *http://geo.spacebel.be/opensearch/request?parentIdentifier={collectionId}&startDate={toi_start}&endDate={toi_end}&bbox={aoi}&httpAccept=application/atom%2Bxml*

Finally, the returned document is a list of entries that include, as shown below:

- The link to the products: in *atom:entry/atom:link[@rel=enclosure]*
- The associated WPS endpoint: in *atom:entry/owc:offering/owc:operation[@code=Execute]/@href*

```
<link href=""http://185.48.233.249/Sentinel-2/MSI/L1C/2018/01/30
/S2B_MSIL1C_20180130T034959_N0206_R104_T47PPT_20180130T064159.SAFE" rel="enclosure"
title="Download" type="application/x-binary"/>
<owc:offering code="http://www.opengis.net/spec/owc-atom/1.0/req/wps">
    <owc:operation method="GET" code="Execute" type="application/xml" href="http://wps-
domain/WPS/endpoint"/>
</owc:offering>
```

The execution of the WPS Process pointed to by the workflow step that consumes the EO Image needs to be performed on the MEP ADES associated with the EO Image. This ensures that the Process will be executed close to the EO Image data location.

7.3.6. EMS Deployment on ADES (Step 6)

The EMS needs to deploy the Application(s) before starting the execution. The deployment request is based on the document provided on step 1 by Alice. It also embeds in the Process Description the information provided by the CWL file inside the *additionalParameters* element. Note that this

duplicates the information in the request, but this agreement was adopted as a compromise between the different implementation approaches.

The MultiSensorNDVI deployment request sent to the ADES is shown below.


```
{
    "processOffering": {
        "process": {
            "id": "MultiSensorNDVI",
            "title": "Multi Sensor NDVI",
            "abstract": "NDVI is calculated after the two bands values Near Infrared and
red. It is calculated by this formula : NDVI = (NIR-Red)/(NIR+Red)",
            "keywords": [
                "NDVI"
            ],
            "owsContext": {
                "offering": {
                    "code": "http://www.opengis.net/eoc/applicationContext/cwl",
                    "content": {
                        "href": "http://some.host/CWL/MultiSensorNDVI.cwl"
                    }
                }
            },
            "inputs": [
                {
                    "id": "inputImage",
                    "title": "Input Image",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        }
                    ],
                    "minOccurs": 1,
                    "maxOccurs": 1,
                    "additionalParameters": [
                        {
                            "role": "http://www.opengis.net/eoc/applicationContext/cwl",
                            "parameters": [
                                {
                                    "name": "position",
                                    "value": "1"
                                },
                                {
                                    "name": "prefix",
                                    "value": "image"
                                },
                                {
                                    "name": "separate",
                                    "value": "false"
```

```
                                },
                                {
                                    "name": "itemSeparator",
                                    "value": "="
                                }
                            ]
                        }
                    ],
                    "owsContext": {
                        "offering": {
                            "code": "anyCode",
                            "content": {
                                "href": "anyRef"
                            }
                        }
                    }
                },
[...]    },
    "deploymentProfile": {
        "deploymentProfileName":
"http://www.opengis.net/profiles/eoc/dockerizedApplication",
        "executionUnit": {
            "reference": "docker.registry.host/multisensorNDVI"
        }
    }
}
```

The client receives a similar deployment confirmation message as described earlier.

7.3.7. EMS Execution on ADES (Step 7)

For each of the workflow steps (or for the single Process in the case of an application), an execute request needs to be sent to the appropriate MEP ADES. The WPS-T REST HTTP POST operation path is */processes/{processId}/jobs*.

The request for the ADES "MultiSensorNDVI" process is shown below.

The request in JSON:

```
{
    "inputs": [
        {
            "id": "files",
            "href": "http://some-host/PROBAV_L1C_20160505_232748_3_V101.HDF5"
        },
        {
            "id": "files",
            "href": "https://some-host/PROBAV_L1C_20160505_232949_3_V101.HDF5"
        }
    ],
    "outputs": [
        {
            "id": "output",
            "transmissionMode": "reference"
        }
    ],
    "mode": "async",
    "response": "document"
}
```

Chapter 8. Execution Management Service

This Chapter describes the implementation experience of the Testbed-14 EOC Participants in charge of providing EMS implementations, signaling any relevant peculiarities of their solutions or issues found.

8.1. Spacebel

The Spacebel EMS implementation is based on the 52° North WPS 2.0 Java implementation and on the Spacebel WPS transactional extension which provides an XML interface. The interface made use of the OpenAPI 3 Specification (OAS3). The XML interface is brokered by a proxy based on a Java JAX-RS server stub generated from the WPS-T REST/JSON OAS3 specification with Swagger CodeGen.

The proxy does not support the GetJobs operation. That operation was proposed in the WPS SWG REST/JSON draft, but is not compliant with the WPS 2.0 official specification, so it cannot be supported when using a JSON proxy on the frontend.

The execution of the workflow is performed using the *CWL-runner* tool which is provided as part of the standard CWL toolset.

The CWL workflow steps perform runs of Docker containers while the EMS invokes the appropriate ADES to execute the processes. Instead of tailoring the CWL-runner or developing a custom CWL engine, the Spacebel approach consists of rewriting the workflow steps to match the desired behavior. This approach allows the use of any alternative CWL engine (e.g. Airflow) for running the workflow.

For each workflow step, the appropriate target ADES is computed depending on its inputs, and a command line step is created for starting a WPS commmand-line client which performs the execute request.

The Spacebel EMS has been deployed on the Google Cloud Platform.

Due to lack of time, the quotation and billing operations have not been implemented.

8.2. Geomatys

The EMS server is a proxy publishing a WPS 2.0 JSON API. It is based on a Spring boot controller using pojo generated with swagger codegen from the openAPI document.

The aims of the EMS are to:

- Validate user credentials and ACL
- Perform product search on the OpenSearch catalogue
- Transfer WPS requests to the relevant ADES.
- Execute CWL Workflow, dispatching each step to different ADES and merging the results

8.2.1. Security

Before performing a deploy/execute operation, the EMS server validates the identity of the user with

the supplied credentials from the OAuth 2.0 OpenId server. Upon validation, if the user has the rights to perform the operation, these credentials are transferred to the ADES, when forwarding the request.

8.2.2. Catalogue communication

The communications with the OpenSearch Catalogue endpoint are made in XML. A data-binding has been generated with JAXB to manipulate Java objects during the search.

8.2.3. ADES forwarding

Geomatys' EMS server uses the collection input parameter to determine to which ADES the request must be forwarded. If there is no EOImage in the input, the request is sent to the PFC ADES.

If multiple collections are involved, multiple ADES can be contacted to perform an Execute request.

The server keeps in memory a mapping between an EMS generated JobId and one or more ADES jobIDs. The same mechanism is applied for quotations and bills.

When receiving a request, if multiple ADES are involved, all ADES responses are merged in one (StatusInfo, Results, Quotations).

8.2.4. Workflow Execution

When executing a workflow, the EMS downloads the CWL file, parses it and then executes each step sequentially. It transforms the CWL input into an Execute request and sends it to the selected ADES. When the ADES has completed the execution of the step, the results are stored in order to be used in another step.

8.2.5. EMS Deployment

The EMS Docker image is hosted on the Geomatys public registry. The EMS server is deployed on the Geomatys cloud infrastructure, using Kubernetes.

8.3. CRIM

Test EMS and ADES servers are based on components of the Open Source software framework *Birdhouse* [8]. The Twitcher [https://github.com/Ouranosinc/twitcher] component acts as a Policy Enforcement Point (PEP) and a WPS 2.0 JSON proxy to WPS 1.0 endpoints. It has been enhanced to comply to the EMS API which involves adding the dynamic deployment of processes and CWL workflow capabilities. The Magpie [https://github.com/Ouranosinc/Magpie] component is used as an adapter to manage ACLs of deployed processes and process permissions (WPS requests) for a given user's credentials.

Although fulfilling different roles, *EMS* and *ADES* components use the same underlying CWL engines and require security access verification. Both of them use common operations that are available, such as:

- dynamic deployment of WPS processes
- parsing inputs from WPS requests to relay them to CWL operations

- WPS 1.0 (XML) & WPS 2.0 (JSON) requests and responses
- authorization validation of access token

CRIM's implementation of the EMS differs from the ADES only in the runtime context used by the CWL tool engine. While the ADES uses the default runtime environment which pulls the Docker image and executes it using the CWL parameters, the EMS uses a custom runtime implementation. That implementation instead chooses the right ADES to dispatch the processing based on the input's data source, makes sure that the process is deployed on the corresponding ADES, then executes and monitors it until it can finally retrieve the outputs on successful completion. This simple approach allows the dispatching of a single process only by invoking the CWL engine as well as executing a full workflow in which case every step is executed by dispatching the processing to a remote ADES and chaining the outputs to the next step.

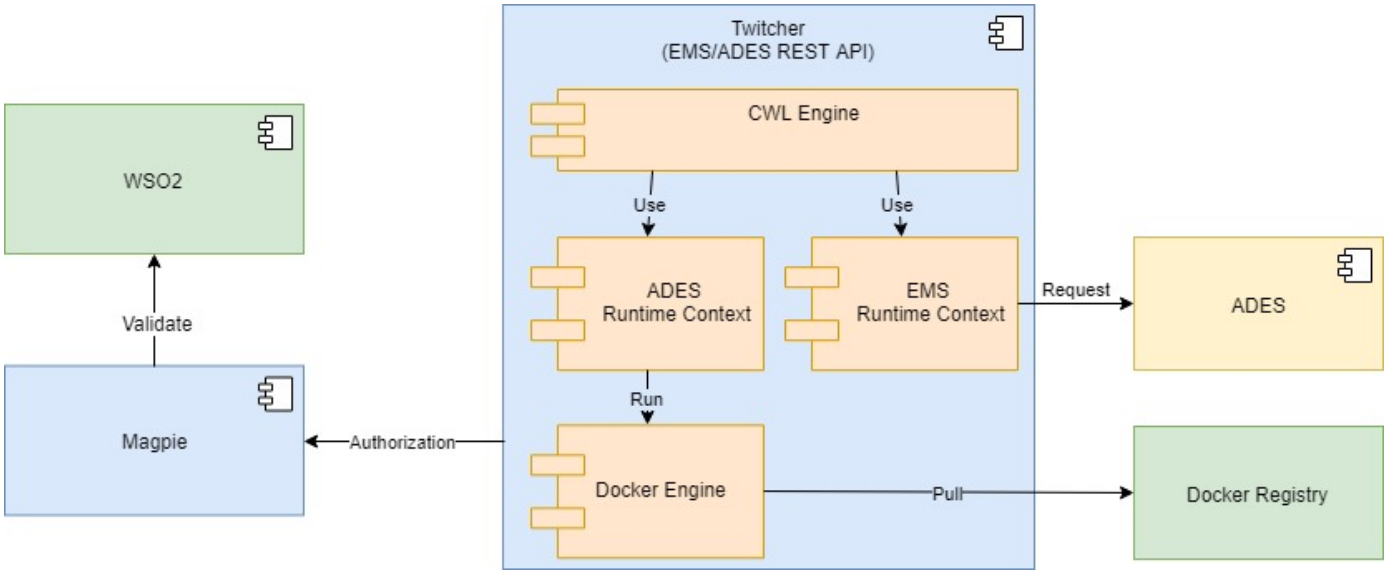

Figure 6. CRIM EMS Components Diagram

The principal requests of the EMS involve workflow operations which are DeployProcessWorkflow and ExecuteWorkflow. The workflow itself is presented as CWL Workflow with Workflow parameters. Additional details on the AP described in this section can be found in [6].

8.3.1. Deployment on Cloud

The EMS and its accompanying applications were deployed on CRIM's cloud infrastructure. Tests were also conducted on the PFC Boreal Cloud. More information on Cloud computing resources described in this subsection can be found in [4].

CRIM Infrastructure

The servers are deployed on three separate VMs, a development ADES and EMS, and a production EMS (*m1.2xlarge, 80GB /dev/vda, 500 GB /dev/vdc*) running ubuntu-18.04 and created on a public OpenStack tenant hosted at CRIM. A monitoring agent, *Zabbix*, is installed on the VM to ensure sufficient operational QoS. The EMS and security components are themselves Docker images pulled from a Docker registry and run using Docker *compose*. Cloud infrastructure setup and automation procedures can be found in [4].

PFC Boreal Cloud

Access to the PFC Boreal Cloud has been made to ensure that an AP could be deployed and run by any ADES hosted on the cloud. Deployment of the EMS has been explored, as well as the security solution of NRCan, DACS. More information on DACS can be found in [4].

8.3.2. WSO2 Security

The *Magpie* component allows the securing of any REST API with user-specific permissions on each endpoint. It is used to restrict the process deployment endpoint to user members of the developer group only. Each deployed process endpoint is authorized for listing and execution only to its owner unless the visibility is updated to public. Provided that the WSO2 token is valid (proper credentials and not expired), the following call will convert the generic OAuth2 *access_token* into a *Magpie auth_tkt* that allows calls to any API route with corresponding user credentials, permissions, and authorizations.

```
{
GET '{host}/magpie/providers/wso2/signin'
Headers: {'Authorization': 'Bearer <access_token>', 'Homepage-Route': '<magpie-relative-
route>'
}
```

While a user can log directly in *Magpie* using its WSO2 credentials the previous step is performed transparently when accessing one of the EMS endpoints with a valid WSO2 token.

The WSO2 provider must be properly setup in the corresponding instance for the previous call to work. Since WSO2 does not allow multiple callback URLs for security reasons (return tokens only to known clients), the WSO2 client ID/secret must be defined beforehand. A new authorized *Magpie* service provider must be defined for every host server, with corresponding client ID/secret pairs deployed on each of their respective instance. The following environment variables are required.

```
WSO2_CLIENT_ID = #####
WSO2_CLIENT_SECRET = #####
WSO2_HOSTNAME = https://eodata-iam.user.eocloud.eu:8080
```

Chapter 9. Application Deployment and Execution Service

This Chapter describes the implementation experience of the Testbed-14 EOC Participants in charge of providing ADES implementations, signaling any relevant peculiarities of their solutions or issues found.

9.1. Spacebel

The ADES implementation is based on the 52° North WPS 2.0 Java implementation and on the Spacebel WPS transactional extension which provides a XML interface. The XML interface is brokered by a proxy based on Java JAX-RS server stub generated from the WPS-T REST/JSON OAS3 specification with Swagger CodeGen.

The proxy does not support the GetJobs operation. That operation was proposed in the WPS SWG REST/JSON draft, but is not compliant with the WPS 2.0 official specification, so it cannot be supported when using a JSON proxy on frontend.

The execution of the Docker container is performed using CWL-runner which interprets the CWL file provided during the application deployment. The input files are downloaded (and renamed to avoid filename collisions) on the ADES file system.

Spacebel ADES has been deployed on IPT Poland cloud and on PFC boreal cloud.

9.2. Geomatys

The Geomatys ADES Server is based on the Examind-server WPS 2.0. It is a Java-based server that uses the Geotoolkit library [http://www.geotoolkit.org/] for data-binding and processing. It handles WPS 1.0/2.0 in XML and JSON. Geomatys added the *Transactional* and *Quotation* part during this Testbed.

The ADES features are to:

- Validate user credentials and ACL
- Deploy/Execute CWL processes
- Download PROBAV product files (authenticated)
- Compute quotations and bills

9.2.1. Security

Before performing a deploy/execute operation, the ADES server validates the identity of the user with the supplied credentials from the OAuth 2.0 OpenId server. The user authentication/authorization is performed within the Spring-security stack of Examind-server.

9.2.2. CWL Process

The deployed process definitions (inputs / outputs / CWL location) are stored in the Examind

[https://www.geomatys.com/en/products/examind-server/] datasource.

During execution, a command-line is created with the specified input parameters to call the native executable *cwl-runner*.

Upon cwl-runner completion, the outputs are extracted and hosted in a WEBDAV service. The execution / monitoring of the process is performed with the Geotoolkit processing framework.

9.2.3. PROBAV product files

The access to PROBAV product files is restricted. When the ADES receives an Execute request with PROBAV URL inputs, it uses the PROBAV Geomatys dedicated account to download the files before transmitting it to *cwl-runner*.

9.2.4. Quotations and bills

The Examind ADES implements the quotation/bill profile with a minimalistic approach. The estimated time and price are calculated based on a ratio of input files' size. On successful quotation request, the quotation response is stored with the Execute request within the ADES database. The quotation has a validity period. An execution request based on the quotation outside of the validity period is refused. Once executed, the quotation is transformed into a bill and sent with the results as a link.

9.2.5. ADES Deployment

The ADES Docker image is hosted on the Geomatys public registry (e.g. **docker pull images.geomatys.com/examind-ades:latest**). The ADES server is deployed on a VITO VM using Docker.

9.3. CubeWerx

9.3.1. ADES endpoints

CubeWerx delivered two ADES's for the EOC thread. A development server located at:

- http://www.pvretano.com/cubewerx/cubeserv/default/wps/3.0.0

and a production server deployed on an Amazon instance provided by Deimos-Imaging and available at:

- https://tb14-deimos.cubewerx.com/cubewerx/cubeserv/default/wps/3.0.0

The development server is an open service without security or access control.

The production service is a protected service and a username/password is required to access the service. The service can be accesses via an OAuth bearer token obtained from the ATOS WSO2 bridge or by accessing the login page here: https://tb14-deimos.cubewerx.com/cubewerx/cwauth?operation=login&callback=https%3A%2F%2Ftb14-deimos.cubewerx.com%2Fcubewerx%2Fcubeserv.
There is an *alice* user with the role of *ApplicationDeveloper* and a *bob* user with the role of *ApplicationUser*.

The security protocols supported by the CubeWerx ADES are described further ahead.

9.3.2. Conformance Classes

All of the existing WPS 2.0 conformance classes are supported by the CubeWerx ADES deployed for TB14:

- Basic WPS
- Synchronous WPS
- Asynchronous WPS
- WPS process model encoding
- Dismiss extension

The supported conformance classes are available at the /conformance path of the REST/JSON API.

9.3.3. Extensions

The CubeWerx ADES implements the WPS Transactional Extension as defined in [1] with modifications to handle exception handling which is simply incorrectly defined in OGC 13-071 using a *FailureReason* parameter in an otherwise valid response. The correct way to handle exceptions is not to generate a valid response and then include a parameter with a failure reason but to either generate a valid response on success or an OGC exception report on failure; which is how the CubeWerx ADES behaves.

The CubeWerx ADES also implements the REST API defined in the EOC thread and described by the OpenAPI document as explained in chapter 7.

The CubeWerx ADES supports message encoding in XML or JSON and can generate responses in JSON, XML and HTML where such responses are defined. For example, the quotation and billing interface defines JSON message encodings and our server only supports JSON for that particular interface.

The HTTP OPTIONS method may be used with an any path to determine the valid methods and representations applicable at that path.

9.3.4. REST API

Content negotiation

Basic content negotiation is as defined in the Hypertext Transfer Protocol — HTTP/1.1 [https://tools.ietf.org/rfc/rfc2616.txt]. The Content-Type header may be used to indicate the representation of any content being sent to the server. The *Accept* header may be used to indicate what output formats are preferred.

Because the CubeWerx ADES supports multiple encodings (JSON, XML and HTML), the server supports an 'f' parameter that allows clients to mint encoding-specific URIs. Valid values for the 'f' parameter are 'json', 'xml' and 'html' (although not all formats are supported for all resources since in some cases, alternate encodings are not defined). As mentioned above, the OPTIONS method may be used to identify which representations are supported.

Link headers

As a general rule, links included in a response payload are also included using link headers. When, however, the number of links exceeds a configurable value, links are only included in the payload to prevent excessively large headers. The current default limit is set to 10 link headers.

Resources and methods

The following table summarizes the resources and methods supported by the CubeWerx ADES.

Table 2. Resource paths implemented by the CubeWerx ADES

Resource path	HTTP Method	Description
All	OPTIONS	Gets the valid methods and representations for the resource
/	GET	Gets the server's landing page
/processes	GET	Get a list of summary descriptions of each visible process
	POST	Add a new process to the ADES; the request content is a Deploy request
/processes/{processId}	GET	Get a detailed description of the specified process
	DELETE	Undeploys the specified process
/processes/{processId}/jobs	GET	Get the list of jobs associated with the specified process
	POST	Executes the process with the specified set of arguments; the body contains an Execute request
/processes/{processId}/jobs/{jobId}	GET	Get the status of the specified job
	DELETE	Cancels the specified job if it is running
/processes/{processId}/jobs/{jobId}/result	GET	Gets the result of an asynchronous job or re-gets the result of a completed job (or 404 if the job results have expired)
/processes/{processId}/quotations	GET	Gets the list of available quotations for the specified process
	POST	Executes the specified quotation
/quotations	GET	Gets the list of all quotations
/quotations/{quotationId}	GET	Get the specified quotation
	POST	Executes the specified quotation (empty POST body)
/bills	GET	Gets the list of available bill identifiers

Resource path	HTTP Method	Description
/bills/{billId}	GET	Gets the specified bill

Given the `entscheidungsproblem`, resource and time limitations of the Testbed, it was not possible to devise a computation to generate accurate quotations to execute any given *Execute* request. Since the goal of the Testbed was to define an API to allow quotations to be requested and process execute requests based on an existing quotation, the CubeWerx ADES defines a quotation heuristic based on the byte size of the inputs. As the size of the inputs increases, so does the cost of executing a particular quotation. Amazon technical staff were contacted about their pricing tool but it has not been possible to implement more sophisticated quotations based on this before the end of the Testbed.

9.3.5. Application program execution

Execution units

Unlike the other MEP ADES's in the EOC thread, which rely on execution units being expressed using CWL, the CubeWerx ADES relies solely on Docker execution units. The primary reason for this is scalability. CWL execution engines do not try and take advantage of any execution elasticity that a hosting environment, such as AWS, might offer, thus limiting scalability.

Application program description

In order to execute an application program, the CubeWerx ADES requires:

- a reference to the Docker image containing the application program
- an application package (see [6]) containing sufficient metadata to map WPS process inputs/output to application program arguments

The reference to the application unit is specified in the execution unit section of the application package. The following fragment illustrates an execution unit that references a Docker image:

```
"executionUnit": [
   {
      "href": "images.geomatys.com/ndvims:latest"
   }
],
```

The mapping from WPS process inputs/output to application program arguments is effected using *additionalParameters* in the input/output descriptions. The following fragment illustrates their use:

```
"additionalParameters": [
   .
   .
   .
   {
      "role": "http://www.opengis.net/eoc/inputBinding",
      "parameters": [
         {
            "name": "position",
            "values": ["1"]
         },
         {
            "name": "prefix",
            "values": ["files"]
         },
         {
            "name": "separate",
            "values": ["false"]
         },
         {
            "name": "itemSeparator",
            "values": ["="]
         }
      ]
   },
   .
   .
   .
]
```

In the absence of such descriptions, the CubeWerx ADES applies default logic to try and create a mapping using the WPS input description alone.

Docker execution

The CubeWerx ADES offers two configurable execution modes for running Docker containers. The first mode uses a local installed Docker instance to directly execute the Docker container and is primarily intended for testing. This is how the CubeWerx development environment is configured.

The second mode relies of Amazon's Fargate compute engine. AWS Fargate is a compute engine for Amazon ECS that allows you to run containers without having to manage servers or clusters. With AWS Fargate, you no longer have to provision, configure, and scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

To use Fargate, two components must be created. First, a cluster must be defined on which the containers will be run. This cluster definition includes elasticity parameters that define how the cluster grows or contracts with load. Second, a task file needs to be created that indicates which container to run, which volumes to map into the container, etc. The cluster definition is created when the ADES is

deployed. A task file is created each time a process is deployed.

9.3.6. Access control model

The CubeWerx ADES implements a full access control model for WPS processes. In a secured and access-controlled environment, a process is owned by the user who deploys a process. The owner can perform all operations on the process including undeploying the process and making the process "visible" to specific other users or roles. Users or roles granted visibility of a process can perform all operations on the process except undeploy or changing the visibility of the process.

Making a process visible or invisible to all users is accessible via the */visibility* access path. Finer-grained control of access must be performed manually by an administrator at this time.

9.3.7. Security considerations

The CubeWerx ADES advertises security requirements objects and security scheme objects in its OpenAPI document. The server's OpenAPI document is accessible via the */api* path.

Security Requirements Object

The security requirements object [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#securityRequirementObject] is a declaration of which security mechanisms can be used to authorize resource access across the API.

The list of values includes alternative security requirement objects that can be used. Only one of the security requirement objects needs to be satisfied to authorize resource access. The field name of the security requirements object is *security*.

The following JSON fragment, from the CubeWerx production ADES, illustrates the top-level security requirements objects:

```
"servers": [
  {
    "url": "https://tb14-deimos.cubewerx.com/cubewerx/cubeserv/default/wps/3.0.0"
  }
],
"security": [
  {
    "httpBearer": [
    ]
  },
  {
    "oauth2": [
      "profile",
      "email",
      "openid"
    ]
  },
  {
    "openIdConnect": [
    ]
  },
  {
    "cwApiKeyQuery": [
    ]
  },
  {
    "cwApiKeyHeader": [
    ]
  },
  {
    "cwAuth": [
    ]
  }
]
```


Individual operations can override this definition using a locally scoped security requirements object [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#operation-object].

Security scheme object

Each security requirement (i.e. "oauth2", "cwApiKeyQuery", etc.) specified at the API or operation level must reference a security scheme object [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#securitySchemeObject] defined in the components section [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md#componentsObject] of a server's OpenAPI document.

Supported schemes relevant to the TB14 EOC thread are *oauth2* and *openIdConnect* using OAuth2's common flows (implicit, password, application and access code) as defined in RFC 6749 [https://tools.ietf.org/html/rfc6749], and OpenID Connect Discovery [https://tools.ietf.org/html/draft-ietf-oauth-discovery-06].

The following JSON fragment illustrates a security schemes object from CubeWerx's ADES:

```
   "securitySchemes": {
      "httpBearer": {
        "type": "http",
        "scheme": "Bearer"
      },
      "oauth2": {
        "type": "oauth2",
        "flows": {
          "implicit": {
            "authorizationUrl": "https://tb14-
deimos.cubewerx.com/cubewerx/cubeserv/authorize",
            "scopes": {
              "profile": "requests access to the end-user's profile",
              "email": "requests access to the end-user's e-mail address",
              "openid": "OpenID Connect scope"
            }
          },
          "password": {
            "tokenUrl": "https://tb14-deimos.cubewerx.com/cubewerx/cubeserv/token",
            "scopes": {
              "profile": "requests access to the end-user's profile",
              "email": "requests access to the end-user's e-mail address",
              "openid": "OpenID Connect scope"
            }
          },
          "authorizationCode": {
            "authorizationUrl": "https://tb14-
deimos.cubewerx.com/cubewerx/cubeserv/authorize",
            "tokenUrl": "https://tb14-deimos.cubewerx.com/cubewerx/cubeserv/token",
            "scopes": {
              "profile": "requests access to the end-user's profile",
              "email": "requests access to the end-user's e-mail address",
              "openid": "OpenID Connect scope"
            }
          }
        }
      },
      "openIdConnect": {
        "type": "openIdConnect",
        "openIdConnectUrl": "https://tb14-deimos.cubewerx.com/cubewerx/cubeserv/.well-
known/openid-configuration"
      },
      "cwApiKeyQuery": {
        "type": "apiKey",
        "name": "apiKey",
        "in": "query"
      },
      "cwApiKeyHeader": {
```

```
        "type": "apiKey",
        "name": "CubeWerx-API-Key",
        "in": "header"
      },
      "cwAuth": {
        "type": "http",
        "scheme": "CwAuth",
        "x-auth-server": "https://tb14-deimos.cubewerx.com/cubewerx/cwauth",
        "description": "CubeWerx-specific scheme."
      }
    }
```

Issues

An issue that arises with the current REST pattern established by WFS 3.0, which is used for the ADES as well, is the "chicken and egg" security issue described here: https://github.com/opengeospatial/WFS_FES/issues/135. Without *a priori* knowledge, a client needs to access an unsecured OpenAPI document at */api* in order to get the necessary information so that the client can establish a secure connection with the server.

A solution to this issue was not discussed in the EOC thread but the above referenced issue mentions a couple of workarounds including the availability of an unsecured */api* document with just enough information in it to allow a client to establish a secure connection.

IANA considerations

Standard media types and link relations are supported.

CORS considerations

CubeWerx has implemented the Cross-Origin Resource Sharing (CORS) mechanism as detailed in "http://www.w3.org/TR/cors/". Specifically, CubeWerx supports the "Access-Control-Request-Method" and "Origin" HTTP request headers, and generates the "Access-Control-Allow-Origin", "Access-Control-Allow-Credentials", "Access-Control-Max-Age", "Access-Control-Allow-Methods", "Access-Control-Allow-Header" and "Access-Control-Expose-Headers" HTTP response headers in accordance with the CORS mechanism. This includes full support for pre-flight requests. The list of *Origins* from which credentialed access is allowed is fully configurable, and the entire mechanism can be disabled if necessary (e.g. to support older environments that are not compatible with CORS).

Chapter 10. Issues of Interest

10.1. OpenSearch results pagination

The OpenSearch specification provide a pagination mechanism for listing search results. For the purpose of the Testbed, some EMS implementations do not support the pagination and retrieve results from the first result page which is limited to 50 entries on the Spacebel Gateway. In operational contexts, the pagination would have to be addressed.

10.1.1. Handling complexity of OpenSearch

Other limitations have been identified as well:

- How to map search results to multiple inputs, for which the unique TOI / AOI have been introduced.
- What are the query parameters that should be supported, some have been identified: AOI, TOI and Collection.

To remove complexity in this Testbed, the OpenSearch query has been embedded in the EMS, but at the expense of the versatility. It has been proposed that the search ability could be packaged as a process which could be embedded in any workflow at design time. The developer can then decide explicitly on which inputs to chain an OpenSearch query, choose the query parameters, limit the number of results, etc. Also, coupling the OpenSearch handling into the EMS makes it hard to maintain, while having a lightweight application dedicated to performing searches is much more agile and can be versioned.

10.2. Quotation API

Even though [3] describes these aspects in more detail, the following is a brief rationale for how the Quotation part of the WPS-T REST API was designed:

- First of all, the complexity of quoting an execution in such a distributed and heterogeneous environment was highlighted. A user using the TEP Client will ask for a quotation which will be routed to the EMS. Particularly in the case of a workflow execution, the EMS in turn will have to ask ADES instances in MEPs for sub-quotations and put them all together to generate a total quote. There are also associated issues in terms of reliability, expiration and legality (a quote will likely be built from sub-quotes from different businesses). Such issues, to limit complexity, were not directly tackled in the Testbed but are very important to address and solve in operational contexts given specifically the financial implications for both users and platform providers.
- To limit the complexity mentioned in the previous point, a simple use case was assumed, wherein quotes have short expiration times and the datasets to be processed are fairly static in time (i.e. there is not often new data in a given AOI). For some applications and use cases, the constant availability of new data over an AOI brings additional complexity and volatility to quotations.
- Additionally, even though in a real environment quotes will be very dynamic, changing in time due to fluctuations in the price of several cost contributors (data, services, manpower, utilities, IT), in the Testbed it was assumed that quotes are static. Also, fictitious numbers, with no relation to actual costs or real currencies, were considered.

- Finally, it was discussed whether there should be a separate quotation operation or if the quotation could be simply an option of the execution operation. The first option was chosen to provide a cleaner interface.

For further information about the parts of the WPS-T REST specification introduced in Chapter 7 that deal with Quotation and Billing - as well as all related issues of interest - the reader is referred to [3].

10.3. Error handling

As mentioned in Chapter 6, the Sponsor in the CFP included a requirement to “make use of standardized error handling and user messaging”. Even though the Testbed Participants did not dedicate significant or special efforts to this requirement in particular, standardized error handling and user messaging has been elegantly included in the solution by relying on standard HTTP error codes (4xx) in the communication between functional blocks. This can be seen extensively in the OpenAPI 3.0 file with the WPS-T REST specification linked to in Chapter 7. A large part of the file’s content is indeed dedicated to the specification of error responses, codes and messages.

10.4. Authentication between EMS and ADES

It has been decided that applications should be deployed on the ADES at runtime in order to determine the best suitable ADES based on the data. While this is good behavior it causes an issue when the EMS is required to deploy an application on Bob’s behalf (which should be forbidden because he is not a developer) when this user wants to execute an application for which permission has been granted to him on the EMS. It has been discussed to deploy every application on every ADES when it is first deployed on the EMS, but that defeats the purpose of on-demand deployment. It has also been discussed to add a trusted channel between EMS and ADES such that a deployment can be always authorized if initiated by a known EMS.

10.5. CRIM API recommendations

- Although including a location in the job execution response’s header is valid according to HTTP 201, it is not extremely obvious that the server created the job because an empty body is returned. Furthermore, other requests of the API return at the very least a code/description in case of error, or details corresponding to the request (e.g. Process Description) when successful. It therefore feels off that this POST returns nothing in the body.
- Process deployment should return a "201 Created" as a new process is created, rather than a 200 OK.
- *ProcessDescription.Inputs.min/maxOccurs* parameters should allow both string - for the "unbounded" value - and integers for any other values.

10.6. WPS Transactional specification

Spacebel has submitted a set of corrections to the WPS-T draft implemented in the context of the Testbed-14 NextGen thread. It has been agreed to base the WPS-T JSON API on this specification.

There are still some minor elements that have been noticed as ambiguous, which will be reported to

the WPS 2.0 SWG.

10.7. WPS and WPS-T REST/JSON API

Generally speaking, the WPS REST/JSON API draft submitted by the SWG sometimes explores contradictory guidance: from a side, it distances from the XML schemas, losing a strict analogy with the specification. On the other side, the liberties taken for applying the REST principles fail to meet the target.

The following comments should be considered to improve the future versions of the API:

- The REST API should not restrict or extend the specification operations, in particular, the GetJobs operation. Also, such operation cannot be implemented by a basic proxy put in front of an XML WPS implementation.
- The REST API should not lose any specification element if there are no reasons to remove it. A lot of XML elements have not been carried to the JSON API.
- The OpenAPI 3.0 specification is a flexible approach for defining APIs, but the flexibility may lead to the complexity in being supported by some languages. In the WPS JSON API, some XML wrapping elements (e.g. *wps:LiteralData* and *wps:ComplexData*, *wps:LiteralDataDomain*) have been removed for wrong reasons, as code generators (e.g. Swagger CodeGen) fail to generate the models (e.g. in Java) because the model misses a needed discriminator and the parsing is ambiguous. Concretely, it is recommended to use the same wrappers as XML in particular when the API defines *oneOf*, *anyOf*, and *allOf*, and also to avoid tiered constructs of these elements.
- The WPS SWG JSON API has been extended to transactional and a set of issues have been fixed and will be reported to the SWG.

10.8. File format, name and extension

During the project, it has been noticed that some applications require that the input files meet some specific criteria:

- A particular file name or file extension should be used.
- A specific format which is more specific that a usual mime type.

10.8.1. Filename and extension

Sometimes applications expect a particular file name (and extension) and the WPS should not break the original name. Multiple means are available to determine the filename, and Spacebel has applied the following procedure if the input is provided by reference (URL):

1. Use the HTTP (non-standard) header "Content-disposition" (mean for the origin server to suggest a default filename) if available.
2. Use the URL filename, and URL extension if present.
3. Use the URL filename, and the execution request mimeType attribute if provided.
4. Use the URL filename, and the default mimeType of the process description in other cases.

The WPS needs to state, in the file, the discovered file name for sharing it with the containerized backend application. Due attention must be paid to avoid filename collision for multiple inputs (e.g. by storing files in distinct directories).

Additionally, the outputs of process executions are potentially consumed as inputs by succeeding workflow steps. Therefore, preserving the application output filename is supported by the corollary assumptions:

1. The steps of the workflow request outputs of WPS process executions by reference.
2. In the output resource HTTP response, the WPS returns a Content-Disposition HTTP header with the original output filename.

10.8.2. Specific format

The process input and output descriptions intend to indicate to the client the formats accepted by the process. The information about the format is limited to the mimeType, and additional restrictions can be provided in metadata fields.

During Testbed-14, the need for providing additional restrictions was highlighted, in particular concerning EO Image inputs, which may be only accepted if the format is the original Sentinel-2 package rather than a variation of this format, or even a format of another satellite (e.g. Landsat-8).

As future work, it is recommended to provide for such inputs a reference to the format definition in the metadata fields.

10.9. Multiple Outputs

The output of an application or script may be a set of files. The WPS specification states the following: “Processes, as well as their inputs and outputs, are elements with identity. A process input may have arbitrarily defined value cardinality, i.e. for a given input, multiple datasets may be submitted for execution. A process output always has a value cardinality of one”. Therefore, the question is how to return an array of files in a WPS response.

The case typically happens when an application receives multiple files as input and applies the same treatment to each of those files. Various approaches have been considered to handle this case:

- Generating multiple files in a single WPS process execution may be considered as a bad practice. Indeed, the WPS client (or the EMS workflow engine) should be seen as the orchestrator and should only manage the scheduling of atomic operations, which means individual executions for each input file.
- Processes returning multiple files may also define a return format (e.g. zip) that allows to return a single file as output. For large files, this may lead to resource and time-consuming tasks when it comes to archiving or deleting the files.
- Repeating the outputs with an identical identifier in the execution response would not be compliant to the WPS specification and could cause response parsing issues on some implementations.
- The use of a Metalink file is recommended as this allows flexibility. However, this requires the parsing of such a file by the client.

- Multipart/mixed content in an HTTP server response is only supported by few browsers but may be implemented by the WPS client. The task of providing downloadable URLs would be then delegated to the client.

10.10. Application execution recommendations

The Testbed-14 experience has allowed identification of problems affecting application execution, some of them specifically related to CWL. It has also allowed development of workarounds for them. Herewith follows a list:

- Cwl-runner mounts volumes with read-only access by default. The option *--no-read-only* can be used to prevent this
- Cwl-runner shall not be run as *root*
- The Stacker application used in the Tested consumes about 8GB of memory. The process runs successfully only after reserving 16 GB of memory.
- For the Stacker application in particular, the input image filename should match the archive subfolder name (so, when downloading the input files, make sure to name them correctly)
- Related to the previous point, this assumption/requirement cannot be met in the context of the ADES. The ADES receives an arbitrary URL, stages the file locally with an arbitrary filename and then shares the file location with the Dockerized application. In general, applications should not rely on assumptions related to particular filenames.
- Finally, in cases like the Stacker application, such limitations/assumptions may come from third-party tools like SNAP (used to read Sentinel and other EO mission data). In those cases, wrapper scripts can be used (for example) to peek inside an archive, obtain (for example) subfolder names and create (for example) symbolic links pointing from the assumed name to the real name (i.e. "tricking" the tool). The general idea is that additional logic might be required to overcome problems created by assumptions.

10.11. processExecuteUrl

One of the participants suggested that the *processExecuteUrl* attribute used in the Process Description should be changed to a *links* section where hypermedia controls with appropriate *rel* links could be used. At least one link for executing the process must be in that section, but other links could be allowed too.

Due to lack of time this has not been tried in Testbed-14.

10.12. Using EO Image HTTP URLs as ADES inputs

As a clarification, even though participants agreed to use HTTP URLs as inputs for the ADES executions, this is not to be understood as implying a download from a remote server, as this would completely defeat the purpose of running the processing close to the data. Instead, the idea is that data to be processed in a given platform can be hosted on that same platform for example in Object Storage buckets (accessed using HTTP) or in a WCS server (accessed using the WCS HTTP API). Such access is still to be considered "local".

Chapter 11. TEP Client

This Chapter provides a brief introduction to the TEP Client through a few screenshots that illustrate some of the main functionality.

The first Figure shows the screen where users can login. The login is implemented using the WSO2 endpoint provided and configured by ATOS. See [3] for more information.

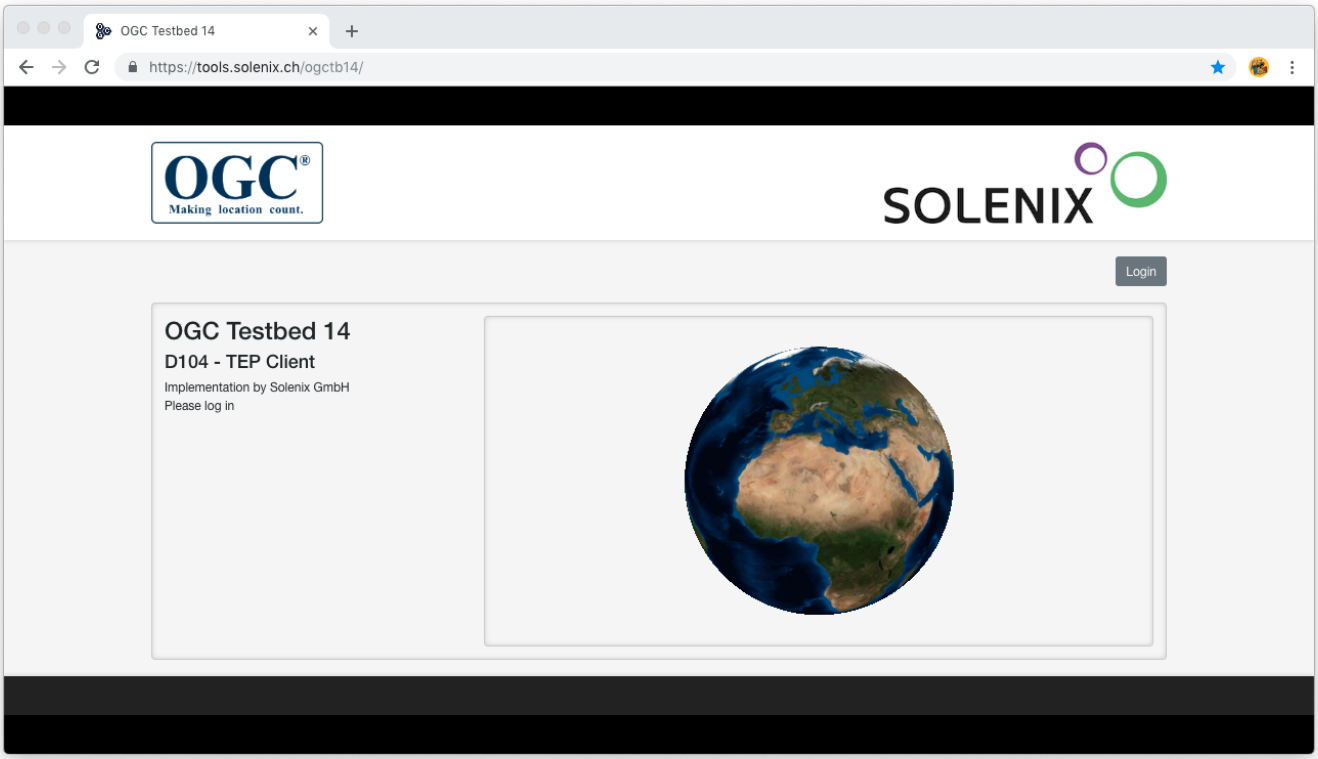

Figure 7. TEP Client initial screen

The second Figure shows the same screen after login. Notice that the user is *alice@ogctestbed14.com* . This is one of the users configured by ATOS.

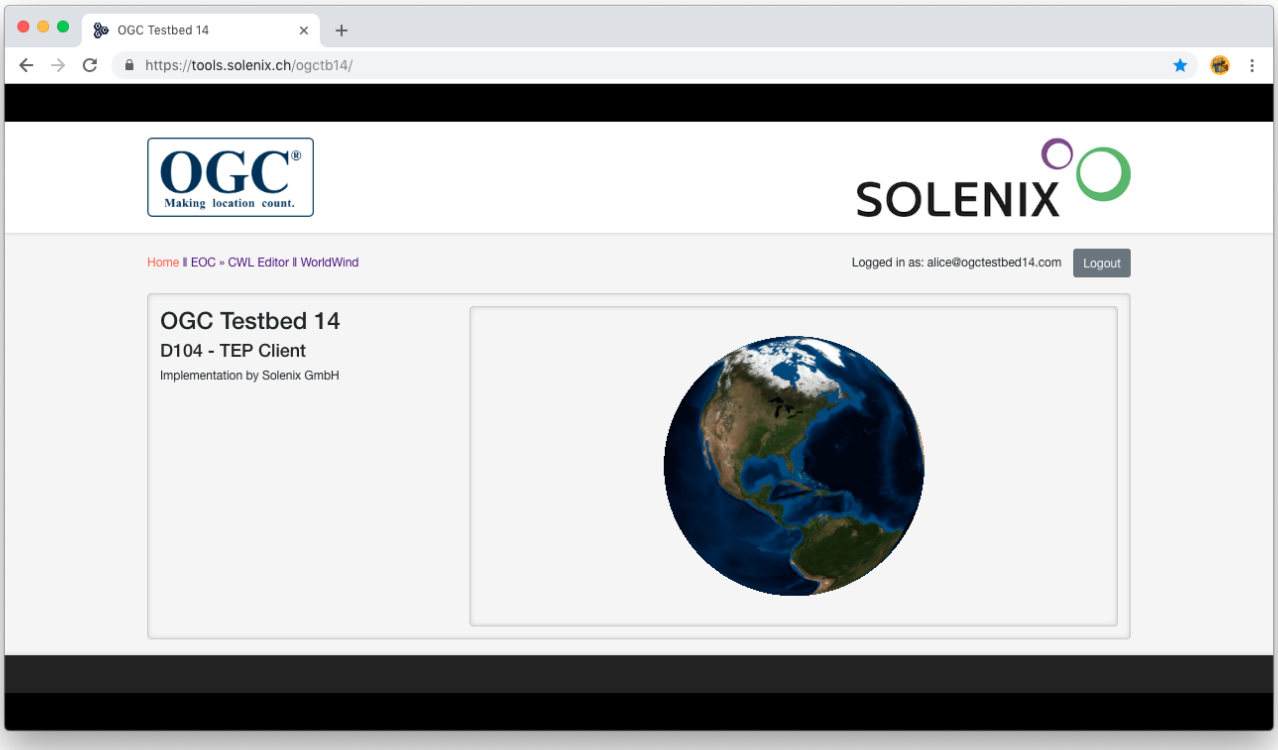

Figure 8. TEP Client initial screen after login

In the next screen, users can select one of the available EMS endpoints (e.g. Geomatys) and browse the Processes exposed by these (e.g. NDVIStacker, NDVIMultiSensorStacker).

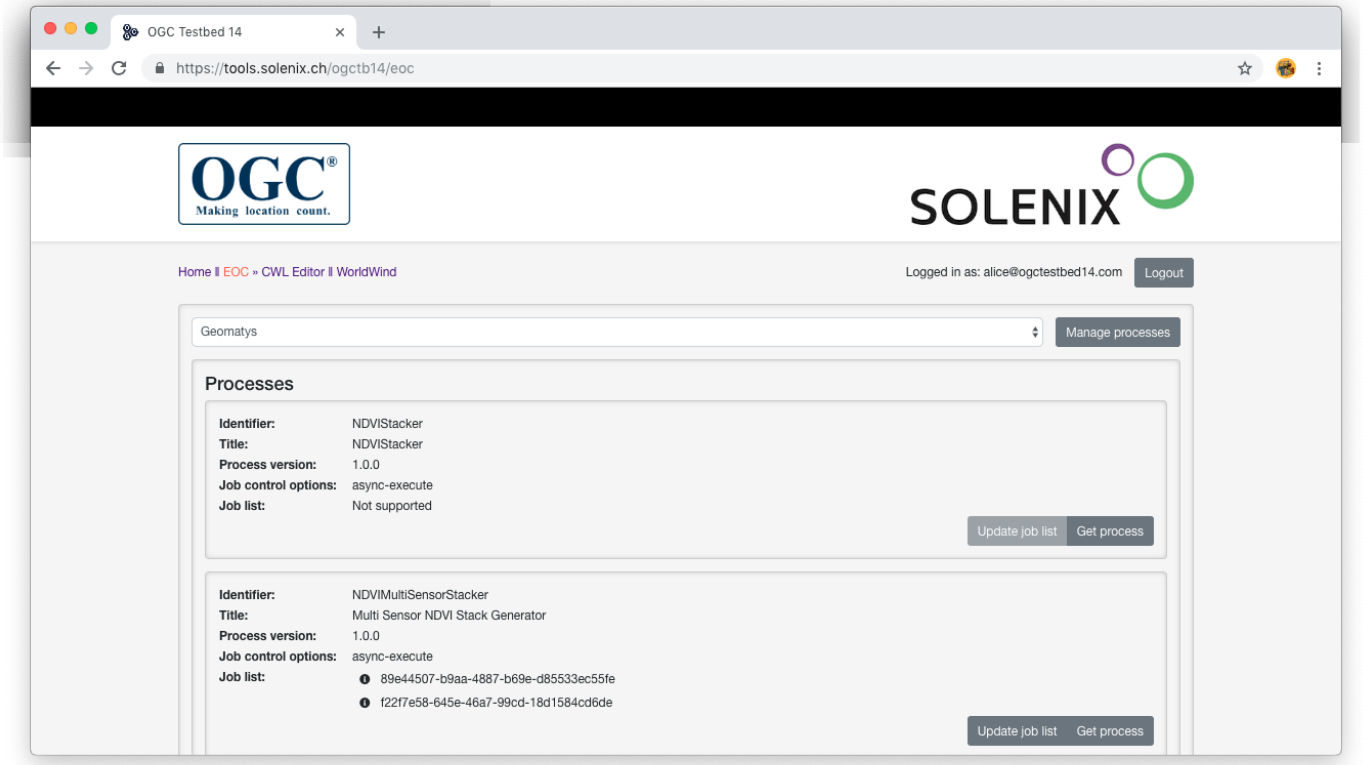

Figure 9. TEP Client EMS process browser

By clicking on *Manage Processes*, it is possible to access the application/process deployment and undeployment functionality ("+ and "-" buttons in the screen below, respectively). Clicking the "+"

button presents a pop-up that can be used to upload a JSON file containing a WPS-T DeployProcess message encoded as described in [6] (*NDVIStackerTest.json* in the screen, for a process/application named *NDVIStackerTest*).

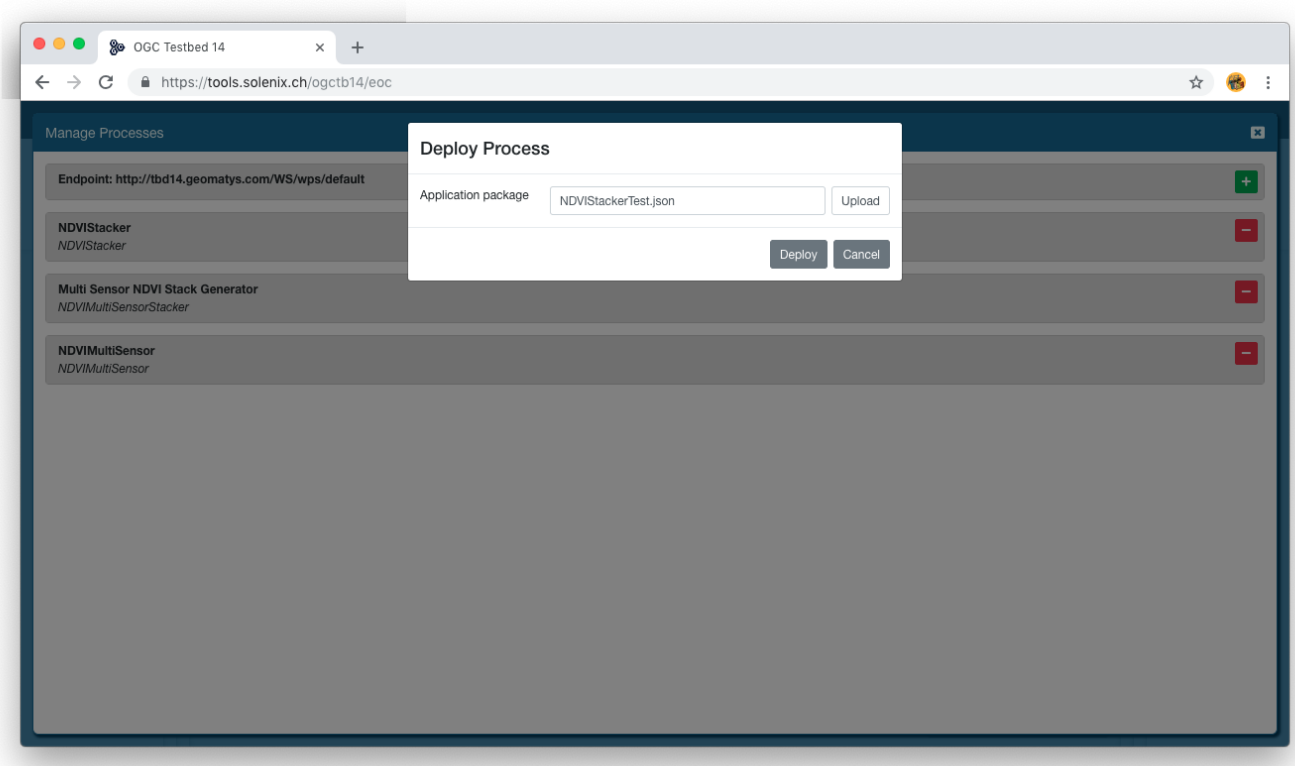

Figure 10. TEP Client Deploy Process screen

The same screen can be used to undeploy the *NDVIStackerTest* application/process using the "-" button.

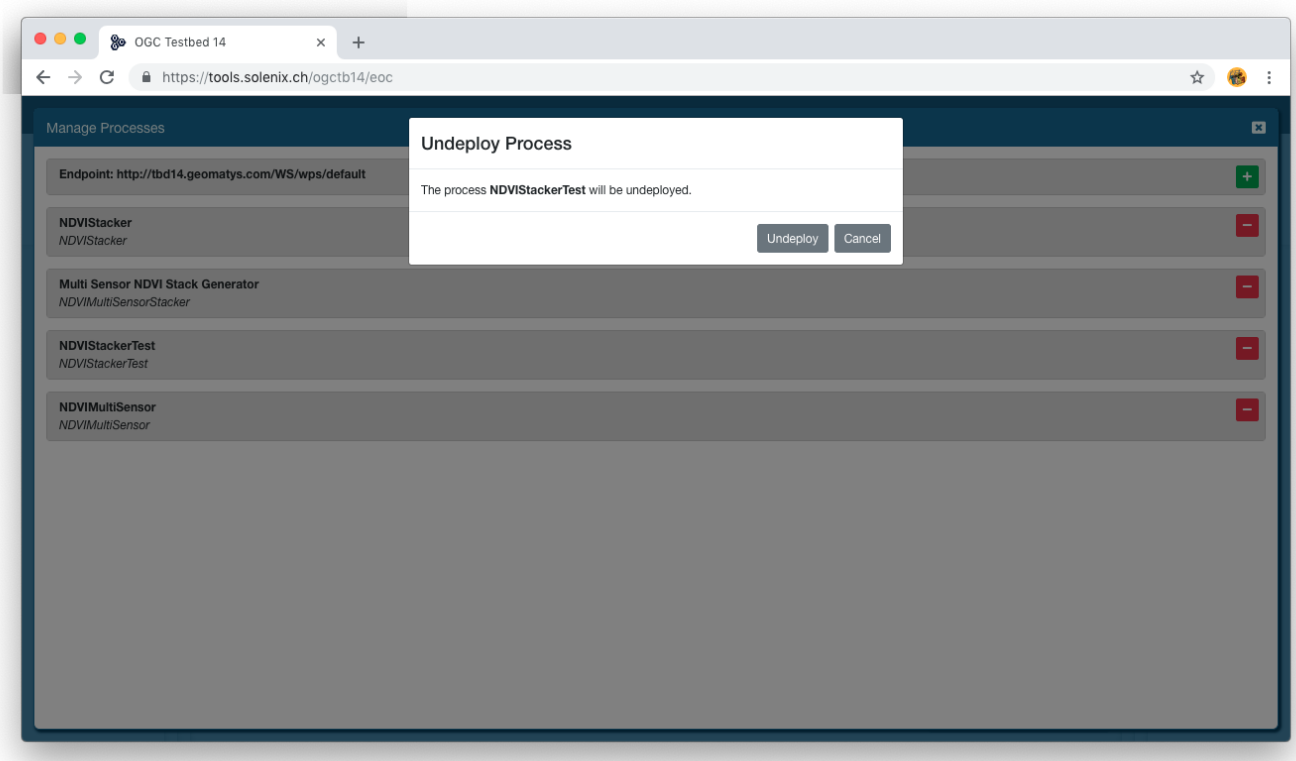

Figure 11. TEP Client Undeploy Process screen

Once a process/application is selected, an application-specific GUI will be dynamically built and presented to the user to collect inputs. The screenshot below shows the GUI for the *NDVIMultiSensor* application, collecting the TOI start time using a calendar widget.

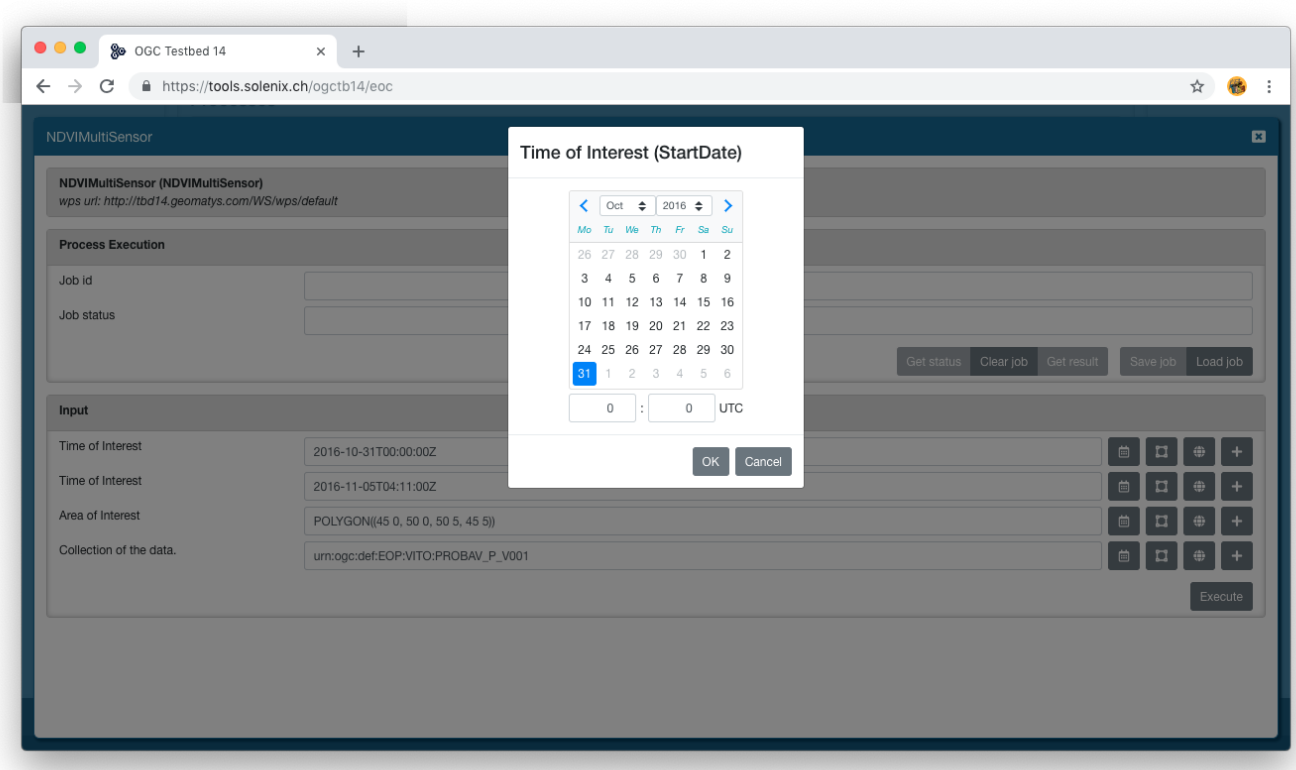

Figure 12. TEP Client Process execution screen

The screen below shows the WebWorldWind 3D globe screen with the selected AOI.

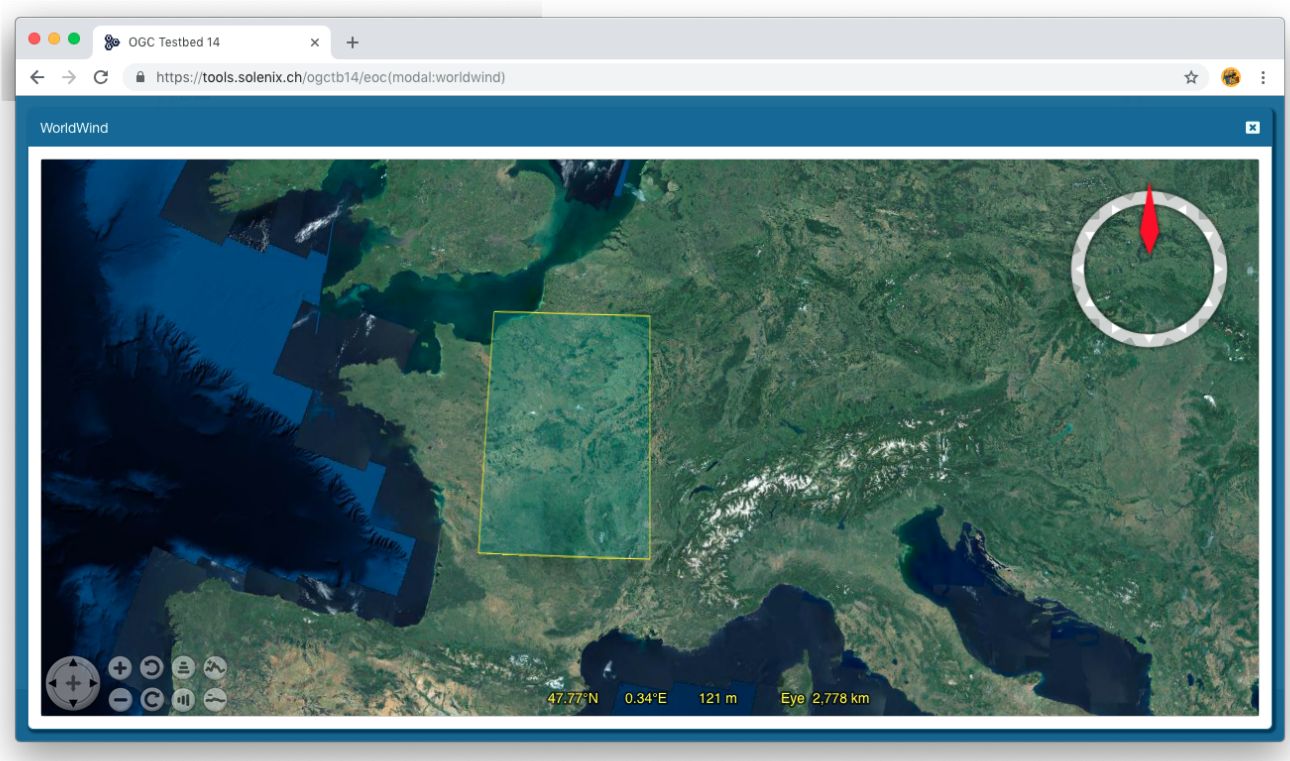

Figure 13. TEP Client 3D globe screen showing AOI

Once the process execution is triggered using the "Execute" button, the top part of the execution screen will provide and update the job status information. In the next screen, it can be seen that the job is running.

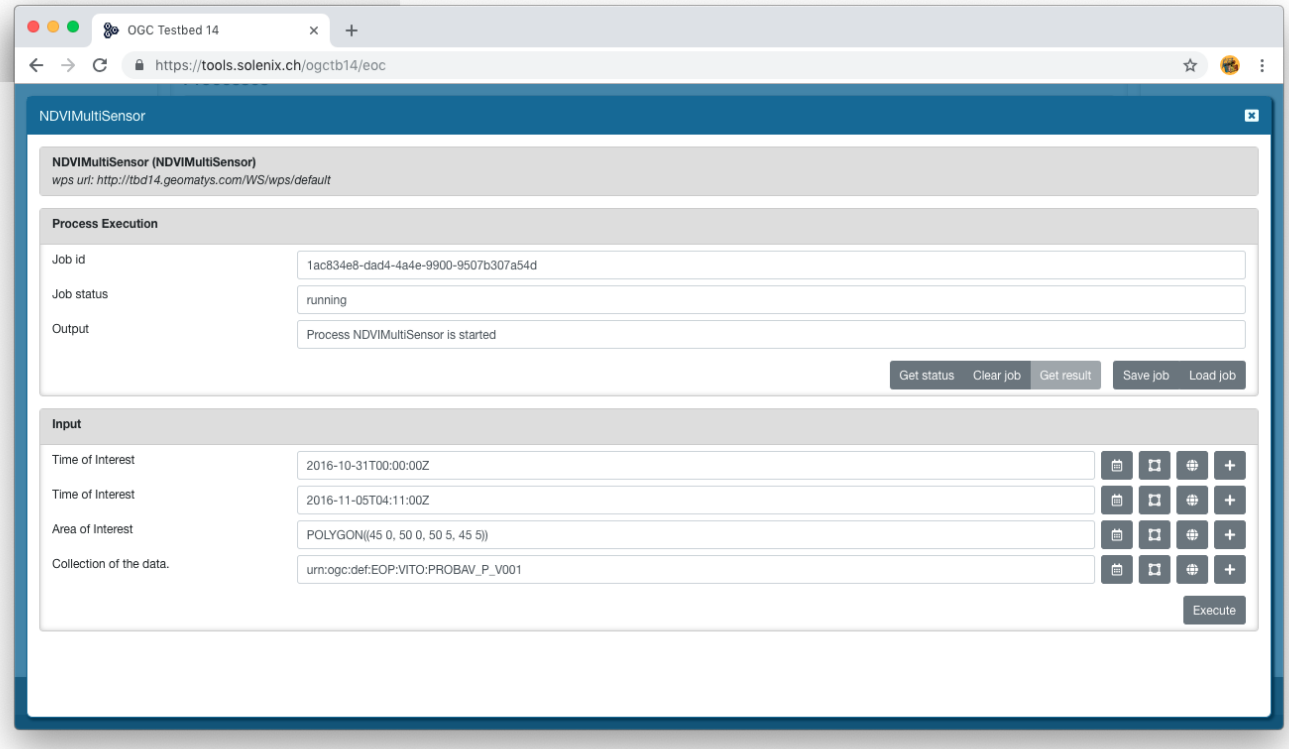

Figure 14. TEP Client job status screen

After the time required to run the application/process, the job status will change from *running* to *succeeded* and the bottom part of the screen will show the application outputs (in this case a NDVI image downloadable using the provided URL).

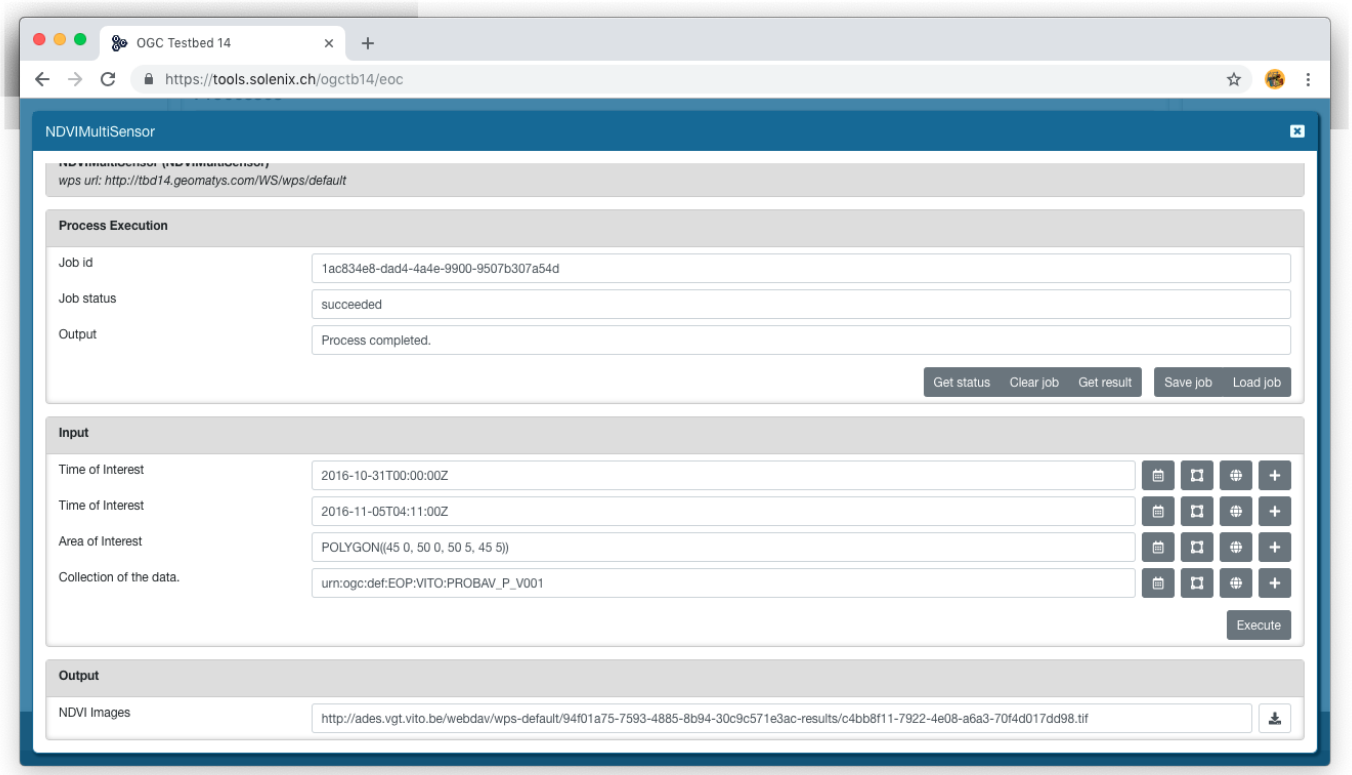

Figure 15. TEP Client job status screen after successful execution

Finally, the screenshot below shows the CWL editor that is integrated in the TEP Client and can be used

to compose CWL workflows.

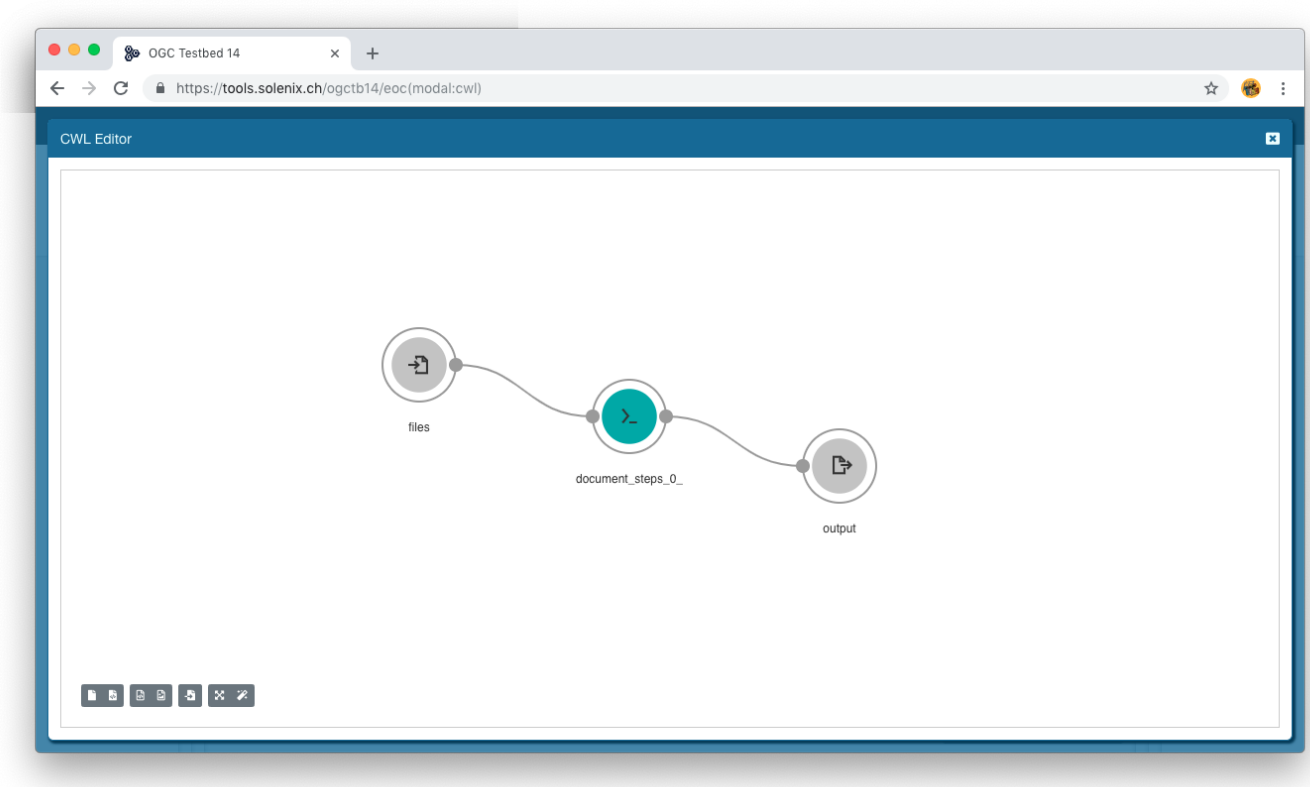

Figure 16. CWL Editor

Appendix A: JSON file for a Deploy Request

JSON file for a deploy request for CRIM's workflow

```
{
    "processDescription": {
        "process": {
            "id": "Workflow",
            "title": "Workflow",
            "abstract": "",
            "keywords": [],
            "inputs": [
                {
                    "id": "input_files",
                    "title": "Input Image",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        }
                    ],
                    "minOccurs": 1,
                    "maxOccurs": "unbounded",
                    "additionalParameters": [
                        {
                            "role":
"http://www.opengis.net/eoc/applicationContext/inputMetadata",
                            "parameters": [
                                {
                                    "name": "EOImage",
                                    "values": [
                                        "true"
                                    ]
                                }
                            ]
                        }
                    ]
                }
            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "Workflow SFS Image",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        }
                    ]
                }
```

```
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "reference"
        ]
    },
    "executionUnit": [
        {
            "href": "https://raw.githubusercontent.com/crim-
ca/testbed14/master/application-packages/Workflow/Workflow.cwl"
        }
    ],
    "deploymentProfileName": "http://www.opengis.net/profiles/eoc/workflow"
}
```

Appendix B: CWL file for a complete workflow

CWL file describing a complete workflow for CRIM's applications

```
{
    "cwlVersion": "v1.0",
    "class": "Workflow",
    "requirements": [
        {
            "class": "StepInputExpressionRequirement"
        },
        {
            "class": "MultipleInputFeatureRequirement"
        }
    ],
    "inputs": {
        "image-s2": "File[]",
        "image-probav" : "File[]",
        "image-deimos" : "File[]"
    },
    "outputs": {
        "classout": {
            "type": "File",
            "outputSource": "sfs/output"
        }
    },
    "steps": {
        "stacker_s2": {
            "run": "Stacker.cwl",
            "in": {
                "files": "image-s2"
            },
            "out": ["output"]
        },
        "stacker_probav": {
            "run": "Stacker.cwl",
            "in": {
                "files": "image-probav"
            },
            "out": ["output"]
        },
        "stacker_deimos": {
           "run": "Stacker.cwl",
           "in": {
                "files": "image-deimos"
            },
            "out": ["output"]
        },
        "stack_creation": {
            "run": "Stacker.cwl",
```

```
            "in": {
                "files": [
                    "stacker_s2/output",
                    "stacker_probav/output",
                    "stacker_deimos/output"
                ]
            },
            "out": ["output"]
        },
        "sfs": {
            "run": "SFS.cwl",
            "in": {
                "source_product": "stack_creation/output"
            },
            "out": ["output"]
        }
    }
}
```

Appendix C: JSON file for an ADES Execute Request

JSON file for an ADES execute request for CRIM's workflow


```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "image-s2",
      "href": "https://ogc-
ems.crim.ca/twitcher/wpsoutputs/S1A_IW_GRDH_1SDV_20170413T224356_20170413T224421_016132_0
1AA52_90ED.SAFE.zip"
    },
    {
      "id": "image-probav",
      "href": "https://ogc-
ems.crim.ca/twitcher/wpsoutputs/S2A_MSIL1C_20180610T154901_N0206_R054_T18TXR_20180610T193
029.SAFE.zip"
    },
    {
      "id": "image-deimos",
      "href": "https://ogc-
ems.crim.ca/twitcher/wpsoutputs/RS2_OK18072_PK188251_DK178156_F22_20090501_110525_HH_HV_S
LC.zip"
    }
  ],
  "outputs": [
    {
      "id": "output",
      "transmissionMode": "reference"
    }
  ]
}
```

Appendix D: JSON file for parameters of a workflow

JSON file describing workflow parameters for CRIM's applications

```
{
    "image-s2": [
        {
            "location":
"./S2A_MSIL1C_20180610T154901_N0206_R054_T18TXR_20180610T193029.SAFE.zip",
            "class": "File"
        },
        {
            "location":
"./S1A_IW_GRDH_1SDV_20170413T224356_20170413T224421_016132_01AA52_90ED.SAFE.zip",
            "class": "File"
        }
    ],
    "image-probav": [
        {
            "location":
"./S2A_MSIL1C_20180610T154901_N0206_R054_T18TXR_20180610T193029.SAFE.zip",
            "class": "File"
        },
        {
            "location":
"./RS2_OK18072_PK188251_DK178156_F22_20090501_110525_HH_HV_SLC.zip",
            "class": "File"
        }
    ],
    "image-deimos": [
        {
            "location":
"./S2A_MSIL1C_20180610T154901_N0206_R054_T18TXR_20180610T193029.SAFE.zip",
            "class": "File"
        },
        {
            "location":
"./S1A_IW_GRDH_1SDV_20170413T224356_20170413T224421_016132_01AA52_90ED.SAFE.zip",
            "class": "File"
        }
    ]
}
```

Appendix E: Revision History

Table 3. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
November 22, 2018	P. Sacramento	1.0	all	issue for submission as pending to OGC TC Charlotte
October 23, 2018	P. Sacramento	.9	all	first version for review

Appendix F: Bibliography

1. Cauchy, A.: OpenGIS® WPS2.0 Transactional Extension Implementation Standard. OGC 13-071r2,Open Geospatial Consortium, http://www.opengeospatial.org/docs/discussion-papers (2013).
2. Goncalves, P., others: OGC Testbed-13: Application Deployment and Execution Service ER. OGC 17-024,Open Geospatial Consortium, http://www.opengeospatial.org/per/17-024.html (2017).
3. Gasperi, J., others: OGC Testbed-14: Authorisation Authentication and Billing Engineering Report. OGC 18-057,Open Geospatial Consortium, http://www.opengis.net/doc/PER/t14-D010 (2018).
4. Chen, C., others: OGC Testbed-13: Cloud ER. OGC 17-035,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-035.html (2017).
5. Goncalves, P., others: OGC Testbed-13: EP Application Package Engineering Report. OGC 17-023,Open Geospatial Consortium, http://docs.opengeospatial.org/per/17-023.html (2017).
6. Sacramento, P., others: OGC Testbed-14: Application Package ER. OGC 18-049,Open Geospatial Consortium, http://www.opengis.net/doc/PER/t14-D008 (2018).
7. Simonis, I., others: OGC Earth Observation Exploitation Platform Hackathon 2018 Engineering Report. Open Geospatial Consortium, http://www.opengeospatial.org/docs/er (2018).
8. Ehbrecht, C., Landry, T., Hempelmann, N., Huard, D., Kindermann, S.: Projects Based on the Web Processing Service Framework Birdhouse. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-4/W8, 43–47 (2018).